2022

# Efficient Evaluation of Probability and Reliability with Digital Integrated Circuits

Suoyue Zhan
*University of Windsor*

# Efficient Evaluation of Probability and Reliability with Digital Integrated Circuits

By

**SUOYUE ZHAN**

A Dissertation
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
at the University of Windsor

Windsor, Ontario, Canada

2022

**Efficient Evaluation of Probability and Reliability with Digital Integrated Circuits**

by

**Suoyue Zhan**

APPROVED BY:

J. Han, External Examiner

University of Alberta

H. Zhang

Department of Biomedical Sciences

H. Wu

Department of Electrical and Computer Engineering

M. Azzouz

Department of Electrical and Computer Engineering

C. Chen, Advisor

Department of Electrical and Computer Engineering

August 30, 2022

# DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION

## I. Co-Authorship

I hereby declare that this thesis incorporates material that is result of joint research, as follows:

- *"Chapters 4 and 5 of the thesis include the outcome of publications which have the following other co-authors: Chunhong Chen (academic advisor). In all cases only my primary contributions towards these publications are included in this thesis, and the contribution of co-author Chunhong Chen was primarily through methodology development, algorithm design, simulation results analysis and manuscript editing".*
- *"Chapter 3 incorporates unpublished material under the supervision of professor Chunhong Chen. In all cases the primary contributions, experimental designs, MATLAB coding, data analysis, interpretation, and writing were performed by myself; The contribution of Chunhong Chen was through the key ideas, interpretation and manuscript writing/editing".*

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

## II. Previous Publication

This thesis includes 3 original papers that have been previously published/submitted to journals/conference for publication, as follows:

| Thesis Chapter | Publication title/full citation | Publication status |
|---|---|---|
| *Chapter [3]* | S. Zhan and C. Chen, “A Hybrid Method for Signal Probability and Reliability Estimation with Combinational Circuits,” *Integration, the VLSI Journal*, 2022 | *“Published”* |
| *Chapter [4]* | S. Zhan and C. Chen, “An Efficient Method for Sequential Circuit Reliability Estimation,” *Proc. 65th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Japan, August 7-10, 2022. | *“Published”* |
| *Chapter [5]* | S. Zhan and C. Chen, “Circuit Reliability Analysis with Consideration of Aging Effect,” *Proc. 35th Symposium on Integrated Circuits and Systems Design* (*SBCCI)*, Brazil, August 22-26, 2022. | *“Published”* |

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as a graduate student at the University of Windsor.

## III. General

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

# ABSTRACT

As complementary metal–oxide–semiconductor (CMOS) devices shrink to nanoscale, digital integrated circuits (ICs) are more susceptible to various environmental parameters, such as temperature, supply voltage, wiring, noise, and fabrication process variations. This would reduce the circuit operation reliability (i.e., the probability that a circuit or component is performing its intended logic function). Signal probability (the probability that a digital signal is producing logic 1) is another factor that measures circuit's dynamic behavior and power dissipation. Research shows that signal probability and reliability within ICs may interact with each other in a complicated way. Generally speaking, as signal probability changes due to input probability variations, so does the signal reliability, and vice versa. This motivates simultaneous evaluation of both for digital ICs towards their performance improvement. However, this evaluation could be a challenge especially for large-scale circuits, due to signal correlations caused by reconvergent fanouts within circuits. Out of two existing evaluation methods, i.e., numerical and analytical methods, the former can give high accuracy level at the cost of expensive computation, while the latter does exactly the opposite.

This thesis provides a hybrid solution by taking advantage of both numerical and analytical methods to achieve fast and accurate evaluation for signal probability and reliability for ICs (including both combinational and sequential circuits). First, we develop a categorization-based analytical model for combinational circuits to deal with a variety of signal correlations. For strongly correlated or independent cases, analytical solutions are applied for accurate results. For cases with moderate correlation strength, we use local bitstream simulations for fast estimation. Our simulation results show that the proposed method is hundreds of times faster than Monte-Carlo (MC) simulation, while keeping almost same level of accuracy.

We then extend the above method to sequential circuits (with finite-state-machine model) for probability and reliability evaluation. Since sequential circuits can be viewed as an unfolded network of combinational logic, our focus is on how both probability and reliability converge to a final stable state over a certain number of cycles/iterations. To improve the efficiency of this convergence process, we propose a two-step-convergence (TSC) model instead of using traditional step-size based convergence. Simulation results show that the proposed method speeds up the process by around 30% on average compared to traditional method while maintaining a high level of accuracy.

Finally, we study the impact of device aging on circuit reliability. After years of operation, CMOS (especially PMOS) devices would experience an increase in their threshold voltage, a phenomenon called Negative Bias Temperature Instability (NBTI). This aging effect leads to the increased gate delay with late arrival time of signals, making circuits temporally unreliable. Threshold voltage changes may also negatively affect the probability that transistors perform intended logical operations, causing them spatially more unreliable. Our investigation focuses on evaluation of the overall reliability at circuit-level by considering both spatial (solely considering the correctness of signal logic values) and temporal (considering the signal arrival time to catch up sampling action) aspects of it. This would help circuit designers predict the circuit lifetime. Simulations on benchmark circuits show that the reliability degradation rate due to aging effect ranges from 1.5% to 8.2% over one-year period, depending on specific circuits.

# ACKNOWLEDGEMENTS

I would like to express my deep sense of gratitude towards my academic advisor Prof. Chunhong Chen for the continuous support of my PhD study and related research. This thesis, as well as the publications, would never be possible without the expertise. He has been extremely patient guiding me into the research world and demonstrating me all the qualities a researcher should have. I could not image having a better mentor than him.

I would like to extend my sincere appreciation to the committee members. Thanks for their valuable comments and advice throughout my PhD study.

Meanwhile, a debt of gratitude is owed to University of Windsor as well as the Department of Electrical and Computer Engineering. Thanks for providing such incredible chance to study here.

Last but not least, I would like to offer my thanks to my family, especially my wife and my parents, for their consistent support and encouragement during the years of my research.

## TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS/SYMBOLS

| | |
|---|---|
| CMOS | Complementary metal–oxide–semiconductor |
| IC | Integrated circuits |
| MC | Monte-Carlo |
| TSC | Two-step-convergence |
| NBTI | Negative bias temperature instability |
| PTM | Probabilistic transfer matrices |
| ITM | Ideal transfer matrix |
| PGM | Probabilistic gate matrix |
| ER | Equivalent reliability model |
| CC-SPRA | Correlation coefficients approach for signal probability-based reliability analysis |
| DI | Dummy input |
| DO | Dummy output |
| VDD | Supply voltage |
| P* | Error-free signal probability |
| P | Signal probability |
| PF | Probability of failure |
| JPV | Error-free joint probability vector |
| JCPM | Joint conditional probability matrix |
| PV | Error-free probability vector |
| RV | Reliability vector |
| CGSP | Cross-gate signal pairs |
| DFF | D-flip-flop |
| SPF | Spatial probability of failure |
| TPF | Temporal probability of failure |
| CLK | Clock signal |
| CDF | Cumulative density function |
| PDD | Probabilistic decision diagram |
| CC | Circuit clustering |
| STA | Static timing analysis |

# LIST OF APPENDICES

# CHAPTER 1
# INTRODUCTION

## 1.1 Research Background

Downscaling of CMOS devices to nanometers raises quite a few new challenges for digital integrated circuit designers [1]. Among them is circuit reliability. With the imprecision of nanoscale fabrication, device reliability can be affected by many hard defects (such as shorts and opens [2-3]) and soft errors which may occur due to environmental variations. High integration density and unsaturated voltage/current are increasingly compounding the problem [4]. The unreliability of fundamental devices will then negatively affect circuit performance at high levels. Thus, it becomes necessary to keep track of not only signal probability, but also signal reliability (or faulty ratio). For large circuits, the difficulties in finding either signal probability or reliability stem from signal correlations due to numerous reconvergent fanouts within circuits. Since the reliability and probability may affect each other, it would make much sense to estimate both of them simultaneously in a circuit. More specifically, for given unreliable circuits, the output reliability depends on their input signal probabilities. Take a two-input AND gate for example. When both inputs have a lower signal probability (i.e., most of time they are logic ‘0’), the output signal would have a higher reliability (i.e., high probability of being a correct value of ‘0’) due to the nature of AND logic. On the other hand, signal probability depends on reliability as well, as will become clear later in the thesis. Since signal reliability is essentially a conditional signal probability (refer to next section for details), evaluating signal reliability is generally more difficult than estimating signal probability.

These challenges, along with the high circuit density, bring up two main requirements for the probability and reliability evaluation method: high accuracy when dealing with correlation, and high efficiency (scalability) when applied to integrated circuits. Currently, there exist numerical and analytical solutions, which are mainly under the zero-delay model and assume gate reliability is a constant.

The numerical method can provide high accuracy results statistically at a cost of extremely long processing time, which makes it a good way to provide benchmark results for probability and reliability evaluation. Analytical methods, different from numerical ones, tend to use some fast models to calculate/estimate signal probability and reliability. Although it is much more efficient and practical to be implemented on integrated circuits, the accuracy level is not satisfying. The major error source is coming from imperfect solutions to signal correlation estimation. Meanwhile, some works' scalability, such as [5], is not good enough with a time complexity of $O(M^2)$, where M is the number of gates.

## 1.2 Objectives

The goal of this thesis is to assist designers in evaluating digital integrated circuit performance accurately and efficiently so that effective adjustments can be applied immediately. More specifically, the performance evaluation includes estimations for signal probability, reliability, and device reliability variations after a certain operation time, which is called the aging effect. All these factors are directly related to critical issues such as size, power consumption, lifetime, etc. Ideally, designers are expecting this evaluation to be accurate and efficient.

However, it is unrealistic to search for an absolute accurate model with maximum efficiency for signal probability and reliability estimations. The best researchers can do is to find a reasonable balance between accuracy and efficiency. Any proposed model should be well-designed to handle signal correlations correctly, and the processing time should be preferably linear/quasi-linear proportional to number of logic gates. Otherwise, the efficiency of the proposed method will be a big concern when applied to integrated circuits.

Due to the lack of full circuit aging effect analysis, we are also trying to introduce a model to help designers evaluate circuit reliability changes with consideration of long-term intense operation. The model should be straightforward, easy to follow and implement to any digital circuit.

## 1.3 Main Contributions

Main contributions of this thesis are as follows:

- developing a new, hybrid approach for signal correlation evaluation, unlike others which are either pure numerical or analytical.

- introducing a categorization system for correlations based on circuit connectivity.

- achieving linear time complexity to circuit size by evaluating reliability in terms of conditional signal probabilities, which allows the use of gate propagation model

- achieving accurate evaluation for error-free probability (the probability that a signal is 'expected' to produce logic '1' with given inputs) and reliability simultaneously during analytical and numerical analysis, getting rid of extra time obtaining signal probability from other methods/tools/software.

- proposing a new convergence method (Two-step convergence) for sequential circuit analysis, reducing the number of iterations required by around 30%

- covering the shortage of circuit-level aging effect evaluation by extending existing device-level analysis.

- creating a new index to describe circuit overall reliability by considering the cross-impact of spatial and temporal reliability simultaneously.

The basic idea of the hybrid model is to categorize signal correlations by investigating a local circuit topology, and then apply specific solution methods accordingly (including analytic computation, bitstream simulation, or their combination). All category solutions, along with the conditional-probability-based gate propagation model, are well-designed to obtain signal error-free probability and reliability simultaneously. It should be noted that actual signal probability can be easily obtained using error-free probability and reliability, but not vice versa. With this strategy, the proposed model achieves not only linear time complexity to circuit size efficiency (benefits from the propagation-based model), but also further efficiency improvement by saving the processing time for signal error-free probabilities. Meanwhile, the simulation shows decent accuracy for the proposed model. When this model is applied to sequential circuits, instead of using traditional step-size convergence, we use the first few iterations as a trial, and obtain a far-better initial value for the following iterations by regression analysis. For evaluations considering the aging effect, we propose a model to look at circuit reliability from a logic perspective (or spatial viewpoint) as well as a delay aspect (or temporal viewpoint), in order to build a single index to evaluate the overall reliability of circuit outputs. We also extend and combine some existing models to accomplish reliability analysis, starting from transistor level to gate level and finally to full circuit reliability estimation.

## 1.4 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 reviews the existing methods on above mentioned topics and their pros and cons. Chapter 3 presents the hybrid model in detail. Chapter 4 illustrates how the proposed hybrid model could be applied onto sequential circuits, as well as introduces the newly developed convergence technique. Chapter 5 focuses on circuit-level aging effect analysis. Finally, chapter 6 concludes the thesis.

# CHAPTER 2

# LITERATURE REVIEW

Models for signal probability and reliability evaluation on combinational and sequential circuits are well discussed, but there is still space for improvement for better accuracy and/or efficiency. For aging effect evaluation, although there are numerous works on device (transistors, gates) reliability changes, little has been done on circuit-levels.

## 2.1 Evaluation of Signal Probability and Reliability Under Zero-delay Model

Gate and wire delay are important factors in circuit design and cannot be ignored in real circuit analysis. However, when obtaining output probability and reliability from a logical values perspective, researchers tend to assume that gate and wire delays are zeros. This zero-delay model allows direct investigation to circuit logical functionality without disturbance coming from temporal issues such as delay and glitches, and the reliability evaluated under zero-delay model is called spatial reliability, meaning that it is only related to circuit connectivity/structure without considering any temporal factors.

### 2.1.1Evaluation for Combinational Circuits

Evaluation of signal reliability is mainly achieved by two methods: numerical method and analytical method. The numerical method, such as Monte-Carlo simulation [6] is to estimate circuit reliability by randomly generating input vectors (with logic values 0 or 1 only) and evaluating the outputs. This approach is straightforward and easy to implement, and its accuracy is generally proportional to the number of iterative simulations. The major drawback of this method is that it requires long processing times, making it almost impossible for large integrated circuits. Therefore, it usually serves as a tool to provide standard results for evaluating the accuracy of other methods.

Analytical methods, on the other hand, represent a class of approaches that use analytical models with gate reliability, which is inspired by Von Neumann’s work

[7]. This reliability, assigned to each gate independently, represents a probability that a given gate maintains output logic value correctly after any logic functions, or equivalently, the probability that bit flip (from 0 to 1 or 1 to 0) event will not happen. These analytical models using gate reliability are much more efficient than numerical methods at a cost of accuracy reduction since the correlations between signals are difficult to evaluate. Another challenge for analytical methods is how to increase the efficiency as much as possible for probability and reliability evaluation.

In 2008, S. Krishnaswamy et al. proposed a probabilistic transfer matrix model (PTM) [8]. This work brings up the idea that all logic gates can be assigned by a pair of transfer matrix ITM and PTM (ideal transfer matrix and probabilistic transfer matrix), encountering error-free and erroneous environment respectively. The entire propagation procedure from input to output is then transferred into matrices multiplications, which speed up the evaluation process significantly. However, since the matrix tensor product is required during computation, the memory capacity needed to store the intermediate results is extremely high. On the other hand, signal correlations are handled by storing all multi-output gates and all correlated signals are considered simultaneously (maximum 10 correlated signals) regarding joint probability. These greatly increases the time complexity of the algorithm, especially for circuits with complex connectivity. Besides, considering the high difficulty level of estimating joint probability accurately for multiple (more than 2) signals, algorithm accuracy can still be a secondary concern in general.

Another work [9] presented a model using Boolean difference calculus to calculate circuit output reliability. Although the time complexity to number of gates is linear, the computational process is based on an assumption that spatial correlation coefficients are already available/obtained from some other methods, which means this method itself is not capable for correlation evaluation. Thus, the accuracy level is entirely based on the accuracy of the applied correlation coefficients.

To better evaluate signal correlations, a multiple-state strategy has been used by [10-12]. The main idea is to generate a set of ‘copies’ circuits and each of them represents a specific logic state (0 or 1) for the reconvergent-fanouts. This method is usually accurate enough since when reconvergent-fanouts are fixed to certain logic values, the other signals’ probability and reliability can be easily calculated accurately. However, the major drawback of this idea is that when multiple reconvergent-fanouts exist, the number of input pattern considered will be increasing exponentially, making the evaluation process much less efficient.

In a following milestone work called “probabilistic gate matrix” (PGM) [13], the signal correlations are handled by investigating output reliability for specific reconvergent fanout input pattern $j$ (with $n$ inputs) and take weighted summation as follows:

$$R_{out} = \sum_{j=0}^{2^n-1} P_j R_{out_j} \tag{2.1}$$

where $R_{out}$ is output reliability, $P_j = P\{inputs = j\}$ , and $R_{out_j}\{output = 1|inputs = j\}$. This approach is pretty accurate but requires a certain amount of time to list all possible input patterns, resulting in a time complexity of $O(M2^{M_f})$, where $M$ is the number of gates and $M_f$ is the number of correlated signals. It should be noted that if we ignore the extra processing time for signal correlations, the fundamental propagation-based model achieves a linear time complexity regarding the number of gates, which is the best in expectation so far. Therefore, this framework is widely used in the following years.

Meanwhile, researchers tried to find solutions based on probabilistic decision diagrams (PDD) [14] as well. This is a good attempt to study circuit probability, reliability, and correlation topologically. Again, the main drawback of the proposed model is that the time complexity is not linear, but highly related to the number of reconvergent-fanouts. For example, from [14], the CPU time for the circuit ‘duke2’ with only 88 gates takes 14.76 seconds, and for the circuit ‘9symml’ with 108 gates it becomes 0.22 seconds. The instability of efficiency greatly reduces the scalability of PDD. A similar problem has been observed in [15] as well.

Bayesian network [16] is another attempt using graphical method for probability and reliability evaluation. However, it is quite hard to address signal correlations using the Bayesian network, and the accuracy level becomes a concern. The efficiency and accuracy are both moderate, but the scalability of this work is still questionable since the processing time to build up corresponding networks can be unacceptably long for integrated circuits.

To make the evaluation method more scalable, [17] has bring up a model using correlation coefficient $C_{AB}$ for signal pair (A, B) which is defined as:

$$C_{AB} = \frac{P(AB)}{P(A)P(B)} \tag{2.2}$$

where $P(k) = \Pr\{k = 1\}, k = A, B, AB$. This model requires only one-pass to the circuit, making it very fast. However, considering the fact that reliability correlation is different from probability correlation, this method becomes less accurate for circuit reliability evaluation. More recently, a similar idea has been used in equivalent reliability model (ER) [18]. With a different definition of the correlation coefficient, the accuracy level of estimation for signal reliability has been improved, but still not enough.

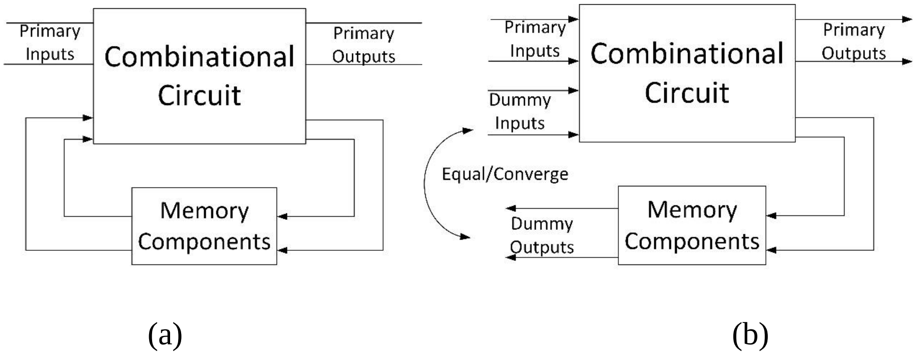
Another example of using correlation coefficients is the bitstream simulation model (CC-SPRA) [5]. Although the big framework is still a propagation-based analytical method, the evaluation of all correlation coefficients is achieved by ‘bitstream simulation’, which is a small-scale Monte-Carlo simulation instead. More specifically, for any given signal pair, bit value sequences are generated using available information. The correlation coefficient is then calculated using information obtained by ‘counting’ the bit values from generated sequences for signals of interest. With a good choice of the length of the bitstream, it provides accurate results without sacrificing too much efficiency. A major drawback of this work is that bitstream generation requires a circuit connectivity analysis from primary inputs. When a signal pair of interest is approaching outputs within integrated circuits, this investigation process will be time-consuming.

In 2011, circuit clustering (CC) [19] has been introduced as a speed-up technique. The main idea is to separate a circuit into multiple clusters following its topological order and calculate conditional signal probabilities based on given values of its previous cluster only. The output probability can then be obtained by multiplying these conditional probabilities all together. Since the maximum size of matrices used to describe conditional probabilities is smaller, the algorithm efficiency has been improved a lot. However, this model is too optimistic since any interconnect signal between clusters will increase the difficulty of finding the individual conditional probability. Therefore, it is only capable for circuits with simple connectivity and clear hierarchy.

### 2.1.2 Evaluation for Sequential Circuits

Since most of the real-world circuits are sequential, there is much research work on this area. From a hardware and design perspective, for example, [20] has developed a low-power, non-volatile and radiation-hardened latch against single event upsets and [21] takes advantage of reconfigurable pulsed latches to achieve low-power design with reliability enhancement.

As for software simulation-based work, a soft-error detection and correction system has been introduced in [22], and [23] has investigated the critical path identification technique for sequential circuit reliability analysis. However, neither of their discussions mentioned how to improve the efficiency for sequential circuit probability and reliability directly.



(a) (b)

Figure 1. (a) Sequential circuit, (b) combinational equivalent of (a)

As discussed before, most of the introduced methods for combinational circuit evaluation can be expanded to sequential circuits. Generally, as shown in [24], a sequential circuit can be transferred into a combinational equivalent by unlooping the memory component as shown in Fig. 1.

It should be noted that this combinational equivalent requires the probability and reliability of the dummy inputs (DIs) are exactly the same as those of the dummy outputs (DOs) since they are initially connected. However, since this information is not given initially, we have to assign certain probability and reliability to DIs first and let the DIs and DOs converge to each other. This brings up a higher efficiency requirement for efficiency of combinational evaluation models since they are applied continuously until circuits' stable status is found.

The current convergence process is mostly driven by the following equation:

$$Q_{DI}^{n+1} = Q_{DI}^{n} + \eta \cdot (Q_{DO}^{n} - Q_{DI}^{n}) \tag{2.3}$$

where $Q$ represents either the probability or reliability element of DIs and DOs, $\eta$ is the step size, and $n$ is the number of iterations. The step size is the key to determining how fast this convergence process is. Larger $\eta$ means a faster approach from DIs to DOs, while smaller $\eta$ are usually used during the fine-tuning process. In the extreme case, if $\eta = 1$, the new DI value is directly substituted by the current DO value. However, the optimal $\eta$ is difficult to find, and for each circuit the best $\eta$ can be different as well.

In [24-26], $\eta = 0.1$ is applied for the entire convergence process without any change, which leads to extremely tedious convergences. In fact, there is no reason to keep $\eta$ to such a small value for the first few iterations, where the probability and reliability of DIs and DOs are still far from each other. Such discussions on the choices of $\eta$ have been done in [27], but no exact solution was given. The only conclusion is that $\eta$ should be a dynamic value, starting with a relatively large one, such as 0.5, and gradually decreasing as DIs and DOs are getting closer.

The tricky part about choosing $\eta$ is that it is sensitive to circuit structure as well as input probability and reliabilities. For any given circuit, the optimal value of $\eta$ is unique, which is also extremely difficult to find. Therefore, it is understandable that current researchers tend to use a universal model for all sequential circuits at a cost of reduction of efficiency.

Another important parameter, which is ignored by researchers, is the initial value of $Q_{DI}^{1}$. A good initial value can significantly reduce the number of iterations required. In most work, DIs shares the same probability and reliability as primary inputs at the beginning of convergence. Although this setup is easy to follow, it is never the best choice. This thesis will discuss how to find a better initial value to speed up the entire convergence process as well.

## 2.2 Evaluation of Signal Probability and Reliability Considering Delay and Aging Effect

In real world circuits, gate delay will affect output probability and reliability, which cannot be ignored. For instance, there is a great chance for the sampled output data to be incorrect if the expected logic value arrives later than the designed guard band, even if the logic value is correct. In contrast to spatial reliability, temporal reliability is used to describe the probability that a signal arrives on time. Besides, gate reliability is usually not a constant either. In fact, after operating intensively for years, transistor threshold voltages tend to increase (especially for PMOS). The reason is that accumulated opposite charges during operation will cancel the upcoming gate voltage and increase the threshold voltage if there is not enough ‘rest’ time to let the charges release, which is known as Negative Bias Temperature Instability (NBTI). Considering many other performance (such as power and area) requirements, circuits are usually designed to have their critical path delays around 10-30% shorter than the required clock period [28]. However, an accurate evaluation of the performance degradation caused by NBTI would be less likely since it strongly depends on dynamic operation conditions, such as supply voltage (VDD), environmental temperature (T), and signal probability (Pin) [29]. Even if a guard band is designed using extreme environmental parameters (such as high temperatures and high supply voltages), the uncertainty of signal probabilities makes it hard to accurately evaluate the aging effect.

Some prior work has successfully built up short-term and long-term estimation models for the change in threshold voltage $\Delta V_{th}$ due to NBTI effect through experiments, as shown in [30, 31, 32]. Based on their results, [33] further simplified the proposed model assuming the environmental conditions (such as supply voltage and temperature) were fixed.

### 2.2.1 Aging Effect on Spatial Reliability

The spatial reliability refers to the probability that a signal will produce the correct logic value, in contrast to the temporal reliability which is related to delay. With the information of $\Delta V_{th}$, researchers in [33] have proposed models for gate-level reliability aging analysis by looking into transistor performance. [33] shows that the designed transistor threshold voltages $V_{th}$ can fluctuate due to oxide thickness variations [34], line edge roughness [35], polysilicon granularity, and the combined effects of all these factors. Simulations and some sample analysis [36-49] conclude that $V_{th}$ fluctuations will lead to unexpected behavior (turning on/off at incorrect gate-to-source voltage), which can be treated as a Gaussian distribution. The probability of failure (PF) for transistors is then calculated by assuming that transistors are ideal switches at its actual threshold voltage. The logic gates, formed by these transistors, are found to be experiencing a certain probability to fail to produce the correct logic function, which has been discussed in [40]. Therefore, the output logic value could be incorrect.

### 2.2.2 Aging Effect on Temporal Reliability

The increased threshold voltage will make the transistors harder to be turned on and reduce drain current, leading to a longer gate propagation delay ([41, 42]). With the extra delay accumulated, there is a chance that the output signal may miss the sampling action of the current clock cycle and produce an incorrect sampling result for the output value, which is called temporal unreliability. Fortunately, it is possible to efficiently calculate gate delay variation using $\Delta V_{th}$ with the linear approximation model for logic gates delay approximation developed in [43]. The output delay is found using static timing analysis (STA) [44, 45], which can be done easily using SPICE tools, as indicated in [46]. The temporal reliability of output can be obtained by comparing the obtained output delay with the designed CLK frequency requirement.

The above-mentioned work provides a clear picture of how aging effects can change single device performance with respect to spatial and temporal reliability. However, it is still a big challenge for designers to accurately determine the lifetime of designed circuits due to the lack of full circuit estimations for reliability aging issues while considering both types of reliability.

# CHAPTER 3 PROBABILITY AND RELIABILITY ESTIMATION FOR COMBINATIONAL CIRCUITS

In this chapter, we first describe some background of digital signal probability and reliability, and then introduce the joint probability vector (JPV) and joint conditional probability matrix (JCPM) for any given signal pair. With this information, we propose a hybrid model based on correlation categorization and show the corresponding solutions. The simulation results and summary are followed.

## 3.1 Signal Probability and Joint Probability Vector

### 3.1.1 Signal Probability

For any signal D in a digital circuit, its error-free probability is defined as follows:

$$\begin{cases} P_D^{0^*} = \Pr\{D^* = 0\} \\ P_D^{1^*} = \Pr\{D^* = 1\} \end{cases} \tag{3.1}$$

where $D^*$ is the error-free version of D (throughout the thesis, the symbol '*' is used to indicate 'error-free', which refers to the condition where all gates and input signals are reliable), and $\boldsymbol{P_D^*} = (P_D^{0^*} \;\; P_D^{1^*})$ is called the error-free probability vector (PV) of signal D with $P_D^{0^*} + P_D^{1^*} = 1$. In an unreliable circuit, the reliability of signal D, (denoted by $r_D$) is defined as the probability that the signal generates an intended logic value, i.e., $r_D = Pr\{D = D^*\}$. Similarly, for a logic gate $g$, its reliability (denoted by $r_g$) is defined as the probability that an intended logic value at its output is produced for given inputs (either erroneous or error-free). It should be mentioned that the reliability for gate output may not necessarily be lower than its input signal reliabilities for given $r_g$ due to logic masking. For instance, if one input of a reliable AND gate has a reliability of 1 but is fixed to logic 0, the output reliability will always be 1 no matter how low the reliability of the other input could be.

### 3.1.2 Joint Probability Vector

For any input pair (A, B) of a logic gate $g$, we define the joint (error-free) probability vector (JPV) and joint conditional probability matrix (JCPM) as follows:

$$JPV: \boldsymbol{P}^*_{\boldsymbol{AB}} = \left(P_{AB}^{00^*}\ \ P_{AB}^{01^*}\ \ P_{AB}^{10^*}\ \ P_{AB}^{11^*}\right) \tag{3.2}$$

where $P_{AB}^{ij^*} = \Pr\{(AB)^* = 'ij'\}, i, j = 0\ or\ 1$ , $\sum P_{AB}^{ij*} = 1$ . If (A, B) are independent, we have $P_{AB}^{ij^*} = P_A^{i^*} \cdot P_B^{j^*}$. If (A, B) are correlated, this value is evaluated otherwise.

## 3.2 Signal Reliability and Joint Conditional Probability Matrix (JCPM)

### 3.2.1 Signal Reliability

The reliability $r_D$ for signal D can be expressed in terms of probabilities of error-free signal $D^*$and conditional probabilities of signal D as follows:

$$r_D = Pr\{D = D^*\}$$

$$= Pr\{D = 1 \cap D^* = 1\} + Pr\{D = 0 \cap D^* = 0\}$$

$$= \Pr\{D = 1 \mid D^* = 1\} \cdot P_D^{1^*} + \Pr\{D = 0 \mid D^* = 0\} \cdot P_D^{0^*}$$

$$= R_D^1 \cdot P_D^{1^*} + R_D^0 \cdot P_D^{0^*} \tag{3.3}$$

where $R_D^1$ $and$ $R_D^0$ are conditional probabilities:

$$\begin{cases} R_D^1 = \Pr\{D = 1 \mid D^* = 1\} \\ R_D^0 = \Pr\{D = 0 \mid D^* = 0\} \end{cases} \tag{3.4}$$

Let $\boldsymbol{R_D} = (R_D^0 \;\; R_D^1)$, which is known as the reliability vector (RV) of signal D. In other words, the reliability of the signal D is associated with a pair of conditional probabilities $(R_D^0 \;\; R_D^1)$.

### 3.2.2 Joint Conditional Probability Matrix

Similar to JPV definition, we have Joint Conditional Probability Matrix defined as follows:

$$JCPM\text{: } \boldsymbol{C_{AB}} = \begin{pmatrix} P_{00}^{00} & P_{00}^{01} & P_{00}^{10} & P_{00}^{11} \\ P_{01}^{00} & P_{01}^{01} & P_{01}^{10} & P_{01}^{11} \\ P_{10}^{00} & P_{10}^{01} & P_{10}^{10} & P_{10}^{11} \\ P_{11}^{00} & P_{11}^{01} & P_{11}^{10} & P_{11}^{11} \end{pmatrix} \tag{3.5}$$

where the element $P_{ij}^{kl} \quad = \Pr\{(AB) = 'kl' \mid (AB)^* = 'ij'\}, i, j, k, l = 0 \; or \; 1,$ and summation of the four elements in each row is 1 unless $P_{AB}^{ij^*} = 0$ for a specific value of $i$ and $j$, in which case $P_{ij}^{kl} \quad (k, l = 0\,, 1)$ in (3.5) is simply defined as 0. Once both $\boldsymbol{P_{AB}^*}$ and $\boldsymbol{C_{AB}}$ in the above (3.4) and (3.5) are available, the PV and RV for the output of gate $g$ can be derived through a probability propagation to be presented in the next section.

TABLE I. CONDITIONAL RELIABILITIES FOR CIRCUIT C432 WITH $r_g = 0.999$

| Node # | RV under Independent assumption | RV Considering signal correlations | Errors in percentage (%) |
|---|---|---|---|
| 1 | (0.9977, 0.9862) | (0.9958, 0.9778) | (0.19, 0.86) |
| 2 | (0.9932, 0.9218) | (0.9866, 0.9681) | (0.67, 4.78) |
| 3 | (0.9953, 0.9346) | (0.9785, 0.9645) | (1.72, 3.10) |
| 4 | (0.9256, 0.9290) | (0.9673, 0.9927) | (4.31, 6.42) |
| 5 | (0.9101, 0.9751) | (0.9692, 0.9813) | (6.10, 0.64) |
| 6 | (0.9032, 0.9574) | (0.9700, 0.9832) | (6.89, 2.63) |
| 7 | (0.8719, 0.9870) | (0.9684, 0.9786) | (9.97, 0.86) |
| Average | − | − | (4.26, 2.75) |

For small circuits, both PVs and RVs for all signals can be found using MC simulation. However, for large circuits with signal correlations, it would be impractical to do so by MC simulation which would be too time-consuming. Generally speaking, the PV and RV for any signal depends on input signal probabilities, signal correlations and/or gate reliabilities. Intuitively, if a circuit contains no signal correlations, the conditional reliability pair $(R_D^0\ R_D^1)$ can be easily propagated through gates from circuit inputs to its outputs. Unfortunately, most circuits have quite a few reconvergent fanouts which lead to signal correlations. To get a sense of what could happen if we ignore these correlations, we did MC simulations for benchmark circuit C432 with 160 gates under the assumption that all signals are independent with gate reliability of 0.999, and that all primary inputs are reliable with their signal probability of 0.5. The results are summarized in Table I, where the errors in evaluating the RV go up to 10%. This indicates that the signal correlations play an important role in circuit reliability evaluation.

## 3.3 Gate Level Probability and Reliability Propagation

In this section, we first show how to obtain the (error-free) probability vector (PV) and reliability vector (RV) for the output D of an unreliable logic gate assuming the availability of $\boldsymbol{P_{AB}^{*}}$ and $\boldsymbol{C_{AB}}$, where A and B are the two inputs of the gate. We then present detailed analysis (either analytic or statistic simulation) by considering various correlations with the signal pair (A, B) to find both $\boldsymbol{P_{AB}^{*}}$ and $\boldsymbol{C_{AB}}$. It should be mentioned that the above obtained PV and RV for the signal D will be required and used to find both JPV and JCPM for other signal pairs (if they are the signal D's transitive fanouts). Thus, the whole computation is a recursive gate-by-gate propagation process, as will become clear later in the thesis. Finally, the pseudo-codes of the overall algorithm are provided with some discussions.

Assume both $\boldsymbol{P_{AB}^{*}}$ and $\boldsymbol{C_{AB}}$ are available and that the logic gate under consideration is an AND gate with reliability of $r_g$. To obtain the PV of its output D (i.e., $\boldsymbol{P_{D}^{*}}$), we partition the $\boldsymbol{P_{AB}^{*}}$ as follows:

$$\boldsymbol{P_{AB}^{*}} = \left(P_{AB}^{00^*}\ P_{AB}^{01^*}\ P_{AB}^{10^*} \ \vdots\ P_{AB}^{11^*}\right) = (\boldsymbol{P0}\ \vdots\ \boldsymbol{P1}) \tag{3.6}$$

where $\boldsymbol{P0}$ = $(P_{AB}^{00^*}\ P_{AB}^{01^*}\ P_{AB}^{10^*})$ and $\boldsymbol{P1}$ = $(P_{AB}^{11^*})$. Under zero-delay model, the $P_{D}^{0^*}$ and $P_{D}^{1^*}$ are the sum of all elements in $\boldsymbol{P0}$ and $\boldsymbol{P1}$, respectively, i.e.,

$$\begin{cases} \boldsymbol{P_{D}^{0^*}} = SUM(\boldsymbol{P0}) \\ \boldsymbol{P_{D}^{1^*}} = SUM(\boldsymbol{P1}) \end{cases} \tag{3.7}$$

The $\boldsymbol{C_{AB}}$ is partitioned accordingly as follows:

$$\boldsymbol{C_{AB}} = \begin{pmatrix} P_{00}^{00} & P_{00}^{01} & P_{00}^{10} & | & P_{00}^{11} \\ P_{01}^{00} & P_{01}^{01} & P_{01}^{10} & | & P_{01}^{11} \\ P_{10}^{00} & P_{10}^{01} & P_{10}^{10} & | & P_{10}^{11} \\ - & - & - & + & - \\ P_{11}^{00} & P_{11}^{01} & P_{11}^{10} & | & P_{11}^{11} \end{pmatrix} = \begin{pmatrix} \boldsymbol{CP1} & \boldsymbol{CP2} \\ \boldsymbol{CP3} & \boldsymbol{CP4} \end{pmatrix} \tag{3.8}$$

If $r_g = 1$, the RV for the output D is expressed as $\boldsymbol{R'_D} = \left(R_D^0{}' \; R_D^1{}'\right)$, where:

$$\begin{cases} R_D^0{}' = SUM(\mathbf{P0} * \mathbf{CP1})/\boldsymbol{P_D^{0^*}} \\ R_D^1{}' = SUM(\mathbf{P1} * \mathbf{CP4})/\boldsymbol{P_D^{1^*}} \end{cases} \tag{3.9}$$

When $r_g$ < 1 in general, consider the following two cases which would lead to a reliable output D: (a) both D*’* (i.e., D when $r_g$ = 1) and the gate are reliable, and (b) both D*’* and the gate are unreliable, which also produces a correct (reliable) value of D. The probability of case (a) is given by $r_g \cdot R_D$*’*, while the probability of case (b) is given by (1 – $r_g$)·(1 – $R_D$*’*). The summation of these two probabilities gives the output reliability, i.e., $R_D$ = $r_g$·$R_D$*’* + (1–$r_g$)·(1–$R_D$*’*) = (2$r_g$–1)·$R_D$*’* + (1–$r_g$). Applying this result to both $R_D^0$ and $R_D^1$ gives the RV for D in a vector form as follows:

$$\boldsymbol{R_D} = (2r_g - 1) \cdot \boldsymbol{R'_D} + \boldsymbol{I_g} \tag{3.10}$$

where

$$\boldsymbol{I_g} = (1 - r_g \quad 1 - r_g) \tag{3.11}$$

For any 2-input gates other than AND logic, similar derivations can be done to find the PV and RV of its output for given $\boldsymbol{P_{AB}^*}$ and $\boldsymbol{C_{AB}}$, except that (3.6) through

(3.8) shall be partitioned in different ways accordingly. The question to ask now is how to find both JPV and JCPM (i.e., $\boldsymbol{P}^{*}_{\boldsymbol{AB}}$ and $\boldsymbol{C}_{\boldsymbol{AB}}$) as defined in (3.4) and (3.5), which will be answered in the following section III.B.

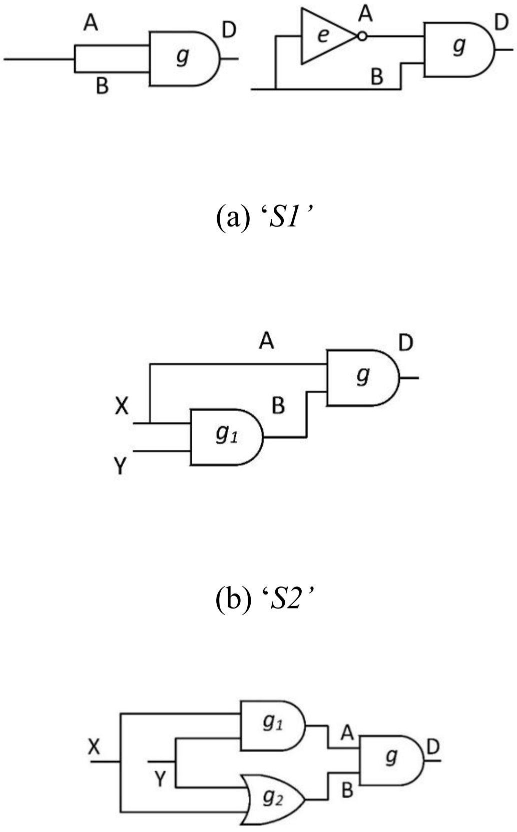
## 3.4 Correlation Categorization and JCPM Estimation

For any signal pair of (A, B), finding its JPV and JCPM requires considerations of the correlations between A and B, which depends on the connectivity of themselves and/or their transitive fan-ins and deserves detailed analysis. In what follows, we first define three different categories, i.e., categories '*S'*', '*N'*' and '*I'*', to represent 'strong', 'not-strong' and 'independent' correlations, respectively, before calculating or estimating both JPV and JCPM.

### 3.4.1 Category 'S'

There are three sub-categories in this strong-correlation category '*S'*'. They are named as '*S1'*', '*S2'*', and '*S3'*', as shown in Fig. 2. '*S1'* refers to a situation where A and B are the same signal (Note that '*S1*' rarely happens for any two-input gate as it would be degenerated to an inverter or buffer. However, we still define it for completeness of correlation categorization).

'*S2'* represents the case where one of A and B is an immediate fan-in of the other. If A and B are driven by two different gates which share same inputs, then it is defined as '*S3'*'. Throughout the thesis, gates (such as $g$ , $g_1$ and $g_2$) in all figures represent any type of 2-input logic gate unless otherwise stated. Also, inverter or buffer would be ignored because they have no effect on correlation category. Therefore, both cases shown in Fig.2 (a) are considered as '*S1'*'. The characteristic of the category '*S'* is that the correlation between A and B is so strong that their JPV and JCPM can be calculated directly, as discussed below. It should be noted that only JCPMs of input pairs in previous levels of gates are treated as 'available', which are obtained before the propagation reaches the current gate.



(a) ‘*S1’*

(b) ‘*S2’*

(c) ‘*S3’*

Figure 2. Examples of correlation category ‘*S’*.

For category ‘*S1’* in the left of Fig. 2 (a) where A and B are a same signal, the JPV and JCPM for (A, B) are given by:

$$\boldsymbol{P}^{*}_{\boldsymbol{AB}} = \begin{pmatrix} P_A^{0^*} & 0 & 0 & P_A^{1^*} \end{pmatrix} \qquad (3.12)$$

$$\boldsymbol{C}_{\boldsymbol{AB}} = \begin{pmatrix} R_A^0 & 0 & 0 & 1-R_A^0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1-R_A^1 & 0 & 0 & R_A^1 \end{pmatrix} \qquad (3.13)$$

For category ‘*S2’* of Fig. 2 (b), the JPV of (A, B) is given by:

$$\boldsymbol{P}_{\boldsymbol{AB}}^{*} = \boldsymbol{P}_{\boldsymbol{XY}}^{*} \cdot \boldsymbol{M}_{\boldsymbol{S_2}}^{*} \tag{3.14}$$

where $\boldsymbol{P}_{\boldsymbol{XY}}^{*}$ is the JPV of (X, Y), which is assumed to be available, and $\boldsymbol{M}_{\boldsymbol{S_2}}^{*}$ is a gate-dependent probability propagation matrix for gate *g1*. For instance, if *g1* is an AND gate, the corresponding $\boldsymbol{M}_{\boldsymbol{S_2}}^{*}$ is given by:

$$\boldsymbol{M}_{\boldsymbol{S}_2}^{*} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.15}$$

The JCPM of (A, B) in Fig. 2(b) can also be calculated by using the JPV and JCPM of (X, Y) as well as the information about gate *g1*. To make the calculation easier, this can be done by initially assuming the *g1*'s reliability of $r_g = 1$ and then extending it to the general case with any value of $r_g$. First, with $r_g = 1$, one can find the JCPM of (A, B), denoted as $\boldsymbol{C_{AB}}'$, depending on the gate type of *g1*. For instance, if *g1* is an AND gate, $\boldsymbol{C_{AB}}'$ is expressed as:

$$\boldsymbol{C_{AB}}' = \begin{pmatrix} P_{00}^{00} & 0 & P_{00}^{10} & P_{00}^{11} \\ 0 & 0 & 0 & 0 \\ P_{10}^{00} & 0 & P_{10}^{10} & P_{10}^{11} \\ P_{11}^{00} & 0 & P_{11}^{10} & P_{11}^{11} \end{pmatrix}_{AB} \tag{3.16}$$

where all elements can be calculated from both JPV and JCPM of (X, Y). For example,

$$\begin{cases} (P_{00}^{00})_{AB} = \dfrac{P_{XY}^{00^*}[(P_{00}^{00})_{XY} + (P_{00}^{01})_{XY}] + P_{XY}^{01^*}[(P_{01}^{00})_{XY} + (P_{01}^{01})_{XY}]}{P_{XY}^{00^*} + P_{XY}^{01^*}} \\ (P_{00}^{10})_{AB} = \dfrac{P_{XY}^{00^*} \cdot (P_{00}^{10})_{XY} + P_{XY}^{01^*} \cdot (P_{01}^{10})_{XY}}{P_{XY}^{00^*} + P_{XY}^{01^*}} \\ (P_{00}^{11})_{AB} = \dfrac{P_{XY}^{00^*} \cdot (P_{00}^{11})_{XY} + P_{XY}^{01^*} \cdot (P_{01}^{11})_{XY}}{P_{XY}^{00^*} + P_{XY}^{01^*}} \end{cases} \tag{3.17}$$

The other elements can be found similarly. For simplicity, (3.16) is rewritten in matrix as:

$$\boldsymbol{C_{AB}}' = (\boldsymbol{M}_{\boldsymbol{S_2}}^{*})^{\boldsymbol{T}} \cdot \boldsymbol{T}_{\boldsymbol{S_2}} \cdot \boldsymbol{C_{XY}} \cdot \boldsymbol{M}_{\boldsymbol{S_2}}^{*} \tag{3.18}$$

where $(\boldsymbol{M}_{\boldsymbol{S_2}}^{*})^{T}$ represents the transpose of $\boldsymbol{M}_{\boldsymbol{S_2}}^{*}$, $\boldsymbol{T}_{\boldsymbol{S_2}}$ is a transition matrix for category '*S2*', depending on the gate type of $g_1$. For instance, if $g_1$ is an AND gate, the $\boldsymbol{T}_{\boldsymbol{S_2}}$ is given by

$$\boldsymbol{T}_{\boldsymbol{S_2}} = \boldsymbol{diag}\left(\frac{P_{XY}^{00^*}}{P_{XY}^{00^*} + P_{XY}^{01^*}}, \frac{P_{XY}^{01^*}}{P_{XY}^{00^*} + P_{XY}^{01^*}}, 1, 1\right) \tag{3.19}$$

Secondly, for a general case with unreliable g1 (i.e., $r_g < 1$), the JCPM of (A, B) is modified to

$$\boldsymbol{C_{AB}} = \boldsymbol{C_{AB}}' \cdot \boldsymbol{M}_{\boldsymbol{g_1}}^{\boldsymbol{B}} \tag{3.20}$$

where $\boldsymbol{C_{AB}}'$ is given by (3.18), and $\boldsymbol{M}_{\boldsymbol{g_1}}^{\boldsymbol{B}}$ represents the modification due to the unreliable $g_1$ and is given by

$$\boldsymbol{M}_{\boldsymbol{g_1}}^{\boldsymbol{B}} = \begin{pmatrix} r_g & 1-r_g & 0 & 0 \\ 1-r_g & r_g & 0 & 0 \\ 0 & 0 & r_g & 1-r_g \\ 0 & 0 & 1-r_g & r_g \end{pmatrix} \tag{3.21}$$

For category '*S3'* of Fig. 2 (c), the computation procedure for the JPV and JCPM is similar to the above category '*S2'* except that an extra gate *g2* shall be considered. More specifically, the JPV of (A, B) in this case is given by:

$$\boldsymbol{P}_{\boldsymbol{AB}}^{*} = \boldsymbol{P}_{\boldsymbol{XY}}^{*} \cdot \boldsymbol{M}_{\boldsymbol{S_3}}^{*} \tag{3.22}$$

where $\boldsymbol{M}_{\boldsymbol{S_3}}^{*}$ is the joint propagation matrix defined by both $g_1$ and $g_2$. In Fig. 2 (c), if the gates $g_1$ and $g_2$ are AND and OR logic, respectively, the corresponding $\boldsymbol{M}_{\boldsymbol{S_3}}^{*}$ is given by:

$$\boldsymbol{M}_{\boldsymbol{S_3}}^{*} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.23}$$

Similar to the derivation of (3.18) and (3.20), we define a transition matrix for category '*S3'* as $\boldsymbol{T}_{\boldsymbol{S_3}}$, and express the JCPM of (A, B) for category '*S3'* of Fig. 2 (c) as:

$$\boldsymbol{C}_{\boldsymbol{AB}} = \left[\left(\boldsymbol{M}_{\boldsymbol{S_3}}^{*}\right)^{\boldsymbol{T}} \cdot \boldsymbol{T}_{\boldsymbol{S_3}} \cdot \boldsymbol{C}_{\boldsymbol{XY}} \cdot \boldsymbol{M}_{\boldsymbol{S_3}}^{*}\right] \cdot \boldsymbol{M}_{\boldsymbol{g_1}}^{\boldsymbol{A}} \cdot \boldsymbol{M}_{\boldsymbol{g_2}}^{\boldsymbol{B}} \tag{3.24}$$

where $\boldsymbol{M}_{\boldsymbol{g_2}}^{\boldsymbol{B}}$ takes the form of (3.21) while $\boldsymbol{M}_{\boldsymbol{g_1}}^{\boldsymbol{A}}$is given by

$$\boldsymbol{M}_{g_1}^{\boldsymbol{A}} = \begin{pmatrix} r_g & 0 & r_g & 0 \\ 0 & 1-r_g & 0 & 1-r_g \\ 1-r_g & 0 & 1-r_g & 0 \\ 0 & r_g & 0 & r_g \end{pmatrix} \tag{3.25}$$

### 3.4.2 Category 'N'

Category '*N*' refers to the 'not-strong' or relatively weak correlation cases for the pair (*A*, *B*), including 3 sub-categories, i.e., '*N1'*, '*N2'* and '*N3'*, as shown in Fig. 3 where all gates are again generic. More specifically, *N1* and *N2* shown in Fig. 3 (a) and (b), respectively, represent two specific correlations where A or B is the reconvergent fanout with more than one gates involved in the reconvergent path, which weaken the strength of correlation and make it more difficult to find the JPV and JCPM of signal pair (A, B). A more general case for 'not-strong' correlations is the category '*N3'*, as shown in Fig. 3 (c) where the dotted lines stand for some possible correlations/connections. It should be noticed that the above category '*S3'* of Fig. 3 (c) is an exception of '*N3'* where X=W and Y=Z, which is considered as a strong correlation.

Unlike category '*S'*, accurate evaluation of both JPV and JCPM for (A, B) under category '*N'* is generally difficult. For example, to find the JPV of (A, B) in '*N1'*, the three-signal joint probability of (X, Z, W) is required, which is unknown (only JCPMs for input pairs are available). To address this issue, a bitstream simulation technique, which is a statistical method proposed by [12], can be used instead (see Section III-*C* for details).

### 3.4.3 Category 'I'

Category '*I'* generally represents any topological structures other than the above categories '*S'* and '*N'* (primary inputs are assumed to be independent). In particular, if a reconvergent fanout does not exist or is too far away from the signal pair (A, B), the correlation can be treated as the category '*I'*, which refers to "independent". This is because of the fact that when the reconvergent fanout stays further away from A or B, their correlation is getting weaker. One could do bitstream simulation [12] to evaluate both JPV and JCPM for (A, B), but it would be very time-consuming. Our simulations showed that when the reconvergent fanout is more than 4 levels away, the correlation strength would be negligibly weak and can thus be treated as an independent case with negligible errors (refer to Section IV for detailed results). For this category ‘I’, the JPV (or JCPM) for (A, B) can be obtained easily and efficiently without simulations, and is given approximately by simply taking the product of individual signal probabilities (or reliabilities) for both A and B as:

$$\boldsymbol{P}^{*}_{\boldsymbol{AB}} = \left(P_A^{0^*}\cdot P_B^{0^*} \quad P_A^{0^*}\cdot P_B^{1^*} \quad P_A^{1^*}\cdot P_B^{0^*} \quad P_A^{1^*}\cdot P_B^{1^*}\right) \tag{3.26}$$

$$C_{mn} = R_A^{mn}\cdot R_B^{mn} \tag{3.27}$$

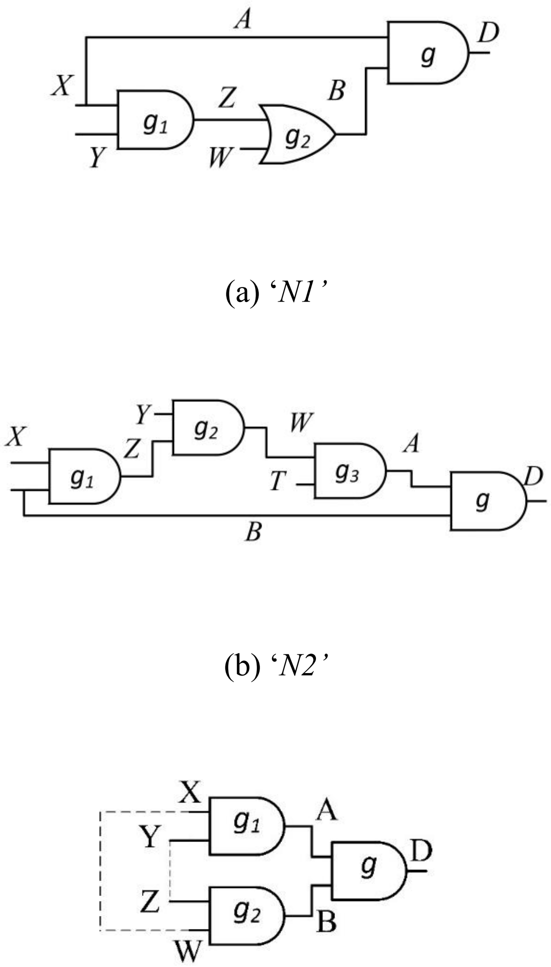
where $C_{mn}$ is the element in the *m*-th row and *n*-th column of $\boldsymbol{C_{AB}}$, and $R_A^{mn}$ and $R_B^{mn}$ are given by:

$$R_A^{mn} = \begin{cases} R_A^0, & if\ m,n = 1\ or\ 2 \\ 1 - R_A^0, & if\ m = 1\ or\ 2\ and\ n = 3\ or\ 4 \\ 1 - R_A^1, & if\ m = 3\ or\ 4\ and\ n = 1\ or\ 2 \\ R_A^1, & if\ m,n = 3\ or\ 4 \end{cases} \tag{3.28}$$

$$R_B^{mn} = \begin{cases} R_B^0, & if\ m,n = 1,3 \\ 1 - R_B^0, & if\ m = 1\ or\ 3\ and\ n = 2\ or\ 4 \\ 1 - R_B^1, & if\ m = 2\ or\ 4\ and\ n = 1\ or\ 3 \\ R_B^1, & if\ m,n = 2\ or\ 4 \end{cases} \tag{3.29}$$

### 3.4.4 Multi-level Category

In terms of correlation strength, all above categories can be listed in a descending order from the strongest to the weakest as: ‘*S1*’, ‘*S2*’, ‘*S3*’, ‘*N1*’, ‘*N2*’,



(a) ‘*N1’*

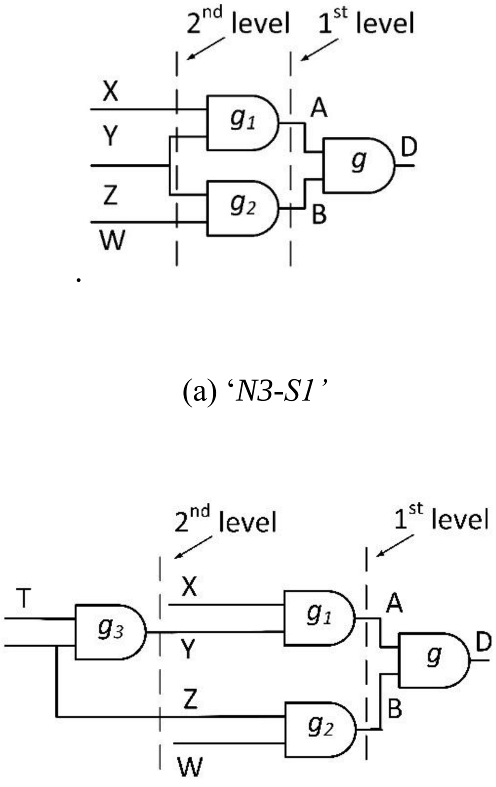
(b) ‘*N2’*

(c) ‘*N3’*

Figure 3. Examples of correlation category ‘*N’*.

‘*N3’*, and ‘*I’*. In all these categories with exception of ‘*N3’*, a reconvergent fanout, which significantly contributes to the correlation of (A, B), is known (or no such a reconvergent fanout exists when it is the category ‘*I’*). For category ‘*N3’*, further investigation is required to find possible correlations among signals X, Y, Z and W (refer to Fig. 3 (c)) by locating any potential reconvergent fanouts. Considering the fact that the correlation between X and Y (or between Z and W) does not directly lead to the correlation between A and B, we are interested only in looking at possible correlations within the four cross-gate signal pairs (CGSPs), i.e., (X, Z), (X, W), (Y, Z) and (Y, W). Like (A, B), each of these pairs has their own category, but is one gate away from (A, B). In order to differentiate these CGSPs from (A, B), we define the correlation for (A, B) as the 1st-level correlation, and those for CGSPs as the 2nd-level correlation. The strongest correlation category among these four CGSPs defines the 2nd-level category. For example, in Fig. 4 (a) where (Y, Z) is category '*S1*' which is the strongest among all CGSPs, the correlation of (A, B) has the 1st-level category of '*N3*' followed by the 2nd-level category of '*S1*'.

Therefore, the correlation of (A, B) belongs to 2-level category, expressed as '*N3*-



(a) '*N3-S1*'

(b) '*N3-S2*'

Figure 4. Examples of 2-level correlation category.

*S1'*. Fig. 4 (b) shows another example with a 2-level category of '*N3*-*S2'*.

Following the above definition, we further proceed to the 3rd-level if the 2nd-level category is '*N3'* again, in order to check if any reconvergent fanout stem can be found in the 3rd-level. It is noted that only CGSP pairs with category '*N3'* in the 2nd-level proceed to the 3rd level. The maximum number of CGSPs at the 3rd-level is $4 \times 4 = 16$ . If the 3rd-level category is still '*N3'*, then we move to the 4th level to check, and so on. The maximum number of CGSPs on level $K$ would be up to $4^{K-1}$. However, considering that the correlations due to reconvergent fanouts at the 5th or higher level is increasingly weak, we go only up to 4 levels for computational efficiency. In case the 4th-level category is '*N'* which represents any category of '*N1'*, '*N2'* or '*N3'*, the correlation of (A, B) is approximately treated as category '*I'* instead of a 4-level category of '*N3*-*N3*-*N3*-*N'*. Also, if the category at the last level for a multi-level category is '*I'*, it would be simply equivalent to single-level category '*I'*.

## 3.5 Bitstream Simulation for JPV and JCPM Estimation

As discussed before, there are no analytical methods available to compute the probability and reliability for category '*N'*. We resort to a local bitstream simulation technique instead to estimate the JPV and JCPM of (A, B). We will start with a single-level category '*N1'* and '*N2'*, then extend our discussions to a general multi-level category.

### 3.5.1 Bitstream Simulation for Single-level Category

Prior to simulation, all the JPV and JCPMs for transitive fan-ins of (A, B), along with signal probabilities and reliabilities, are assumed to be available. We take single-level category *'N1'* for example to show how the bitstreams are generated. Similar procedure also applies to single-level category '*N2'* with considerations of just one more gate on the reconvergent path.

For category '*N1*' (refer to Fig. 2 (a)), we first generate an error-free bitstream sequence for the reconvergent fanout X (or A), denoted as $Seq_X^*$ (with length of $L$), using the error-free probability vector $\boldsymbol{P_X^*}$, and then generate the corresponding error-free bitstream sequence for Y, denoted as $Seq_Y^*$, based on $Seq_X^*$ and $\boldsymbol{P_{XY}^*} = (P_{XY}^{00^*}\ P_{XY}^{01^*}\ P_{XY}^{10^*}\ P_{XY}^{11^*})$. More specifically, if X = '0' is first generated, then the bit-value of Y is generated using the following conditional probability vector:

$$Pr\{\boldsymbol{Y}|X=0\} = (\frac{P_{XY}^{00^*}}{P_{XY}^{00^*}+P_{XY}^{01^*}} \quad \frac{P_{XY}^{01^*}}{P_{XY}^{00^*}+P_{XY}^{01^*}}) \qquad (3.30)$$

If X = '1' otherwise, we use the following conditional probability vector instead to generate the bit-value for Y:

$$Pr\{\boldsymbol{Y} \mid X=1\} = \left(\frac{P_{XY}^{10^*}}{P_{XY}^{10^*}+P_{XY}^{11^*}} \quad \frac{P_{XY}^{11^*}}{P_{XY}^{10^*}+P_{XY}^{11^*}}\right) \qquad (3.31)$$

With the above $Seq_X^*$ and $Seq_Y^*$, we then generate the (erroneous) bit-value sequence for 'XY' using the JCPM of (X, Y):

$$\boldsymbol{C_{XY}} = \begin{pmatrix} P_{00}^{00} & P_{00}^{01} & P_{00}^{10} & P_{00}^{11} \\ P_{01}^{00} & P_{01}^{01} & P_{01}^{10} & P_{01}^{11} \\ P_{10}^{00} & P_{10}^{01} & P_{10}^{10} & P_{10}^{11} \\ P_{11}^{00} & P_{11}^{01} & P_{11}^{10} & P_{11}^{11} \end{pmatrix}_{XY} \qquad (3.32)$$

For instance, if $(\mathrm{XY})^* = 00$ at a certain bit of $Seq_X^*$ and $Seq_Y^*$, then 'XY' is generated using the joint conditional probability from the first row of $\boldsymbol{C_{XY}}$, i.e.,

$$Pr\{\boldsymbol{XY} \mid (XY)^* = 00\} = (P_{00}^{00} \quad P_{00}^{01} \quad P_{00}^{10} \quad P_{00}^{11})_{XY} \qquad (3.33)$$

For any other values of (XY)$^*$, simply take the corresponding row from $\boldsymbol{C_{XY}}$**.** This process is repeated $L$ times to obtain two bitstream sequences for (erroneous) X and Y, denoted as $Seq_X$ and $Seq_Y$, respectively. Once $Seq_X^*$, $Seq_Y^*, Seq_X$ and $Seq_Y$ are available, the bitstream sequence for the signal Z (denoted as $Seq_Z^*$ and $Seq_Z$) can be obtained accordingly by propagating (X, Y) through $g_1$ to its output Z: $Seq_Z^*$ is generated by applying the logic operation on $Seq_X^*$ and $Seq_Y^*$, while $Seq_Z$ is generated by applying the logic operation on $Seq_X$ and $Seq_Y$. It should be noted that $g_1$ is assumed to be reliable when producing $Seq_Z^*$, while its gate reliability of $r_g$ is taken into considerations for producing $Seq_Z$.

To further generate the bitstream sequence of B in Fig. 2 (a), we need to use the JPV and JCPM of (Z, W). First, $Seq_W^*$ can be easily obtained by following the similar procedure of generating the above $Seq_Y^*$. The $Seq_W$ can be generated by taking two elements from $\boldsymbol{C_{ZW}}$, depending on the specific bit-values in $Seq_W^*$, $Seq_Z^*$and $Seq_Z$ and $\boldsymbol{C_{ZW}}$. The bit-values in $Seq_W^*$ and $Seq_Z^*$ define the row of the two elements, while the bit-value in $Seq_Z$ define their column. For instance, if (ZW)$^* = 00$ and Z $= 0$, the conditional probability vector of W is given by:

$$Pr\{\boldsymbol{W} \mid (ZW)^* = 00 \ \cap \ Z = 0\} = (P_{00}^{00} \quad P_{00}^{01})_{ZW} \tag{3.34}$$

where $P_{00}^{00}$ and $P_{00}^{01}$ are the two elements in the 1st row and 1st two columns of $\boldsymbol{C_{ZW}}$. Thus, the bit-value of W is generated by using the probability vector of $\left(\frac{P_{00}^{00}}{P_{00}^{00} \ +P_{00}^{01}} \quad \frac{P_{00}^{01}}{P_{00}^{00} \ +P_{00}^{01}}\right)$. Once $Seq_Z^*$, $Seq_W^*, Seq_Z$ and $Seq_W$ are available, the bitstream sequence for the signal B (denoted as $Seq_B^*$ and $Seq_B$) can be finally obtained by propagating the bitstream sequences of both Z and W through $g_2$ to its output.

With the above-generated $Seq_A^*$ , $Seq_A$, $Seq_B^*$ and $Seq_B$, the occurrences of $(\mathrm{AB})^*$ to take '00', '01', '10' or '11 can be counted. Dividing them by the total length of bitstreams $L$ gives an estimate of $\boldsymbol{P_{AB}^*} = \left(P_{AB}^{00^*}\ P_{AB}^{01^*}\ P_{AB}^{10^*}\ P_{AB}^{11^*}\right)$. The total 16 elements in $\boldsymbol{C_{AB}}$ (i.e., JCPM for (A, B)) are estimated by the frequency of occurrences for 'AB' = '00', '01', '10' or '11' given $(\mathrm{AB})^*$ = '*ij*', where *i*, *j* = 0, 1. For instance, $P_{00}^{01}$ (i.e., the element in the 1st-row and 2nd-column of $\boldsymbol{C_{AB}}$) is given by the occurrences of 'AB' = '01' given $(\mathrm{AB})^*$ = '00'. With $\boldsymbol{P_{AB}^*}$ and $\boldsymbol{C_{AB}}$, both probability vector and reliability vector for the output D in Fig. 2 (a) can be obtained easily by following the method discussed in Section III-*A*.

To better illustrate the proposed method, we show an example using benchmark circuit C17 (Fig. 4) before presenting the pseudocodes of our algorithm.

In C17, all gates are NAND type, and (D, E, F, G, H) are primary input, while (Q, T) are primary outputs. We assume all primary inputs are independent with an error-free probability of 0.5 and reliability of 1, i.e., $\boldsymbol{P_{D\sim H}^*} = (\mathbf{0.5}, \mathbf{0.5}), \boldsymbol{R_K} = (\mathbf{1}, \mathbf{1})$. In this example, we go through K - (L, M) - T to find the probability and reliability of T. For K, since (F, G) are independent, the JPV and JCPM are calculated by simple multiplication as follows:

$$\boldsymbol{P_{FG}^*} = (0.25\ \ 0.25\ 0.25\ \ 0.25) = (\mathbf{P0}, \mathbf{P1}) \tag{3.35}$$

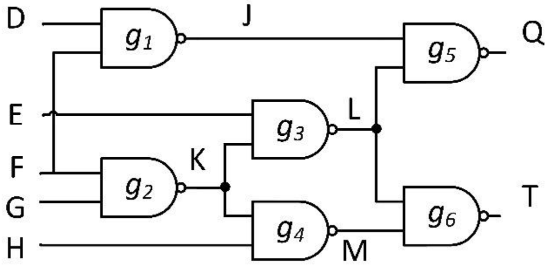


Figure 5. Benchmark Circuit C17

$$\boldsymbol{C_{FG}} = \begin{pmatrix} 1 & 0 & 0 & | & 0 \\ 0 & 1 & 0 & | & 0 \\ 0 & 0 & 1 & | & 0 \\ - & - & - & + & - \\ 0 & 0 & 0 & | & 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{CP1} & \boldsymbol{CP2} \\ \boldsymbol{CP3} & \boldsymbol{CP4} \end{pmatrix} \qquad (3.36)$$

where ***P0*** = (0.25 0.25 0.25) and ***P1*** = (0.25). Under this case, it is easy to find $P_K^* = (0.25,\ 0.75)$.

To calculate $\boldsymbol{R_K}$**,** we firstly assume gate *g*2 is reliable. The CPM of K is given by: $\boldsymbol{C_K}' = \begin{pmatrix} R_K^0 & 1-R_K^0 \\ 1-R_K^1 & R_K^1 \end{pmatrix}$

$$\begin{cases} {R_K^0}' = \dfrac{SUM(\mathbf{P0}\cdot\mathbf{CP1})}{\boldsymbol{P_D^{0^*}}} = \dfrac{0.25}{0.25} = 1 \\ {R_K^1}' = \dfrac{SUM(\mathbf{P1}\cdot\mathbf{CP4})}{\boldsymbol{P_D^{1^*}}} = \dfrac{0.25+0.25+0.25}{0.75} = 1 \end{cases} \qquad (3.37)$$

If gate *g*2 is not reliable and has reliability $r_{g_2} = 0.9$, we apply (3.11) to obtain

$$\boldsymbol{C_K} = \boldsymbol{C_K}' \cdot \boldsymbol{M_g} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}, \qquad (3.38)$$

which means that the RV of K is $\boldsymbol{R_K} = (0.9, 0.9)$.

Then we move on to signal L and M. Again, since E and H are primary inputs and K is the output of (F, G), both (E, K) and (H, K) are independent pairs according to our assumption. The procedure of finding PV and RV for L and M is exactly the same as above in deriving (3.35) - (3.38).

Once the PV and RV for both L and M are calculated, we then tend to find out the PV and RV for T. In this case, (L, M) is an ‘N3-S1’ correlation, which requires bitstream simulation. Following the previous discussions, we firstly look at CGSPs of inputs of (L, M) to find the most correlated pair, which is (K, K) with an ‘S1’ correlation. Then the bitstream for error-free and erroneous value of (K, K) pair is generated based on JPV and JCPM:

$$JPV\colon \boldsymbol{P}_{\boldsymbol{KK}}^{*} = \left(P_K^{0^*}, 0,0, P_K^{1^*}\right) = (0.25,0,0,0.75) \tag{3.39}$$

$$JCPM\colon \boldsymbol{C}_{\boldsymbol{KK}} = \begin{pmatrix} 0.9 & 0 & 0 & 0.1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & 0.9 \end{pmatrix} \tag{3.40}$$

Assume we've generated 10-bit length bitstreams $\boldsymbol{Seq}_{\boldsymbol{K}}^{*}$ and $\boldsymbol{Seq}_{\boldsymbol{K}}$ for K as follows:

| Error-free $\boldsymbol{Seq}_{\boldsymbol{K}}^{*}$ | 0 | **0** | 1 | 1 | **1** | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| Erroneous $\boldsymbol{Seq}_{\boldsymbol{K}}$ | 0 | **1** | 1 | 1 | **0** | 0 | 0 | 0 | 0 | 0 |

The two bits in bold represent the actual logic value of K is not the same as the error-free value.

The next step is to find the second most correlated signal among (E, K) and (K, H) (if no 'S1' exists, then 4 pairs will be involved here). Since both of them are independent, we randomly choose one, say (E, K), to generate the $\boldsymbol{Seq}_{\boldsymbol{E}}^{*}$ and $\boldsymbol{Seq}_{\boldsymbol{E}}$ for E (they should be equal since E is a primary input with reliability of 1), based on bit values of $\boldsymbol{Seq}_{\boldsymbol{K}}^{*}, \boldsymbol{Seq}_{\boldsymbol{K}}$ and $\boldsymbol{C}_{\boldsymbol{EK}}$ (Note that $\boldsymbol{C}_{\boldsymbol{EK}}$ has been calculated already when finding PV and RV for L):

$$\boldsymbol{P}_{\boldsymbol{EK}}^{*} = (0.125,0.375,0.125,0.375) \tag{3.41}$$

$$\boldsymbol{C}_{\boldsymbol{EK}} = \begin{pmatrix} 0.9 & 0.1 & 0 & 0 \\ 0.1 & 0.9 & 0 & 0 \\ 0 & 0 & 0.9 & 0.1 \\ 0 & 0 & 0.1 & 0.9 \end{pmatrix} \tag{3.42}$$

Let's assume we are at the first bit generation of E, where $\boldsymbol{Seq}_{\boldsymbol{K}}^{*} = 0$ and $\boldsymbol{Seq}_{\boldsymbol{K}} = 0$. From $\boldsymbol{P}_{\boldsymbol{EK}}^{*}$, we take out the two probabilities representing two '0's on K value, as $(P_{EK}^{00^*}, P_{EK}^{10^*}) = (0.125,0.125)$. Therefore, E will have a probability of 0.125/0.25=0.5 to generate a '0' on the first bit of $\boldsymbol{Seq}_{\boldsymbol{E}}^{*}$. Assume $\boldsymbol{Seq}_{\boldsymbol{E}}^{*}$ is fully generated with the following values:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Error-free $\boldsymbol{Seq_K^*}$ | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| Error-free $\boldsymbol{Seq_E^*}$ | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |

For the first bit of $\boldsymbol{Seq_E}$ generation, since $\boldsymbol{Seq_K^*} = 0$, $\boldsymbol{Seq_K} = 0$ and $\boldsymbol{Seq_E^*} = \mathbf{1}$, we need to find the terms in $\boldsymbol{C_{EK}}$ which matches the given values, as highlighted in (3.42): $(P_{10}^{00}, P_{10}^{10}) = (0, 0.9)$. Therefore, E will have a probability of 0.9/(0+0.9) = 1 to generate a '1' at the first bit of $\boldsymbol{Seq_E}$. The final $\boldsymbol{Seq_E}$ could be as follows**:**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Error-free $\boldsymbol{Seq_K^*}$ | 0 | **0** | 1 | 1 | **1** | 0 | 0 | 0 | 0 | 0 |
| Error-free $\boldsymbol{Seq_E^*}$ | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| Erroneous $\boldsymbol{Seq_K}$ | 0 | **1** | 1 | 1 | **0** | 0 | 0 | 0 | 0 | 0 |
| Erroneous $\boldsymbol{Seq_E}$ | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |

Since E is one of the primary inputs which are assumed to be reliable, the $\boldsymbol{Seq_E}$ and $\boldsymbol{Seq_E^*}$ are the same in this case. However, they are usually different for internal nodes of circuits.

Once the $\boldsymbol{Seq_E}$ is fully generated, we then propagate to L by using:

$$\begin{cases} \boldsymbol{Seq_L^*} = \boldsymbol{NAND}(\boldsymbol{Seq_E^*}, \boldsymbol{Seq_K^*}) \\ \boldsymbol{Seq_L} = \boldsymbol{NAND}(\boldsymbol{Seq_E}, \boldsymbol{Seq_K}), with\ r_{g_3} = 0.9 \end{cases} \tag{3.43}$$

where $r_{g_3} = 0.9$ means there are 90% chance for $g_3$ to perform the correct logic 'NAND', and 10% chance to produce the opposite logic output of 'NAND', or actually 'AND' logic. Using above sequences, we can have:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Error-free $\boldsymbol{Seq_L^*}$ | 1 | 1 | 1 | 0 | 1 | **1** | 1 | 1 | 1 | 1 |
| Erroneous $\boldsymbol{Seq_L}$ | 1 | 0 | 1 | 0 | 1 | **0** | 1 | 1 | 1 | 1 |

where the bit values in bold represent that the NAND gate performs incorrect logic function due to gate unreliability.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Error-free $\boldsymbol{Seq_L^*}$ | **1** | **1** | 1 | 0 | **1** | 1 | **1** | **1** | 1 | 1 |
| Error-free $\boldsymbol{Seq_M^*}$ | **1** | **1** | 0 | 1 | **1** | 0 | **1** | **1** | 0 | 0 |
| Erroneous $\boldsymbol{Seq_L}$ | 1 | 0 | 1 | 0 | **1** | 0 | 1 | **1** | 1 | 1 |
| Erroneous $\boldsymbol{Seq_M}$ | 1 | 1 | 0 | 1 | **0** | 0 | 1 | **0** | 0 | 0 |

Similarly, we can find $\boldsymbol{Seq_M^*}$ and $\boldsymbol{Seq_M}$**.** After that, by counting the frequencies of each logic value combinations, we can find the $\boldsymbol{P_{LM}^*}$ and $\boldsymbol{C_{LM}}$ accordingly. For example in the following, if there are five bits (out of ten bits in total) giving $\boldsymbol{L^*} = \boldsymbol{1}, \boldsymbol{M^*} = \boldsymbol{1}$, and among these five bits, two of them are giving $\boldsymbol{L} = \boldsymbol{1}, \boldsymbol{M} = \boldsymbol{0}$, given the bitstream length is 10, then we can estimate $\boldsymbol{P_{LM}^{11^*}} = \frac{5}{10} = 0.5$, and $(\boldsymbol{C_{11}^{10}})_{LM} = \frac{2}{5} = 0.4$. Signal probability $\boldsymbol{P_L^*}$ and $\boldsymbol{P_M^*}$ can be found similarly. When $\boldsymbol{P_{LM}^*}$ and $\boldsymbol{C_{LM}}$ are available, we can easily propagate through $g_6$ using (3.8) - ( 3.10).

Since the signal correlations have been taken into account in the above bitstream-producing process, the results would be highly accurate if the bitstream sequences are long enough. Also, the process is still fast even for long sequences because only a few gates and signals in a small local structure are involved. In this work, we chose *L* = 1000 as the bitstream length for all simulations with the best trade-off between accuracy and efficiency, as will be verified in Section IV.

### 3.5.2 Bitstream Simulation for Multi-level Category

For multi-level category (refer to Fig. 3), finding the JPV and JCPM for (A, B) would be more difficult. The reasons are two-fold: First, the correlation of (A, B) depends on the correlations among their immediate fan-ins (i.e., X, Y, Z and W in Fig. 3), which potentially depend further on those of their transitive fan-ins, making the probability and reliability estimation even harder. Secondly, multi-level category may involve multiple reconvergent fanout stems, which require some investigation on what order shall be followed in generating bitstream sequences so that the correlation of (A, B) can be captured to a maximum extent.

In the following, we present a heuristic method to generate bitstreams for (A, B). The general idea is to first generate both error-free and erroneous bitstream sequences for the four signals X, Y, Z and W, and then propagate them through the driving gates of A and B (i.e., $g_1$ and $g_2$ in Fig. 3) to obtain the bitstreams of (A, B) and its JPV and JCPM accordingly.

Since the CGSPs with strongest correlations are generally a main contributor to the correlation of (A, B), they represent a perfect candidate as a starting point of bitstream simulation. This is to ensure that the correlation information among multiple signals can be captured as much as possible. Consider a simple example of Fig. 4 (a), where (Y, Z) are strongly correlated while other CGSPs are independent. If the bitstreams of (Y, Z) are generated first, followed by bitstreams of X and W, the correlation information can be fully captured. However, if we begin with bitstreams of (X, W) instead, followed by bitstreams of Y and Z using bit-values of X and W, respectively, then the bitstreams of Y would be independent of Z, which would be totally untrue. Therefore, if the last level of multi-level category is any category other than '*I*' (it cannot be '*N3*'), then we choose that defining pair (i.e., the signal pair which defines the specific category at the level) as the starting candidate for bitstream simulations, and generate its bitstreams by either using the JPV and JCPM from analytic computation (for category '*S*'), or single-level bitstream simulations (for category '*N1*' and '*N2*'). If the last level is category '*I*' otherwise, no bitstream simulations will be needed.

Once the above starting CGSP is found, we then proceed to generate all bitstreams for the signals at the same level as this starting CGSP, before propagating them to the next level closer to (A, B). In doing so, there are two general rules to follow:

*Rule 1*: After generating the bitstreams for the starting CGSP, we choose the next signal pair as one with strongest correlation among those signal pairs (including both CGSPs and non-CGSPs) containing only one signal with available bitstreams.

*Rule 2:* In case the JPV and/or JCPM for a certain signal pair is not available for generating bitstreams, another set of new bitstream simulations is required, which shall be discarded once the JPV and JCPM are found.

To elaborate the above two rules and their importance, we take Fig. 4 (b) as an example. The starting CGSP is (Y, Z) whose JPV and JCPM can be obtained from an analytic computation as discussed in Section III-B. Following *Rule 1*, the second pair will be the one of (X, Y), (X, Z), (Y, W) or (Z, W), whichever has the strongest correlation. If one fails to follow *Rule 1* by taking (X, W) as the second pair whose bitstreams would be generated using both $\boldsymbol{P}_{\boldsymbol{XW}}^{*}$ and $\boldsymbol{C}_{\boldsymbol{XW}}$. This would imply that the bitstreams of (X, W) are independent of (Y, Z), and overlook the possible correlations within other signal pairs of (X, Y), (X, Z), (Y, W) and (Z, W). Similarly, if the third signal for bitstream generation is X, the third pair for bitstream simulations will be the one with strongest correlation among (X, W), (Y, W) and (Z, W) by following *Rule 1*.

Note that in case the JPV and JCPM for the above second pair (say, it is (Y, W)) is not available yet, another new bitstream process would be required to estimate them. As stated in the above *Rule 2*, these new bitstreams for Y shall not interfere with the existing $Seq_Y^*$ or $Seq_Y$ . The $Seq_W^*$ and $Seq_W$ will be generated next by using the obtained JPV and ICPM of (Y, W) along with the existing $Seq_Y^*$ or $Seq_Y$ . This applies to all signal pairs involved to ensure that the most correlated information is kept.

The above bitstream simulation procedure applies to all multi-level categories. The only difference is that as the number of levels increases, more signal pairs need to be considered. For instance, for 3-level category with 8 signals at the 3rd-level, each of the two signals from the starting pair can pair with any of other 6 signals, leading to a total of 2×6=12 candidate pairs to be considered when choosing the next (second) pair for bitstream generation. Each of the 3 signals with available bitstreams can then pair with any of the remaining 5 signals, resulting in a total of 3×5=15 signal pairs to be considered when choosing the 3rd pair for their bitstream generation, and so on. For a 4-level category in which the 4th-level is category '*S'*, we start with this defining pair of category '*S'* for bitstream simulations with a total of 16 signals at 4th level, leaving 2×14=28 candidate pairs to be considered when choosing the second pair for bitstream generation, followed by the 3rd pair, and so on. This procedure continues until all bitstreams for the signals at all levels have been generated.

## 3.6 Algorithm Description

In summary, the proposed method looks at one gate at a time in a topological order and finds the JPV and JCPM for its input signal pair before propagating to its output for estimation of error-free signal probability vector and reliability vector. This represents a hybrid model since both JPV and JCPM are found by either analytic computation or local bitstream simulation, depending on the specific signal correlation (single-level or multi-level) category. This can provide a high level of accuracy while maintaining the computational efficiency for signal probability and reliability evaluation with large circuits.

The whole procedure is described in the following algorithm:

*Hybrid Estimation Algorithm*

***Input****:* Circuit with probabilities of independent PIs and *M* gates with gate reliability. All PIs are assumed to be reliable.

***Output****:* All signal probabilities and reliabilities in the circuit.

***Procedure***:

```
 {Sort all $M$ gates in a topologic order;
 for $i$ = 1 to $M$, do
    Let (A, B) be input signal pair of gate $i$, and D the output;
    /* Category identification */
    Determine the correlation category for (A, B) (see Section III-B);
    /* Estimation on JPV of $\boldsymbol{P}_{AB}^{*}$ and JCPM of $\boldsymbol{C}_{AB}$ */
    if (A, B) is category 'I'
        Estimate $\boldsymbol{P}_{AB}^{*}$ and $\boldsymbol{C}_{AB}$ by eq. (3.26) - (3.29);
    else if (A, B) is single-level category 'S'
            Find $\boldsymbol{P}_{AB}^{*}$ and $\boldsymbol{C}_{AB}$ by analytic computation.
            (3.12) - ( 3.25) or similar ones according to the type of gate $i$;
         else if (A, B) is single-level category 'N1' or 'N2'
```

```
                Find $\boldsymbol{P}^{*}_{\boldsymbol{AB}}$ and $\boldsymbol{C}_{\boldsymbol{AB}}$ by bitstream simulations (refer to
                Section III-C for details);
            else   /* (A, B) is a multi-level category */
                Find $\boldsymbol{P}^{*}_{\boldsymbol{AB}}$ and $\boldsymbol{C}_{\boldsymbol{AB}}$ by bitstreams or combination of bitstreams and
                analytic computation;
        Calculate the probability and reliability for D using eqs. (3.7) - ( 3.11) or
        similar ones according to the type of gate i;
    end for
}
```

The category identification is one of the important steps in the above algorithm and is elaborated as follows: For any given gate, we simply take its two inputs (A and B), and compare their driving signals to see if they share a common signal or not. If they do (examples shown in Fig. 1), then the (single-level) category '*S'* is identified. If not, keep searching their driving signals at the next logic level (once a common signal is found, it could be identified as either category '*N'* or a multi-level category, with examples shown in Figs. 2 and 3), and so on. For instance, to identify whether (A, B) belongs to either category '*N1'* or '*N2'*, one can check if one of A and B (say, A) is also an input of another gate (other than gate *g* in the figures). If this is the case, then we can travel from this input through a few gates to check if it meets the other signal (say, B). If yes, then it belongs to '*N1'* or '*N2'*, depending on the number of gates it goes through (refer to Fig. 2 (a) and (b)). Otherwise, it is identified as category 'I'. If none of A and B is an input of another gate instead, then it belongs to category '*N3'* which would lead to a multi-level category. For multi-level category, we do a similar traversal for each CGSP signal pair at the next level to identify the category based on the definition of multi-level category. If the category remains '*N3'* for up to 4 levels, then it is identified as category 'I' as well (i.e., treated approximately as independent). As one can see, the above search process is like a (local) graph traversal plus checking the local structure against those in Figs. 1 through 3.

It can be seen from the above algorithm that the processing time is mostly spent in the category identification and bitstream generation/propagation. To identify the category for each signal pair of (A, B), the maximum number of gates (for up to four levels) to be considered is (1+2+4+8+16) = 31. For a circuit with *M* gates, the identifying process takes *O*(*M*) time. On the other hand, the computation time for bitstream simulation is linearly proportional to the length of bitstream sequences (*L*) since the number of local signals to be considered for each pair of (A, B) is typically in tens. Also, *L* is usually few thousands (in this work, we chose *L* = 1000). Thus, the overall time complexity for the algorithm is *O*(*L·M).* Meanwhile, the value of L is directly related to the algorithm accuracy. The estimation results would generally be more accurate with longer bitstream generated. According to [47], the general error for L=1000 will be within $5 \cdot 10^{-4}$ while considering the computational round-up.

## 3.7 Simulation Results and Performance Comparison

The proposed method was implemented using MATLAB on a DELL desktop (OS: Windows 10) with CPU frequency of 3.2 GHz and 8GB RAM. All simulations were conducted on benchmark circuits under the assumption that all primary inputs are reliable and independent of each other with their signal probability of 0.5. The length of bitstream sequences was set to *L*=1000.

First of all, in order to justify our previous assumption that the signal correlations can be treated approximately as an independent case as long as the reconvergent fanout is more than 4 levels away, we made an example circuit with 31 gates, as shown in Fig. 6 which represents a strongly-correlated 5-level category being treated approximately as category '*I'* (it would otherwise be identified as category of '*N3-N3-N3-N3-S1*'), where different types of gates are mixed at different levels. More specifically, all 16 gates at level 4 are NOR logic, all 8 gates at level 3 are NAND logic, all 4 gates at level 2 are NOR logic, both gates at level 1 are AND logic, and the last gate to the output D is an OR logic. Each of 16 inputs at level 5 is shared by two gates (i.e., signals #1, #3, #5, …, #15 are connected to signals #32, #31, #30, …, #25, respectively, while signals #2, #4, …, #16 are connected to signals #17, #18, …, #24, respectively, as shown in Fig. 6). Assume all these 16 inputs are reliable with signal probability of 0.5 and that all gate reliabilities are set to $r_g$ = 0.95. Since the two inputs of all gates (except the last gate G31) in the figure are independent, we can calculate the error-free signal probability vector $\boldsymbol{P_i}^*$ and signal reliability vector $\boldsymbol{R_i}$ at level $i$ (for $i$ = 4, 3, 2, 1) using (3.6) through (3.11) (or similar equations, depending on the specific gate type) as follows:

$$\boldsymbol{P_4}^* = [0.75 \quad 0.25],\ \boldsymbol{R_4} = [0.95 \quad 0.95];$$

$$\boldsymbol{P_3}^* = [0.0625 \quad 0.9375],\ \boldsymbol{R_3} = [0.8623 \quad 0.9316];$$

$$\boldsymbol{P_2}^* = [0.8828 \quad 0.1172],\ \boldsymbol{R_2} = [0.8349 \quad 0.7815];$$

$$\boldsymbol{P_1}^* = \boldsymbol{P_A}^* = \boldsymbol{P_B}^* = [0.9863 \quad 0.0137],$$

$$\boldsymbol{R_1} = \boldsymbol{R_A} = \boldsymbol{R_B} = [0.9062 \quad 0.5995].$$

If we assume A and B are independent as well, the approximate JPV and JCPM for the signal pair (A, B) can be calculated by (3.26) through (3.29) as:

$$\text{JPV(appr.):}\ \ \boldsymbol{P_{AB}}^{*} = [0.9727 \quad 0.0135 \qquad\qquad 0.0135 \quad 0.0003]$$

$$\text{JCPM(appr.):}\ \ \boldsymbol{C_{AB}} = \begin{pmatrix} 0.8212 & 0.0850 & 0.0850 & 0.0088 \\ 0.3629 & 0.5433 & 0.0376 & 0.0562 \\ 0.3629 & 0.0376 & 0.5433 & 0.0562 \\ 0.1604 & 0.2401 & 0.2401 & 0.3594 \end{pmatrix}$$

Finally, we can use equations similar to (3.6) through (3.11) for the OR gate of G31 to find $\boldsymbol{P_D}^{*}$, $\boldsymbol{R_D}$ and $r_D$ at the output signal D as:

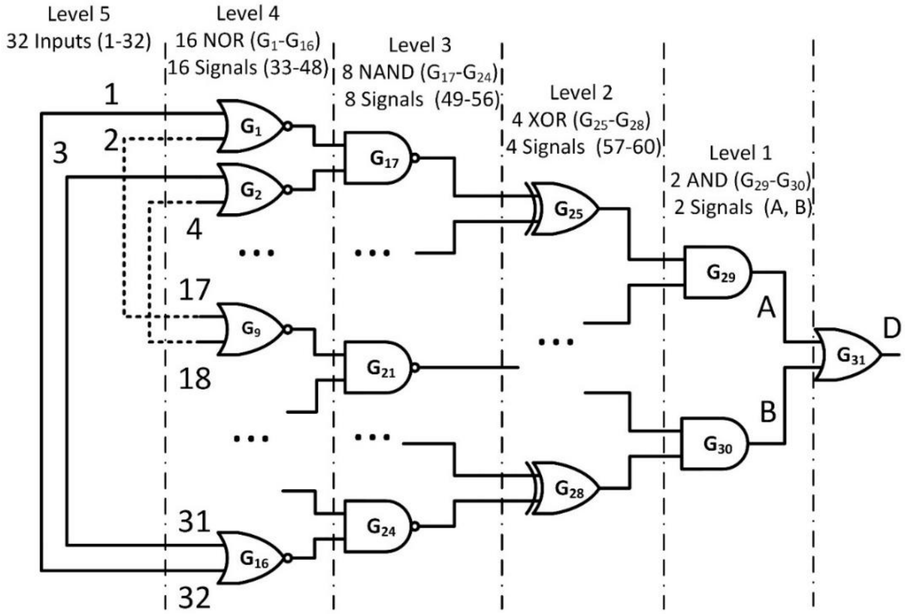


Figure 6. An example circuit with 5-level category of ‘*N3-N3-N3-N3-S1*’ being treated as category ‘*I*’ for approximation.

$\boldsymbol{P_D}^*$ = [0.9727   0.0273], $\boldsymbol{R_D}$ = [0.7891   0.6246] and $r_D$ = 0.7846.

For comparison, we also performed MC simulations (with 1 million runs) considering the correlations between A and B, and obtained the results below:

$$\text{JPV(mc):}\ \boldsymbol{P}^*_{\boldsymbol{AB}} = (0.9735\ \ 0.0127\ \ 0.0127\ \ \ 0.0011)$$

$$\text{JCPM (MC):}\ \boldsymbol{C_{AB}} = \begin{pmatrix} 0.8222 & 0.0842 & 0.0843 & 0.0093 \\ 0.3550 & 0.5233 & 0.0487 & 0.0730 \\ 0.3408 & 0.0536 & 0.5215 & 0.0841 \\ 0.1674 & 0.2447 & 0.2291 & 0.3588 \end{pmatrix}$$

which lead to the values of $\boldsymbol{P_D}^*$, $\boldsymbol{R_D}$ and $r_D$ (again by using similar equations to (3.6) through (3.11) as well as (3.2)) for the OR logic of gate G31) as:

$\boldsymbol{P_D}^*$ = [0.9735   0.0265], $\boldsymbol{R_D}$ = [0.7899   0.6432] and $r_D$ = 0.7860.

Comparison of the above approximate results with MC simulations shows that the absolute errors for $P_D^{1*}$ (the dominant element of $\boldsymbol{P_D}^*$) and $r_D$ are: (0.9735 – 0.9727) = 0.0008 and (0.7860 – 0.7846) = 0.0014, which translate into the percentage errors of around 0.08% and 0.18% (including possible numerical errors during the simulation or calculation), respectively. It should also be mentioned that this was just the results from the worst case of Fig. 6 with possibly strongest signal correlations at level 5. Therefore, it can be expected that under all category '*I*' which generally has weaker correlations at level 5 and beyond, the relative errors caused by our approximation are typically even less (depending on specific structures) and can thus be reasonably ignored for efficient evaluation.

TABLE II. Statistics on Frequency of Occurrences (%) for Different Correlation Categories

| Circuit | # Gates ($M$) | Single-level category | | | Multi-level category ($ML$) | Category by analytic method ($S$ + $I$) | Category by bitstream simulation ($N$ + $ML$) |
|---|---|---|---|---|---|---|---|
| | | $S$ | $N$ | $I$ | | | |
| C432 | 216 | 1 | 3 | 57 | 39 | 58 | 42 |
| C499 | 246 | 1 | 0 | 73 | 26 | 74 | 26 |
| C880 | 435 | 9 | 0 | 77 | 14 | 86 | 14 |
| C1355 | 590 | 35 | 1 | 44 | 20 | 79 | 21 |
| C1908 | 1057 | 1 | 0 | 81 | 18 | 82 | 18 |
| C2670 | 1400 | 4 | 0 | 67 | 29 | 71 | 29 |
| C5315 | 2973 | 5 | 0 | 65 | 30 | 70 | 30 |
| C6288 | 2416 | 40 | 10 | 21 | 29 | 61 | 39 |
| C7552 | 4042 | 3 | 0 | 63 | 34 | 66 | 34 |
| Log2 | 45083 | 6 | 2 | 54 | 38 | 60 | 40 |

Table II shows specific benchmarks for our simulations with some statistics, including their sizes and occurrence frequencies for all different correlation categories. All these circuits are ISCAS'85 benchmarks [48], except *Log2* which is a relatively large arithmetic circuit from the EPFL [49]. As can be seen from the table, the majority of correlations belong to single-level category of '*S'* or '*I'*, for which an analytic method applies. This would generally help improve the accuracy level of the proposed method. For instance, for circuit *C6288* where the category '*S'* accounts for as high as 40% (which means that a great number of signals are strongly correlated with each other), the accuracy level for both signal probability and reliability estimation with the proposed method is much better than that of the CC-SPRA, as can be seen from Table III shown below. The circuit *C1355* also has

TABLE III. Probability and Reliability Estimation Results by Proposed Method and CC-SPRA [5]

| Circuit | $E_{aP}$ | | $E_{aR}$ ($r_g$ = 0.95) | |
|---|---|---|---|---|
| | Prop. method | CC-SPRA | Prop. method | CC-SPRA |
| C432 | 0.0090 | 0.0239 | 0.0026 | 0.0317 |
| C499 | 0.0086 | 0.0018 | 0.0033 | 0.0032 |
| C880 | 0.0089 | 0.0150 | 0.0061 | 0.0061 |
| C1355 | 0.0094 | 0.0257 | 0.0088 | 0.0027 |
| C1908 | 0.0092 | 0.0200 | 0.0055 | 0.0071 |
| C2670 | 0.0160 | 0.0398 | 0.0146 | 0.0124 |
| C5315 | 0.0114 | 0.2500 | 0.0105 | 0.0124 |
| C6288 | 0.0154 | 0.0300 | 0.0179 | 0.0374 |
| C7552 | 0.0195 | 0.0238 | 0.0143 | 0.0098 |
| Log2 | 0.0253 | − | 0.0288 | − |
| Average* | 0.0119 | 0.0478 | 0.0092 | 0.0136 |

* The average is taken with circuit *Log2* excluded.

a high percentage (35%) for category '*S*', and the proposed method gives a much better result for this circuit than CC-SPRA in terms of signal probability estimation (see Table III). However, in terms of signal reliability of *C1355*, the proposed method produces slightly more errors than the CC-SPRA. This is most likely due to a relatively high percentage (44%) of category '*I*' with the circuit.

To compare the proposed method with CC-SPRA [12], we measured their average errors in both signal probability and reliability against MC simulation results (with $10^5$ runs), as summarized in Table III, where $E_{aP}$ is the average absolute errors of estimated signal probability ($P^*$) for all outputs against MC simulation results, and $E_{aR}$ is the average absolute errors of estimated signal reliability (calculated by $r_D = R_D^1 \cdot P_D^{1^*} + R_D^0 \cdot P_D^{0^*}$) for all outputs against MC simulation results (defined as $r_D = \Pr\{D = D^*\}$). For a fair comparison, the average of $E_{aP}$ or $E_{aR}$ in the table was taken with circuit *Log2* being excluded (the result for this circuit is not available from the CC-SPRA). It can be seen from Table III that the proposed method achieves absolute errors of around 0.01 for signal probability estimation (except *Log2*), which are much better than errors of around 0.05 with the CC-SPRA. The reliability estimation results from the proposed method are slightly better than those from CC-SPRA, on average.

The differences between the proposed method and CC-SPRA are summarized as follows. First, the CC-SPRA handles the correlation coefficients for a signal pair by either analytic computation (only when A=B) or bitstream simulations, while the proposed method provides two extra accurate solutions under categories '*S2*' and '*S3*'. For the case of '*S2*' in particular, the CC-SPRA would generate the bitstreams for 3 signals of (A, X, Y), and calibrate the JCPM of (A, B). Secondly, in producing bitstreams for 3 signals in general, the CC-SPRA does not necessarily take the signal pair with the strongest correlation to start with, leading to a doubtful level of accuracy. Finally, the bitstream technique in CC-SPRA is applied for up to 3 signals at a time. However, the proposed method considers more than 4 signals at a time for generation of signal bitstreams, with the potential of capturing signal correlations to a maximum extent.

The errors ($E_{aR}$) in estimating the output reliability by the proposed method for different values of $r_g$ are reported in Table IV, where the average CPU time was calculated by taking the average of computation times over three different values of $r_g$. The last two columns of this table show the speedup factors (against MC simulations) with both the proposed method and CC-SPRA for comparison. As can be seen from the table, the average errors (again, the average is taken with circuit *Log2* excluded) in reliability estimation are getting smaller as a higher value of $r_g$ is applied. The average speed-up factor of the proposed method against MC simulation is 123.97 (except the circuit *Log2*), compared to 192.34 with the CC-SPRA [12]. This is mainly due to the fact that the proposed algorithm typically generates more bitstreams for the JCPM estimation than CC-SPRA. In the CC-SPRA, for a gate at certain level, the correlation coefficient propagation process

TABLE IV. OUTPUT RELIABILITY ESTIMATION ERRORS, CPU TIME AND SPEEDUP FACTOR FOR THE PROPOSED METHOD WITH COMPARISON

| Circuit | $E_{aR}$ | | | Avg. CPU time (s) | Speedup with prop. | Speedup with CC-SPRA |
|---|---|---|---|---|---|---|
| | $r_g$=0.90 | $r_g$=0.95 | $r_g$=0.99 | | | |
| C432 | 0.0060 | 0.0026 | 0.0012 | 0.39 | 160.52 | 108.09 |
| C499 | 0.0054 | 0.0033 | 0.0015 | 0.76 | 182.24 | 195.43 |
| C880 | 0.0122 | 0.0061 | 0.0023 | 1.18 | 109.93 | 194.22 |
| C1355 | 0.0119 | 0.0088 | 0.0077 | 1.55 | 110.21 | 281.85 |
| C1908 | 0.0117 | 0.0055 | 0.0024 | 2.34 | 136.18 | 352.41 |
| C2670 | 0.0158 | 0.0146 | 0.0096 | 3.99 | 107.84 | 99.06 |
| C5315 | 0.0223 | 0.0105 | 0.0078 | 8.13 | 75.56 | 321.36 |
| C6288 | 0.0242 | 0.0179 | 0.0131 | 6.41 | 135.63 | 34.56 |
| C7552 | 0.0217 | 0.0143 | 0.0096 | 13.22 | 97.54 | 144.11 |
| Log2 | 0.0392 | 0.0288 | 0.0201 | 453.47 | 214.78 | - |
| Average* | 0.0146 | 0.0093 | 0.0061 | − | 123.97 | 192.34 |

* The average is taken with circuit *Log2* excluded.

always starts from the first level (i.e., primary inputs) with a maximum of 3 signal bitstreams to be generated per level, compared to up to 64 bitstreams (with a maximum of 4 levels considered) for the proposed method regardless of levels as mentioned in Section III. However, this also means that as the number of levels increases in large circuits, the CC-SPRA would require more signal bitstreams and simulation time, making the proposed method more scalable than the CC-SPRA. A particular example is the EPFL circuit *Log2* (with 45k gates) shown in Table IV. The size of this circuit is around 10 times as large as circuit *C7552*. While the processing time is more than 30 times as long (this is mainly due to the fact that matrix operations involved with large circuits slow down the computation), the speed-up factor for *Log2* is 215, which is much greater than those for other circuits and is also better than the average of 192 for the CC-SPRA. The main reason is that the correlation evaluation for CC-SPRA requires a reconvergent-fanout investigation back all the way to the input, resulting in a time complexity of O(M$^2$), while the proposed method stops at maximum 4 levels of gate investigation, making it possible to have an upper-bound for the processing time of any input pair correlation evaluation. This suggests that the proposed method can be more scalable and more advantageous for larger circuits. Also, it should be mentioned that the scalability has become increasingly important for any CAD methods/algorithms, given today's large-scale circuits.

## 3.8 Summary

In this chapter, we have proposed a hybrid method to estimate both signal probability and reliability for combinational circuits by categorizing all signal pairs based on their correlation strength. The signals pairs with strong correlations are handled by an analytic computation, leading to an accurate propagation of signal probability and reliability through logic gates. Those with relatively weak correlations are processed using local bitstream simulations which take signal correlations into consideration (to a maximum extent) with high efficiency. The signal pairs with extremely weak correlations are treated approximately as independent. This combination of analysis and simulation makes the proposed model competitive in terms of the tradeoff between accuracy and efficiency by estimating both signal probability and reliability simultaneously. Comparing to the recent CC-SPRA method, the proposed method is 3.59% and 0.44% more accurate regarding probability and reliability estimation, and the time complexity is improved from CC-SPRA's O($M^2$) to O(M).

# CHAPTER 4 PROBABILITY AND RELIABILITY ESTIMATION FOR SEQUENTIAL CIRCUITS

In this chapter we focus on sequential circuit reliability analysis by considering combinational logic and memory components. This involves how both probability and reliability at the inputs of combinational logic are to be updated over a number of iterations until they converge to final stable values. For any sequential circuit, the total processing time equals to: $t_{comb-Eq} \cdot \mathrm{N}_{iter}$, where $t_{comb-Eq}$ represents the estimation time for its combinational equivalent, and $\mathrm{N}_{iter}$ is the number of iterations needed before convergence.  Therefore, the speed-up of convergence process becomes comparably important to that of combinational equivalent evaluation. In this chapter, we introduce a two-step convergence technique and show how it reduces the number of iterations. The simulation results and summary are followed as well.

## 4.1 Combinational Equivalent of Sequential Circuits

Sequential circuits contain both combinational logic and memory components which introduce feedback loops into the circuit. Since most memory components are simply D-Flip-Flops (DFFs), one can simply break all loops by creating dummy inputs DIs (i.e., the outputs of DFFs) and dummy outputs DOs (i.e., the inputs of DFFs) to transform sequential circuits into a sequence of combinational logic networks, as in Fig.1.

## 4.2 Convergence Process Analysis and Two-Step-Convergence (TSC) Method

### 4.2.1 Convergence Process for Sequential Circuit

Since the statistics of DIs are unknown initially, one can assume they all have signal probability of 0.5 (i.e., fully random) and reliability of 1 before starting the probability and reliability propagation through the combinational logic (as is discussed in the previous section). The resulting statistics (i.e., both probability and reliability) at the DOs would generally be different from that of DIs, and thus an iterative process is required to reach a stable status (or a convergence point). During this iterative process, we keep updating both probability and reliability at DIs based on the differences between DIs and DOs until these differences in both probability and reliability (including $R^0$ and $R^1$) are less than a specified threshold value of $\varepsilon$ (assuming DFFs are reliable). This iterative process can be described by:

$$Q_{DI}^{n+1} = Q_{DI}^{n} + \eta \cdot (Q_{DO}^{n} - Q_{DI}^{n}) \tag{4.1}$$

where $n$ is the number of iteration, $Q$ represents either probability or reliability element of DI or DO, and $\eta$ is an adjustment parameter (or step size) within [0, 1]. When $\eta = 1$, we are simply using DO's statistics in the *n*-th iteration as DI's in the (*n*+1)-th iteration. A larger value of $\eta$ means a faster step towards the convergence point but may take more time in the later stage of the process. A typical value of $\eta$ is chosen as 0.1. While it was claimed by [27] that a good way of choosing $\eta$ is to change it dynamically with an initial value of 0.5, some recent work, such as [5] suggested a universal value of 0.1 for $\eta$ to make the convergence process smoother, which is less efficient.

### 4.2.2 Two-Step-Convergence Method

In addition to the $\eta$ value, the choice of initial statistics can affect the convergence process as well. If the initial reliabilities for DIs are set to the maximum of 1, the reliabilities of DOs would be less than those of DIs due to unreliable gates. Gradual reduction in DIs' reliability will only result in decreasing reliabilities at DOs, which means that the reliability convergence under this case would be monotonous. This suggests that a simple linear regression can be applied to find a set of initial reliability values to speed up the entire process. More specifically, we can firstly do $N_{reg}$ trial estimations to find out the 'trends' of the input and output statistics for a specific pair of (DI, DO) with a relatively large value of $\eta$ (say 0.4), and then apply a linear regression on input and output datasets independently. The intersection point of the two lines serves as a new initial value for the following fine-tuning process with $\eta$ being a smaller value (say 0.1). This method is called two-step convergence (TSC). The reason we choose a relatively large $\eta$ to begin with is that the proposed method contains bitstream simulations which may introduce random fluctuations during the process. Therefore, to reduce the impact of randomness, the step size $\eta$ should be large enough to provide enough tolerance and maintain the accuracy level of regression. It should be noted that the initial values of $R^0$ and $R^1$ are found independently.

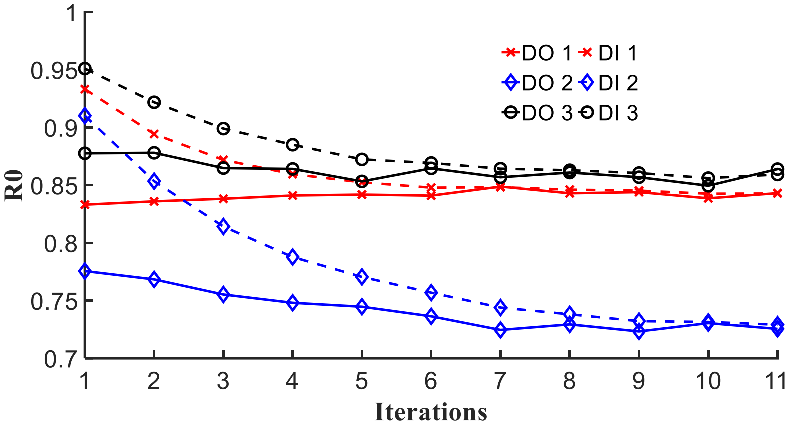
As an illustration example, Fig. 7 shows a visualized comparison of convergence progress for $R^0$ (using $\varepsilon = 0.003,\ r_g = 0.9$) with a universal $\eta = 0.4$ and TSC (using $N_{reg} = 3, \eta = 0.4$ initially and $\eta = 0.1$ after regression) on the benchmark circuit S27 with 3 DFFs. In Fig. 7, the dashed and solid lines represent DOs and DIs, respectively.

It can be seen from the figure that the TSC halves the required number of iterations to reach the convergence, and the new initial points calculated from the regression model are much closer to the actual convergence point.

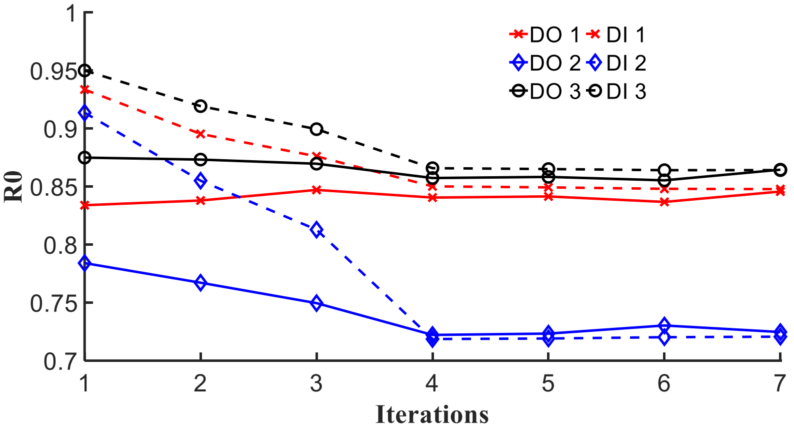
More circuit results are reported in Table V, where the $f$ is a speedup factor that compares the number of iterations required when using TSC versus choosing $\eta = 0.4$ universally, and is defined as

$$f = \left(\frac{N_{\eta=0.4} - N_{TSC}}{N_{\eta=0.4}}\right) \cdot 100\% \qquad (4.2)$$

where $N_{\eta=0.4}$ and $N_{TSC}$ are the entries in the 3rd and 4th column of Table V, respectively, with an average improvement of around 28% in terms of iterations.



(a) $\eta = 0.4$ (universal value)



(b) TSC

Figure 7. Convergence process of $R^0$ in circuit S27 with 3 DFFs.

## 4.3 Simulation Results and Performance Comparison

The proposed method was implemented using MATLAB on a DELL desktop (OS: Windows 10) with CPU frequency of 3.2 GHz and 8GB RAM. Simulations

Table V. COMPARISON OF NUMBER OF ITERATIONS TO CONVERGE

| **Circuit** | $\boldsymbol{\eta = 0.1}$ | $\boldsymbol{\eta = 0.4}$ | $\boldsymbol{\eta = 0.4}$ **(TSC)** | $\boldsymbol{f(\%)}$ |
|---|---|---|---|---|
| S27 | 40 | 11 | 7 | 36.36 |
| S298 | 74 | 14 | 12 | 14.28 |
| S349 | 79 | 21 | 14 | 33.33 |
| S444 | 80 | 21 | 16 | 23.81 |
| S526 | 73 | 26 | 15 | 42.31 |
| S635 | 88 | 24 | 23 | 4.17 |
| S820 | 62 | 14 | 9 | 35.71 |
| Average | - | - | - | 28.53 |

were conducted on benchmark circuits under the assumption that all primary inputs are reliable and independent of each other with their signal probabilities of 0.5. The length of bitstream sequences was set to $L$=1000, and the gate reliability to $r_g = 0.9$. The TSC parameters were set as $\varepsilon = 0.003$, $N_{reg} = 3$, $\eta = 0.4$ (during first step) or 0.1 (during second step).

Detailed simulation results for applying TSC on ISCAS’89 benchmark circuit S27 are firstly presented in Table VI. In Table VI, we show data for reliability converge process. The 3rd and 4th column shows the corresponding coefficient obtained from linear regression using first 3 iterations. The former value represents slop of the linear regression and latter one is the corresponding Y-axis intercept. The 5th and 6th column compares the new initial calculated by TSC and the final converged value. The last two columns compare the gap remained to reach convergence for 4th iteration (Converged values-4th loop initials) using TSC and traditional regression with η=0.4. The 0s reported for row ‘R1’ of DFF2 is because of a probability equals to 0 for that signal, which means logic 1 reliability is not applicable to it.

It could be seen that by using TSC, the remained gaps were reduced significantly, except for R1 estimation of DFF2 which simply reaches 0. The largest gap difference happens for R1 estimation of DFF1, up to 0.0609. The average improvement regarding remained absolute gap is 0.0343, which could cause at least a few more iterations to be covered for traditional method.

TABLE VI S27 CONVERGENCE PROCESS INFORMATION

| Number of DFF | Reliability element | Linear Regression coefficient (slop, y intercept) | | New Initials | Conv. values | Remained absolute gap (4th loop) | |
|---|---|---|---|---|---|---|---|
| | | DI | DO | | | TSC | Traditional |
| DFF1 | R0 | (-0.0294,0.9591) | (0.0056,0.8260) | 0.8470 | 0.8423 | 0.0047 | 0.0198 |
| | R1 | (-0.0726,0.9301) | (-0.0134,0.6703) | 0.6125 | 0.6184 | 0.0059 | 0.0668 |
| DFF2 | R0 | (-0.0501,0.9577) | (-0.0141,0.7908) | 0.7252 | 0.7249 | 0.0003 | 0.0568 |
| | R1 | (-0.3000,0.8000) | (0.0000,0.0000) | 0 | 0 | 0 | 0 |
| DFF3 | R0 | (-0.0267,0.9767) | (-0.0069,0.8862) | 0.8548 | 0.8552 | 0.0004 | 0.0295 |
| | R1 | (-0.0395,0.9627) | (-0.0072,0.8215) | 0.7896 | 0.7891 | 0.0005 | 0.0446 |
| Average | - | - | - | - | - | 0.0020 | 0.0363 |

Since the efficiency of the convergence process will not affect the final converged value, the output reliability accuracy for S27, along with some other ISCAS’89 benchmark circuits results are shown in Table VII, with the comparison between the proposed TSC and Seq-RE [24] in terms of both accuracy and speed on some ISCAS’89 benchmark circuits, where the average error in reliability refers to the average of reliability estimation errors (in percentage) over all outputs with a circuit. Due to hardware limitations, we were not able to conduct the same MC simulations as in [24] with $10^{12}$ full-circuit iterations. Instead, $10^{5}$ MC iterations were used in this work. It was found that the differences in MC results from $10^{5}$ versus $10^{12}$ iterations are marginal (for instance, the absolute difference is less than 0.00004 with S27). It can be seen from the table that the speedup factor with TSC is still around $10^{2}$ times faster than Seq-RE. There are mainly two reasons behind this efficiency. First, the Seq-RE uses a universal value of $\eta = 0.1$, which significantly slows down the entire convergence process. Secondly, our TSC uses direct calculations for popular correlation categories of ‘S2’ and ‘S3’, while the Seq-RE requires full bitstreams to evaluate these two correlations.

Table VII. ACCURACY AND SPEED COMPARISON OF TSC AND SEQ-RE

| **Circuit** | **# Gates** | **# DFFs** | **Average errors in reliability (%)** | | **Speed-up factor against MC** | |
|---|---|---|---|---|---|---|
| | | | ***TSC*** | ***Seq-RE*** | ***TSC*** | ***Seq-RE*** |
| S27 | 10 | 3 | 0.03 | 0.01 | $3.46\times10^2$ | $4.00\times10^4$ |
| S298 | 119 | 14 | 0.57 | 1.00 | $1.39\times10^2$ | $1.16\times10^4$ |
| S349 | 161 | 15 | 0.40 | 1.50 | $1.46\times10^2$ | $1.50\times10^4$ |
| S444 | 181 | 21 | 0.68 | 2.60 | $1.57\times10^2$ | $1.76\times10^4$ |
| S526 | 193 | 21 | 0.58 | 2.30 | $1.48\times10^2$ | $1.69\times10^4$ |
| S635 | 286 | 32 | 0.70 | 2.10 | $1.43\times10^2$ | $1.81\times10^4$ |
| S820 | 288 | 5 | 0.78 | 2.20 | $1.39\times10^2$ | $1.85\times10^4$ |
| S1196 | 529 | 18 | 1.04 | 2.84 | $1.88\times10^2$ | $2.47\times10^4$ |
| S1488 | 653 | 103 | 1.18 | 2.00 | $1.71\times10^2$ | $2.59\times10^4$ |
| S13207 | 7951 | 638 | 3.82 | - | $2.44\times10^2$ | - |
| Ave* | - | - | 0.66 | 1.83 | $1.75\times10^2$ | $2.09\times10^4$ |

*S13207 is excluded while calculating the average.

Further detailed performance of the proposed TSC method is summarized in Table VIII, where *P* and *R* refer to probability and reliability, respectively, and DFF maximum errors represent the maximum absolute difference between DIs and DOs when the convergence is reached. The results from Table VIII again show a high level of efficiency and accuracy with the proposed method. Since the TSC uses simulation-based estimation, random fluctuations may result in an increasing number of iterations prior to convergence when circuits involve more DFFs.

Table VIII. DETAILED PERFORMANCE OF THE PROPOSED TSC

| **Circuit** | **Average relative errors in outputs (%)** | | **DFF max. errors** | | **# Conv. iterations** | | **CPU Time (s)** |
|---|---|---|---|---|---|---|---|
| | ***P*** | ***R*** | ***P*** | ***R*** | ***P*** | ***R*** | |
| S27 | 0.03 | 0.03 | 0.0013 | 0.0008 | 11 | 7 | 0.27 |
| S298 | 0.85 | 0.57 | 0.0021 | 0.0021 | 44 | 12 | 10.32 |
| S349 | 0.56 | 0.40 | 0.0028 | 0.0029 | 65 | 14 | 18.75 |
| S444 | 0.72 | 0.68 | 0.0023 | 0.0028 | 73 | 16 | 26.63 |
| S526 | 0.82 | 0.58 | 0.0027 | 0.0015 | 78 | 15 | 33.57 |
| S635 | 0.89 | 0.70 | 0.0027 | 0.0029 | 82 | 23 | 49.24 |
| S820 | 0.74 | 0.78 | 0.0023 | 0.0014 | 25 | 9 | 19.98 |
| S1196 | 1.24 | 1.04 | 0.0030 | 0.0028 | 31 | 24 | 197.72 |

## 4.4 Summary

In this chapter, we have proposed a fast and effective hybrid method for sequential circuit probability and reliability estimation. In combinational logic, we combined both analytical and statistical methods to reach a good balance between efficiency and accuracy in estimation. To speed up the convergence for sequential circuits, a two-step convergence method was applied to further reduce the number of iterations required. This TSC reaches average 30%, maximum 42% improvement of convergence process efficiency on simulated benchmark sequential circuits. The high efficiency of combinational model, combined with fast convergence process, leads to a decent speed up factor, which is much higher than what is achieved by Seq-RE.

# CHAPTER 5 RELIABILITY ESTIMATION WITH CONSIDERATION OF AGING EFFECT

The model discussed in Chapter 3 and 4 are coming with assumptions that circuit is under zero-delay and the gate reliability is a constant. In real-world circuits, the gate delay cannot be ignored, and after operating intensively for long time, devices are experiencing aging effect and therefore reduce circuit reliability. A key reliability issue is the Negative-bias temperature instability (NBTI) on PMOS, which is caused by operating with negative gate-to-source voltage. Without enough rest time, the positive charges trapped underneath the gate will partially cancel the negative gate-to-source voltage and hence make it harder for the transistors to be turned on, hence increase threshold voltage. This threshold voltage change will not only affect the delay of the transistors, but also change the behavior acting as a switch, leading to possible incorrect logic value at the output.

In this chapter, we first introduce a model using a single index for signal reliability by combining spatial and temporal reliability, followed by discussions on how to obtain spatial and temporal reliability separately. We then present simulation results and summarize our work.

## 5.1 Spatial Reliability and Temporal Reliability

### 5.1.1 Spatial Reliability and Spatial Probability of Failure (SPF)

For a combinational circuit, if either an output signal arrives late, or its logic value is incorrect, the sampled output value is considered as incorrect, and therefore we have:

$$pf_C = pf_C^T + (1 - pf_C^T) \cdot pf_C^s \tag{5.1}$$

where $pf_C^T$ represents the probability that the signal arrives late and $pf_C^s = 1 - r_C^S$, which is the probability that the produced logic value is incorrect.

### 5.1.2 Temporal Reliability and Temporal Probability of Failure (TPF)

In sequential circuits, if a logic value arrives too late and exceeds the guard band, the sampled output would be the value generated by the previous clock (CLK) cycle, and there is a probability that the sampled data of current CLK cycle to be accidentally correct. There are two possibilities to consider: 1. Last cycle logic value was wrong, and the error free value of current cycle has an expected switching. 2. Last cycle logic value is correct, and the error free value of current cycle has no expected switching. However, it is impractical to accurately find out if a switch is happening for each CLK cycle. Hence, we use output error free probability to find the overall probability of switching activity. The output PF is then approximated as follows:

$$\begin{aligned} pf_C \approx pf_C^T * [pf_C^S * (1 - P_{SWI}) + (1 - pf_C^S) * P_{SWI}] \\ +(1 - pf_C^T) \cdot pf_C^S \end{aligned} \tag{5.2}$$

where $P_{SWI} = 2 \cdot P_C^* \cdot (1 - P_C^*)$ represents the switching probability. It can be seen that overall PF is reduced compared to a combinational case. Under extreme cases, if $pf_C^T = 0$ (i.e., the signal is always arriving on time), then $pf_C = pf_C^S$ according to (5.2) which provides a same value as (5.1). If $pf_C^S = 0$, we have $pf_C = pf_C^T P_{sw1}$, which means that the output will always be correct if the signal arrives on time. If the signal arrives late otherwise, the output could still be correct unless there is a switching event.

In the following section, we will introduce how to calculate/estimate the PFs first, and then find the signal overall reliability using (5.1) or (5.2) accordingly.

## 5.2 Threshold Variation under Aging Effect

In this section, we modify some of previously developed model and introduce how to apply it into full circuit reliability estimations with detailed error analysis.

As mentioned above, with the NBTI effect, the threshold value of MOSFETs will increase, causing performance variation of logic gates. There are several analytical NBTI models that have been introduced in [30, 31, 32], and a simplified version (for an inverter) was proposed in [34] as follows:

$$\Delta V_{th} = b \cdot (1 - P_{in})^n \cdot t^n \tag{5.3}$$

where $b = 3.9 \cdot 10^{-3}$, $P_{in}$ is the input probability, $t$ is the time that the device have been working for, and $n$ is a time component factor with a standard value of 0.16. However, since the input probabilities for a two-input logic gates (such as NAND gate) are usually different, we take the smaller probability to estimate $\Delta V_{th}$ in (5.3) to ensure that the model covers the worst case.

## 5.3 Estimation of $SPF$

### 5.3.1 Estimation of SPF for MOSFETs

For $pf_C^S$ estimation, our focus is on identifying PF for every individual gate $pf_g$, i.e., the probability that the gate is *not* performing the logic function correctly, by investigating the transistor performance. According to [32], the designed transistor threshold voltages $V_{th}$ can fluctuate due to oxide thickness variations [23], line edge roughness [24], polysilicon granularity, and the combined effects of all these factors. Simulation and some sample analysis [25-29] concluded that the real $V_{th}$ fluctuations can be treated as a *Gaussian* distribution $N(V_{TH}, \sigma_{TH}^2)$, where the $V_{TH}$ represents the designed threshold and $\sigma_{TH}$ is the standard deviation found from the real threshold voltage. The PF for PMOS and NMOS can be given as follows (assuming the transistor behaves like a binary switch around the threshold voltage):

$$\begin{cases} pf_p(v_{in}) = 0.5 \cdot erfc\left(\dfrac{\left|v_{in} - \left[V_{DD} + V_{thp}\right]\right|}{\sigma_{TH,p} \cdot \sqrt{2}}\right) \\ pf_n(v_{in}) = 0.5 \cdot erfc\left(\dfrac{\left|v_{in} - V_{thn}\right|}{\sigma_{TH,n} \cdot \sqrt{2}}\right) \end{cases} \tag{5.4}$$

where $\sigma_{TH} \approx t_{ox}{N_A}^{0.4}/(L_{eff}W_{eff})^{0.5}$ (stands for both NMOS and PMOS), $t_{ox}$ is the oxide thickness, $N_A$ is the channel doping, $L_{eff}$, $W_{eff}$ are the effective length and width of the channel, which are all determined by the given technology, and $v_{in}$ is the input voltage to the transistors. When the threshold voltage changes due to aging, the $pf_n$and $pf_p$ would be different from their original values, and any logic gate containing MOSFETS will perform differently.

### 5.3.2 Estimation of SPF for Logic Gates and Integrated Circuits

The gate PF can then be calculated using information provided by (5.4). Take NAND gate as an example, as shown in Fig.8 where the four transistors are independent of each other. The probability of failure of the NAND gate with different input patterns (A, B), as mentioned in [19], is given by:

$$\boldsymbol{pf(AB)_{NAND}} = \begin{cases} pf(00) = pf_{T1}^0 \cdot pf_{T2}^0 + pf_{T3}^0 \cdot pf_{T4}^0 - \\ \quad pf_{T1}^0 \cdot pf_{T2}^0 \cdot pf_{T3}^0 \cdot pf_{T4}^0 \\ pf(01) = pf_{T1}^1 \cdot (1 - pf_{T2}^0) + pf_{T3}^0 \cdot (1 - pf_{T4}^1) - \\ \quad pf_{T1}^1 \cdot (1 - pf_{T2}^0) \cdot pf_{T3}^0 \cdot (1 - pf_{T4}^1) \\ pf(10) = (1 - pf_{T1}^0) \cdot pf_{T2}^1 + (1 - pf_{T3}^1) \cdot pf_{T4}^0 - \\ \quad pf_{T1}^0 \cdot (1 - pf_{T2}^1) \cdot pf_{T3}^1 \cdot (1 - pf_{T4}^0) \\ pf(11) = 1 - (1 - pf_{T1}^1) \cdot (1 - pf_{T2}^1) \cdot \\ \quad (1 - pf_{T3}^1) \cdot (1 - pf_{T4}^1) \end{cases} \tag{5.5}$$

where $pf_{Ti}^j$ $(for\ i = 1, 2, 3, 4, and\ \ j = 0, 1)$ represents the PF for transistor $i$ using (5.4) when $v_{in}$ takes any value that represents logic $j$. However, it is a non-trivial task to find the exact $v_{in}$ when input signals (A, B) arrive. The best solution would be letting the $v_{in}$ be two standard values for logic 0 and 1 separately.

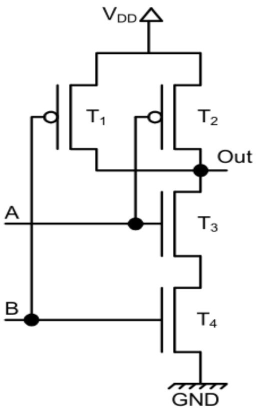


Figure 8. NAND logic gate

Considering the fact that $v_{in}$ is barely exceeding the designed noise margin, we can let the designed $v_{in}$ be the following values with the assumption that the noise margin is at 50% point of the two intervals [0, $V_{th}$] and [$V_{th}, V_{DD}$] for both PMOS and NMOS transistors (taking the more strict boundaries):

$$\begin{cases} v_{in}^{0} = \min\left(0.5 \cdot (V_{DD} + V_{thp}\right), 0.5 \cdot V_{thn}) \\ v_{in}^{1} = 1 - \min(0.5 \cdot (-V_{thp}), 0.5 \cdot (V_{DD} - V_{thn})) \end{cases} \tag{5.6}$$

where $V_{DD} = 1V$ throughout this thesis. When $pf_{NAND}(ij)$ $(for\ i = 0, 1, j = 0, 1)$ are found, the gate reliability is then given by:

$$pf_{NAND} = \sum P_{NAND}(ij)\, pf(ij) \tag{5.7}$$

where $P_{NAND}(ij)$ represents the joint probability that (A, B) takes ‘$ij$’ pattern, which is well discussed in any propagation-based reliability estimation work (such as [7]).

For any other logic gates, equations similar to (5.5) can be easily derived. With the availability of reliability for every single gate, it is possible to use the methodology introduced in chapter 3 to find the output SPFs.

## 5.4 Estimation of TPF

### 5.4.1 Gate Delay Distribution

Estimation of $pf_C^T$ is different from that of $pf_C^S$ since we only consider the impact of delay. It has shown [33, 34] that the delay of logic gates can be treated as a linear function of $V_{th}$. Thus, when $\Delta V_{th}$ goes up due to aging, the delay increment of gate $i$ can be estimated as a linear function:

$$t_{p_i} = a_{0_{inv}} + a_{1_{inv}} \cdot \Delta V_{th}/b \tag{5.8}$$

where $t_{p_i}$ is the real gate transition delay, $a_{0_i}$ is the original intrinsic delay, and $a_{1_i}$ is a constant. The $a_{0_i}$ and $a_{1_i}$ values for different gate type with given technology can be calibrated using data simulated from PSPICE tools.

### 5.4.2 Circuit Delay Distribution and TPF Estimation

Since $\Delta V_{th}$ is estimated as a deterministic value (instead of a distribution), the degraded threshold voltage for a specific gate G can be expressed as $V_{th} \sim N(V_{TH,G} + \Delta V_{th,G}, \sigma_{TH}^2)$. The delay distribution can be found using (5.8) as: $D_i \sim N(a_{0_i} + a_{1_i} * \Delta V_{th}/b, \sigma_D^2)$, and the $\sigma_D$ is a constant introduced by $\sigma_{TH}$ : $\sigma_D = a_1 \cdot \frac{\sigma_{TH}}{b}$. By adding all gate delay distributions along the path, the distribution of output delay $D_O$, which is also a *Gaussian* distribution, is found as follows:

$$D_O \sim N\left(\sum_{i=1}^{NG} a_{0_i} + a_{1_i} \cdot \frac{\Delta V_{th}}{b}, \sum_{i=1}^{NG} \sigma_{D_i}^2\right) \tag{5.9}$$

where NG is the number of gates along the path from primary inputs to a specific output. We can then find the $pf_O^T$ using the cumulative density function (CDF) as:

$$pf_O^T = 1 - cdf(D_O = D_{GB}) \tag{5.10}$$

where $D_{GB}$ represents the designed guard band delay which equals to $\frac{1}{CLK\ Frequency}$.

## 5.5 Algorithm Description

The propagation-based method used to find signal probabilities and joint probabilities is out of this thesis's scope but is well-developed in other prior works such as [16]. In this work, we simply use MC simulation to find this information. The pseudo-code of the proposed algorithm is given below:

```
Algorithm:
Begin:
 Read in circuit information (including input probabilities,
   initial gate PF, initial MOSFET threshold voltage
   distributions, circuit operation time t, and gate netlist);
 Sort all gates in a topologic order;
 Set delay indicator for all signals to 0;
 for i = 1 : NG (# gates)
     Spatial PF
         a) Find threshold voltage increment at time t:
            $\Delta V_{th} = b \cdot (1 - P_{in})^n \cdot t^n$;
        b) Find signal probability ($P_A$and $P_B$) and their joint
            probability ($P_{GATE}$) by MC simulations;
            c) Find $pf_{GATE}$ using (5.4-5.7);
            d) Find $P_O^*$and $r_0$ at outputs by propagation analysis
              or MC simulation;
      Timing Analysis
          Find gate delay $t_{p_i}$ at time t by (5.8);
    End for
    An extra step for sequential circuit only:
      Repeat the above for-loop until a stable state is
          reached
    End extra step
    Find output spatial PF: $pf_O^s = 1 - r_o$;
    Find output delay distribution using (5.9) and $t_{p_i}$;
    Find output temporal PF $pf_O^T$ using (5.10);
    Find $pf_O$ by (5.1) for combinational circuits or by
          (5.2) for sequential circuits.
  End
```

## 5.6 Error Analysis

The first source of errors comes from the propagation model we chose. Theoretically speaking, this error can be significantly reduced if Monte-Carlo simulation is applied in this step. The second error source lies in the assumption that $v_{in}^0$ and $v_{in}^1$ used in section 3.2 are two universal constants. In a real circuit, the environmental noise is always involved even if the circuit is always working under the worst case.

Table IX shows the difference between our proposed method and Monte-Carlo simulation for NAND PF regarding multiple $V_{th}$ (to simulate possible degradation). In the MC simulation, a white noise with standard deviation of

TABLE IX. COMPARISON OF $pf_{NAND}$ EVALUATED BY PROPOSED METHOD AND MC SIMULATION

| $V_{thN}$, $V_{thP}$ (V) | $\boldsymbol{pf(00)}$ | $\boldsymbol{pf(01)}$ | $\boldsymbol{pf(10)}$ | $\boldsymbol{pf(11)}$ | $\boldsymbol{pf_{NAND}}$ |
|---|---|---|---|---|---|
| 0.6, -0.6 | 1.32e-13 | 6.66e-31 | 6.66e-31 | 7.28e-7 | 1.82e-7 |
| 0.6, -0.6 (MC) | 3.34e-11 | 1.31e-20 | 5.00e-23 | 7.91e-6 | 1.977e-6 |
| 0.75, -0.75 | 0.0024 | 5.58e-50 | 5.58e-50 | 0.0963 | 0.0247 |
| 0.75, -0.75 (MC) | 0.0124 | 8.89e-41 | 2.79e-44 | 0.2093 | 0.0554 |
| 0.8, -0.8 | 0.25 | 1.49e-61 | 1.49e-61 | 0.75 | 0.25 |
| 0.8, -0.8 (MC) | 0.0553 | 1.36e-56 | 1.74e-54 | 0.4143 | 0.1174 |

$\frac{Noise\ Margin}{12}$ is added to $v_{in}^{0}$ and $v_{in}^{1}$ to simulate the worst-case scenario in real-world, i.e., the device is always receiving inputs with voltage of maximum/minimum logic 0/1 requirements, with possible environmental noise. We assume the initial designed Vth for NMOS and PMOS are 0.6V and -0.6V, respectively, and compare the PF when Vth increases to different values. The $pf_{NAND}$ is calculated assuming all four patterns have equal probability. Fig. 9 shows the overall trends under the same settings.

It can be seen that our model overestimated the unreliability when $\Delta V_{th}$ reaches one-third of the original threshold voltage. The reason is that the transistor failure rate decreases in logarithm scale according to (5.4). When noise is considered, the PF of transistors drops dramatically while the proposed model assumes the PF is always at the maximum point. Nonetheless, the proposed model still shows the correct trend of how the PF for NAND gate changes. If the given noise has a smaller standard deviation, the two curves shown in Fig. 9 would be closer to each other and finally overlap when no noise exists.

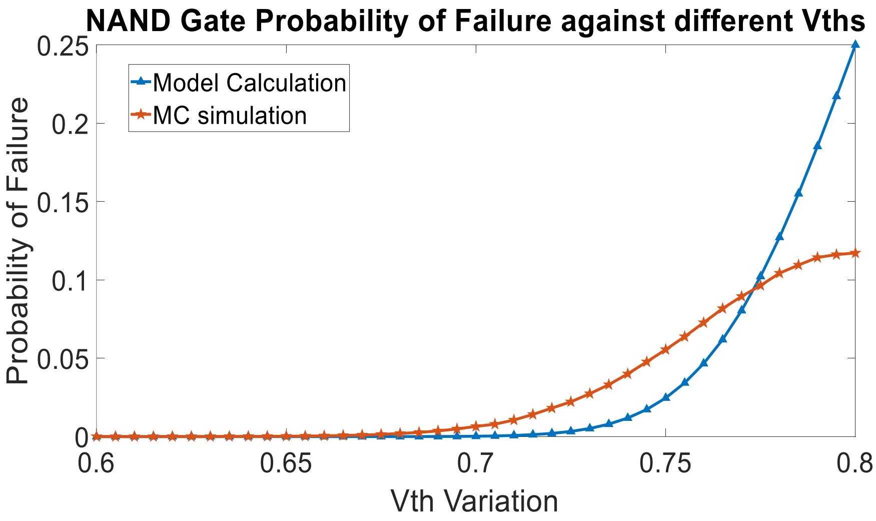


Figure 9. $pf_{NAND}$ trends as $V_{th}$ changes

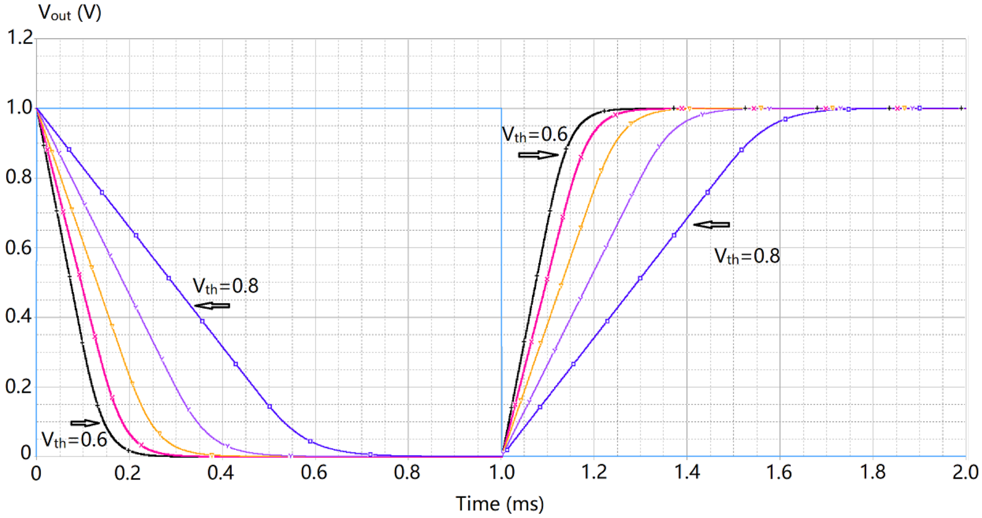


Figure 10. The transient process of output voltage for NAND gate

The third source of errors has something to do with the linear model of gate propagation delay evaluation. Fig. 9 shows the simulated delay of NAND gate using PSPICE with default MOSFET parameters, except the absolute threshold voltage of both PMOS and NMOS ranging from [0.6V, 0.8V] with increment of 0.05V (with the order from left to right in Fig. 10). The input A is set to be logic 1 and input B transits from 0 to 1 and back to 0.

It can be seen from the figure that when $\Delta V_{th}$ increases linearly, the $t_{pHL}$ and $t_{pLH}$ are not sharing same increment. However, when $\Delta V_{th}$ is relatively small, the resulting delay increase can be treated as quasi-linear.

## 5.7 Simulation Results

In this section, we show some circuit simulation results using the proposed method. We firstly take a close look at ISCAS'85 benchmark circuit C17, as shown in Fig. 5. Assuming the original gate PF is 0.1 (or gate reliability is 0.9), all primary inputs have probability of 0.5 and PF of 0, $a_{0_{NAND}} = 50$ , $a_{1_{NAND}} = 0.59$ (calibrated from PSPICE simulation data), $\sigma_{TH} = 30.28 * 10^{-3}V$ for 35nm technology for both NMOS and PMOS for simplicity, initial threshold voltage for PMOS and NMOS are -0.6V and 0.6V, respectively, and designed CLK frequency is 5GHz, i.e., $D_{GB} = 200ps$ (these settings are case sensitive and subject to change). Using (5.8), the standard deviation of delay can be found to be 4.58ps.

Table X. Mean of Signal Delay for C17

| **Node** | **Delay/Original Delay (ps)** | |
|---|---|---|
| | ***1 month*** | ***1 year*** |
| Inputs | 0/0 | 0/0 |
| 10, 11 | 59.04/50 | 63.45/50 |
| 16, 19 | 118.07/100 | 126.90/100 |
| 22, 23 | 176.72/150 | 189.77/150 |

Table XI. Spatial PFs for Elements in C17

| **Elements** | **Increment of PF** | | **Initial PF** |
|---|---|---|---|
| | ***1 month*** | ***1 year*** | |
| G1-G4 | 1.89E-8 | 4.42E-7 | 0.1 |
| G5, G6 | 2.43E-8 | 6.13E-7 | 0.1 |
| Output 22 | 2.10E-5 | 0.0006 | 0.224585 |
| Output 23 | 8.00E-6 | 0.0002 | 0.239580 |

The mean of signal delay is shown in Table X for $\mathrm{t} = 2.63 \cdot 10^6 s, 3.16 \cdot 10^7 s$ (i.e. a month, and a year).

Using (5.10), the corresponding PFs for one month was calculated as: $pf_{22}^T = pf_{23}^T = 0.0075$, which increases to 0.0328 after being degraded over one-year period. Considering the fact that originally the $pf_{22}^T = 1.4E^{-4}$ (without considering aging issue), the degradation of transistors worsens the circuit performance significantly based on the temporal analysis. With this information in mind, the designers can either reduce the CLK frequency or implement the circuit with shorter original delay to improve the 1-year performance and extend the circuit's expected lifetime. It should be noted that even for the same circuit with different input probabilities, these values vary. For example, if the input probabilities are zero, then the first two gates would be experiencing much more degradation and hence increasing the overall delay.

As for $pf_{22}^S$, the gate PF increment is different as well due to the input signal probability difference. The detailed changes of element (including gates and outputs) spatial PFs are shown in Table XI along with their initials, where the data of two outputs are simulated using Monte-Carlo simulation with $10^6$ iterations.

With information from Table X, Table XI and (5.1), given circuit operation time of 1 year, we have:

$$pf_{22} = pf_{22}^T + (1 - pf_{22}^T) \cdot pf_{22}^s = 0.2500$$

$$pf_{23} = pf_{23}^T + (1 - pf_{23}^T) \cdot pf_{23}^s = 0.2645$$

or

$$r_{22} = 0.7500, r_{23} = 0.7355$$

which represents around 3% degradation when compared to the reliabilities of $r_{22} = 0.7745$ and $r_{23} = 0.7606$ without considering aging effect. This is mostly due to an extra delay involved in this case, and the impact of spatial reliability was observed to be marginal. However, under situations where the $\Delta V_{th}$ varies significantly, the spatial reliability can play a critical role, as shown in Table VIII, and should be considered.

Similar simulations were also applied to other circuits with the results shown in Table XI. The reliability degradation rate is calculated by $\frac{pf(Aging)-pf(No\ Aging)}{1-pf(No\ Aging)} \cdot 100\%$. It should be noted that the designed delay guard band was set to be 15% more than the original delay, while the initial intrinsic delay was assumed to be 70ps for each gate (although they have different degradation rate) for simulations. The initial gate PF was set to 0.01. Again, these settings can vary according to specific situations. It can be seen from Table XI that the circuit degradation rate ranges from 1.5% to 8.2%. In addition, it was found that the $pf_O^S$ value does not change much regardless of the aging effect. Instead, the $pf_O^T$ is the main contributor to the reliability degradation. This suggests that the designers should focus on delay optimization to efficiently prolong the life of circuit operation.

Although the presented results are not verified using any hardware experiments with above-mentioned assumptions, this work shows an example of how to consider both spatial and temporal reliabilities and can be easily applied to practical cases by simply substituting the assumptions.

Table XII. Circuit Reliability Simulations on Some ISCAS’85 and ISCAS’89 Circuits with 1-Year Operation

| Circuit | Avg. Parameters with Aging | | | Avg. Parameters without Aging | | | Reliability Degradation Rate (%) |
|---|---|---|---|---|---|---|---|
| | $\boldsymbol{pf_O^T}$ | $\boldsymbol{pf_O^S}$ | $\boldsymbol{pf}$ | $\boldsymbol{pf_O^T}$ | $\boldsymbol{pf_O^S}$ | $\boldsymbol{pf}$ | |
| C432 | 0.023 | 0.081 | 0.110 | 0.000 | 0.078 | 0.078 | 3.47 |
| C499 | 0.049 | 0.102 | 0.144 | 0.000 | 0.098 | 0.098 | 5.10 |
| C880 | 0.062 | 0.163 | 0.212 | 0.000 | 0.156 | 0.156 | 6.64 |
| C2670 | 0.038 | 0.149 | 0.182 | 0.000 | 0.133 | 0.133 | 5.65 |
| C7552 | 0.044 | 0.207 | 0.243 | 0.000 | 0.175 | 0.175 | 8.24 |
| S27 | 0.069 | 0.030 | 0.060 | 0.000 | 0.029 | 0.029 | 3.19 |
| S298 | 0.051 | 0.062 | 0.071 | 0.000 | 0.057 | 0.057 | 1.48 |
| S349 | 0.022 | 0.080 | 0.093 | 0.000 | 0.074 | 0.074 | 2.05 |
| S444 | 0.034 | 0.075 | 0.089 | 0.000 | 0.070 | 0.070 | 2.04 |

## 5.8 Summary

In this chapter, we have proposed a new circuit-level reliability evaluation model to estimate signal reliability variations under aging effect. The model takes both spatial and temporal reliabilities into consideration and studies the reliability variations caused by the aging/NBTI effect. This helps designers predict potential performance degradation for circuits to operate over an extended time. Simulations on benchmark circuits have shown that the reliability degradation rate ranges from 1.5% to 8.2% over one-year period of operation, depending on specific circuits

# CHAPTER 6

# CONCLUSION AND FUTURE WORK

We have proposed a hybrid method to estimate both signal probability and reliability for combinational circuits by categorizing all signal pairs based on their correlation strength, under the assumption of zero-delay and constant gate reliability. The signal pairs with strong correlations are handled by analytic computation, leading to an accurate propagation of signal probability and reliability through logic gates. Those with relatively weak correlations are processed using local bitstream simulations which take signal correlations into consideration (to a maximum extent) with high efficiency. Additionally, we have proved that reconvergent-fanouts can be ignored when they are too far away from signals of interest through simulation. Therefore, signal pairs with extremely weak correlations are treated approximately as independent. This combination of analysis and simulation makes the proposed model competitive in terms of the tradeoff between accuracy and efficiency in estimating both signal probability and reliability simultaneously. While maintaining high accuracy, the efficiency improvement is twofold: 1. The proposed method has a linear time complexity. 2. The CPU time required for obtaining signal probability prior to reliability evaluation has been saved. These improvements significantly strengthen method scalability. For sequential circuit evaluation, we introduced a TSC model to speed up the convergence process. With the help of trial iterations, TSC can reduce the number of iterations needed by up to 42%, on average 28%.

Without the zero-delay and constant gate reliability assumptions, we further investigate signal reliability considering aging effect. We extend some existing device-level models and present circuit-level aging effect evaluations. We innovatively combine both temporal and spatial reliability into a proposed new index to help designers better predict circuit performance under aging effects. The simulation results show that circuit reliability degradation ranges from 1.5% to 8.2% over a one-year period of operation.

The suggested future work includes:

- Searching for better solutions for category 'N' correlation.

  Although the bitstream can provide accurate results, it is relatively time-consuming compared to analytical approaches. For better efficiency, this part of process time should be further shortened.

- Searching for supportive theory for TSC.

  We had an observation that when all gates are set to be reliable except for one specific gate, the output reliability is following a quasi-linear trend to this gate reliability. It might be a direction to find theoretical support for TSC linear regression model.

- Considering more factors for temporal reliability model

  In Chapter 5, the proposed temporal reliability model only considers late arrival. However, glitches as well as early arrivals of signals can affect circuit probability/reliability. Moreover, the CLK generator is experiencing aging effect, and the CLK signal may not be ideal after intense long-term operation, which should be considered simultaneously.

# REFERENCES/BIBLIOGRAPHY

[1] A. H. El-Maleh and K. A. K. Daud, “Simulation-based method for synthesizing soft error tolerant combinational circuits,” *IEEE Trans. Rel.*, vol. 64, no. 3, pp. 935–948, Sep. 2015.

[2] S. Borkar, “Designing reliable systems from unreliable components: The challenges of transistor variability and degradation,” *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov./Dec. 2005.

[3] N. R. Shanbhag *et al.*, “The search for alternative computational paradigms,” *IEEE Des. Test Comput.*, vol. 25, no. 4, pp. 334–344, Jul./Aug. 2008.

[4] C. Constantinescu, “Trends and challenges in VLSI circuit reliability,” *IEEE Micro*, vol. 23, no. 4, pp. 14-19, Jul./Aug. 2003

[5] H. Jahanirad, “CC-SPRA: Correlation coefficients approach for signal probability-based reliability analysis,” *IEEE Transactions on VLSI Systems*, vol. 27, no. 4, pp. 927-939, Apr. 2019.

[6] R.Y. Rubinstein and D.P. Kroese, *Simulation and the Monte Carlo Method*, 2nd Edition, John Wiley & Sons, New York, 2007

[7] J. von Neumann, “Probabilistic logics and synthesis of reliable organisms from unreliable components,” *Automata Studies*, Eds. Princeton, NJ, USA: Princeton Press, pp. 43–98, 2016.

[8] S. Krishnaswamy *et al*, “Probabilistic transfer matrices in symbolic reliability analysis of logic circuits,” *ACM Trans. Des. Automat. Electron. Syst.*, vol. 13, no. 1, pp. 1–35, Jan. 2008.

[9] N. Mohyuddin, E. Pakbaznia, and M. Pedram, “Probabilistic error propagation in logic circuits using the Boolean difference calculus,” *Proc. IEEE Int. Conf. Comput. Design*, pp. 7–13 Oct. 2008

[10] D. T. Franco, M. C. Vasconcelos, L. Naviner, and J.-F. Naviner, “Signal probability for reliability evaluation of logic circuits,” *Microelectron. Rel.*, vol. 48, pp. 1586–1591, Aug./Sep. 2008.

[11] J. T. Flaquer, J. M. Daveau, L. Naviner, and P. Roche, “Fast reliability analysis of combinatorial logic circuits using conditional probabilities,” *Microelectron. Rel.*, vol. 50, pp. 1215–1218, Sep./Nov. 2010.

[12] M. R. Choudhury and K. Mohanram, “Reliability analysis of logic circuits,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 3, pp. 392–405, Mar. 2009.

[13] J. Han *et al*, “Reliability evaluation of logic circuits using probabilistic gate models,” *Microelectron. Rel.*, vol. 51,no. 2, pp. 468–476, Feb. 2011.

[14] A. Abdollahi, “Probabilistic decision diagrams for exact probabilistic analysis,” *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (CAD)* , pp. 266–272, Nov. 2007

[15] Bo Yang *et al*, "An approach for digital Circuit Error/Reliability Propagation Analysis based on Conditional Probability," *2016 27th Irish Signals and Systems Conference (ISSC)*, pp. 1-6, Jun. 2016.

[16] T. Rejimon, K. Lingasubramanian, and S. Bhanja, “Probabilistic error modeling for nano-domain logic circuits,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 1, pp. 55–65, Jan. 2009.

[17] S. Ercolani, M. Favalli, M. Damiani, P. Olivo, and B. Ricco, “Estimate of signal probability in combinational logic networks,” *Proc. 1st Eur. Test Conf.*, pp. 132–138, Apr. 1989

[18] C. Chen and R. Xiao, “A fast model for analysis and improvement of gate-level circuit reliability,” *Integration*, vol. 50, pp. 107–115, Jun. 2015.

[19] J. T. Flaquer *et al*, "An approach to reduce computational cost in combinatorial logic netlist reliability analysis using circuit clustering and conditional probabilities," *2011 IEEE 17th International On-Line Testing Symposium*, pp. 98-103, 2011.

[20] Ramin Rajaei, “Single Event Double Node Upset Tolerance in MOS/Spintronic Sequential and Combinational Logic Circuits,” *Microelectronics Reliability*, vol. 69, pp 109-114, Dec. 2016.

[21] W. Elsharkasy *et al*, “Reliability Enhancement of Low-Power Sequential Circuits Using Reconfigurable Pulsed Latches,” *IEEE Transactions on Circuits and Systems I : Regular Pape*rs, vol. 64, no. 7, pp. 1803-1814, Jul. 2017

[22] M. Krstić *et al*, “Enhanced architectures for soft error detection and correction in combinational and sequential circuits,” *Microelectronics Reliability*, vol. 56 pp 212-220, Jan. 2016

[23] R. Ubar et al, “Fast identification of true critical paths in sequential circuits,” *Mircoelectronics reliability*, vol. 81, pp 252-261, Feb. 2018.

[24] H. Jahanirad, “Efficient reliability evaluation of combinational and sequential logic circuits,” *J Comput Electron* 18, 343–355, Dec. 2018.

[25] K. Mohammadi *et al*, "Fast Reliability Analysis Method for Sequential Logic Circuits," *2011 21st International Conference on Systems Engineering*, pp. 352-356, 2011.

[26] H. Jahanirad and K. Mohammadi “SEQUENTIAL LOGIC CIRCUITS RELIABILITY ANALYSIS,” *Journal of Circuits, Systems and Computers*, vol. 21, no. 5, 2012.

[27] B. Srinivasu and K. Sridharan, "A transistor-level probabilistic approach for reliability analysis of arithmetic circuits with applications to emerging technologies," *IEEE Transactions on Reliability,* vol. 66, no. 2, pp. 440-457, Jun. 2017.

[28] B. C. Paul *et al*, “Temporal performance degradation under NBTI: Estimation and design for improved reliability of nanoscale circuits,” *DATE*, pp. 780–785, 2006.

[29] W. Wang *et al*, “The impact of nbti on the performance of combinational and sequential circuits” *DAC*, pp. 364–369, Jun. 2007.

[30] M. A. Alam and S. Mahapatra. “A comprehensive model of PMOS NBTI degradation,” *Microelectronics Reliability*, vol. 45, 71–81, Aug. 2005.

[31] S. Bhardwaj *et al*. “Predictive modeling of the nbti effect for reliable design,” *CICC*, pp. 189–192, Sep. 2006.

[32] S. V. Kumar *et al*. “An analytical model for negative bias temperature instability,” *ICCAD*, pp. 493–496, 2006.

[33] W. Wang, *et al “*An efficient method to identify critical gates under circuit aging.” *IEEE/ACM International Conference on Computer-Aided Design*, pp. 735-740, Nov. 2007.

[34] A. Asenov *et al*. “Intrinsic threshold voltage fluctuations in decanano MOSFETs due to local oxide thickness variations.” *IEEE Trans. Electr. Dev.,* vol. 49, pp. 112–119, Jan. 2002.

[35] G. Roy *et al*. “Simulation study of individual and combined sources of intrinsic parameter fluctuations in conventional nano-MOSFETs,” *IEEE Trans. Electr. Dev.,* vol. 53, pp. 3063–3070, Dec., 2006.

[36] K. Kuhn *et al*. “Managing process variations in Intel’s 45nm CMOS technology.” *Intel Tech. J.*, vol. 12, pp. 93–109, Jun. 2008.

[37] B. Cheng *et al.* “Evaluation of intrinsic parameter fluctuations on 45, 32 and 22nm technology node LP n-MOSFETs.” *Proc. European Solid-State Dev. Res. Conf. ESSDERC’08,* pp. 47–50, Sep., 2008.

[38] C. Alexander *et al*, “Random dopant-induced drain current variation in nano-MOSFETs: A three-dimensional self-consistent Monte Carlo simulation study using “ab initio” ionized impurity scattering.” *IEEE Trans. Electr. Dev.,* vol. 55, pp. 3251–3258, Nov., 2008.

[39] Y. Li *et al.* “Large-scale atomistic approach to random dopant-induced characteristic variability in nanoscale CMOS digital and high-frequency integrated circuits.” *ICCAD*, pp. 278–285, Nov. 2008.

[40] Ibrahim, W., and Beiu, V., “Reliability of NAND-2 CMOS gates from threshold voltage variations.” *IEEE International Conference on Innovations in Information Technology (IIT)*, pp. 135-139, 2009

[41] De, S *et al*, “Negative bias temperature instability (NBTI) effects on p-Si/n-InGaAs hybrid CMOSFETs for digital applications,” *Microsystem Technologies*, vol. 26, no. 4, pp. 1173-1178, 2020.

[42] Aryan, N. P. et al, “From an analytic NBTI device model to reliability assessment of complex digital circuits” *IEEE 20th International On-Line Testing Symposium (IOLTS)*, pp. 19-24, Jul. 2014

[43] T. Sakurai, “Alpha-Power Law MOSFET Model and its Applications to CMOS Inverter Delay and Other Formulas”, *IEEE Journal of Solid-State Circuits*, vol. 25, No.2, Apr. 1990.

[44] R. Chadha and J. bhasker, “Static Timing Analaysis for Nanometer Designs”, *Springer*, ISBN 978-0-387-93819-6, 2009

[45] L. Scheffer *et al*, “Electronic Design Automation for Integrated Circuits Handbook”, vol. II, Chapter 8. ISBN 0-8493-3096-3. 2005

[46] J. A. Delport and C. J. Fourie, "A Static Timing Analysis Tool for RSFQ and ERSFQ Superconducting Digital Circuit Applications," *IEEE Transactions on Applied Superconductivity*, vol. 28, no. 5, pp. 1-5, , Art no. 1300705, Aug., 2018.

[47] J. Han *et al*. "A Stochastic Computational Approach for Accurate and Efficient Reliability Evaluation," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1336 - 1350, Jun., 2014.

[48] M. Hansen *et al*, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," *IEEE Design and Test*, vol. 16, no. 3, pp. 72-80, Jul./Sep.. 1999

[49] L. Amarú *et al*, “The EPFL combinational benchmark suite,” *Proc. the 24th International Workshop on Logic & Synthesis (IWLS)*, pp. 57-61. 2015.

[50] S. Zhan and C. Chen, “Circuit Reliability Analysis with Consideration of Aging Effect,” *Proc. 35th Symposium on Integrated Circuits and Systems Design (SBCCI)*, Aug. 2022.

[51] S. Zhan and C. Chen, “An Efficient Method for Sequential Circuit Reliability Estimation,” *Proc. 65th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug. 2022.

[52] S. Zhan and C. Chen, “A Hybrid Method for Signal Probability and Reliability Estimation with Combinational Circuits,” *Integration, the VLSI Journal* (In press), 2022.

APPENDICES

## Appendix A. Important MATLAB Code

**The MATLAB code used for this thesis is attached in this appendix. The code and functions are in same order as the chapters.**

**The following codes are used for obtaining MC simulation results for combinational circuits.**

```
clear
clc;
Circuit = {'C17'};
% % load(['Circuit_decomp/',Circuit{1},'.mat']);
load([Circuit{1},'.mat']);


%% Basic Parameters
% Num_node   = max(max(Gates));
% Num_gate   = size(Gates,1);
% Num_input  = size(Input,1);
% Num_output = size(Output,1);
Ind        = Gates(:,1:3);
MC         = 10^6;
Pin        = 0.5*ones(1,Num_input);
rin        = ones(1,Num_input);
Gates(Gates(:,5)==7,5)=8;
rg = 0.9;
% small nodes first, large nodes second
tic
% for i = 1:Num_gate
%     if Gates(i,2)>Gates(i,3)
%         Gates(i,2:3) = [Gates(i,3),Gates(i,2)];
%     end
% end
%% Counters Initialization
tic;
NodevecStar        = zeros(Num_node,1);
Nodevec            = zeros(Num_node,1);

Pstarc             = zeros(Num_node,1);
rallc              = zeros(Num_node,2);
%% MC simulation
for i=1:MC
    NodevecStar        = zeros(Num_node,1);
    %% Input Setup regarding to Pin, Node Reliability Setup
    for k = 1:Num_input
        NodevecStar(Input(k)) = (rand(1)<Pin(k));
        if rand(1)<rin(k)
            Nodevec(Input(k)) = NodevecStar(Input(k));
        else
            Nodevec(Input(k)) = 1-NodevecStar(Input(k));
        end
    end
    for j = 1:Num_gate
        x=Gates(j,:);
        %% simulate error-free value and real value
        %% Gate type: 1-NAND, 2-AND, 3-NOR, 4-OR, 5-NOT, 6-
BUFF, 7-XNOR, 8-XOR
        switch x(5)
            case {1}
                NodevecStar(x(1)) = 1-
and(NodevecStar(x(2)), NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = 1-and(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = and(Nodevec(x(2)),
Nodevec(x(3)));
                end
            case {2}
                NodevecStar(x(1)) = and(NodevecStar(x(2)),
NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = and(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = 1-and(Nodevec(x(2)),
Nodevec(x(3)));
                end
            case {3}
                NodevecStar(x(1)) = 1-or(NodevecStar(x(2)),
NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = 1-or(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = or(Nodevec(x(2)),
Nodevec(x(3)));
                end
            case {4}
```

```
                NodevecStar(x(1)) = or(NodevecStar(x(2)),
NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = or(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = 1-or(Nodevec(x(2)),
Nodevec(x(3)));
                end

            case {5}
                NodevecStar(x(1)) = not(NodevecStar(x(2)));
                if rand(1)<rg
                    Nodevec(x(1)) = not(Nodevec(x(2)));
                else
                    Nodevec(x(1)) = 1-not(Nodevec(x(2)));
                end

            case {6}
                NodevecStar(x(1)) = NodevecStar(x(2));
                if rand(1)<rg
                    Nodevec(x(1)) = Nodevec(x(2));
                else
                    Nodevec(x(1)) = 1-Nodevec(x(2));
                end

            case {7}
                NodevecStar(x(1)) = 1-
xor(NodevecStar(x(2)), NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = 1-xor(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = xor(Nodevec(x(2)),
Nodevec(x(3)));
                end

            case {8}
                NodevecStar(x(1)) = xor(NodevecStar(x(2)),
NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = xor(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = 1-xor(Nodevec(x(2)),
Nodevec(x(3)));
```

```
                end

        end
        %% Calculate Node Probability
        bitStreamMatStar(:,i)= NodevecStar;
        bitStreamMat(:,i)    = Nodevec;
        if NodevecStar(x(1)) == 1
            Pstarc(x(1)) = Pstarc(x(1)) + 1;
            if Nodevec(x(1)) == 1
                rallc(x(1),2) = rallc(x(1),2)+1;
            end
        else
            if Nodevec(x(1)) == 0
                rallc(x(1),1) = rallc(x(1),1)+1;
            end
        end

    end
    i
end

G_ind = [Gates(:,1:3),Gates(:,5)];
r = rallc./[MC-Pstarc,Pstarc];
TMC = toc;
toc
sum(bitStreamMat(22,:)==bitStreamMatStar(22,:))/MC
sum(bitStreamMat(23,:)==bitStreamMatStar(23,:))/MC
% save bitStreamC17.mat

% eval(['save
BitStreamMat\bitStream',num2str(rg),'_',Circuit{1},'_',num2
str(MC),'.mat bitStreamMat bitStreamMatStar'])
% eval(['save bitStream',Circuit{1},'.mat bitStreamMat
bitStreamMatStar'])
```

**The following codes are used to find frequency of occurrences for signal correlations within circuit**

```
clear
clc;
close all

Circuit = {'C1355'};
% load(['Circuit\',Circuit{1},'.mat']);
load(['Circuit_decomp\',Circuit{1},'.mat'])


Num_node   = max(max(Gates));
Num_gate   = size(Gates,1);
Num_input  = size(Input,1);
Num_output = size(Output,1);


%% Gate type: 1-NAND, 2-AND, 3-NOR, 4-OR, 5-NOT, 6-BUFF, 7-
XNOR, 8-XOR
%% Initialization
Gates(Gates(:,4)==7,4)=8;
gatestrx   =
{'NAND','AND','NOR','OR','NOT','BUFF','XNOR','XOR'};
invPair = Gates(Gates(:,3)==0,[2,1,5]); % signal 1, signal
2, Type (1 = not, 2 = buff)
tic;
cMat = []; % Correlation type matrix
sixc = 0;
C = [];
% bitStream_MC;
tic
for ipn = 1:Num_gate
    gateVec    = Gates(ipn,:);
   [cMat,sixc] = CorrCate(gateVec(2),gateVec(3), cMat,
Gates, Input, invPair);
    ipn

end
T = toc;
Type = {'S1';'S2';'S3';'N1';'N2';'N3';'I';'N3-2';'N3-
3';'N3-4';'Approx.I'};
Occurence = zeros(11,1);
for i = 1:size(cMat)
    X = cMat(i,:);
    Occurence(X(3))= Occurence(X(3))+1;
    if X(3)==6
        if X(4)~=6
            Occurence(8)=Occurence(8)+1;
        elseif X(5)~=6
            Occurence(9)=Occurence(9)+1;
        elseif X(6)~=6
            Occurence(10)=Occurence(10)+1;
        else
            Occurence(11)=Occurence(11)+1;
        end
    end
end
Occurence = Occurence([1,2,3,4,5,7,11,8,9,10]);
Type = Type([1,2,3,4,5,7,11,8,9,10]);
CTable = rows2vars(table(Type,Occurence));

Per =
[sum(Occurence(1:3)),sum(Occurence(4:5)),sum(Occurence(6:7)
),...
    Occurence(8),Occurence(9),Occurence(10)]/Num_gate
sum(Per)

sum(Occurence(1:10))


function [cMat,sixc] = CorrCate(n1, n2, cMat,  Gates,
Input,...
    invPair)
%find out correlation categories for each gate of a given
circuit
%% correlation type and order detection
sixc = 0;
h = [0,0,0,0]; % correlation type
Lis1 = n1; Lis2 = n2; % Lists for input pair detection on
level 1-3s
if n2~=0
    [v1,s1] = in_L4New(n1, Gates, Input);
    [v2,s2] = in_L4New(n2, Gates, Input);
    G1 = Gates(Gates(:,1)==n1,:);
    G2 = Gates(Gates(:,1)==n2,:);

    hi2 = [];
    for f = 1:4 %correlation detection flag
        for i1 = 1:size(Lis1,2)
            for i2 = 1:size(Lis2,2)
```

```
                hi1 = corrType(Lis1(i1),Lis2(i2), Gates,
Input, invPair);
                hi2 = [hi2;Lis1(i1),Lis2(i2),hi1];
            end
        end
        hi2 = sortrows(hi2,3);
        h(f) = hi2(1,3);
        if hi2(1,3)~=6
            break;
        else
            Lis1 = v1(s1(f)+1:s1(f+1));
            Lis2 = v2(s2(f)+1:s2(f+1));
        end
    end
    if h == [6,6,6,6]
        sixc = sixc + 1;
        h = [7,0,0,0];
    end
else
    h = [7,0,0,0]; %independent for inverter
end
cMat = [cMat;n1,n2,h];
end
function [h] = corrType(n1, n2, Gates, Input, invPair)
h = 0;
% [v1,s1] = in_L3New(n1, Gates, Input);
% [v2,s2] = in_L3New(n2, Gates, Input);
[v1,s1] = in_L4New(n1, Gates, Input);
[v2,s2] = in_L4New(n2, Gates, Input);
% G1 = Gates(Gates(:,1)==n1,:);
% G2 = Gates(Gates(:,1)==n2,:);

if n1==n2 || ismember([n1,n2],invPair(:,[1,2]),'rows')...
        || ismember([n2,n1],invPair(:,[1,2]),'rows')
    h = 1; % S1
elseif sum(ismember(v2(1:s2(1)),v1(s1(1)+1:s1(2))))||...
        sum(ismember(v1(1:s1(1)),v2(s2(1)+1:s2(2))))
    h = 2; % S2
elseif ~isempty(v1(s1(1)+1:s1(2))) && ...

(sum(ismember(v1(s1(1)+1:s1(2)),v2(s2(1)+1:s2(2))),'all')==
2||sum(ismember(v2(s2(1)+1:s2(2)),v1(s1(1)+1:s1(2))),'all')
==2)
    h = 3; % S3
elseif sum(ismember(v2(1:s2(1)),v1(s1(2)+1:s1(3))))||...
        sum(ismember(v1(1:s1(1)),v2(s2(2)+1:s2(3))))
    h = 4; % N1
elseif sum(ismember(v2(1:s2(1)),v1(s1(3)+1:s1(4))))||...
        sum(ismember(v1(1:s1(1)),v2(s2(3)+1:s2(4))))
    h = 5; % N2
elseif ismember(n1,Input) || ismember(n2,Input) && n1~=n2
    h = 7; % I, independent is the least correlated case
else
    h = 6;
end
end

function [v,s] = in_L4New(node1,Gates,Input)
%% return inputs from previous 4 levels. Inverter and
buffer is not counted
v = [];
s = zeros(1,5);
L = 1;
vOut = node1;
if ismember(node1,Input)
    v = node1;
    s(1) = 1;
else
    while L < 6
        i = 1;
        vPresent{L} = [vOut];
        [vL1,vL2] = deal([],[]);
        while i < size(vOut,2)+1
            if ismember(vOut(1,i),Input)
                i = i+1;
                continue;
            else
                vInP = Gates(Gates(:,1) == vOut(1,i),:);
                if ismember(vInP(5),[5,6])
                    vL1 = [vL1,vInP(2)];
                    vOut(1,i) = vL1(end);
                else
                    vL2 = [vL2,vInP(2:3)];
                    i = i+1;
                end
            end
        end
        vPresent{L} = [vPresent{L},vL1];
        s(L) = size(vPresent{L},2);
        vOut = vL2;
        L = L+1;
    end
    for i = 1:L-1
        v = [v,vPresent{i}];
    end
    s = cumsum(s);
end

end
```

**The following codes are used for category S calculations**

```
function [nP,jP,jC] = pS1(n1,n2,P,R,Gates,PM,C,Input,...
    invPair,bitStreamMatStar,rg)
% node pair-nP, joint Probability-jP, joint reliability-jR
jP  = [P(n1,1), 0, 0, P(n1,2)];
jC  = [R(n1,1), 0, 0, 1-R(n1,1);0,0,0,0;0,0,0,0;1-
R(n1,2),0,0,R(n1,2)];
MB  = [rg,1-rg,0,0;1-rg,rg,0,0;0,0,rg,1-rg;0,0,1-rg,rg];
if n1==n2
    nP = [n1, n1];
else
    [n1,n2] = isinput(n1,n2,Gates);
    nP = [n1,n2]; % min first in PM matrix
    if invPair(invPair(:,1:2)==[n1,n2],3)==5
        jP = [0,P(n1,1),P(n1,2),0];
        jRS = [0,0,0,0;0,R(n1,1),1-R(n1,1),0;0,1-
R(n1,2),R(n1,2),0;0,0,0,0];
        jC = jRS*MB;
    else
        jC = jC*MB;
    end
end
end


function [nP,jP,jR] =
pS2(n1,n2,P,R,Gates,PM,C,Input,invPair,bitStreamMatStar)
[v1,s1] = in_L3New(n1, Gates, Input);
[v2,s2] = in_L3New(n2, Gates, Input);
G1 = Gates(Gates(:,1)==n1,:);
G2 = Gates(Gates(:,1)==n2,:);

fOutInv = 0;
fPairSwap = 0;
fInInv = 0;
if sum(ismember(v1(1:s1(1)),v2(s2(1)+1:s2(2))))
    L1 = v1(1:s1(1));
    x = L1(ismember(L1,v2(s2(1)+1:s2(2))));
    x = x(1);
    if x ~= n1
        fInInv = 1;
    end
    if ~isempty(invPair)
        if ismember(n2,invPair(:,1))
            vecInv2 = invPair(invPair(:,1)==n2,:);
```

```
                z = vecInv2(2);
                if vecInv2(3)==1
                    fOutInv = 1;
                end
                if ismember(z,invPair(:,1))
                    z = invPair(invPair(:,1)==z,2);
                    if vecInv2(3)==1
                        fOutInv = 1;
                    end
                end
            else
                z = n2;
            end
        else
            z = n2;
        end
        G = Gates(Gates(:,1)==z,:);
        inPair = G(2:3);
        inPair(inPair==0) = [];
        xPrime = invPair(invPair(:,2)==x,1);
        if ismember(xPrime,inPair)
            fInInv = 1;
            x = xPrime;
        end
        y = inPair(inPair~=x);
        gateType = G(4);
    elseif sum(ismember(v2(1:s2(1)),v1(s1(1)+1:s1(2))))
        L1 = v2(1:s2(1));
        x = L1(ismember(L1,v1(s1(1)+1:s1(2))));
        x = x(1);
        if x ~= n2
            fInInv = 1;
        end
        fPairSwap = 1;
        if ~isempty(invPair)
            if ismember(n1,invPair(:,1))
                vecInv1 = invPair(invPair(:,1)==n1,:);
                z = vecInv1(2);
                if vecInv1(3)==1
                    fOutInv = 1;
                end
                if ismember(z,invPair(:,1))
                    z = invPair(invPair(:,1)==z,2);
                    if vecInv1(3)==1
                        fOutInv = 1;
                    end
                end
            else
                z = n1;
            end
        else
            z = n1;
        end

        G = Gates(Gates(:,1)==z,:);
        inPair = G(2:3);
        inPair(inPair==0) = [];
        xPrime = invPair(invPair(:,2)==x,1);
        if ismember(xPrime,inPair)
            fInInv = 1;
            x = xPrime;
        end
        y = inPair(inPair~=x);
        gateType = G(4);
    end
    pVec = PM(sum(ismember(PM(:,1:2),[x,y]),2)==2,:);
    CXY  = C{sum(ismember(PM(:,1:2),[x,y]),2)==2};
    if size(pVec,1)>1
        pVec = pVec(1,:);
    end
    if x == pVec(1)
        pXY = pVec(3:6);
    else
        pXY = pVec([3,5,4,6]);
        CXY = CXY([1,3,2,4],[1,3,2,4]);
    end
    %% Gate type: 1-NAND, 2-AND, 3-NOR, 4-OR, 5-NOT, 6-BUFF, 7-
    XNOR, 8-XOR
    mAND = [1,0,0,0;1,0,0,0;0,0,1,0;0,0,0,1];
    mOR  = [1,0,0,0;0,1,0,0;0,0,0,1;0,0,0,1];
    mXOR = [1,0,0,0;0,1,0,0;0,0,0,1;0,0,1,0];
    mBUF = [1,0,0,0;0,0,0,0;0,0,0,0;0,0,0,1];
    switch gateType
        case {1,2}
            mT = mAND;
            TM =
    (pXY./[pXY(1)+pXY(2),pXY(1)+pXY(2),pXY(3),pXY(4)]);
        case {3,4}
            mT = mOR;
            TM =
    (pXY./[pXY(1),pXY(2),pXY(3)+pXY(4),pXY(3)+pXY(4)]);
        case {5,6}
```

```
        mT = mBUF;
    case {7,8}
        mT = mXOR;
        TM = [1,1,1,1];
end
if mod(gateType,2)==1
    mModify = [0,1,0,0;1,0,0,0;0,0,0,1;0,0,1,0];
else
    mModify = 1;
end
jP = pXY * mT * mModify;
jC = mT'*diag(TM)*CXY*mT;
if fOutInv == 1
    jP = jP([2,1,4,3]);
    jC = jC([2,1,4,3],[2,1,4,3]);
end
if fInInv == 1
    jP = jP([3,4,1,2]);
     jC = jC([3,4,1,2],[3,4,1,2]);
end
nP = [n1,n2];
if fPairSwap == 1
    jP = jP([1,3,2,4]);
end
end

function [nP,jP,jR] = pS3(n1, n2, Pstar, Gates, PM, C, ...
    Input,invPair,bitStreamMatStar,bitStreamMat)
[v1,s1] = in_L3New(n1, Gates, Input);
[v2,s2] = in_L3New(n2, Gates, Input);
G1 = Gates(Gates(:,1)==n1,:);
G2 = Gates(Gates(:,1)==n2,:);

fOutInvA = 0;fOutInvB = 0;
fPairSwap = 0;
fInInvXA = 0;fInInvXB = 0; fInInvYA = 0; fInInvYB = 0;
x =
if s1(1)==2
    fOutInvA = 1;
end
if s2(1)==2
    fOutInvB = 1;
end
if ~ismember(x,[v1(s(1)+1),v1(s(1)+2)])
    fInInvXA = 1;
end
```

```
if sum(ismember(v1(1:s1(1)),v2(s2(1)+1:s2(2))))
    L1 = v1(1:s1(1));
    x = L1(ismember(L1,v2(s2(1)+1:s2(2))));
    x = x(1);
    if x ~= n1
        fInInv = 1;
    end
    if ~isempty(invPair)
        if ismember(n2,invPair(:,1))
            vecInv2 = invPair(invPair(:,1)==n2,:);
            z = vecInv2(2);
            if vecInv2(3)==1
                fOutInv = 1;
            end
            if ismember(z,invPair(:,1))
                z = invPair(invPair(:,1)==z,2);
                if vecInv2(3)==1
                    fOutInv = 1;
                end
            end
        else
            z = n2;
        end
    else
        z = n2;
    end
    G = Gates(Gates(:,1)==z,:);
    inPair = G(2:3);
    inPair(inPair==0) = [];
    xPrime = invPair(invPair(:,2)==x,1);
    if ismember(xPrime,inPair)
        fInInv = 1;
        x = xPrime;
    end
    y = inPair(inPair~=x);
    gateType = G(4);
elseif sum(ismember(v2(1:s2(1)),v1(s1(1)+1:s1(2))))
    L1 = v2(1:s2(1));
    x = L1(ismember(L1,v1(s1(1)+1:s1(2))));
    x = x(1);
    if x ~= n2
        fInInv = 1;
    end
    fPairSwap = 1;
    if ~isempty(invPair)
```

```
        if ismember(n1,invPair(:,1))
            vecInv1 = invPair(invPair(:,1)==n1,:);
            z = vecInv1(2);
            if vecInv1(3)==1
                fOutInv = 1;
            end
            if ismember(z,invPair(:,1))
                z = invPair(invPair(:,1)==z,2);
                if vecInv1(3)==1
                    fOutInv = 1;
                end
            end
        else
            z = n1;
        end
    else
        z = n1;
    end

    G = Gates(Gates(:,1)==z,:);
    inPair = G(2:3);
    inPair(inPair==0) = [];
    xPrime = invPair(invPair(:,2)==x,1);
    if ismember(xPrime,inPair)
        fInInv = 1;
        x = xPrime;
    end
    y = inPair(inPair~=x);
    gateType = G(4);
end
pVec = PM(sum(ismember(PM(:,1:2),[x,y]),2)==2,:);
CXY  = C{sum(ismember(PM(:,1:2),[x,y]),2)==2};
if size(pVec,1)>1
    pVec = pVec(1,:);
end
if x == pVec(1)
    pXY = pVec(3:6);
else
    pXY = pVec([3,5,4,6]);
    CXY = CXY([1,3,2,4],[1,3,2,4]);
end
%% Gate type: 1-NAND, 2-AND, 3-NOR, 4-OR, 5-NOT, 6-BUFF, 7-
XNOR, 8-XOR
mAND = [1,0,0,0;1,0,0,0;0,0,1,0;0,0,0,1];
mOR  = [1,0,0,0;0,1,0,0;0,0,0,1;0,0,0,1];
mXOR = [1,0,0,0;0,1,0,0;0,0,0,1;0,0,1,0];
```

```
mBUF = [1,0,0,0;0,0,0,0;0,0,0,0;0,0,0,1];
switch gateType
    case {1,2}
        mT = mAND;
        TM =
(pXY./[pXY(1)+pXY(2),pXY(1)+pXY(2),pXY(3),pXY(4)]);
    case {3,4}
        mT = mOR;
        TM =
(pXY./[pXY(1),pXY(2),pXY(3)+pXY(4),pXY(3)+pXY(4)]);
    case {5,6}
        mT = mBUF;
    case {7,8}
        mT = mXOR;
        TM = [1,1,1,1];
end
if mod(gateType,2)==1
    mModify = [0,1,0,0;1,0,0,0;0,0,0,1;0,0,1,0];
else
    mModify = 1;
end
jP = pXY * mT * mModify;
jC = mT'*diag(TM)*CXY*mT
if fOutInv == 1
    jP = jP([2,1,4,3]);
    jC = jC([2,1,4,3],[2,1,4,3]);
end
if fInInv == 1
    jP = jP([3,4,1,2]);
    jC = jC([3,4,1,2],[3,4,1,2]);
end
nP = [n1,n2];
if fPairSwap == 1
    jP = jP([1,3,2,4]);
end
end
```

**The following codes are used for bitstream generations.**
**For generation based on one existed sequence:**

```
function [SimPvec,SimRvec] =
bitGen1(n1,SimP2,SimR2,CJM,bitLength)
%generate bitstream for n1 based on exist n2
SimPvec = [n1];
SimRvec = [n1];
for ilength = 1:bitLength
    ran = rand(1);
    if [SimP2(ilength),SimR2(ilength)]==[0,0]
        CJMvec = [CJM(1,1),CJM(1,3),CJM(3,1),CJM(3,3)];
        CJMvec = cumsum(CJMvec/sum(CJMvec));
        if ran < CJMvec(1)
            SimPvec = [SimPvec,0];
            SimRvec = [SimRvec,0];
        elseif ran<CJM(2)
            SimPvec = [SimPvec,0];
            SimRvec = [SimRvec,1];
        elseif ran<CJM(3)
            SimPvec = [SimPvec,1];
            SimRvec = [SimRvec,0];
        else
            SimPvec = [SimPvec,1];
            SimRvec = [SimRvec,1];
        end
    elseif [SimP2(ilength),SimR2(ilength)]==[0,1]
        CJMvec = [CJM(1,2),CJM(1,4),CJM(3,2),CJM(3,4)];
        CJMvec = cumsum(CJMvec/sum(CJMvec));
        if ran < CJMvec(1)
            SimPvec = [SimPvec,0];
            SimRvec = [SimRvec,0];
        elseif ran<CJM(2)
            SimPvec = [SimPvec,0];
            SimRvec = [SimRvec,1];
        elseif ran<CJM(3)
            SimPvec = [SimPvec,1];
            SimRvec = [SimRvec,0];
        else
            SimPvec = [SimPvec,1];
            SimRvec = [SimRvec,1];
        end
    elseif [SimP2(ilength),SimR2(ilength)]==[1,0]
        CJMvec = [CJM(2,1),CJM(2,3),CJM(4,1),CJM(4,3)];
        CJMvec = cumsum(CJMvec/sum(CJMvec));
        if ran < CJMvec(1)
            SimPvec = [SimPvec,0];
            SimRvec = [SimRvec,0];
        elseif ran<CJM(2)
            SimPvec = [SimPvec,0];
            SimRvec = [SimRvec,1];
        elseif ran<CJM(3)
            SimPvec = [SimPvec,1];
            SimRvec = [SimRvec,0];
        else
            SimPvec = [SimPvec,1];
            SimRvec = [SimRvec,1];
        end
    else
        CJMvec = [CJM(2,2),CJM(2,4),CJM(4,2),CJM(4,4)];
        CJMvec = cumsum(CJMvec/sum(CJMvec));
        if ran < CJMvec(1)
            SimPvec = [SimPvec,0];
            SimRvec = [SimRvec,0];
        elseif ran<CJM(2)
            SimPvec = [SimPvec,0];
            SimRvec = [SimRvec,1];
        elseif ran<CJM(3)
            SimPvec = [SimPvec,1];
            SimRvec = [SimRvec,0];
        else
            SimPvec = [SimPvec,1];
            SimRvec = [SimRvec,1];
        end
    end
end
SimPvec = [SimPvec;SimP2];
SimRvec = [SimRvec;SimR2];
end
```

**Generating 2 bitstreams when both signals are not generated**

```
 function [SimPvec,SimRvec] =
bitGen2(n1,n2,pVec,CJM,bitLength)
% generate 2 bitstreams when both n1 and n2 are not
generated
SimPvec = [n1;n2];
SimRvec = [n1;n2];
for ilength = 1:bitLength
    pVec = cumsum(pVec);
    ranP = rand(1);
    ranR = rand(1);
    if ranP<pVec(1)
        SimPvec = [SimPvec,[0;0]];
        CJMvec = cumsum(CJM(1,:));
        if ranR < CJMvec(1)
            SimRvec = [SimRvec,[0;0]];
        elseif ranR<CJMvec(2)
            SimRvec = [SimRvec,[0;1]];
        elseif ranR<CJMvec(3)
            SimRvec = [SimRvec,[1;0]];
        else
            SimRvec = [SimRvec,[1;1]];
        end
    elseif ranP<pVec(2)
        SimPvec = [SimPvec,[0;1]];
        CJMvec = cumsum(CJM(2,:));
        if ranR < CJMvec(1)
            SimRvec = [SimRvec,[0;0]];
        elseif ranR<CJMvec(2)
            SimRvec = [SimRvec,[0;1]];
        elseif ranR<CJMvec(3)
            SimRvec = [SimRvec,[1;0]];
        else
            SimRvec = [SimRvec,[1;1]];
        end
    elseif ranP<pVec(3)
        SimPvec = [SimPvec,[1;0]];
        CJMvec = cumsum(CJM(3,:));
        if ranR < CJMvec(1)
            SimRvec = [SimRvec,[0;0]];
        elseif ranR<CJMvec(2)
            SimRvec = [SimRvec,[0;1]];
        elseif ranR<CJMvec(3)
            SimRvec = [SimRvec,[1;0]];
        else
```

```
            SimRvec = [SimRvec,[1;1]];
        end
    else
        SimPvec = [SimPvec,[1;1]];
        CJMvec = cumsum(CJM(4,:));
        if ranR < CJMvec(1)
            SimRvec = [SimRvec,[0;0]];
        elseif ranR<CJMvec(2)
            SimRvec = [SimRvec,[0;1]];
        elseif ranR<CJMvec(3)
            SimRvec = [SimRvec,[1;0]];
        else
            SimRvec = [SimRvec,[1;1]];
        end
    end
end
end
```

**The following codes are used to deal with category N**

```
function [nP,jP,jC] =
pS4(n1,n2,P,R,Gates,PM,C,Input,invPair,bitStreamMatStar,bit
StreamMat,rg,bitLength)
SimPbit = [];
SimRbit = [];
path = findpathN1(n1,n2,Gates,Input, invPair);
for iBit = 1:size(path,2)
    G = Gates(Gates(:,1)==path(iBit),:);
    %% input pair bitstream generation
    if G(3)~=0
        pVec =
PM(sum(ismember(PM(:,1:2),[G(2),G(3)]),2)==2,:);
        CJM  =
C{sum(ismember(PM(:,1:2),[G(2),G(3)]),2)==2};
        if G(2) == pVec(1)
            pVec = pVec(3:6);
        else
            pVec = pVec([3,5,4,6]);
            CJM = CXY([1,3,2,4],[1,3,2,4]);
        end
        if ~isempty(SimPbit)
            n1Exi = ismember(G(2),SimPbit(:,1)); % check if
bitstream existed or not
            n2Exi = ismember(G(3),SimPbit(:,1));
        else
            n1Exi = 0;
            n2Exi = 0;
        end
        if n1Exi
            SimP2 = SimPbit(SimPbit(:,1)==G(2),:);
            SimR2 = SimPbit(SimRbit(:,1)==G(2),:);
            [SimPvec,SimRvec] =
bitGen1(G(3),SimP2,SimR2,CJM([1,3,2,4],[1,3,2,4]),bitLength
);
            SimPbit = [SimPbit;SimPvec(1,:)];
            SimRbit = [SimRbit;SimRvec(1,:)];
        elseif n2Exi
            SimP2 = SimPbit(SimPbit(:,1)==G(3),:);
            SimR2 = SimPbit(SimRbit(:,1)==G(3),:);
            [SimPvec,SimRvec] =
bitGen1(G(2),SimP2,SimR2,CJM,bitLength);
            SimPbit = [SimPbit;SimPvec(1,:)];
            SimRbit = [SimRbit;SimRvec(1,:)];
        else
```

```
            [SimPvec,SimRvec] =
bitGen2(G(2),G(3),pVec,CJM,bitLength);
            SimPbit = [SimPbit;SimPvec];
            SimRbit = [SimRbit;SimRvec];
        end
        %% propagate to output for current gate on path
        SimPOut(1) = G(1);
        SimROut(1) = G(1);
        for iprop = 1:bitLength
            switch G(end)
                case {1}
                    SimPOut(1,iprop+1) = 1-
and(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(1,iprop+1) = 1-
and(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
                        SimROut(1,iprop+1) =
and(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end
                case {2}
                    SimPOut(1,iprop+1) =
and(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(1,iprop+1) =
and(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
                        SimROut(1,iprop+1) = 1-
and(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end
                case {3}
                    SimPOut(1,iprop+1) = 1-
or(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(1,iprop+1) = 1-
or(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
                        SimROut(1,iprop+1) =
or(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end
                case {4}
                    SimPOut(1,iprop+1) =
or(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(1,iprop+1) =
or(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
```

```
                    else
                        SimROut(iprop+1) = 1-
or(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end
                case {7}
                    SimPOut(1,iprop+1) = 1-
xor(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(1,iprop+1) = 1-
xor(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
                        SimROut(1,iprop+1) =
xor(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end

                case {8}
                    SimPOut(1,iprop+1) =
xor(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(1,iprop+1) =
xor(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
                        SimROut(iprop+1) = 1-
xor(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end
            end
        end
        SimPbit = [SimPbit;SimPOut];
        SimRbit = [SimRbit;SimROut];
    else
        SimPOut(1) = G(1);
        SimROut(1) = G(1);
        if ~isempty(SimPbit)
            n1Exi = ismember(G(2),SimPbit(:,1));
        else
            n1Exi = 0;
        end
        if n1Exi
            SimPvec = SimPbit(SimPbit(:,1)==G(2),:);
            SimRvec = SimRbit(SimPbit(:,1)==G(2),:);
        else
            SimPvec(1) = G(2);
            SimRvec(1) = G(2);
            for ilength = 1:bitLength
                ranP = rand(1);
                ranR = rand(1);
                if ranP < P(G(2),1)
                    SimPvec(ilength+1) = 0;
                    if ranR < R(G(2),1)
                        SimRvec(ilength+1) = 0;
                    else
                        SimPvec(ilength+1) = 1;
                    end
                else
                    SimPvec(ilength+1) = 1;
                    if ranR < R(G(2),2)
                        SimRvec(ilength+1) = 1;
                    else
                        SimPvec(ilength+1) = 0;
                    end
                end
            end
        end
        for iprop = 1:bitLength
            switch G(end)
                case {5}
                    SimPOut(iprop+1) = 1-
SimPvec(1,iprop+1);
                    if rand(1)<rg
                        SimROut(iprop+1) = 1-
SimPvec(1,iprop+1);
                    else
                        SimROut(iprop+1) =
SimPvec(1,iprop+1);
                    end
                case {6}
                    SimPOut(iprop+1) = SimPvec(1,iprop+1);
                    if rand(1)<rg
                        SimROut(iprop+1) =
SimPvec(1,iprop+1);
                    else
                        SimROut(iprop+1) = 1-
SimPvec(1,iprop+1);
                    end
            end
        end
        SimPbit = [SimPbit;SimPOut];
        SimRbit = [SimRbit;SimROut];
        if ~n1Exi
            SimPbit = [SimPbit;SimPvec];
            SimRbit = [SimRbit;SimRvec];
        end
```

```
    end


end
nP = [n1,n2];
[jP,jC] = bitCount(n1,n2,SimPbit,SimRbit);
end




function [nP,jP,jC] =
pS5(n1,n2,P,R,Gates,PM,C,Input,invPair,bitStreamMatStar,bit
StreamMat,rg,bitLength)
SimPbit = [];
SimRbit = [];
path = findpathN2(n1,n2,Gates,Input, invPair);
for iBit = 1:size(path,2)
    G = Gates(Gates(:,1)==path(iBit),:);
    %% input pair bitstream generation
    if G(3)~=0
        pVec =
PM(sum(ismember(PM(:,1:2),[G(2),G(3)]),2)==2,:);
        CJM  =
C{sum(ismember(PM(:,1:2),[G(2),G(3)]),2)==2};
        if G(2) == pVec(1)
            pVec = pVec(3:6);
        else
            pVec = pVec([3,5,4,6]);
            CJM = CXY([1,3,2,4],[1,3,2,4]);
        end
        if ~isempty(SimPbit)
            n1Exi = ismember(G(2),SimPbit(:,1)); % check if
bitstream existed or not
            n2Exi = ismember(G(3),SimPbit(:,1));
        else
            n1Exi = 0;
            n2Exi = 0;
        end
        if n1Exi
            SimP2 = SimPbit(SimPbit(:,1)==G(2),:);
            SimR2 = SimPbit(SimRbit(:,1)==G(2),:);
            [SimPvec,SimRvec] =
bitGen1(G(3),SimP2,SimR2,CJM([1,3,2,4],[1,3,2,4]),bitLength
);
            SimPbit = [SimPbit;SimPvec(1,:)];
            SimRbit = [SimRbit;SimRvec(1,:)];
        elseif n2Exi
            SimP2 = SimPbit(SimPbit(:,1)==G(3),:);
            SimR2 = SimPbit(SimRbit(:,1)==G(3),:);
            [SimPvec,SimRvec] =
bitGen1(G(2),SimP2,SimR2,CJM,bitLength);
            SimPbit = [SimPbit;SimPvec(1,:)];
            SimRbit = [SimRbit;SimRvec(1,:)];
        else
            [SimPvec,SimRvec] =
bitGen2(G(2),G(3),pVec,CJM,bitLength);
            SimPbit = [SimPbit;SimPvec];
            SimRbit = [SimRbit;SimRvec];
        end
        %% propagate to output for current gate on path
        SimPOut(1) = G(1);
        SimROut(1) = G(1);
        for iprop = 1:bitLength
            switch G(end)
                case {1}
                    SimPOut(iprop+1) = 1-
and(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(iprop+1) = 1-
and(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
                        SimROut(iprop+1) =
and(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end
                case {2}
                    SimPOut(iprop+1) =
and(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(iprop+1) =
and(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
                        SimROut(iprop+1) = 1-
and(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end
                case {3}
                    SimPOut(iprop+1) = 1-
or(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(iprop+1) = 1-
or(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
```

```
                        SimROut(iprop+1) =
or(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end
                case {4}
                    SimPOut(iprop+1) =
or(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(iprop+1) =
or(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
                        SimROut(iprop+1) = 1-
or(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end
                case {7}
                    SimPOut(iprop+1) = 1-
xor(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(iprop+1) = 1-
xor(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
                        SimROut(iprop+1) =
xor(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end

                case {8}
                    SimPOut(iprop+1) =
xor(SimPvec(1,iprop+1), SimPvec(2,iprop+1));
                    if rand(1)<rg
                        SimROut(iprop+1) =
xor(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    else
                        SimROut(iprop+1) = 1-
xor(SimRvec(1,iprop+1), SimRvec(2,iprop+1));
                    end
            end
        end
        SimPbit = [SimPbit;SimPOut];
        SimRbit = [SimRbit;SimROut];
    else
        SimPOut(1) = G(1);
        SimROut(1) = G(1);
        if ~isempty(SimPbit)
            n1Exi = ismember(G(2),SimPbit(:,1));
        else
            n1Exi = 0;
        end
        if n1Exi
            SimPvec = SimPbit(SimPbit(:,1)==G(2),:);
            SimRvec = SimRbit(SimPbit(:,1)==G(2),:);
        else
            SimPvec(1) = G(2);
            SimRvec(1) = G(2);
            for ilength = 1:bitLength
                ranP = rand(1);
                ranR = rand(1);
                if ranP < P(G(2),1)
                    SimPvec(ilength+1) = 0;
                    if ranR < R(G(2),1)
                        SimRvec(ilength+1) = 0;
                    else
                        SimPvec(ilength+1) = 1;
                    end
                else
                    SimPvec(ilength+1) = 1;
                    if ranR < R(G(2),2)
                        SimRvec(ilength+1) = 1;
                    else
                        SimPvec(ilength+1) = 0;
                    end
                end
            end
        end
        for iprop = 1:bitLength
            switch G(end)
                case {5}
                    SimPOut(iprop+1) = 1-
SimPvec(1,iprop+1);
                    if rand(1)<rg
                        SimROut(iprop+1) = 1-
SimPvec(1,iprop+1);
                    else
                        SimROut(iprop+1) =
SimPvec(1,iprop+1);
                    end
                case {6}
                    SimPOut(iprop+1) = SimPvec(1,iprop+1);
                    if rand(1)<rg
                        SimROut(iprop+1) =
SimPvec(1,iprop+1);
                    else
                        SimROut(iprop+1) = 1-
SimPvec(1,iprop+1);
```

```
                    end
            end
        end
        SimPbit = [SimPbit;SimPOut];
        SimRbit = [SimRbit;SimROut];
        if ~n1Exi
            SimPbit = [SimPbit;SimPvec];
            SimRbit = [SimRbit;SimRvec];
        end
    end
end
nP = [n1,n2];
[jP,jC] = bitCount(n1,n2,SimPbit,SimRbit);
end



function [nP,jP,jC] = pS7(n1,n2,P,R)
nP = [n1,n2];
jP =
[P(n1,1)*P(n2,1),P(n1,1)*P(n2,2),P(n1,2)*P(n2,1),P(n1,2)*P(
n2,2)];
jC = [R(n1,1)*R(n2,1),R(n1,1)*(1-R(n2,1)),(1-
R(n1,1))*R(n2,1),(1-R(n1,1))*(1-R(n2,1));...
    R(n1,1)*(1-R(n2,2)),R(n1,1)*R(n2,2),(1-R(n1,1))*(1-
R(n2,2)),(1-R(n1,1))*R(n2,2);...
    (1-R(n1,2))*R(n2,1),(1-R(n1,2))*(1-
R(n2,1)),R(n1,2)*R(n2,1),R(n1,2)*(1-R(n2,1));...
    (1-R(n1,2))*(1-R(n2,2)),(1-R(n1,2))*R(n2,2),R(n1,2)*(1-
R(n2,2)),R(n1,2)*R(n2,2)];
end
```

**The following codes are used for gate propagation, from (A, B) to D**

```
 % M, First row is signal pair, second row is P_j, third
row is R_j
% AND

pIJ = PM(ipn,3:6);
P(gateVec(1),:)   = [1-pIJ(4),pIJ(4)];
CJ = C{ipn};
R(gateVec(1),:) =
[sum(pIJ(1:3)*CJ(1:3,1:3))/P(gateVec(1),1),sum(pIJ(4)*CJ(4,
4))/P(gateVec(1),2)];
R(gateVec(1),:) = R(gateVec(1),:).*(2*rg-1)+[1-rg,1-rg];


% M, First row is signal pair, second row is P_j, third row
is R_j
% NAND
pIJ = PM(ipn,3:6);
P(gateVec(1),:)   = [pIJ(4),1-pIJ(4)];
CJ = C{ipn};
R(gateVec(1),:) =
[sum(pIJ(4)*CJ(4,4))/P(gateVec(1),1),sum(pIJ(1:3)*CJ(1:3,1:
3))/P(gateVec(1),2)];
R(gateVec(1),:) = R(gateVec(1),:).*(2*rg-1)+[1-rg,1-rg];

% M, First row is signal pair, second row is P_j, third row
is R_j
% NOR
pIJ = PM(ipn,3:6);
P(gateVec(1),:)   = [1-pIJ(1),pIJ(1)];
CJ = C{ipn};
R(gateVec(1),:) =
[sum(pIJ(2:4)*CJ(2:4,2:4))/P(gateVec(1),1),sum(pIJ(1)*CJ(1,
1))/P(gateVec(1),2)];
R(gateVec(1),:) = R(gateVec(1),:).*(2*rg-1)+[1-rg,1-rg];

% M, First row is signal pair, second row is P_j, third row
is R_j
% NOT
pInputJoint = P(gateVec(2),:);
rInputJoint = R(gateVec(2),:);
P(gateVec(1),:)   = [pInputJoint(2),pInputJoint(1)];
R(gateVec(1),:)   = [rInputJoint(2),rInputJoint(1)]*(2*rg-
1)+[1-rg,1-rg];
```

```
% M, First row is signal pair, second row is P_j, third row
is R_j
% OR

pIJ = PM(ipn,3:6);
P(gateVec(1),:)   = [pIJ(1),1-pIJ(1)];
CJ = C{ipn};
R(gateVec(1),:) =
[sum(pIJ(1)*CJ(1,1))/P(gateVec(1),1),sum(pIJ(2:4)*CJ(2:4,2:
4))/P(gateVec(1),2)];
R(gateVec(1),:) = R(gateVec(1),:).*(2*rg-1)+[1-rg,1-rg];

% M, First row is signal pair, second row is P_j, third row
is R_j
% XNOR
pInputJoint = PM(ipn,3:6);
CJ = C{ipn};
P(gateVec(1),:)=
[pInputJoint(2)+pInputJoint(3),pInputJoint(1)+pInputJoint(4
)];


% M, First row is signal pair, second row is P_j, third row
is R_j
% XOR
pIJ = PM(ipn,3:6);
P(gateVec(1),:)= [pIJ(1)+pIJ(4),pIJ(2)+pIJ(3)];
CJ = C{ipn};
R(gateVec(1),:) =
[sum(pIJ([1,4])*CJ([1,4],[1,4]))/P(gateVec(1),1),sum(pIJ([2
,3])*CJ([2,3],[2,3]))/P(gateVec(1),2)];
R(gateVec(1),:) = R(gateVec(1),:).*(2*rg-1)+[1-rg,1-rg];

% M, First row is signal pair, second row is P_j, third row
is R_j
% BUFF
pInputJoint = P(gateVec(2),:);
rInputJoint = R(gateVec(2),:);
P(gateVec(1),:)   = [pInputJoint(1),pInputJoint(2)];
R(gateVec(1),:)   = [rInputJoint(1),rInputJoint(2)]*(2*rg-
1)+[1-rg,1-rg];
```

The following codes are used to investigate convergence process for sequential circuit, using traditional convergence or TSC

```
clear
clc;
close all;
load('Circuit\s27.mat');

%% Basic Parameters
Num_node   = max(max(Gates));
Num_gate   = size(Gates,1);
Num_input  = size(Input,1);
Num_output = size(Output,1);
Ind        = Gates(:,1:3);
MC         = 2*10^4;
rg         = 0.9;
[R0, R1]   = deal(zeros(Num_node,1),zeros(Num_node,1));
Pin        = 0.5*ones(1,Num_input);
r0_in       = ones(1,Num_input);
r1_in       = ones(1,Num_input);
ita = 0.4;
itarec(1,1) = ita;
i1 = 0; i0=0;


Gates(Gates(:,5)==7,5)=8;
CC = 1;
exitflag = 0;
exitP = 0;
exitR = 0;

%% Counters Initialization

Nodevec            = zeros(Num_node,1);
NodevecStar        = zeros(Num_node,1);

%% MC simulation
%% P* Convergence

clear   R1rec R0rec Prec R1recIn R0recIn  PrecIn;
CC = 1;
exitflag = 0;
[R0, R1]   = deal(zeros(Num_node,1),zeros(Num_node,1));
```

```
Pin        = 0.5*ones(1,Num_input);
r0_in       = ones(1,Num_input);
r1_in       = ones(1,Num_input);
tic;
while (CC==1 || CC < 100) && (exitP == 0)
    Pstarc     = zeros(Num_node,1);
    for i=1:MC
        NodevecStar        = zeros(Num_node,1);
        for k = 1:Num_input
            NodevecStar(Input(k)) = (rand(1)<Pin(k));
        end
        % Gate type: 1-NAND, 2-AND, 3-NOR, 4-OR, 5-NOT, 6-
BUFF, 7-XNOR, 8-XOR
        for j = 1:Num_gate
            x=Gates(j,:);
            switch x(5)
                case {1}
                    NodevecStar(x(1)) = 1-
and(NodevecStar(x(2)), NodevecStar(x(3)));
                case {2}
                    NodevecStar(x(1)) =
and(NodevecStar(x(2)), NodevecStar(x(3)));
                case {3}
                    NodevecStar(x(1)) = 1-
or(NodevecStar(x(2)), NodevecStar(x(3)));
                case {4}
                    NodevecStar(x(1)) =
or(NodevecStar(x(2)), NodevecStar(x(3)));
                case {5}
                    NodevecStar(x(1)) =
not(NodevecStar(x(2)));
                case {6}
                    NodevecStar(x(1)) = NodevecStar(x(2));
                case {7}
                    NodevecStar(x(1)) = 1-
xor(NodevecStar(x(2)), NodevecStar(x(3)));
                case {8}
                    NodevecStar(x(1)) =
xor(NodevecStar(x(2)), NodevecStar(x(3)));
            end
            % Calculate Node Probability
            if NodevecStar(x(1)) == 1
                Pstarc(x(1)) = Pstarc(x(1)) + 1;
            end
        end
    end
    PS = Pstarc./MC;
    PS(Input) = Pin;
    for iDFF = 1:size(indDFF,1)
        xD = indDFF(iDFF,:);
        xIn = find(Input==xD(1));
        Pdif(iDFF,CC+1) = PS(xD(2))-PS(xD(1));
        Pin(xIn) = Pin(xIn)-ita*(Pin(xIn)-PS(xD(2)));
        Prec(iDFF,CC)  = PS(xD(2));
        PrecIn(iDFF,CC)  = Pin(xIn);
    end
    CC = CC+1
    if max(abs(Pdif(:,end)),[],'all')<0.003
        numPConv = CC-1;
        exitP = 1;
    end
end
Prec(Prec>0.99) = 1;
Prec(Prec<0.01) = 0;
PrecIn(Prec>0.99) = 1;
PrecIn(Prec<0.01) = 0;
TPstar = toc;
save pStarConv.mat



%% R convergence
clear;
clc;
load pStarConv.mat
for reg_length = 3
    PinDFF = Prec(:,end);
    Pin    = 0.5*ones(1,Num_input);
    Pin(1,end-size(PinDFF,1)+1:end)  = PinDFF;
    TCC = 0;
    clear   R1rec R0rec R1recIn R0recIn R0 R1;
    CC = 1;
    exitflag = 0;
    [R0, R1]   = deal(zeros(Num_node,1),zeros(Num_node,1));
    r0_in       = ones(1,Num_input);
    r1_in       = ones(1,Num_input);
    tic;
    %         while (CC==1 || CC <50) && (exitP == 0 ||
exitR == 0)
    while (CC==1 || CC < 100) && (exitR == 0)
        [R0, R1]   =
deal(zeros(Num_node,1),zeros(Num_node,1));
```

```
        Pstarc     = zeros(Num_node,1);
        for i=1:MC
            NodevecStar        = zeros(Num_node,1);
            % Input Setup regarding to Pin, Node
Reliability Setup
            for k = 1:Num_input
                NodevecStar(Input(k)) = (rand(1)<Pin(k));
                if (NodevecStar(Input(k))==1 &&
rand(1)<r1_in(k))||...
                        (NodevecStar(Input(k))==0 &&
rand(1)<r0_in(k))
                    Nodevec(Input(k)) =
NodevecStar(Input(k));
                else
                    Nodevec(Input(k)) = 1-
NodevecStar(Input(k));
                end
            end
            for j = 1:Num_gate
                x=Gates(j,:);
                % simulate error-free value and real value
                % Gate type: 1-NAND, 2-AND, 3-NOR, 4-OR, 5-
NOT, 6-BUFF, 7-XNOR, 8-XOR
                switch x(5)
                    case {1}
                        NodevecStar(x(1)) = 1-
and(NodevecStar(x(2)), NodevecStar(x(3)));
                        if rand(1)<rg
                            Nodevec(x(1)) = 1-
and(Nodevec(x(2)), Nodevec(x(3)));
                        else
                            Nodevec(x(1)) =
and(Nodevec(x(2)), Nodevec(x(3)));
                        end
                    case {2}
                        NodevecStar(x(1)) =
and(NodevecStar(x(2)), NodevecStar(x(3)));
                        if rand(1)<rg
                            Nodevec(x(1)) =
and(Nodevec(x(2)), Nodevec(x(3)));
                        else
                            Nodevec(x(1)) = 1-
and(Nodevec(x(2)), Nodevec(x(3)));
                        end
                    case {3}
```

```
                        NodevecStar(x(1)) = 1-
or(NodevecStar(x(2)), NodevecStar(x(3)));
                        if rand(1)<rg
                            Nodevec(x(1)) = 1-
or(Nodevec(x(2)), Nodevec(x(3)));
                        else
                            Nodevec(x(1)) =
or(Nodevec(x(2)), Nodevec(x(3)));
                        end
                    case {4}
                        NodevecStar(x(1)) =
or(NodevecStar(x(2)), NodevecStar(x(3)));
                        if rand(1)<rg
                            Nodevec(x(1)) =
or(Nodevec(x(2)), Nodevec(x(3)));
                        else
                            Nodevec(x(1)) = 1-
or(Nodevec(x(2)), Nodevec(x(3)));
                        end
                    case {5}
                        NodevecStar(x(1)) =
not(NodevecStar(x(2)));
                        if rand(1)<rg
                            Nodevec(x(1)) =
not(Nodevec(x(2)));
                        else
                            Nodevec(x(1)) = 1-
not(Nodevec(x(2)));
                        end

                    case {6}
                        NodevecStar(x(1)) =
NodevecStar(x(2));
                        if rand(1)<rg
                            Nodevec(x(1)) = Nodevec(x(2));
                        else
                            Nodevec(x(1)) = 1-
Nodevec(x(2));
                        end

                    case {7}
                        NodevecStar(x(1)) = 1-
xor(NodevecStar(x(2)), NodevecStar(x(3)));
                        if rand(1)<rg
                            Nodevec(x(1)) = 1-
xor(Nodevec(x(2)), Nodevec(x(3)));
```

```
                        else
                            Nodevec(x(1)) =
xor(Nodevec(x(2)), Nodevec(x(3)));
                        end

                    case {8}
                        NodevecStar(x(1)) =
xor(NodevecStar(x(2)), NodevecStar(x(3)));
                        if rand(1)<rg
                            Nodevec(x(1)) =
xor(Nodevec(x(2)), Nodevec(x(3)));
                        else
                            Nodevec(x(1)) = 1-
xor(Nodevec(x(2)), Nodevec(x(3)));
                        end

                end
                % Calculate Node Probability and
reliability
                if NodevecStar(x(1)) == 1
                    if x(1)==30
                        i1 = i1+1;
                    end
                    Pstarc(x(1)) = Pstarc(x(1)) + 1;
                    if NodevecStar(x(1)) == Nodevec(x(1))
                        R1(x(1)) = R1(x(1))+1;
                    end
                else
                    if x(1)==30
                        i0 = i0+1;
                    end
                    if NodevecStar(x(1)) == Nodevec(x(1))
                        R0(x(1)) = R0(x(1))+1;
                    end
                end

            end
        end
        R1 = R1./Pstarc;
        R0 = R0./(MC-Pstarc);

        R1(isnan(R1)) = 0;
        R0(isnan(R0)) = 0;

        R1(Input) = r1_in';
        R0(Input) = r0_in';
        for iDFF = 1:size(indDFF,1)
            xD = indDFF(iDFF,:);
            xIn = find(Input==xD(1));
            R0dif(iDFF,CC) = R0(xD(2))-R0(xD(1));
            R1dif(iDFF,CC) = R1(xD(2))-R1(xD(1));

            r0_in(xIn) = r0_in(xIn)-ita*(r0_in(xIn)-
R0(xD(2)));
            r1_in(xIn) = r1_in(xIn)-ita*(r1_in(xIn)-
R1(xD(2)));

            R1rec(iDFF,CC) = R1(xD(2));
            R0rec(iDFF,CC) = R0(xD(2));

            R1recIn(iDFF,CC) = r1_in(xIn);
            R0recIn(iDFF,CC) = r0_in(xIn);

            if Pin(xIn) <= 10^-2
                r1_in(xIn) = 0;
            elseif Pin(xIn) >= 1-10^-2
                r0_in(xIn) = 0;
            end
        end
        CC = CC+1
        if
mean(abs([R0dif(:,end),R1dif(:,end)]),'all')<0.003
            numRConv = CC-1;
            exitR = 1;
        end
        % regression
%         if CC == reg_length+1
%             for iDFF = 1:size(indDFF,1)
%                 X = [1:reg_length];
%                 %                      xD =
indDFF(iDFF,:);
%                 xD = indDFF(iDFF,:);
%                 xIn = find(Input==xD(1));
%
%                 mR1O = fitlm(X,R1rec(iDFF,(CC-
reg_length):(CC-1))); mR1In = fitlm(X,R1recIn(iDFF,(CC-
reg_length):(CC-1)));
%                 coR1O(:,iDFF) =
mR1O.Coefficients.Estimate([2,1]);
%                 coR1In(:,iDFF) =
mR1In.Coefficients.Estimate([2,1]);
```

```
%                 r1_in(xIn) =
(coR1O(1,iDFF)*coR1In(2,iDFF)-
coR1O(2,iDFF)*coR1In(1,iDFF))/(coR1O(1,iDFF)-
coR1In(1,iDFF));
%
%                 mR0O = fitlm(X,R0rec(iDFF,(CC-
reg_length):(CC-1))); mR0In = fitlm(X,R0recIn(iDFF,(CC-
reg_length):(CC-1)));
%                 coR0O(:,iDFF) =
mR0O.Coefficients.Estimate([2,1]);
%                 coR0In(:,iDFF) =
mR0In.Coefficients.Estimate([2,1]);
%                 r0_in(xIn) =
(coR0O(1,iDFF)*coR0In(2,iDFF)-
coR0O(2,iDFF)*coR0In(1,iDFF))/(coR0O(1,iDFF)-
coR0In(1,iDFF));
%
%             end
%
%             if r0_in(xIn)<0
%                 r0_in(xIn) = 0;
%             end
%             if r0_in(xIn)>1
%                 r0_in(xIn) = 1;
%             end
%
%             if r1_in(xIn)<0
%                 r1_in(xIn) = 0;
%             end
%             if r1_in(xIn)>1
%                 r1_in(xIn) = 1;
%             end
%
%             if Pin(xIn) == 0
%                 r1_in(xIn) = 0;
%             elseif Pin(xIn) == 1
%                 r0_in(xIn) = 0;
%             end
%
%
%             ita = 0.1;
%
%         end
    end
    TCC = toc;
    %     T(reg_length-2)=mean(TCC);
end
% T = T/10;

figure(1)
color = ['r','b','k'];
linespec = ['>','h','p'];
hold
for i = 1:size(indDFF,1)
    plot(R1rec(i,:),['--
',linespec(i),color(i)],'LineWidth',2)
    plot(R1recIn(i,:),['-
',linespec(i),color(i)],'LineWidth',2)
end
set(gca,'FontSize',40)
legend('DI 1','DO 1','DI 2','DO 2','DI 3','DO
3','NumColumns',2,'Orientation','horizontal')
xlabel('Iterations')
ylabel('R1')
legend('DO 1','DI 1','DO 2','DI 2','DO 3','DI
3','NumColumns',2,'Orientation','horizontal')
xlabel('Iterations','FontName','Times New
Roman','fontweight','bold')
ylabel('R1','FontName','Times New
Roman','fontweight','bold')

figure(2)
color = ['r','b','k'];
linespec = ['>','h','p'];
hold
for i = 1:size(indDFF,1)
    plot(R0rec(i,:),['--
',linespec(i),color(i)],'LineWidth',2)
    plot(R0recIn(i,:),['-
',linespec(i),color(i)],'LineWidth',2)
end
set(gca,'FontSize',40)
legend('DI 1','DO 1','DI 2','DO 2','DI 3','DO
3','NumColumns',2,'Orientation','horizontal')
xlabel('Iterations')
ylabel('R0')

% set(gca,'FontSize',30)
% title('R0')
% figure(3)
% hold
% for i = 1:size(indDFF,1)
%     plot(R1rec(i,:),'-o')
% %     plot(R1recIn(i,:),'-o')
% end
% title('R1')
% set(gca,'FontSize',30)
% %
%
legend('DO 1','DI 1','DO 2','DI 2','DO 3','DI
3','NumColumns',2,'Orientation','horizontal')
xlabel('Iterations','FontName','Times New
Roman','fontweight','bold')
ylabel('R0','FontName','Times New
Roman','fontweight','bold')
title('')
set(gca, 'fontsize', 40)
set(gca, 'linewidth',4)
xlim([1,7])
ylim([0.7,1])
xticklabels([1:11])
```

**The following codes are used to verify that level-5 correlation could be treated as independent**

```
%% Level 5 Independency Assumption MC simulation
clear;
clc;
close all;

Num_node = 63;
Num_gate = 31;
Num_input = 32;
NUm_output = 1;
Input     = [1:32]';
Output    = 63;
MC = 1000000;
Pin = 0.5;
rin = 1;
rg = 0.95;

%% Gate matrix generation
Gates = [];
for i = 1:16
    Gates(i,2:3) = [2*i-1,2*i];
end
Gates(13:16,2:3) = [15,13;11,9;7,5;3,1;];
Gates(9:12,2:3)  = [2,4;6,8;10,12;14,16];
Gates(:,1) = [33:48]';

Gates(17:24,1) = [49:56]';
Gates(17:24,2) = 33:2:47';
Gates(17:24,3) = 34:2:48';

Gates(25:28,1) = [57:60]';
Gates(25:28,2) = 49:2:55';
Gates(25:28,3) = 50:2:56';

Gates(29:30,1) = [61;62];
Gates(29:30,2) = [57;59];
Gates(29:30,3) = [58;60];

Gates(31,:)     = [63,61,62];
Gates(:,4)      = rg;

% Gate type: 1-NAND, 2-AND, 3-NOR, 4-OR, 5-NOT, 6-BUFF, 7-
XNOR, 8-XOR
% randomly generate gates
% GateType       = [1,2,3,4,7,8];
% Gatein         = randi(6,1,31);
%generate gates as L5-NOR, L4-NAND, L3-XOR, L2-AND, L1-OR
Gates(1:16,5)  = 3*ones(16,1);
Gates(17:24,5) = ones(8,1);
Gates(25:28,5) = 8*ones(4,1);
Gates(29:30,5) = 2*ones(2,1);
Gates(31,5)    = 4;
% GateTypeStr    = {'NAND';'AND';'NOR';'OR';'XNOR';'XOR'};
% for m = 1:30
%     GateTypeRecord{1,m} = GateTypeStr(Gatein(m),:);
% end
NodevecStar        = zeros(Num_node,1);
Nodevec            = zeros(Num_node,1);

Pstarc             = zeros(Num_node,1);
rallc              = zeros(Num_node,2);
%% MC simulation
for i=1:MC
    NodevecStar        = zeros(Num_node,1);
    %% Input Setup regarding to Pin, Node Reliability Setup
    for k = 1:Num_input
        NodevecStar(Input(k)) = (rand(1)<Pin);
%         if rand(1)<rin(k)
            Nodevec(Input(k)) = NodevecStar(Input(k));
%         else
%             Nodevec(Input(k)) = 1-NodevecStar(Input(k));
%         end
    end
    for j = 1:Num_gate
        x=Gates(j,:);
        %% simulate error-free value and real value
        %% Gate type: 1-NAND, 2-AND, 3-NOR, 4-OR, 5-NOT, 6-
BUFF, 7-XNOR, 8-XOR
        switch x(5)
            case {1}
                NodevecStar(x(1)) = 1-
and(NodevecStar(x(2)), NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = 1-and(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = and(Nodevec(x(2)),
Nodevec(x(3)));
                end
            case {2}
```

```
                NodevecStar(x(1)) = and(NodevecStar(x(2)),
NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = and(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = 1-and(Nodevec(x(2)),
Nodevec(x(3)));
                end
            case {3}
                NodevecStar(x(1)) = 1-or(NodevecStar(x(2)),
NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = 1-or(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = or(Nodevec(x(2)),
Nodevec(x(3)));
                end
            case {4}
                NodevecStar(x(1)) = or(NodevecStar(x(2)),
NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = or(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = 1-or(Nodevec(x(2)),
Nodevec(x(3)));
                end

            case {5}
                NodevecStar(x(1)) = not(NodevecStar(x(2)));
                if rand(1)<rg
                    Nodevec(x(1)) = not(Nodevec(x(2)));
                else
                    Nodevec(x(1)) = 1-not(Nodevec(x(2)));
                end

            case {6}
                NodevecStar(x(1)) = NodevecStar(x(2));
                if rand(1)<rg
                    Nodevec(x(1)) = Nodevec(x(2));
                else
                    Nodevec(x(1)) = 1-Nodevec(x(2));
                end
            case {7}
                NodevecStar(x(1)) = 1-
xor(NodevecStar(x(2)), NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = 1-xor(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = xor(Nodevec(x(2)),
Nodevec(x(3)));
                end

            case {8}
                NodevecStar(x(1)) = xor(NodevecStar(x(2)),
NodevecStar(x(3)));
                if rand(1)<rg
                    Nodevec(x(1)) = xor(Nodevec(x(2)),
Nodevec(x(3)));
                else
                    Nodevec(x(1)) = 1-xor(Nodevec(x(2)),
Nodevec(x(3)));
                end

        end
        %% Calculate Node Probability
        bitStreamMatStar(:,i)= NodevecStar;
        bitStreamMat(:,i)    = Nodevec;
        if NodevecStar(x(1)) == 1
            Pstarc(x(1)) = Pstarc(x(1)) + 1;
            if Nodevec(x(1)) == 1
                rallc(x(1),2) = rallc(x(1),2)+1;
            end
        else
            if Nodevec(x(1)) == 0
                rallc(x(1),1) = rallc(x(1),1)+1;
            end
        end

    end
    i
end

G_ind = [Gates(:,1:3),Gates(:,5)];
r = rallc./[MC-Pstarc,Pstarc];
P = [MC-Pstarc,Pstarc]/MC;

%% A = 61, B = 62, D = 63
```

```
A = 61; B = 62; D = 63;
[jP,jC] = bSMP(61,62,bitStreamMatStar,bitStreamMat,MC);

PA      = P(A,:);
PB      = P(B,:);
jPIN    = PA'*PB;
jPIN    = jPIN'

RA      = r(A,:);
RB      = r(B,:);
RAM     = [RA(1),RA(1),1-RA(1),1-RA(1);...
           RA(1),RA(1),1-RA(1),1-RA(1);...
           1-RA(2),1-RA(2),RA(2),RA(2);...
           1-RA(2),1-RA(2),RA(2),RA(2);]
RBM     = [RB(1),1-RB(1),RB(1),1-RB(1);...
           1-RB(2),RB(2),1-RB(2),RB(2);...
           RB(1),1-RB(1),RB(1),1-RB(1);...
           1-RB(2),RB(2),1-RB(2),RB(2);]
jCIN    = RAM.*RBM;
jPE     = jPIN(:)-jP(:)
jCE     = jCIN-jC

P(63,:)-[0.9735,0.0265];
r(63,:)-[0.7891,0.6246];
rDMC = P(63,1)*r(63,1)+P(63,2)*r(63,2)
rDIN = R(4,1)*PC(4,1)+R(4,2)*PC(4,2)
rDMC-rDIN
rDINMC = R(5,1)*PC(5,1)+R(5,2)*PC(5,2);
```

**The following codes are used to obtain JCPM, JPV and signal probabilities from bitstreams**

```
function [jP,jC] =
bSMP(node1,node2,bitStreamMatStar,bitStreamMat,bitLength)
bitLength = size(bitStreamMatStar,2);
MC = bitLength;
bSMS = bitStreamMatStar;
bsm1S = bSMS(node1,:);
bsm2S = bSMS(node2,:);
bR = [bitStreamMat(node1,:);bitStreamMat(node2,:)];
bsmSCounter = zeros(1,4);
bRcounter = zeros(4,4);
%% one signal
for ibsm = 1:MC
    if [bsm1S(ibsm),bsm2S(ibsm)] == [0,0]
        bsmSCounter(1,1) = bsmSCounter(1,1)+1;
        if bR(:,ibsm) == [0;0]
            bRcounter(1,1) = bRcounter(1,1)+1;
        elseif bR(:,ibsm) == [0;1]
            bRcounter(1,2) = bRcounter(1,2)+1;
        elseif bR(:,ibsm) == [1;0]
            bRcounter(1,3) = bRcounter(1,3)+1;
        elseif bR(:,ibsm) == [1;1]
            bRcounter(1,4) = bRcounter(1,4)+1;
        end
    end
    if [bsm1S(ibsm),bsm2S(ibsm)] == [0,1]
        bsmSCounter(1,2) = bsmSCounter(1,2)+1;
        if bR(:,ibsm) == [0;0]
            bRcounter(2,1) = bRcounter(2,1)+1;
        elseif bR(:,ibsm) == [0;1]
            bRcounter(2,2) = bRcounter(2,2)+1;
        elseif bR(:,ibsm) == [1;0]
            bRcounter(2,3) = bRcounter(2,3)+1;
        elseif bR(:,ibsm) == [1;1]
            bRcounter(2,4) = bRcounter(2,4)+1;
        end
    end
    if [bsm1S(ibsm),bsm2S(ibsm)] == [1,0]
        bsmSCounter(1,3) = bsmSCounter(1,3)+1;
        if bR(:,ibsm) == [0;0]
            bRcounter(3,1) = bRcounter(3,1)+1;
        elseif bR(:,ibsm) == [0;1]
            bRcounter(3,2) = bRcounter(3,2)+1;
        elseif bR(:,ibsm) == [1;0]
```

```
            bRcounter(3,3) = bRcounter(3,3)+1;
        elseif bR(:,ibsm) == [1;1]
            bRcounter(3,4) = bRcounter(3,4)+1;
        end
    end
    if [bsm1S(ibsm),bsm2S(ibsm)] == [1,1]
        bsmSCounter(1,4) = bsmSCounter(1,4)+1;
        if bR(:,ibsm) == [0;0]
            bRcounter(4,1) = bRcounter(4,1)+1;
        elseif bR(:,ibsm) == [0;1]
            bRcounter(4,2) = bRcounter(4,2)+1;
        elseif bR(:,ibsm) == [1;0]
            bRcounter(4,3) = bRcounter(4,3)+1;
        elseif bR(:,ibsm) == [1;1]
            bRcounter(4,4) = bRcounter(4,4)+1;
        end
    end

end
jC = bRcounter./bsmSCounter';
jP = bsmSCounter/MC;
```

**The following codes are used to find aging effects on circuit delay**

```
 clear
clc

load C17.mat
load bitStreamC17.mat

% initialization
ain = 50;
aj  = 0.59;
t   = 2.6*10^6*12;
P = (sum(bitStreamMat')/size(bitStreamMat,2))';
PStar = (sum(bitStreamMatStar')/size(bitStreamMat,2))';
D(Input,:) = 0;

for i = 1:Num_gate
    x = Gates(i,:);
    Dpin = min(P(x(2)), P(x(3)));
    tpi = ain+aj*((1-Dpin)*t).^0.16;
    D(x(1),:) = max(D(x(2),:),D(x(3),:)) + tpi;
end
```

**The following codes are used to find transistor probability of failure regarding**

```
 %%for 35nm tech
delta = 30.28*10^-3;
VDD    = 1;
for i = 1:99
VthN   = 0+i*0.01;
x = 0:0.001:1;
y = 0.5*erfc(abs(x-VthN)/(delta*sqrt(2)));
% plot(y)
% set(gca, 'YSCALE', 'log');
Pin_N = 0.9;
pfN0  = y(floor(VthN/0.002));
pfN1  = y(floor((1+VthN)/0.002));
pfN(i)   = Pin_N * pfN1 + (1-Pin_N)*pfN0;
% display(pfN)
end
plot(pfN)
xlabel('Vth')
```

```
ylabel('Probability of Failure')
xticklabels(0:0.1:0.9)
set(gca, 'FontSize', 40)
set(gca, 'YSCALE', 'log');
title('NMOS probability of Fialure with Vth variation')
```

**The following codes are used to calculate the probability of failure for NAND gate, and compare to MC simulation results**

```
clear;
clc;
%% NAND gate PF calculation
% Initialization
for ith = 1:41
    VthNi = 0.6;
    VthPi = -0.6;
    deltaP = 30.28*10^-3;
    deltaN = 30.28*10^-3;
    VDD  = 1;
    MC = 10^4;
    for i = 1:MC
        % 0 and 1 PF for NMOS and PMOS
        % nmHN  = (1+VthNi)/2; nmLN  = VthNi/2; % noise
margin High/Low for NMOS
        % nmHP  = 1+VthPi/2;   nmLP  = (1+VthPi)/2; % noise
margin High/Low for PMOS
        mu = [(1+VthPi)/2,1-(1-VthNi)/2;...
            (1+VthPi)/2,1-(1-VthNi)/2];

        sigma = [(1+VthPi)/12,(1-VthNi)/12;...
            (1+VthPi)/12,(1-VthNi)/12];
%         sigma = [0,0;...
%             0,0];

        vin =
[normrnd(mu(1,1),sigma(1,1)),normrnd(mu(1,2),sigma(1,2));..
. %[A=0, A=1;]

normrnd(mu(2,1),sigma(2,1)),normrnd(mu(2,2),sigma(2,2))];
%[B=0, B=1;]
        vin(vin>1) = 1;
        vin(vin<0) = 0;

        VthN = 0.6+(ith-1)*0.005;
        VthP = -0.6-(ith-1)*0.005;

        pf   = zeros(4,2); % T1, T2, T3, T4 Pf0 and Pf1
        for ipf = 1:2
            pf(1,ipf) = 0.5*erfc(abs(vin(2,ipf)-
(VDD+VthP))/(deltaP*sqrt(2)));
            pf(2,ipf) = 0.5*erfc(abs(vin(1,ipf)-
(VDD+VthP))/(deltaP*sqrt(2)));
            pf(3,ipf) = 0.5*erfc(abs(vin(1,ipf)-
VthN)/(deltaP*sqrt(2)));
            pf(4,ipf) = 0.5*erfc(abs(vin(2,ipf)-
VthN)/(deltaP*sqrt(2)));
        end
        % pfN0 = 0.5*erfc(abs(nmLN-VthN)/(deltaN*sqrt(2)));
        % pfN1 = 0.5*erfc(abs(nmHN-VthN)/(deltaN*sqrt(2)));
        % pfP0 = 0.5*erfc(abs(nmLP-
(VDD+VthP))/(deltaP*sqrt(2)));
        % pfP1 = 0.5*erfc(abs(nmHP-
(VDD+VthP))/(deltaP*sqrt(2)));

        % MC values
        % pfN0 = 0.5*erfc(abs(0-VthN)/(deltaN*sqrt(2)));
        % pfN1 = 0.5*erfc(abs(1-VthN)/(deltaN*sqrt(2)));
        % pfP0 = 0.5*erfc(abs(0-
(VDD+VthP))/(deltaP*sqrt(2)));
        % pfP1 = 0.5*erfc(abs(1-
(VDD+VthP))/(deltaP*sqrt(2)));


        % PF of NAND gate for different patterns
        %     ipat(i) = ceil(rand(1)/0.25);
        %     switch ipat(i)
        %         case 1
        %             PF(i,1) =
pf(1,1)*pf(2,1)+pf(3,1)*pf(4,1)-...
        %                 pf(1,1)*pf(2,1)*pf(3,1)*pf(4,1);
        %         case 2
        %             PF(i,1) = pf(1,2)*(1-
pf(2,1))+pf(3,1)*(1-pf(4,2))-...
        %                 pf(1,2)*(1-pf(2,1))*pf(3,1)*(1-
pf(4,2));
        %         case 3
        %             PF(i,1) = pf(2,2)*(1-
pf(1,1))+pf(4,1)*(1-pf(3,2))-...
        %                 pf(2,2)*(1-pf(1,1))*pf(4,1)*(1-
pf(3,2));
```

```
        %         case 4
        %             PF(i,1) = 1-(1-pf(1,2))*(1-
pf(2,2))*(1-pf(3,2))*(1-pf(4,2));
        %     end

        PF(i,1) = pf(1,1)*pf(2,1)+pf(3,1)*pf(4,1)-...
            pf(1,1)*pf(2,1)*pf(3,1)*pf(4,1);
        PF(i,2) = pf(1,2)*(1-pf(2,1))+pf(3,1)*(1-pf(4,2))-
...
            pf(1,2)*(1-pf(2,1))*pf(3,1)*(1-pf(4,2));
        PF(i,3) = pf(2,2)*(1-pf(1,1))+pf(4,1)*(1-pf(3,2))-
...
            pf(2,2)*(1-pf(1,1))*pf(4,1)*(1-pf(3,2));
        PF(i,4) = 1-(1-pf(1,2))*(1-pf(2,2))*(1-pf(3,2))*(1-
pf(4,2));


    end
    PFG(ith) = mean(PF,'all');
    PFM(ith,:) = mean(PF);
    ith
end
save NAND_MC.mat PFG PFM

plot([0.6:0.005:0.8], PF,'-^','LineWidth',4);
hold;
plot([0.6:0.005:0.8], PFG,'-p','LineWidth',4);
xlim([0.6 0.8]);
set(gca,'Fontsize',40);
xlabel('Vth Variation')
ylabel('Probability of Failure')
title('NAND Gate Probability of Failure against different
Vths' )
legend('Model Calculation', 'MC simulation')
```

# VITA AUCTORIS

| | |
|---|---|
| NAME: | SUOYUE ZHAN |
| PLACE OF BIRTH: | HUBEI, CHINA |
| YEAR OF BIRTH: | 1993 |
| EDUCATION: | University of Electronic Science and Technology of China, B.Sc., Chengdu, SICHUAN, China, 2015<br>University of Windsor, M.Sc., Windsor, ON, 2018<br>University of Windsor, Ph.D., Windsor, ON, 2022 |