CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

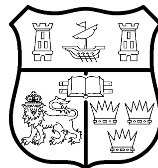
Title	Electronic design automation methodologies for digital VLSI circuit reliability analysis and optimisation
Authors	Yang, Bo
Publication date	2020-08-17
Original Citation	Yang, B. 2020. Electronic design automation methodologies for digital VLSI circuit reliability analysis and optimisation. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2020, Bo Yang. - https://creativecommons.org/licenses/by-nc-nd/4.0/
Download date	2025-04-24 20:26:55
Item downloaded from	https://hdl.handle.net/10468/10536

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Electronic Design Automation Methodologies for Digital VLSI Circuit Reliability Analysis and Optimisation

Bo Yang

A thesis submitted for the degree of
Doctor of Philosophy



UNIVERSITY COLLEGE CORK, IRELAND

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

August 2020

Head of School: Dr. Jorge Oliveira

Supervisor: Dr. Emanuel M. Popovici

Contents

List of Figures . . . iv
List of Tables . . . vi
List of Publications . . . vii
Acknowledgements . . . x
Abstract . . . xii
Preliminary . . . xiv

1 Introduction 1
1.1 *CMOS* Design Flow and Electronic Design Automation . . . 3
1.2 New Challenges . . . 5
1.3 Approach . . . 10
1.4 The Organization of the Contents . . . 15

2 Background 16
2.1 Graph and AND-Inverter Graph . . . 17
2.2 Conditional Probability and Theorem of Total Probability . . . 21
2.3 Probability Density Function . . . 22
2.4 Monte-Carlo Simulation . . . 26
2.5 Pearson Correlation Coefficient . . . 28
2.6 Artificial Neural Network and Backpropagation . . . 29
2.7 Chapter Summary . . . 35

3 Analytical Soft Error Propagation, Reliability Analysis, and Optimisation for *VLSI* Combinational Circuits 36
3.1 *CPEP*: Conditional Probabilistic Error/Reliability Propagation Analysis . . . 38
3.1.1 Basic Concepts . . . 38
3.1.2 The Independent Probabilistic Gate Model . . . 39
3.1.3 Re-convergent Fanouts . . . 43
3.2 Reliability Evaluator Implementation . . . 46
3.2.1 Consideration of Multiple Outputs . . . 46
3.2.2 Data Structure . . . 47
3.2.3 Computation Algorithm . . . 48
3.3 Test Environment Setup and Comparison Results . . . 49
3.4 Synthesis for Reliability Enhancement based on Cut-rewriting Technique . . . 52
3.4.1 Reliability Optimisation: the Algorithm Implementation 54

3.4.2 Experimental Results 56
3.4.2.1 *cm162a* – A Case Study 56
3.4.2.2 Evaluation of Benchmark Circuits 58
3.5 Chapter Summary 62

4 Voltage Scalable *SSTA* Gate Modelling Techniques 63
4.1 Process Variations and their Impacts on Timing Analysis 64
4.2 *SPICE* Monte-Carlo based *SSTA* 67
4.3 An Analytical Analysis of V_{dd} Dependent τ Estimation 69
4.3.1 Drain to Source Current Model 71
4.3.2 Consideration of Threshold Voltage Variation 73
4.3.3 Propagation Delay Estimation 74
4.3.4 *PDF* Generation Model for *CMOS* Gates 80
4.4 *ANN-SSTA*: *ANN-based SSTA* Gate Model 82
4.4.1 Circuit Model for *ANN* Training Dataset Generation . . . 84
4.4.2 Modelling Strategy 1 86
4.4.3 Modelling Strategy 2 101
4.5 Chapter Summary 110

5 Synthesizing an *MLP-based* Combinational Logic Circuit 112
5.1 *ANN VLSI* Implementations 115
5.1.1 Capacitive Network 116
5.1.2 Charge Trapping Devices 118
5.1.3 Memristors 120
5.2 *MLP* Universal Boolean Function Approximator 123
5.2.1 Boolean Function Approximation using *MLP* 123
5.2.2 Learning Algorithm for Boolean Logic Function 126
5.3 Model Transfer Considerations 128
5.3.1 Regularization and Pruning 129
5.3.2 Parameter Quantization 132
5.3.3 Parameter Variations 133
5.4 Chapter Summary 135

6 Circuit Linearization with Bi-decomposition 138
6.1 Boolean Function Linearity Basic Concepts 140
6.2 Bi-Decomposition and Vectorial Bi-Decomposition 141
6.3 Linear Decomposition of Adders 145
6.4 Chapter Summary 147

7 Conclusions and Future Work **149**

A Derivation of Back-propagation Algorithm **A1**

B Levenberg-Marquardt Algorithm **B1**

C *HSPICE* variation analyses **C1**

D SPICE Simulation Code Snippets **D1**

List of Figures

1.1 The *NMOS* symbol and its construction structure 2
1.2 *CMOS* design pentagon 6

2.1 Directed graph and Directed acyclic graph 19
2.2 A sample circuit and its *AIG* representation 20
2.3 Theorem of total probability Venn diagram 22
2.4 *GD* curves with $\mu = 2$ and different σ 25
2.5 *IGD* curves with $\mu = 2$ and different λ 26
2.6 Estimating the area enclosed by curve A 27
2.7 Scatter plot of τ_{pd} against C_{load} demonstrating a linear correlation 28
2.8 Biological neuron simplified structure 29
2.9 Artificial neuron model 30
2.10 Typical multilayer perceptron neural network 31

3.1 Erroneous gate model 38
3.2 *RFO* structures 44
3.3 Cut example and locally equivalent circuit 53
3.4 A case study: *MCNC* benchmark *cm162a* 57

4.1 The variation source classifications 65
4.2 *MC-based SSTA* basic workflow 68
4.3 The falling transition of an *Inverter* 70
4.4 Operating regions of *NMOS* in a falling transition 71
4.5 V_{th} vs. V_{ds} linear fitting 72
4.6 I_{ds} vs. V_{gs} curve fitting @$V_{ds} = 0.9, 0.5, 0.2$ V 73
4.7 I_{ds} vs. V_{gs} with $+/-10$ % V_{th0} variation 74
4.8 *Inverter* circuit topology 75
4.9 Delay vs. V_{th0} @$V_{dd} = 0.5, 0.7, 0.9$ V, respectively 78
4.9 Delay vs. V_{th0} @$V_{dd} = 0.5, 0.7, 0.9$ V, respectively 79
4.10 Comparison of *PDF* curves with *SPICE MC* simulation 80
4.10 Comparison of *PDF* curves with *SPICE MC* simulation 81
4.11 The *ANN* τ_{pd} estimation model framework flowchart 82
4.12 The *SPICE* simulation circuit model for gate or cell characterisation 85
4.13 Performance measurement demonstration 88
4.14 Heatmap of Pearson correlation coefficients between performance and parameter variances 90

4.15 Error histogram of the trained *NAND* model 92
4.15 Error histogram of the trained *NAND* model 93
4.16 Regression diagram of 10^4-sample test of the trained *NAND* model 93
4.16 Regression diagram of 10^4-sample test of the trained *NAND* model 94
4.17 Regression diagram of 10^4-sample Gaussian variation prediction $@V_{dd} = 0.65$ V, $C_{load} = 5$ fF, $C_{pa} = 5$ fF and $C_{pb} = 5$ fF 96
4.18 A Majority Voter circuit composed of basic *CMOS* gates 99
4.19 Regression diagram and *PDF* plots of a Majority Voter $@V_{dd} = 0.5V$ 100
4.20 Function of μ and λ against V_{dd} 101
4.20 Function of μ and λ against V_{dd} 102
4.21 The regression diagrams for $S2$ *ANN* prediction models 104
4.22 Regression diagrams of the switch event estimation trained model 105
4.23 Single *NAND* test, two different setups, 500 bits uniformly distributed random input vectors were simulated for each setup . . 106
4.24 The simulated *NAND-chain* circuit 107
4.25 Model prediction vs. *MC* @ $V_{dd} = 0.45$ V, simulated on 500 randomly generated input vectors, *SPICE* runtime: > 1 h, Model runtime: ~ 1 s 108
4.26 The results of the *NAND-chain* simulation 109

5.1 The *EDA* flow for synthesizing an *MLP-based* logic gate 114
5.2 Capacitive network *ANN* architecture [122] 117
5.3 Charge trapping device *ANN* architecture [123] 119
5.4 Memristive network *ANN* architecture [124] 120
5.5 The diagram graph of *AND3* and *XOR3* 125
5.6 The variation analyses 135

6.1 Reliability enhancement system framework architecture 139
6.2 Linear separation on *Parity* function 142
6.3 Strong and Weak Bi-decompositions 143
6.4 Vectorial XOR-Bi-decomposition on *Carry* function 146

A.1 Compact *ANN* model for *BP* demonstration A1

List of Tables

3.1 Truth table of unreliable *AND* gate 41
3.2 Truth table of the error injection *XOR* gate 42
3.3 Accuracy and performance evaluation for different gate errors (ϵ) 51
3.4 *RWREL* performance evaluation on different benchmark circuits (gate error $\epsilon = 0.001$) 60
3.5 Area, delay and power analyses 61

4.1 Delay estimation results at different voltages with the nominal V_{th0} 78
4.2 Training dataset generation *SPICE MC* simulation setup and parameters 87
4.3 *HSPICE* generated *.mc0* file contents 89
4.4 *ANN* architecture and training setup 91
4.5 Training results of *NAND ANN* model 92
4.6 Training results of some commonly used *CMOS* gates or cells . . 95
4.7 Relative estimation errors (%) of the predicted P_τ for *NAND* gate 98
4.8 Summary of the capacitances for various cells 99
4.9 *ANN* architecture and training setup 103
4.10 *ANN* training setup 105
4.11 *NAND* chain circuit setup 107

5.1 Comparison between the architectures 122
5.2 Basic logic gates and their representations 124
5.3 Trained *XOR* models with 2, 3, 4, 6, 8 and 10 inputs under 2- and/or 3- layer configurations 126
5.4 *XOR3* layer outputs 126
5.5 Training algorithms comparison on benchmark circuits 127
5.6 Training results using the *BR* and Pruning and the estimated number of components 131
5.7 Training *ANN* with reduced precision 133

6.1 Global decomposition experimental results 147

List of Publications

This thesis is based on the following list of publications during my Ph.D. work:

- Bo Yang, Satish Grandhi, Christian Spagnol, Emanuel Popovici, Sorin Cotofana, "An approach for digital circuit error/reliability propagation analysis based on conditional probability," 27th IEEE Irish Signals and Systems Conference (ISSC), 2016, pp. 1-6
- Bo Yang, E. Popovici, M. A. Quille, A. Amann and S. Cotofana, "A supply voltage-dependent variation aware reliability evaluation model," 2016 IEEE/ACM International Symposium on Nanoscale Architectures, Beijing, 2016, pp. 79-84
- Satish Grandhi, Bo Yang, Christian Spagnol, Emanuel Popovici, "An EDA framework for reliability estimation and optimization of combinational circuits," Journal of Low Power Electronics 12(3), 2016, pp. 242-258
- B. Steinbach, Emanuel Popovici, Daniel Rodas Bautista, Bo Yang, "Synthesis techniques for reliability using Bi-decompositions," Book chapter in "Further Improvements in the Boolean Domain," Cambridge Scholars Publishing, 2019, pp. 321-335
- Bo Yang, Sorin Cotofana, Emanuel Popovici, "Synthesizing a multilayer perceptron based combinational logic circuit," submitted to Integration, the VLSI Journal, 2020
- Bo Yang, Sorin Cotofana, Emanuel Popovici, "A novel ANN-based dependent statistical static timing analysis combinational gate model," submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020

I, Bo Yang, certify that this thesis is my own work, and it has not been submitted for another degree at University College Cork or elsewhere.

Except for Chapter 3, which in part is a collaboration with my colleague, Dr. Satish Kumar Grandhi (who is a major contributor for Section 3.4), the remaining sections are primarily my contribution.

Bo Yang

To my beloved mother and friends.

To this generous and beautiful planet.

Acknowledgements

Science is looking carefully at the patterns of nature and trying to understand their origin, that's what Copernicus, Kepler, Newton, Einstein, Galileo, Curie, Dirac, Feynman, Von Neumann, and a thousand others have discovered. There are regularities in nature, and there are patterns as far as the eye can see on every scale from atoms to galaxies. And those patterns are a reflection of the simplicity and beauty of the underlying laws of nature. As a Ph.D. researcher, I have been one of the seekers of those beauties that are glorious for my whole life.

First of all, I want to say "thank you" to my supervisor, Dr. Emanuel Popovici, a man who is filled with ideas, inspirations, and passions. To me, he is a supervisor, but more than that, a friend. He gave me not only the supervision and guidance during my Ph.D., but also supports on my life during my stay these years in Ireland.

I want to express my thanks to the *EU FP7 i-RISC* project consortium and the collaborators who provided many valuable help and suggestions to me. I am particularly grateful to Prof. Sorin Cotofana, who is a research collaborator, helped me a lot with my publications. And I won't forget to mention the wonderful time when I was working with my dear colleagues, Dr. Satish Grandhi, Dr. Christian Spagnol, and Dr. Nicoleta Cucu-Laurenciu. Their advice and help make my Ph.D. work smoothly started. Together with them, we did many nice pieces of work.

I would take this opportunity to say "thank you" to all my families, especially my mother, who is always standing behind me and supporting me no matter what happens. My gratitude is also given to Dr. Han Shao, thanks for your friendship during the journey in these years.

My special thanks will be given to all my friends in Ireland, Dr. Bruno Srbinovski, Dr. Juan Pablo and his wife Mia, Dr. Fiona Edwards Murphy, Mark O'Sullivan,

Antonietta Saggese, my landlord Mr. Richard Courtney, Mr. He Zhu, Dr. Xing Ouyang, Dr. Xi Cao, Mr. Yuan Hu, Dr. Xing Dai and his wife Dr. Guannan Wei, Mr. Nicu Vlasin, Dr. Yineng Wang and his wife Ms. Lin Lin, Mr. Yubo Wang, Mr. Xin Feng, Dr. Siming Zheng, Mr. Chenjun Xu, Miss Naxuan Hao, Ms. Shuoshuo Zhou, Mr. Bin Zeng, Dr. Zili Li and his wife Ms. Lu Zhou, Prof. Guangbo Hao, Mr. Su Liu and Miss Yichen Wang; Also those will be given to all my old friends in China, Mr. Yingming Pang, Mr. Yueming Zhang, Mr. Bowen Jiang, Mr. Haitao Mu, Ms. Lin Wang, Mr. Xiaolong Wang, Mr. Jiuhai Yuan, Mr. Tianyang Sheng, Mr. Zhuoran Zhou, Mr. Bo Li, Mr. Feng Shen and his wife Ms. Ping Xu, Ms. Mengru Huang, Ms. Ran Zhu, Mr. Xuanchi Liu, Mr. Yi Lu, Miss Haoyi Zhang, Ms. Ting Chen and Mr. Shen Yu.

I will never forget about my previous teachers and supervisors, Ms. Yin Zhang, Ms. Wenpei Zhuang, Dr. Liusong Yang, Prof. Wei Zhao, Prof. Jiaxin Liu, Mr. Yonghui Zhao and Mr. Ming Li.

At the moment when I was writing this thesis, the human world was being swept by a world-wide disaster – a deadly disease *Covid-19* (2019 Novel Coronavirus Disease). I witnessed the weakness of humans in the face of nature, but I was also touched by the courage and tenacity of the Homo sapiens. Be thankful to the scientists, be thankful to the doctors, and be thankful to all people who are saving lives and fighting against the virus.

Abstract

As the continuous scaling on both size and the operating voltage of the *Very Large Scale Integrated* (VLSI) circuits for improved power and area efficiencies, keeping the acceptable reliability has become an increasingly significant challenge of the digital circuits in addition to the power consumption and area.

In this thesis, a number of modern *Electronic Design Automation* (EDA) algorithms and approaches emphasizing the digital circuit reliability analysis and optimisation were investigated. In this work, the reliability is categorized into two aspects, i.e., the reliability related to the soft error events in the circuit and the timing related reliability. In terms of the soft error events, an error may be injected into the circuit unexpectedly, which may eventually impact the correctness of the computation executed by the circuit. What makes the analysis complex is that when an error occurred somewhere in the circuit, the error may or may not propagate through the circuit and be reflected on the output due to the *masking phenomena*. *Monte-Carlo* (MC) simulation was the standard method to solve the problem in the traditional workflow until the circuit scale became sufficiently large that it is out of the tractable computation performance. Thus, analytical approaches for soft error propagation algorithms were under research for decades. In this work, a conditional probability based soft error propagation algorithm, *CPEP*, that can achieve significant performance boost compared to the *MC* simulation, while maintaining high accuracy was developed. In addition, building on the foundation of the *CPEP* reliability analysis algorithm, a complete *EDA* framework to enhance the reliability during the circuit synthesis was proposed.

The timing related reliability is a measure of the probabilities that the circuit outputs can properly switch to the desired voltage level at a certain delay (i.e., *Cut-off delay*). The analysis of this kind of reliability is a process of *Statistical Static Timing Analysis* (SSTA), which has become a very active research area in the last decades. In our work, the *Artificial Neural Network* (ANN) function approximator based *SSTA* gate models for accurate and fast estimation of the

propagation delay, τ_{pd}, distribution (hence the reliability) was explored. All these methods for reliability analysis are fully compatible with existing electronics design flows using classical Boolean circuit synthesis methods.

In the quest to design reliable, efficient circuits, alternative circuit architectures were investigated, which may not follow the classical *CMOS* design flow. *ANN*s are such architectures that promise more efficient implementations of digital circuits and can be a viable alternative to the current *CMOS-based VLSI* circuit architectures. A synthesis framework for *Multilayer Perceptron* (MLP) based Boolean logic gates was investigated. Experimental results show that with this architecture, the logic circuit can be implemented effectively, resulting in smaller propagation delay and also the more predictable variations. A typical Boolean circuit consists of many types of gates. Implementing the linear circuits (i.e., *XOR* networks) is particularly difficult for an *MLP* Boolean function approximator. Thus, a circuit linearization framework based on Boolean function Bi-decomposition was proposed to separate the linear part from the non-linear part (composed of the rest of gate types). Then, an *MLP-based* circuit can be used to implement the non-linear part more effectively while the linear part can be part of an error control coding scheme, which can further improve circuit reliability.

Preliminary

Abbreviates

AIG	And-Inverter Graph
ANN	Artificial Neural Network
BFS	Breadth-first Search
BP	Backpropagation
BR	Bayesian Regularization
CDF	Cumulative Density Function
CP	Conditional Probability
CPEP	Conditional Probability-based Error Propagation
DAG	Directed Acyclic Graph
DFS	Depth-first Search
DG	Directed Graph
EP	Error Probability
GD	Gaussian Distribution
IGD	Inverse Gaussian Distribution
MAC	Multiply-Accumulate
MC	Monte-Carlo
MLP	Multilayer Perceptron
MOO	Multi-objective Optimisation
NPN	Negation-Permutation-Negation
PCC	Pearson Correlation Coefficient
PDF	Probability Density Function
PI	Primary Input
PO	Primary Output
RFO	Reconvergent Fanout
RTL	Register Transfer Level

RV	Random Variable
SER	Soft Error Rate
SP	Static Probability
SSTA	Statistical Static Timing Analysis
TTP	Theorem of Total Probability

Symbols

μ	*GD* and *IGD* parameter, i.e., mean
λ	*IGD* parameter, shape adjustment
σ^2	*GD* parameter, variance
σ	Standard deviation
τ_{pd}	Propagation delay
τ_{co}	Cut-off delay
P_τ	Propagation delay distribution (PDF)
MP_τ	Mixture distribution of the propagation delay
$P_\tau\|_{01\rightarrow11}$	P_τ of transition $01 \rightarrow 11$
$w_{j,i}^{(l)}$	*ANN* weight from neuron i to neuron j of layer l
$b_i^{(l)}$	*ANN* bias of neuron i of layer l
$\boldsymbol{W}^{(l)}$	*ANN* weight matrix of layer l
$\boldsymbol{b}^{(l)}$	*ANN* bias vector of layer l

Chapter 1

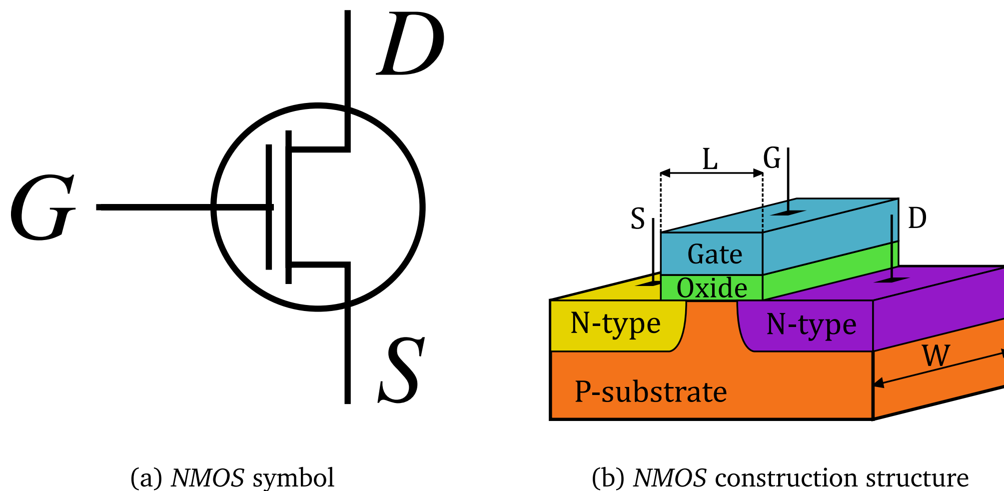
Introduction

Integrated Circuits (ICs) form the basis for the microelectronic industries and play a vital role in our lives and contribute significantly to the global economy [1]. Since the invention of the *transistor* in the 1950s, we are on a continuous quest to optimise, add new features and use transistors and *IC*s in new applications, which improved the quality of life of many. Transistors become smaller every year, driven by Moore's Law predictions and laid down by the *International Semiconductor Roadmap* [2, 3, 4], integration is at unprecedented levels, and the number of applications touches every aspect of our life.

At present, the dominant technology is *Metal-Oxide-Semiconductor Field-Effect Transistor* (MOSFET)/*Complementary MOS* (CMOS). The transistor size in commercial applications is 7 nm (while 5 nm is picking up), while more than 1.2 trillion transistors are integrated on the same chip [5]. Not surprisingly, this chip is aimed at *Artificial Intelligence* (AI) applications where researchers try to mimic the function and the number of neurons in the human brain. This tremendous growth in the number of transistors and integration was driven by (a) the developments on the solid-state semiconductor devices physics; (b) the progress of the semiconductor manufacturing process and packaging technologies; (c) the development of the new computing system architectures as well as memory; (d) and the improvements of the circuit design methodologies through *(micro)electronic design automation* (EDA) tools. The last two drivers

are also the focus of this thesis, in which tools for reliability analysis and optimisation of digital integrated circuits are being developed. New *EDA* design methodologies accompany and enable continuous scaling of the digital circuits.

Firstly, the basic structure of the *MOSFET* will be explained. Figure 1.1a and Figure 1.1b depict the symbol and basic structure of an *N-type MOSFET* (NMOS), respectively.



(a) *NMOS* symbol (b) *NMOS* construction structure

Figure 1.1: The *NMOS* symbol and its construction structure

As shown in the figure, the traditional *MOSFET* has a planar structure, the *channel length* (L), and *channel width* (W) are the two metrics of most importance in a *CMOS* circuit design. Also, the scaling of *MOSFET* is referred to as the reduction of the channel length L. With the continuous scaling down of the transistors (i.e., the *MOSFET*s), more *MOSFET*s can be integrated into a single die, which results in the following advantages:

- *Lower Cost:* The number of transistors per unit area, and the manufacturable die area are both getting larger, which leads to much lower manufacturing cost;
- *Higher Performance:* As the transistor is getting smaller (e.g., the channel length), higher channel current while lower junction and parasitic capacitances can be achieved, which leads to higher speed;

- *Lower Power Consumption:* Lower dynamic power consumption primarily due to the reduction in the operating voltage (smaller gate-oxide thickness), and lower static power consumption due to the reduction in the leakage current (new gate materials).

All these advantages of *MOSFET* translate in enabling a myriad of applications from battery operated to high performance digital integrated circuits.

1.1 *CMOS* Design Flow and Electronic Design Automation

As the scale of the *CMOS* digital circuits is counted in billions of *MOSFET*s in modern digital *CMOS* technology, the digital *CMOS Very Large Scale Integrated* (VLSI) circuit design is *EDA* driven. With *EDA* tools, a set of algorithms are performed to assist the designers to design, analyse, verify, optimize, and produce highly efficient and reliable *VLSI* circuits. With many decades of development, the *CMOS VLSI* industry has formed a mature design flow. The conventional synchronous digital *VLSI* circuit design flow is often partitioned into three phases, i.e., logic design/logic synthesis, physical design and fabrication, of which the first two stages may take advantage of the methodologies proposed in this work:

Logic design and synthesis (Front-end design) The logic designers focus on the circuit architecture, algorithms, and functionality implementation using the *Hardware Description Languages* (HDL), commonly *Verilog* or *VHDL*. The output of this phase is the circuit *Register Transfer Level* (RTL) description. The circuit functionality is then verified by performing a pre-synthesis simulation. During this phase, the circuit representation has only mathematical meanings, and mostly the functionality of the circuit is verified. The circuit performance, when evaluated at this phase, gives the designer only some preliminary guidance on the performance of the designed circuit. The main advantage of the evalua-

tion, although not accurate, it can be performed very fast, requiring relatively modest computing resources.

Synthesis tools translate the *RTL* to the gate level netlist that describes the circuit with a graph that is composed of the logic gates/standard cells (nodes) and their connections (edges). In the synthesis phase, the circuit representation is annotated with physical meanings, and the result of synthesis directly affects the results of the physical design. Thus, it is significant to have accurate analysis and optimisation performed in this phase in order to detect design flaws and to improve the design performance at this early stage.

Physical design/implementation (Back-end design) This phase is where the synthesized netlist is physically laid-out and routed, and the final mask layout file (typically *GDSII* file as industry-standard format) for manufacturing is generated. The goal of physical design is to properly plan the layout area to achieve minimized routing costs and optimized circuit performance and reliability. At this stage, most of the process technology and implementation information are available. Thus the circuit specifications can be very accurately evaluated. Also, this is the last chance for designers to verify the circuit functionalities and performance metrics before sending the design to the semiconductor device foundry. Compared with the front end design analysis and verification tasks, these tasks are orders of magnitude slower and require significant computational resources.

The key challenges are that front end design performance, reliability, or power estimation to be as close as possible to the back end design estimations or indeed to speed-up the physical design performance estimation drastically without compromising on the accuracy of the model.

1.2 New Challenges

Although many advantages of the *CMOS* technology scaling exist, the rate of scaling is slowing down and approaching the limit (i.e., the end of Moore's law) [6, 7, 8, 9], primarily due to the following reasons:

- The transistor channel is reduced down to a few nano meters. Thus the device physics is showing quantum phenomenon;
- Increasing device variability, due to the uncertainties in the fabrication process;
- The manufacturability is getting harder, thus lowering cost-effectiveness;
- The high device density prevents the heat from dissipating.

These new challenges raise increasing new demands for ultra-low power/low voltage devices, circuits, architectures, and systems. To continue advancing the development of the integrated circuits, one direction under research is the post-silicon materials, devices such as the *Single-Electron Transistors* (SETs) [10, 11], *Memristors* [12], and *Carbon Nanotube FETs* (CNFETs) [13, 14]. Another direction is to improve the *MOSFET* structures and *CMOS* process technologies such as the *FinFet* [15], *Gate-All-Around Nanowires* (GAA NWs) [16], etc. On the other hand, the most viable direction in the coming decades is to develop new circuit architectures and optimisation approaches based on the current *CMOS* technology (which is the main area of study in this thesis).

Due to the new challenges, the *VLSI* system design and optimisation goals for the most advanced *CMOS* technologies are getting even more complex. The so-called *Multi-objective Optimisation* (MOO) [17] is usually required to obtain a highly efficient computation system, as more serious trade-offs are exhibited among the design metrics, normally area, performance, and energy consumption [17]. Furthermore, reliability is becoming another important optimisation goal. In order to minimize the circuit energy consumption, voltage scaling is known to be one of the most effective approaches, due to the quadratic relation-

ship between the dynamic power consumption and the operating voltage V_{dd}. However, the voltage scaling causes the performance as well as reliability to degrade quickly. These trade-offs can be illustrated using the pentagon in Figure 1.2. As the pentagon indicates, lowering V_{dd} will directly reduce power con-

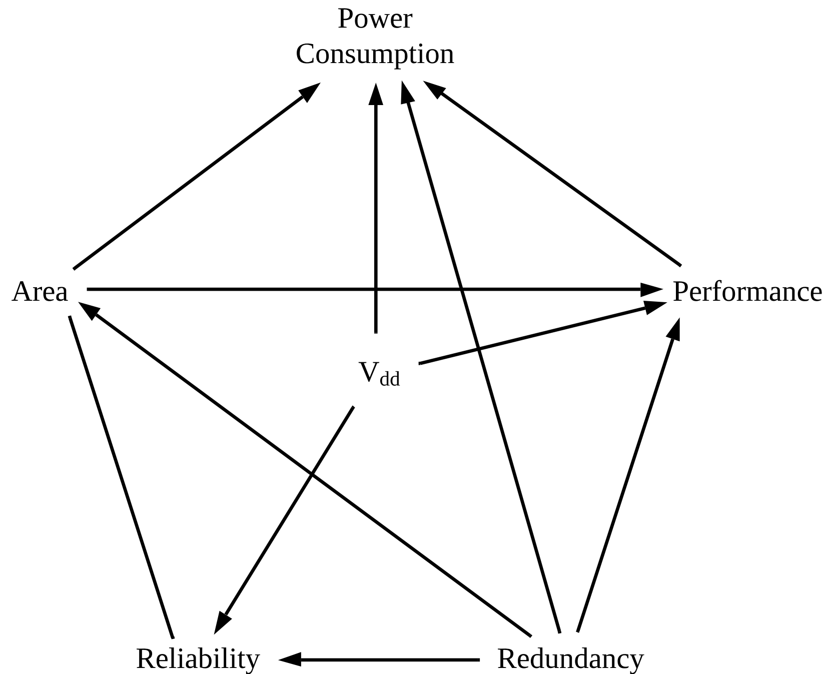


Figure 1.2: *CMOS* design pentagon

sumption, while degrading the performance and reliability. Typically, adding redundant logic to the circuit can protect the reliability from degrading (compared to a canonical logic), but causes the area and power consumption to increase.

Thus, the key point in the voltage scaling optimisation process is to find the optimal V_{dd} to balance the power consumption improvement and performance and reliability degradations. In other words, efficient *EDA* algorithms to analyse, evaluate, and improve the circuit speed, and reliability become a significant requirement in the *CMOS VLSI* design flow. While *EDA* tools for power consumption, area, and performance have been extensively investigated for the past decades, the research on *EDA* tools for reliability is a relatively new concept.

The term *Reliability* is formally defined by *IEEE* as the measures of the ability to generate units to perform their intended function under specified service conditions during a period [18]. In the context of *VLSI* circuits, the reliability is the probability that the correct computation results are obtained at the *Primary Outputs* (POs) of a digital circuit [19] under specified conditions and durations. Therefore, the main target of the study on reliability is the uncertainties and the probabilistic behaviours of the circuits that are composed of the unreliable nano-meter transistors (e.g., the ones operating at low V_{dd}).

The nano-meter transistors operating at lower V_{dd} have the following sources of uncertainties [20, 21]:

- The manufacturing process is even harder to control, which leads to larger device variabilities;
- The lower *over-drive voltage* causes increased *propagation delay*, τ_{pd}, thereby increases the possibility that some transistors cannot switch properly at a certain delay;
- The noise margin gets tighter, which results in a more erroneous circuit.

Thus, the circuit reliability issues are mostly caused by *device variations* due to the manufacturability and *soft errors* that originate from *crosstalk*, *thermal noise*, and *power supply noise*, etc.[20, 21, 22, 23]

Soft errors The soft error is often called the *Transient Fault*, as it is a type of temporary fault that normally has no lasting effects on a certain circuit. A soft error can be generated in both of the memory elements (e.g., registers) and the combinational elements. If a soft error event occurs in a memory element, the error may immediately be latched, which will become an error source of the subsequent level of combinational circuits. However, if that occurs in a combinational element, the error has to propagate through one or some (*Reconvergent Fanout* (RFO) may exist) of the combinational paths to produce a final faulty computation result. Thus, to be "successfully" latched by the subse-

quent memory element, an error must pass through the following three masking mechanisms [24]:

- *Logic Masking:* the errors may be masked by the logic function of a certain logic gate. For example, an error will be masked by a "0" input of an *AND* gate. This is the most common masking mechanism in the combinational circuits.
- *Electrical Masking:* very fast transitioning glitches will be masked by a logic gate with a larger propagation delay. In other words, the error glitches must be wide enough to be "successfully" propagated through a logic gate. This mechanism has more effect on an ultra-low voltage circuit, due to the larger propagation delay.
- *Timing-window Masking:* the error must be captured by a memory element to become a final faulty result, and even to propagate to the next level of the combinational circuit.

As stated above, to predict the error propagation in the combinational circuits can be a challenging task from the *EDA* perspective.

Device variations The device variations, i.e., *Process*, *Voltage*, *Temperature* (PVT) variations, contribute to the uncertainties of the circuit propagation delay. In other words, the propagation delay is very sensitive to the device variabilities. Besides, the deviation in the propagation delay can result in faulty output in a circuit when some of the elements in a path cannot be properly switching at a certain time.

However, to accurately predict the propagation delay variation is a very challenging task, especially for the current advanced nano-meter *CMOS* technologies, as the standard *Static Timing Analysis* (STA) shows limitations and weaknesses to deal with the process variabilities.

Conventionally, to deal with the variation in propagation delay caused by variability, *Corner-based STA* is used to validate the timing properties considering

the process variations. Based on the corner cases, margins are supposed to be large enough to ensure the circuit will run properly under any circumstance. It is obvious that the corner-based approaches make overly pessimistic or optimistic predictions [25] and are difficult to be used for *MOO* purposes. Besides, these approaches completely lose their prediction power when *intra-die* variations are introduced [26], as will be detailed in Section 4.1.

On the other hand, *Statistical Static Timing Analysis* (SSTA) [27, 28, 29] has been attracting more and more attention to deal with this challenging task of profiling the distribution of the propagation delay, P_τ, against the *PVT* variations. One of the biggest differences in *SSTA* from *STA* is that the critical paths are not deterministic in the context of *SSTA* because that in some certain circumstances a shorter path with fewer logic levels that is not likely to be a critical path in *STA* may have bigger τ_{pd} due to the comprehensive impacts of process parameter fluctuations. Also, the propagation of P_τ through the circuit becomes statistical rather than deterministic as in the *STA*. However, many challenges are still under research to make more practically useful *SSTA*, such as more efficient *SSTA* gate models, more precise propagation algorithms, etc.

In addition to the temporary faults introduced above, it is worth to mention that reliability also refers to the physical phenomena of the *CMOS* devices related to permanent failure and *ageing* caused by *Hot Carrier Injection* [30, 31, 32] and *Negative-bias temperature instability* [33, 34], etc. However, while these are also very important topics of research, they are not the aimed subjects of research in our work.

In this thesis, *EDA* methodologies are proposed for digital circuit analysis and optimisation, emphasizing on the reliability, within the framework of the project *EU FP7 FET-Open i-RISC* [35]. The main goal of the *i-RISC* project is to develop methodologies that allow the reliable computing system to be constructed with the unreliable components.

1.3 Approach

As stated above, the reliability of the ultra-low-power and low voltage digital *VLSI* circuits, especially operating in the near-threshold region, is determined mainly by the noise margin and device variations of the circuits. Thus, in our context, reliability is investigated based on the above two aspects. The reduced noise margin will be reflected by a higher probability that the soft errors can be injected into a circuit and finally result in incorrect computation outcomes at the circuit *Primary outputs* (POs). On the other hand, the device variation-related timing reliability refers to the probability that a circuit output can properly switch to the desired voltage level within a certain delay called the *Cut-off delay*, τ_{co}. Methodologies to efficiently analyse and optimize these reliabilities proposed in our approaches will be discussed in detail in this thesis.

Analytical methodology for soft error analyses and reliability-driven synthesis To optimize the soft error-related reliability of a circuit, a fast and accurate gate-level reliability evaluation tool is primarily needed. Thus, a new analytical gate-level soft error propagation algorithm based on conditional probability and the *Theorem of Total Probability* (TTP) called *CPEP* is developed.

Historically, Von Neumann pioneered the art of probabilistic error analysis and defined any system to be reliable only if the probability to provide correct output is greater than a certain threshold [36]. Authors in [37, 38] investigated the soft error impacts on the circuits, [39, 40, 41, 42] focus on building the fault injection models, while more efforts have been invested in the area of gate-level soft error analysis, such as [40, 43, 44, 45, 46, 47, 48, 49]. A gate-level soft error propagation analysis algorithm is also proposed in this thesis, but has better performance for large scale combinational circuits.

The traditional approach to performing reliability analysis begins with elementary *SPICE* simulations, which is not feasible for practically relevant circuits due to its prohibitive computation time and excessive resource requirements. The complexity and accuracy of the simulation-based techniques strongly de-

pend on the scale of circuit and test vector selection [43]. Complementing the simulation methods, several analytical methods such as those using *Probabilistic Transfer Matrices* (PTMs) [44], *Probabilistic Gate Models* (PGMs) [45] and *probabilistic decision diagrams* (PDDs) [46], *Bayesian networks* [47] and observability based algorithm [48] have also been proposed to investigate the circuit behaviour in the presence of faults. However, the *PTM* suffers from massive matrix storage and manipulation overhead that results in its inapplicability to large circuits, and the *Bayesian network* approach is applicable to medium scale circuits but is intractable for large circuits.

The biggest advantage of the proposed probabilistic model when compared to other methodologies is that the algorithm can dramatically improve the speed of solving the soft error propagation problem through the *VLSI* combinational circuit with acceptable accuracy loss while taking correlations such as *re-convergent fanouts* (RFOs) into account. Specifically, the circuit is represented by a data structure similar to the *AND-Inverter Graph* (AIG) [50, 51], which is a graph with "*AND* gate" nodes and optionally "inverted" edges. The graph is traversed in a topological order from a *PO* to *Primary inputs* (PIs). During the traversal, *RFO*s will be resolved by "simplifying" the *AIG* recursively until the circuit is converted into an *AND-Inverter Tree* (*AIT*, i.e., *acyclic AIG*), then the independent error model can be used. Finally, *TTP* will be employed to get final error probability, hence the reliability of a certain output. This model enables fast computation of the soft error propagation problem while keeping the high prediction accuracy compared to the methods in the literature. Especially, the test results on the *MCNC* benchmark circuits show that up to 10^5 performance boosts within $3\ \%$ accuracy loss can be obtained compared to *Monte-Carlo* (MC) simulation-based methods.

Then, based on the efficient *CPEP* evaluation tool, a reliability-driven combinational circuit synthesis approach based on the *cut rewriting* technique was proposed. Reliability-driven logic synthesis is increasingly more critical approach to improve the reliability of the circuit, especially in the recent decade. In

general, to improve the reliability through logic synthesis, redundancy must be added to the circuit to increase the probability to logically mask the soft errors. *Automatic Test Pattern Generation* (ATPG) based rewiring method, yet structurally-different implementations to reduce the *Soft Error Rate* (SER) was implemented in [52]. In [53], the circuit output *SER* is reduced through localized circuit restructuring by taking advantage of don't-care-based re-synthesis and local rewriting. In [54], a technique to improve the circuit robustness to soft errors based on Redundancy Addition and Removal (RAR) by eliminating gates with a large contribution to the overall SER is proposed. Efficient algorithms for synthesizing approximate circuits for simultaneous masking of logical and timing errors were proposed in [55].

The method proposed in this thesis employs the similar concept of area-driven synthesis algorithm used by the synthesis tool *ABC* [56], which iteratively enumerate *cut* of an *AIG*, boolean matching the function of the *cut* then try to replace the current *cut* with a more robust one from the functionally equivalent set of cuts. Our method allows up to $75\ \%$ improvement of the reliability through a test set on the *MCNC* benchmark circuits.

V_{dd} dependent *SSTA* and timing reliability models Timing reliability means the probability that the circuit output can switch properly at a certain delay, τ_{co} (i.e., the cut-off delay). In other words, the timing reliability is profiled by the *Cumulative Distribution Function* (CDF) of the τ_{pd}.

SSTA is one of the latest methodologies to analyse the distribution of the circuit propagation delay, P_{τ}, against the presence of process parameter variabilities.

One example of the V_{th}-based *SSTA* gate model was presented in [57], which approximates the delay *PDF* using the Gaussian distribution function. This is appropriate only when the *CMOS* transistors operate in the strong inversion regime. However, for low supply voltages, the propagation delay profile is highly non-Gaussian [58]. Therefore, various distributions were used to model the propagation delay profile of circuits for weak and moderate inversion

regimes, such as the Log-Normal Distribution [59].

Linear composition methodology [60, 61] was recently proposed to model the propagation delay profile, which has good fitting accuracy. When operating in the weak inversion region, instead of the Gaussian distribution, the propagation delay was observed more likely to follow the *Inverse Gaussian Distribution* (IGD) (i.e., $\tau_{pd} \sim \text{IG}(\mu, \lambda)$), which is a two-parameter continuous probability distribution with support on $(0, +\infty)$. For any generic *CMOS* gate, a pair of constants μ and λ can be achieved by curve fitting the histogram of propagation delay data from *SPICE MC* simulation. Once μ and λ are achieved, the propagation delay *PDF* can be propagated through a circuit by linearly compositing the μ's and λ's of each gate respectively. Although the linear composition method has good accuracy in estimating the P_τ, its independent predictive power is limited as they merely fit the distribution to pre-existing data. In other words, once some of the simulation environment setups (e.g., V_{dd} values etc.) are changed, the whole fitting process has to be repeated to generate new μ and λ constant pairs. The limitation matters when the model is used in the *MOO* design flow where a V_{dd} associated function is needed.

In this thesis, V_{dd} dependent *Artificial Neural Network* (ANN) function approximator based *SSTA* gate models were investigated to help designers to quickly evaluate the τ_{pd} distribution, in order to properly optimize the circuit design. Within this framework, two modelling strategies were proposed, i.e., *ANN* τ_{pd} *estimation with MC simulation-based* methodology and *ANN IGD parameters estimation* based methodology. Both of the methods take the circuit setup (e.g., V_{dd}, Load and driver strength, etc.) and parameter variations (e.g., V_{th} and L) into consideration. Using the proposed *SSTA* gate modelling techniques, over 10^5 performance improvements can be achieved compared to the *SPICE MC* simulations.

***MLP-based* logic gates synthesis** As stated above, the development of *CMOS* circuits is facing many challenges as the *VLSI* technology is challenging Moore's

law. The designers and manufacturers are making a great effort to exploit every percentage of the *CMOS* circuit performance and tweaking the design parameters to optimize and balance among the performance metrics. In addition to optimizing and tweaking the *CMOS-based VLSI* technology, researchers are looking for more efficient new computation paradigms as well as new nanoscale post-silicon devices. While it is a commonly accepted fact that much more effort and time is needed before the maturity of new nanoscale post-silicon devices, new computing architectures are quite promising to replace the current *CMOS-based VLSI* technology in the near future.

In this thesis, an innovative *VLSI* combinatorial logic circuit design architecture inspired by the *Multilayer Perceptron* (MLP) will be described in detail. The current circuit design paradigm that takes advantage of the *MLP* concept is the so-called *Threshold Logic Gate* (TLG) [62, 63, 64, 65], in which the weighted sum of the inputs is compared with a threshold value to determine the output value. However, the *TLG* has limited ability to implement complex logic gates. A synthsis framework for the *MLP-based* combinatorial logic circuit model with *Sigmoid*-like transfer function is proposed in this thesis. While the logic function approximation capability of an *MLP* has been discussed in some research articles [66, 67], there are few articles in the literature discussing the implementation of complex logic functions. Every aspect of the *EDA* considerations for the *MLP-based* logic gates, including the training of the *MLP* and the techniques to enable an efficient model transfer for hardware implementations, are elaborated and discussed.

Some benchmark circuit blocks were tested and analysed with the proposed framework with promising results.

Linear decomposition of combinational circuits It is found that the *MLP* approximating the linear Boolean functions (i.e., Boolean functions contains only *XOR* gates) is not as efficient as the non-linear Boolean functions, thereby a methodology to separate the linear and non-linear parts of a combinational

circuit may help to improve the efficiency of synthesizing the Boolean function with the *MLP-based* architecture. A *Bi-decomposition* [68, 69] based circuit linearization methodology was proposed in our work. The methodology and its application on *Adders* will be explained and analysed in detail in this thesis. In addition, the future integration of the linear decomposition and *MLP-based* logic gates will be discussed in the corresponding later chapter.

1.4 The Organization of the Contents

The contents of this thesis will be organized as follows. Some background mathematical tools and concepts employed in our work will first be introduced in Chapter 2. In Chapter 3, a novel analytical method for soft error propagation and reliability analysis based on conditional probability will be proposed. Additionally, a cut rewriting based synthesis framework for reliability will also be proposed in this chapter. The novel V_{dd} dependent *SSTA* gate models for timing reliability analyses will be described in Chapter 4. The synthesis flow for the *MLP-based* combinational circuit architecture will be introduced in Chapter 5. Then, the linear decomposition of combinational circuits will be briefly introduced in Chapter 6. Finally, the thesis will be concluded in Chapter 7.

Chapter 2

Background

As stated in Chapter 1, the *Electronic Design Automation* (EDA) tools encompass a set of mathematical models and algorithms to assist the designers to produce highly efficient *Very Large Scale Integrated* (VLSI) circuits. Therefore, in this chapter, some important mathematical tools and concepts employed in my Ph.D. work will be briefly reviewed.

A *Graph* is a data structure that is suitable for representing the digital circuit and is extensively used in our error propagation and *Statistical Static Timing Analysis* (SSTA) models. Specifically, the *Directed Acyclic Graph* (DAG) resembles the structure of combinational circuits, while the *Directed Graph* (DG) captures the structure of digital circuits with possible sequential elements, such as *flip-flops*. Thus, in Section 2.1, the basic concept of *Graph* and its traversal algorithm, as well as the *AND-Inverter Graph* (AIG), will be explained.

Section 2.2 describes the *Conditional Probability* (CP) and the *Theorem of Total Probability* (TTP), which are critical tools for the error probability propagation approach developed in Chapter 3, especially when dealing with the circuits taking *Re-convergent Fanouts* (RFOs) into consideration.

The *Probability Density Function* (PDF), *Cumulative Density Function* (CDF), *Gaussian Distribution* (GD), *Inverse Gaussian Distribution* (IGD), and their properties will be presented in Section 2.3. The *PDF* and *CDF* are important concepts

for the *SSTA* and timing reliability, *GD* is one of the most commonly used distribution functions when modelling the process parameter variabilities, and the *IGD* is used in our *SSTA* gate model to fit the propagation delay distribution, especially in the low-voltage (sub- or near-threshold) regime.

Monte-Carlo (MC) simulation, stated in Section 2.4, is the most basic statistical analysis tool and also employed as reference methods for our analysis algorithms in Chapter 3 and Chapter 4.

In Section 2.5, the concept of *Pearson Correlation Coefficients* (the *correlation matrix*) will be introduced. It is employed when analyzing the dependencies between two *Random Variables* (RVs), hence guiding the selection of features for training the *Artificial Neural Network* (ANN) in Chapter 4.

Last but not least, the basic *ANN* or *Multilayer Perceptron* (MLP), which is intensively employed to approximate functions, will be presented in Section 2.6. For example, in Chapter 4, *MLP*s are used to construct novel *SSTA* gate models, and in Chapter 5, the Boolean function approximation capability of the *MLP* is employed to build the *MLP-based* logic gates.

2.1 Graph and AND-Inverter Graph

Graph Theory is one of the most important mathematical concepts, which studies the pairwise relationships between objects. It is widely used in the areas of computation algorithms, linguistics, and biology, etc. The *Graph* is defined as follows:

Definition 2.1 *(**Graph** [70]) A graph G is an ordered triple ($V(G)$, $E(G)$, ψ_G) consisting of a nonempty set $V(G)$ of **vertices**, a set $E(G)$, disjoint from $V(G)$, of **edges**, and an **incidence function** ψ_G that associates with each edge of G an **unordered** pair of (not necessarily distinct) vertices of G. If e is an edge and u and v are vertices such that $\psi_G(e) = uv$, then e is said to **join** u and v; the vertices u and v are called the **ends** of e.*

If, on the other hand, the ψ_G *associates with each edge of* G *an* ***ordered*** *pair of (not necessarily distinct) vertices of* G*, the graph is called* ***Directed Graph*** *(DG).*

Another special case of *DG* is the *Directed Acyclic Graph* (DAG), which is defined next:

Definition 2.2 ***(Directed Acyclic Graph (DAG))*** *In a DG, if there are no directed cycles, such that there is no such a path to start at any vertex* v *and follow a consistently-directed sequence of edges that eventually loops back to* v *again, the Graph is called a Directed Acyclic Graph.*

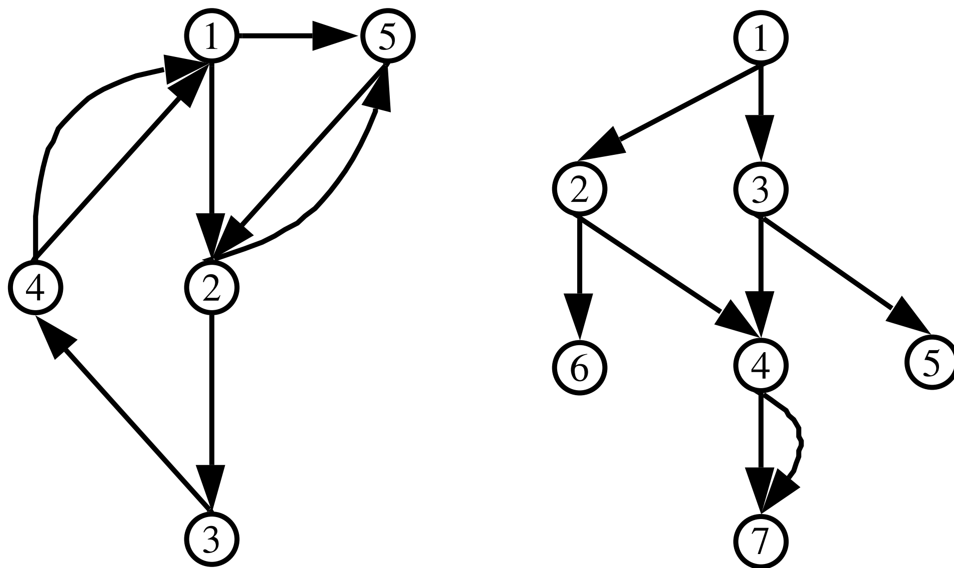
A special case is that if there is exactly only one path from one vertex to each of its subsequent vertices in a *DAG*, then the structure is called a *Tree*.

In Figure 2.1a, an example *DG* is depicted. The circles represent the nodes, and the arrows represent the directed edges. Note that loops (i.e., starting from one node, the same node can eventually be reached along a path) exist in the graph, for example, the paths $\{1 \to 2 \to 3 \to 4 \to 1\}$, $\{2 \to 5 \to 2\}$ and $\{1 \to 5 \to 2 \to 3 \to 4 \to 1\}$, etc. The loops can represent the feedbacks in a digital circuit.

In Figure 2.1b, an example *DAG* is shown. Note that in the graph, no loop exists, as all edges are feedforward throughout the graph from root nodes to leaf nodes, although multiple paths may exist from one node to another.

The traversal of a *DAG* or *Tree* is one of the most basic problems in the computation algorithms. There are commonly two types of traversing strategy, i.e., the *Depth-first Search* (DFS) and *Breadth-first Search* (BFS). The *DFS* visits the child vertices along the depth of a particular path before exploring in its breadth direction and the *BFS* vice versa. According to the order of visiting and searching, each of the two strategies can be subclassed into *Inorder Search* and *Preorder Search*.

Inorder means that the child of the certain vertex will be explored before the vertex is visited (i.e., some operations are performed on the vertex), while *Preorder* means that the certain vertex will be visited before its child is explored.



(a) The typical directed graph (b) The typical directed acyclic graph

Figure 2.1: Directed graph and Directed acyclic graph

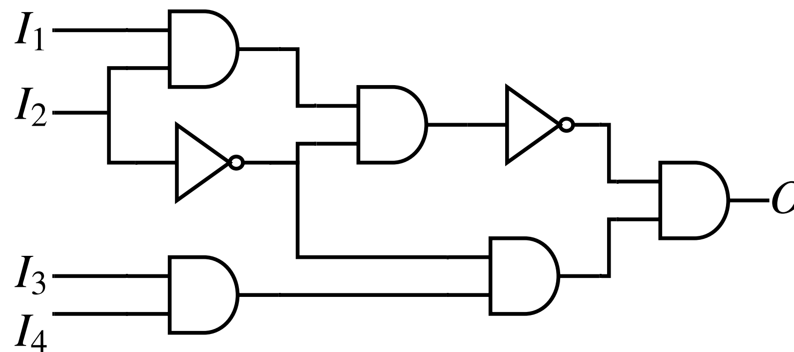
Consider the *Depth-first Inorder Search* of the *DAG* in Figure 2.1b as an example. The searching algorithm will first choose a "root" node, 1, search its first child, 2, and recursively the first child of 2, 6, which is a leaf node without any child. Then 6 will be visited before exploring the second child of 2, 4, so one so forth. Eventually, the visiting order of this *DAG* should be $\{6, 7, 4, 2, 5, 3, 1\}$.

Note that the above order shows that the traversing algorithm visits a certain vertex exactly once. Thus no duplicated nodes should present in the visiting list.

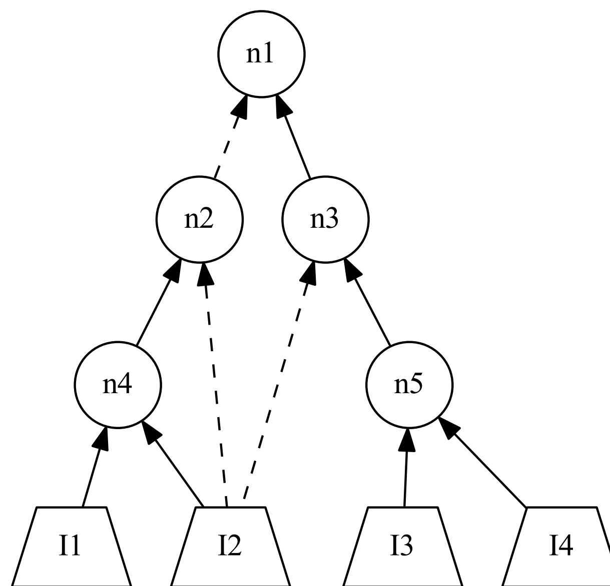
Similarly, the visiting order with the *Depth-first Preorder Search* should be $\{1, 2, 6, 4, 7, 3, 5\}$. This order is also called the *Topological Ordering* of a *DAG*.

AND-Inverter Graphs (AIGs) are *DG*s, of which each node corresponds to a two-input *AND* gate and the edges, representing the node interconnections, optionally having an *Inverter*. The *AIG* is a common and convenient method to represent a digital circuit. A sample circuit in the schematic and the corresponding *AIG* form is presented in Figure 2.2a and Figure 2.2b, respectively. In the *AIG*,

the circles represent *AND* nodes, the solid lines represent direct gate connections, and the dashed lines represent connections through an *Inverter*.



(a) The sample gate-level circuit



(b) The corresponding *AIG* representation

Figure 2.2: A sample circuit and its *AIG* representation

*AIG*s are used as a basic data structure to model logic circuits. For many applications in circuit design, it becomes a natural choice mainly due to the following reasons:

- Using *AIG* makes it simple and straightforward to implement the *EDA* algorithms proposed in this thesis, because traversing an *AIG* can be simply modelled as traversing the *DAG*s;

- It is well known that all logic circuits can be constructed using *AND* and *Inverter* gates;
- It will be much easier to integrate the proposed algorithms to some open-source *EDA* tools such as *ABC* [56], which uses *AIG* as a data structure to model logic circuits as well;
- The *i-RISC* project uses *AIG* as the main data structure.

2.2 Conditional Probability and Theorem of Total Probability

In probability theory, one of the most important concepts is the dependencies of events and conditional probability [71].

Definition 2.3 ***(Conditional Probability)*** *The conditional probability is the probability of an event B under the condition that another event A has been known to occur, called the probability of B given A, and denoted by $P(B|A)$ with:*

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \qquad \textit{or} \qquad P(A \cap B) = P(B|A)P(A)$$

The definition tells that the joint probability of two events A and B can be obtained by multiplying the probability of an event A and the probability of B in the case that A is known to have occurred.

Definition 2.4 ***(Independent Events)*** *Two events A and B are independent if and only if:*

$$P(B|A) = P(B) \qquad \textit{or} \qquad P(A|B) = P(A) \qquad \textit{or} \qquad P(A \cap B) = P(A)P(B)$$

, assuming the existence of the conditional probabilities. Otherwise, A and B are dependent.

The definition of independent events indicates that

- All probabilities can be treated as conditional probabilities;
- The joint probability of two independent events A and B can be obtained by simply multiplying their probabilities.

Sometimes, the probability of an event A is difficult to obtain directly, but the sample space S can be partitioned into several mutually exclusive events $B_1, B_2, \ldots B_k$, in this case, the *Theorem of Total Probability* (TTP) can be applied as shown in Figure 2.3

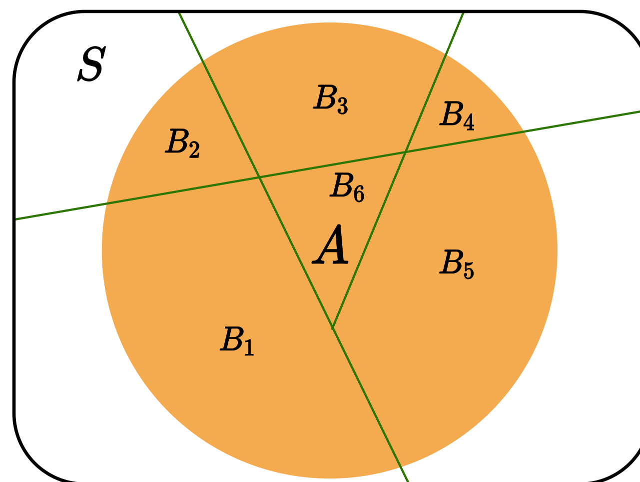


Figure 2.3: Theorem of total probability Venn diagram

Definition 2.5 ***(Theorem of Total Probability)*** *If the events $B_1, B_2, \ldots, B_k$ constitute a partition of the sample space S such that $P(B_i) \neq 0$ for $i = 1, 2, \ldots, k$, then for any event A of S,*

$$P(A) = \sum_{i=1}^{k} P(B_i \cap A) = \sum_{i=1}^{k} P(A|B_i)P(B_i) \tag{2.1}$$

2.3 Probability Density Function

Probability Density Function (PDF) is defined in terms of *Random Variables* (RV).

Definition 2.6 ***(Random Variable)*** *is a* ***function*** *$x(\zeta)$ that associates a real number x_i with each element (observation) ζ_i in the sample space S. A capital letter X is by convention used to denote a random variable, and the corresponding*

lower case, x is one of its sample values.

If an *RV*, X, can take an infinite number of possible values in the sample space S, then the *RV* is called *continuous RV*, the circuit propagation delay τ_{pd} is one example of continuous *RV*s.

Given a continuous *RV*, X, the probability can only be calculated on certain event occurrence such as $\{X \leq x_0\}$ or $\{x_1 \leq X \leq x_2\}$, yet the probability of any event that sample takes an exact value $X = x_0$ should assume to be zero. Thus the *PDF* is defined as:

Definition 2.7 ***(Probability Density function)*** *is a function $f(x)$ of continuous RV X that satisfies:*

- $f(x) \geq 0$*, for all* $x \in R$*;*
- $\int_{-\infty}^{+\infty} f(x)dx = 1$*;*
- $P(x_1 \leq X \leq x_2) = \int_{x_1}^{x_2} f(x)dx$*; and*
- $P(X = x_0) = 0$

Another useful function for *RV*s is the *Cumulative Distribution Function* (CDF), $F(x)$, that is defined as:

Definition 2.8 ***(Cumulative Distribution Function)*** *is a function $F(x)$ of continuous RV, X, with density function f(x) that satisfies:*

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(x)dx, \qquad x \in [-\infty, +\infty] \tag{2.2}$$

An *RV* has two basic properties, *Mean* and *Variance*.

Definition 2.9 ***(Mean of an RV)*** *Let X be an RV with probability distribution $f(x)$. The **mean** or **expected value**, μ_X, of X is:*

$$\mu_X = E(X) = \int_{-\infty}^{\infty} xf(x)dx \tag{2.3}$$

Definition 2.10 ***(Variance of an RV)*** *Let X be an RV with probability distribu-*

tion $f(x)$ and mean μ. The variance, σ_X^2 of X is:

$$\begin{aligned} \sigma_X^2 = E[(X-\mu)^2] &= \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx \\ &= E(X^2) - \mu^2 \end{aligned} \quad (2.4)$$

The positive square root of the variance, σ_X, is called the ***standard deviation*** *of X.*

The *Sum* and *Max* function of two *RV*s are also important concepts.

Theorem 2.1 ***(Sum of two independent RVs)*** *Let X and **Y** be two independent RVs with distribution $f_X(x)$ and $f_Y(y)$. Then the sum $Z = X + Y$ is an RV with distribution $f_Z(z)$, where f_Z is the* ***convolution*** *of f_X and f_Y.*

$$\begin{aligned} f_Z(z) = (f_X * f_Y)(z) &= \int_{-\infty}^{\infty} f_X(z-y)f_Y(y)dy \\ &= \int_{-\infty}^{\infty} f_Y(z-x)f_X(x)dx \end{aligned} \quad (2.5)$$

Theorem 2.2 ***(Maximum of two independent RVs)*** *Let X and Y be two independent RVs with distribution $f_X(x)$ and $f_Y(y)$. Then the maximum $Z = max\{X,Y\}$ is an RV with distribution $f_Z(z)$, where f_Z is given by:*

$$f_Z(z) = F_X(z)f_Y(z) + f_X(z)F_Y(z) \quad (2.6)$$

Three typical *PDF* types are employed most in my work, i.e., *Uniform Distribution*, *Gaussian Distribution* (GD) and *Inverse Gaussian Distribution* (IGD).

Definition 2.11 ***(Uniform distribution)*** *X is said to be uniformly distributed in the interval (a,b), $-\infty < a < b < \infty$, if:*

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \textit{otherwise} \end{cases} \quad (2.7)$$

A uniform distributed RV is denoted by $X \sim U(a,b)$.

Definition 2.12 ***(Gaussian distribution)*** *X is a Gaussian RV with parameters μ and σ^2 if its density function is given by:*

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \times exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.8}$$

A Gaussian distributed RV is denoted by $X \sim N(\mu, \sigma^2)$.

As the *GD* parameters, μ and σ indicate, the *GD* is well defined by the mean, and standard deviation of X. *GD* example curves are depicted in Figure 2.4. It can be seen that the *GD* is defined on $x \in (-\infty, +\infty)$, and μ determines the position of the curve, while σ determines the shape. Larger σ produces larger variance, which leads to a more flat curve shape, and vice versa.

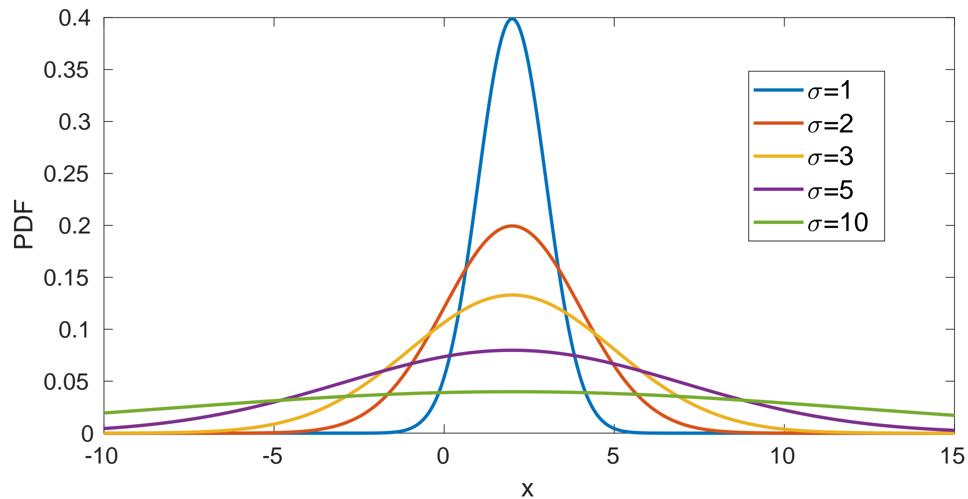


Figure 2.4: *GD* curves with $\mu = 2$ and different σ

Definition 2.13 ***(Inverse Gaussian distribution)*** *X is an Inverse Gaussian RV with parameters μ and λ if its density function is given by:*

$$f(x) = \sqrt{\frac{\lambda}{2\pi x^3}} \times exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right) \quad x > 0, \mu > 0, \lambda > 0 \tag{2.9}$$

IGD captures the distribution of the time a *Brownian motion* particle with positive drift takes to reach a fixed positive level, while the *GD* describes the level at a fixed time of Brownian motion. Sample *IGD* curves are shown in Figure 2.5.

As shown in the figure, the *IGD* curve is defined only on $x > 0$, and μ is the

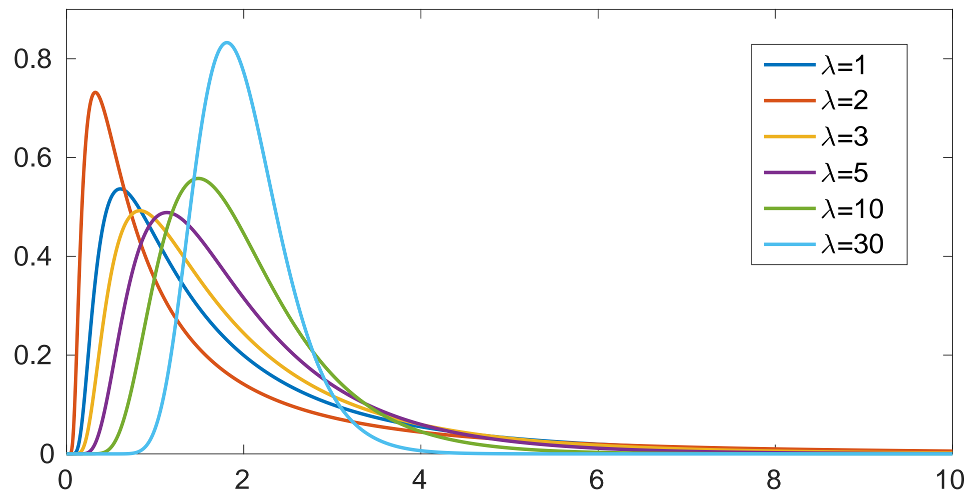


Figure 2.5: *IGD* curves with $\mu = 2$ and different λ

mean value, and λ is a shape adjustment parameter. Larger λ makes the peak more approaching to the mean value, i.e., more *GD*-like.

2.4 Monte-Carlo Simulation

Monte-Carlo (MC) simulation, as its name implies, is a method relying on simulations instead of the analytics. The basic idea is to repeatedly simulate many random or selected samples from a given set in a parametric space then measure the interesting resultant events. As long as the simulation repetition is sufficiently large, a statistical view of the results will appear due to the *Law of Large Numbers* [72].

MC method was originally used as a method for estimating the area of an enclosed curve stochastically and hence numerical integral approximation, especially more efficient in the scenarios of high dimensional integrals compared to the traditional numerical methods such as the *Midpoint Estimation*.

The procedure of *MC* simulation methodologies is simple yet powerful. Taking enclosed curve area estimation as an example and considering the curve shown in Figure 2.6, when estimating the area enclosed by the curve A, a number of samples $(x_i, y_i) \in S$ will be uniformly randomly [1] generated. The generated

[1]Notice that randomly generating samples is a traditional method, there are also advanced

samples usually act as inputs to a system when doing the repeated simulations on the system. In the case of the area estimation, the simulation procedure is just to "plot" the sampled points onto the figure. Then a measurement will be performed on each sample, and in this case, the measurement is an observation of whether the plotted point is inside or outside the enclosed curve A.

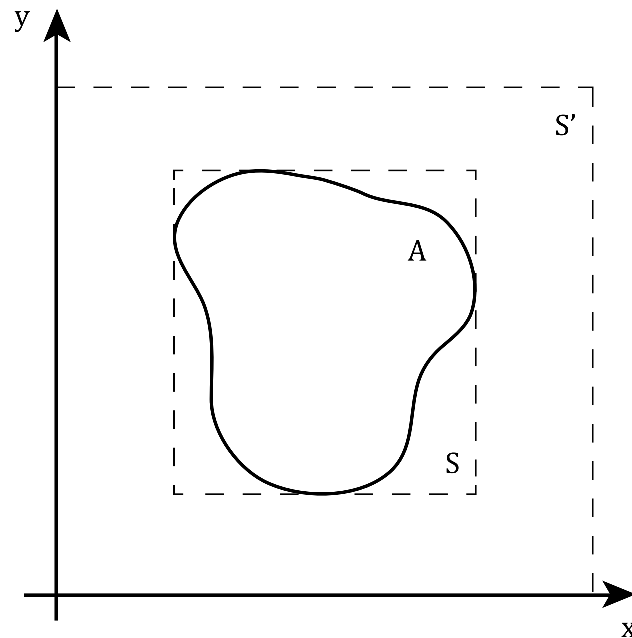


Figure 2.6: Estimating the area enclosed by curve A

Once the total number of samples, N, the area of sample space S has been given, and the number of points that fall inside curve A, n, was measured, the area enclosed by the curve A, $\hat{A}_A$, can be approximated using the Equation 2.10.

$$\hat{A}_A = A_S \times \frac{n}{N} \tag{2.10}$$

From the area calculation example described above, it is obvious that employing the following methods can improve the efficiency of *MC* simulation:

- Sampling needs to be performed on proper space, e.g., sampling on S is obviously of more efficiency on S' shown in Figure 2.6

sampling methodologies which may dramatically improve the performance of *MC* simulation in some scenarios.

- Make the samples as evenly distributed as possible on the whole space, which helps to reduce the total number of samples needed hence reduce the cost of doing the simulations.

2.5 Pearson Correlation Coefficient

In statistics, it is a common requirement to obtain a view of correlations between *RV*s. *Pearson Correlation Coefficient* (PCC) [73] is one of the simplest methods that is a measure of linear correlations between two *RV*s, X and Y, defined by Equation 2.11.

$$\begin{aligned}\rho(X,Y) &= \frac{cov(X,Y)}{\sigma_X\sigma_Y} \\ &= \frac{E[X\cdot Y]-E[X]\cdot E[Y]}{\sigma_X\sigma_Y}\end{aligned} \tag{2.11}$$

The equation reveals that the *PCC* is obtained by normalising the covariance with the product of standard deviations.

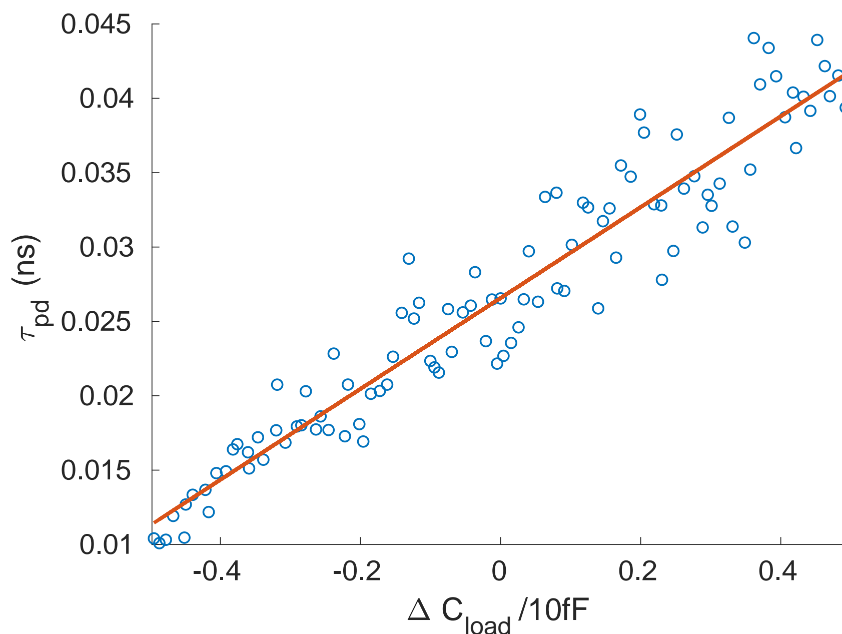


Figure 2.7: Scatter plot of τ_{pd} against C_{load} demonstrating a linear correlation

Figure 2.7 is a typical scatter plot generated using data from *HSPICE* simulation detailed in Section 4.4 with a linear fitting line. It is obvious from this figure that a strong linear correlation exists between τ_{pd} and C_{load}, i.e. $\tau_{pd} \propto C_{load}$, as expected, the *PCC* value is 0.9559, or there is $96\ \%$ confidence that τ_{pd} is linearly correlated to C_{load}.

For N *RV*s, $X_1, X_2, \ldots, X_N$, the correlation coefficients are commonly organised as an $N \times N$ symmetric matrix of which each entry is a correlation coefficient between each element of *RV*s X_i and X_j, $\rho(X_i, X_j)$, where $i, j = 1, 2, \ldots, N$.

2.6 Artificial Neural Network and Backpropagation

The *Artificial Neural Network* (ANN) mimics the functioning of the biological brains, which transports electric signals among a large amount of highly connected neurons. A simplified biological neuron model, shown in Figure 2.8, is composed of a *synapse*, a *cell body*, and a long nerve fibre called the *axon*. The synapse is a contact point of the axon of one neuron and the cell body of

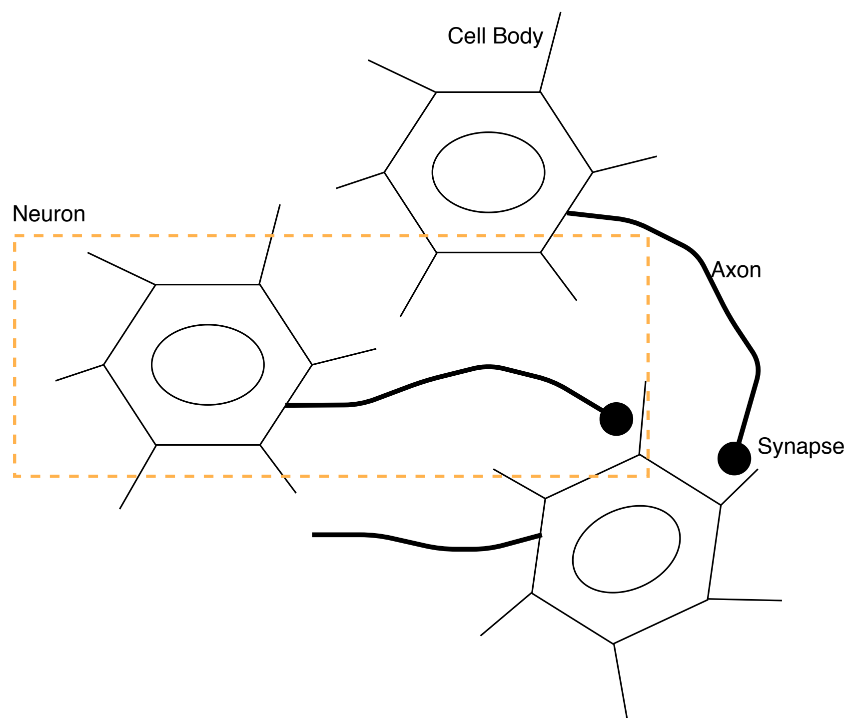


Figure 2.8: Biological neuron simplified structure

the other neuron, and typically one neuron cell body receives electric signals from many other neurons through their synapses[74]. The cell body operates as an adder processing unit that sums all properly weighted electric signals from other neurons, and then the cell body will decide whether to fire a signal depending on a comparison between the multiply and summing result and a threshold.

Inspired by the working principle of the biological neuron, an artificial neuron can be modelled, as shown in Figure 2.9.

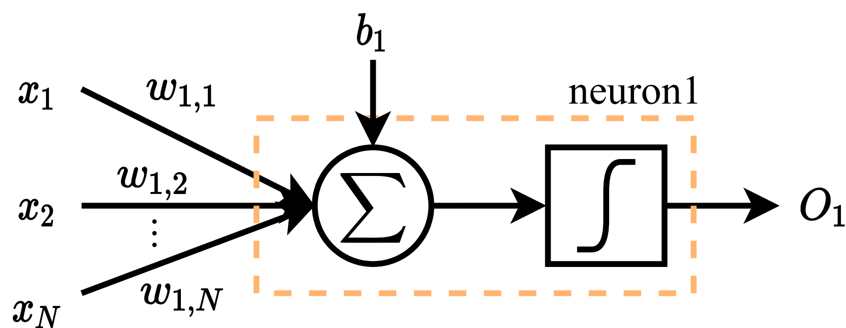


Figure 2.9: Artificial neuron model

In the diagram, $x_1, x_2, \ldots, x_N$ are N inputs, $w_{1,1}, w_{1,2}, \ldots, w_{1,N}$ are the corresponding weights, $\sum$ performs the summation of all weighted inputs, b_1 is an optional bias, the activation is normally represented by a non-linear transfer function, $f(t)$, such as *Purelin*, *Sigmoid*, and *Hardlim*. The most used activation is the *Sigmoid* function throughout this thesis. The *Sigmoid* can be treated as a "soft limit" function, as expressed by Equation 2.12.

$$f = \frac{1}{1 + e^{-x}} \qquad x \in [0, 1] \tag{2.12}$$

With the given activation function, the neuron function can be expressed as Equation 2.13. As can be seen from the equation, a neuron output is a non-linear function of a linear network output that is a *linear combination* of all inputs.

$$a_1 = f_1(w_{1,1} \times x_1 + w_{1,2} \times x_2 + \cdots + w_{1,N} \times x_N + b_1) \quad (2.13)$$

The equation can be written in the compact linear algebra form, as shown in Equation 2.14

$$a_1 = f_1(\boldsymbol{w}_1 \cdot \boldsymbol{x} + b_1) \quad (2.14)$$

The *MLP* is a type of *ANN* that has a feedforward architecture with two or more trainable layers [75]. Figure 2.10 depicts a typical two-layer *MLP-ANN* architecture. The architecture can be represented by a *DAG*, which is organized into two trainable layers, i.e., a hidden layer and an output layer. Employing the terminologies from neuroscience, nodes in the diagram represent the artificial neurons, and the edges represent the synapses.

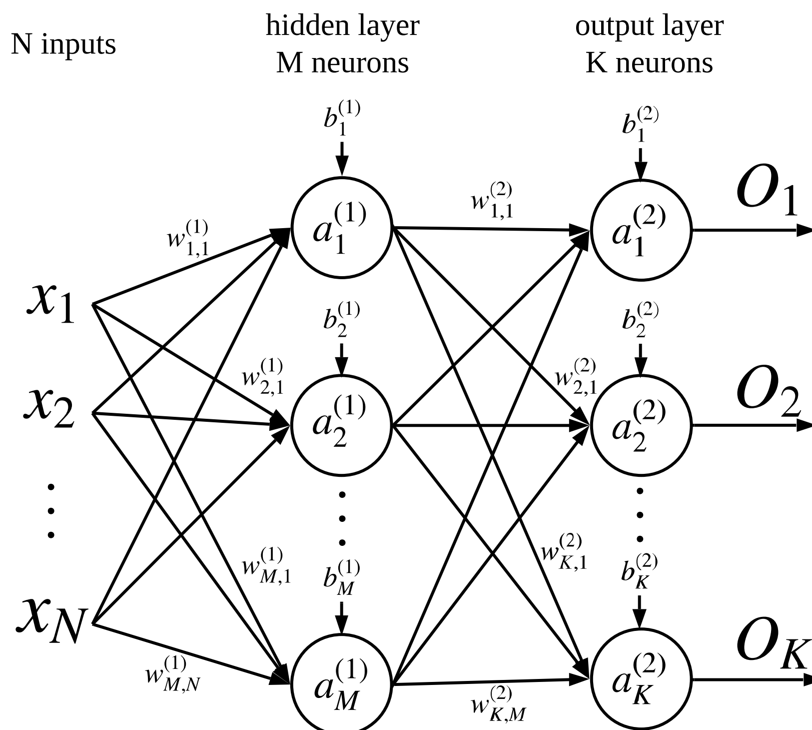


Figure 2.10: Typical multilayer perceptron neural network

By convention, the label $w_{i,j}^{(l)}$ on the edges represents the weights in the l-th layer from input node/neuron j to the corresponding output neuron i; the label

$b_i^{(l)}$ denotes the additional biases taken by the neurons in the l-th layer, and the label $a_i^{(l)}$ inside the neurons denotes the output of the neuron activation functions.

Similar to the neuron function of single neuron shown in Equation 2.13, an *MLP* can be represented by Equation 2.15.

$$a_i^{(l)} = f_i \left[\sum_j \left(w_{i,j}^{(l)} \times a_j^{(l-1)} \right) + b_i^{(l)} \right] \tag{2.15}$$

Or the more compact matrix version in Equation 2.16.

$$\boldsymbol{a}^{(l)} = \boldsymbol{f} \left(\boldsymbol{W}^{(l)} \cdot \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)} \right), \tag{2.16}$$

where $\boldsymbol{a}^{(0)} = \boldsymbol{x}$, $\boldsymbol{a}^{(L)} = \boldsymbol{O}$, $\boldsymbol{x}$ and $\boldsymbol{O}$ are the input and output vectors, respectively. The weight matrices $\boldsymbol{W}^{(l)}$, together with the bias vectors $\boldsymbol{b}^{(l)}$, are called the *network parameters* in the rest of the thesis. The reason why *ANN* is useful is that the network parameters are trainable, in other words, some combinations of the network parameters can be found so that the *ANN* can perform some specific "mapping" operations, which is one of the most important and useful properties of the *ANN*. Specifically, it has been proven that the *MLP* with a finite number of neurons can be used to approximate any continuous function with any desired precision, which holds even when the network contains only a single layer [76]. In fact, an *MLP* can approximate an arbitrary function, regardless of the type of the function (continuous or Boolean). This property is also known as the *Universal Function Approximation*.

The process that is numerically finding the optimal network parameters is called the network *training* or *learning process*, hence the algorithms trying to find the optimized solutions in a learning process are called *training* or *learning algorithms*. Although the most common learning rules include unsupervised, supervised, and reinforcement learning, supervised learning is the one extensively used in this thesis. As the name suggests, supervised learning trains the

network using the existing data, either from experimental measurements, historical observations or theoretical estimations, etc.

The basic idea of a typical learning algorithm is to make small modifications on the network parameters iteratively until a measurement of the learning performance, i.e., a cost function, is optimized. For example, Equation 2.17 is a commonly used cost function in regression and classification, the *Mean Squared Error* (MSE).

$$C = \epsilon(t, \hat{y}) = \frac{1}{N}\|\boldsymbol{t} - \hat{\boldsymbol{y}}\|^2 = \frac{1}{N}\sum_{N}(t - \hat{y})^2, \tag{2.17}$$

where t denotes the target output of the network and $\hat{y}$ is the predicted value. Once a minimal value of the cost function is found, the predicted vector has the shortest distance to the target vector in the *Euclidean* space.

The parameter updating scheme distinguishes the various training algorithms. An intuitive and naive method is to make a small perturbation on each of the parameters until the cost function is improved in this step/iteration. This method soon becomes infeasible once the network is getting bigger, and the problem is getting more complex.

The emergence of derivative-based training algorithms, along with the use of *Backpropagation* (BP), makes the training of deep learning networks more practical for solving real-world problems. Contrary to the perturbation-based training algorithms, *BP* compatible derivative-based training algorithms are trying to evaluate the contribution of each parameter to reducing/increasing the cost function, and then update the parameters, accordingly. In other words, the *BP* algorithm enhances the parameters making a positive impact on reducing the cost function, and vice versa.

This process can be interpreted by deriving the partial derivative of the cost function with respect to each of the parameters, which can be denoted by the *Jacobian Matrix* in Equation 2.18.

$$
\boldsymbol{J}_k = \begin{bmatrix}
\frac{\partial e_{1,k}}{\partial x_{1,k}} & \frac{\partial e_{1,k}}{\partial x_{2,k}} & \cdots & \frac{\partial e_{1,k}}{\partial x_{p,k}} & \cdots & \frac{\partial e_{1,k}}{\partial x_{P,k}} \\
\frac{\partial e_{2,k}}{\partial x_{1,k}} & \frac{\partial e_{2,k}}{\partial x_{2,k}} & \cdots & \frac{\partial e_{2,k}}{\partial x_{p,k}} & \cdots & \frac{\partial e_{2,k}}{\partial x_{P,k}} \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\frac{\partial e_{m,k}}{\partial x_{1,k}} & \frac{\partial e_{m,k}}{\partial x_{2,k}} & \cdots & \frac{\partial e_{m,k}}{\partial x_{p,k}} & \cdots & \frac{\partial e_{m,k}}{\partial x_{P,k}} \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\frac{\partial e_{M,k}}{\partial x_{1,k}} & \frac{\partial e_{M,k}}{\partial x_{2,k}} & \cdots & \frac{\partial e_{M,k}}{\partial x_{p,k}} & \cdots & \frac{\partial e_{M,k}}{\partial x_{P,k}}
\end{bmatrix}, \tag{2.18}
$$

where k is sample index, J_k is the *Jacobian Matrix* of sample k, $\frac{\partial e_{m,k}}{\partial x_p}$ denotes the partial derivative of the m-th output error $e_{m,k}$ with respect to the p-th parameter $x_{p,k}$ resulting in $n \times m$ entries.

Depending upon the order of partial derivative used by a specific training algorithm, the modern popular training algorithms can be classified into two categories, the first-order and second-order training algorithms [77]. In the first-order algorithms, i.e., *Gradient Descent-based* methods such as *Stochastic Gradient Descent (SGD)* and *Adam* [78], etc., the *Jacobian Matrix* (often reduced to gradient vector) is used, while in the second-order algorithms such as *Newton's Method* [79], the *Hessian Matrix* must be used. As the calculation of the *Hessian Matrix* is very computationally intensive and thus is infeasible in most of the real-world large network training tasks, some methodologies such as *BFGS Quasi-Newton* [79] and *Levenberg-Marquardt* (LM) [80, 81] have been invented to approximate the *Hessian Matrix* using first-order derivatives.

First-order algorithms are extensively used in training of the modern large scale deep neural networks due to their advantages in computation speed[77]. On the other hand, second-order algorithms are less computationally efficient yet can achieve higher accuracy in training the moderate scale shallow networks [82].

A derivation of the full *BP* algorithm is presented in Appendix A. Besides, as the *LM* algorithm is crucial in *ANN* function approximator, the derivation of the *LM* algorithm is also depicted in Appendix B.

2.7 Chapter Summary

In this chapter, some common mathematical tools and basic concepts used in this thesis were discussed.

Firstly, the *DAG*, *AIG* and the traversing algorithms were introduced. Secondly, the conditional probability and the Theorem of total probability were discussed. These are the basic tools for the soft error propagation and reliability optimisation algorithms presented in Chapter 3. Thirdly, the concept of probability density function and some common density functions used in the thesis such as Gaussian Distribution, Inverse Gaussian Distribution were presented, which are useful for the timing-related reliability analysis models proposed in Chapter 4. Then, the concept and application of the Monte-Carlo simulation method were introduced, which is used as reference methods for the reliability analysis algorithms proposed in the thesis. Finally, the Pearson correlation coefficient and the Artificial neural network were introduced.

Chapter 3

Analytical Soft Error Propagation, Reliability Analysis, and Optimisation for *VLSI* Combinational Circuits

As the noise margin is getting tighter, hence the digital circuit is more vulnerable to noise, for lower operating voltage, it is essential to evaluate and improve the reliability at an early stage of the circuit design. Error propagation mechanism is a complex problem in the area of *Electronic Design Automation* (EDA), as the masking effects exist, and the correlations like *Reconvergent Fanout* (RFO) further increase the complexity of the problem requiring lengthy simulations. Thus, to effectively optimize the error resilience of the circuit, efficient error propagation and reliability analysis tools are needed. In this chapter, a novel and efficient method for combinational circuit error propagation analysis called *Conditional Probabilistic Error/Reliability Propagation Analysis* (CPEP) algorithm will be presented first. The data structure used to represent the circuit is based on the *AND-Inverter Graph* (AIG). However, the algorithm is quite generic in nature and can be applied for any logic gate.

Then, based on the reliability estimates, a novel *EDA* algorithm for reliability-driven synthesis based on *cut-rewriting* technique is proposed. As it has been noticed that adding redundant nodes to the circuit will increase the possibilities that the errors were masked while propagating through the circuit [19], by rewriting a certain cut with a better reliability subcircuit from a set of pre-computed *Negation-Permutation-Negation* (NPN) equivalent replacement structures, the reliability of the circuit can be enhanced.

The rewriting technique is also employed in the *ABC* [56] for area driven synthesis, where the area is the optimisation target, and the rewriting is driven by trying to pick the cut with the least number of nodes. Thus, the basic rewriting algorithm used in the *ABC* was directly adapted to our reliability-driven synthesis algorithm, which drastically simplifies the process. In our work, 4-*input cuts* based rewriting, i.e., rewriting from a subset of 4-input *NPN* equivalent functions, is employed to improve the reliability. In order to demonstrate the working principle of our reliability enhancement algorithm, the *MCNC* benchmark circuit *cm162a* was explained in detail as a case study.

Also, the performance of our reliability analysis and enhancement approaches are evaluated on a set of *MCNC* benchmark circuits. Experimental results show that with the proposed *CPEP* framework, average accuracy degradation within $3\,\%$ and more than 10^3 times faster are achieved when compared to *Monte-Carlo* (MC) simulations. Meanwhile, with the *RWREL* algorithm, the average reduction in output error of up to $14\,\%$ while a peak improvement of up to $75.5\,\%$ is observed while keeping an acceptable area overhead of $6.57\,\%$ that results in $13.52\,\%$ higher power consumption.

The rest of the chapter is organized as follows: In Section 3.1, the probabilistic methodology, *CPEP*, is introduced, and the data structure, mathematical gate error models, and the correlation issues due to the *RFO*s are presented. Section 3.2 presents the top-level algorithm and the consideration of multiple outputs. Section 3.3 states the experimental results of the *CPEP* algorithm on several benchmark circuits and the comparison with *MC* Simulation method. Finally, in

section 3.4, the synthesis algorithm, *RWREL*, for the reliability enhancement is presented.

3.1 *CPEP*: Conditional Probabilistic Error/Reliability Propagation Analysis

3.1.1 Basic Concepts

Assuming all *Primary Inputs* (PIs) are error-free, when a fault randomly occurs in one of the gates in the circuit, the error may propagate through the circuit or not due to logic masking, electrical masking, and timing window masking mechanisms [83]. Thus, an unreliable logic gate is modelled as an ideal logic gate cascaded with a *fault injection XOR* gate. In other words, an unreliable gate is composed of three probabilistic gates, i.e., an *ideal* logic gate, a *fault injection XOR* gate, and a *fault PI* connecting to the *fault injection XOR* gate, portrayed by the *U* part in Figure 3.1. This model moves the entire error statistics onto the output, which is a fundamental assumption within the algorithm. The benefit of employing this model is that a fault injection gate can be treated as a normal logic gate and the fault of a gate can be treated as a normal input from a *PI* when traversing the graph, which dramatically simplifies the algorithm.

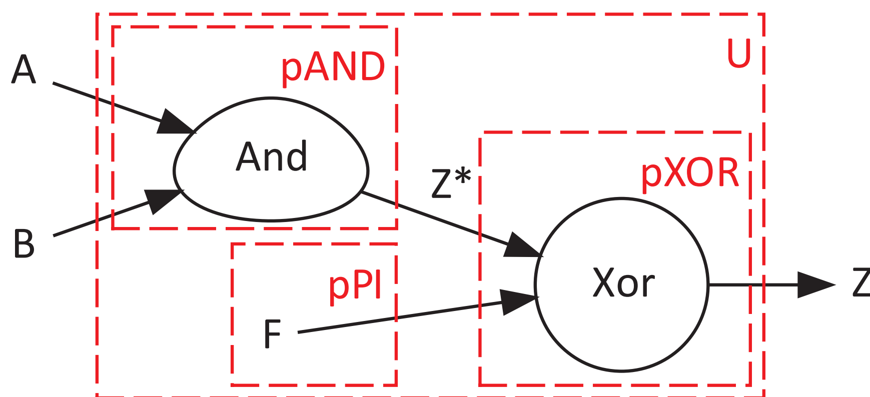


Figure 3.1: Erroneous gate model

Figure 3.1 graphically presents an unreliable *AND* gate model. As shown in

the diagram, an unreliable *AND* gate can be modelled as an ideal (fault-free) *AND* gate (represented by the "egg" shape in the rest of the thesis)followed by a fault injection *XOR* (represented by the circle) that determines the stochastic faulty behaviour by toggling the *fault PI* (denoted by the "F") with a pre-defined probability.

The data structure employed is based on the *AIG* [50, 51], which contains only 2-input *AND* gates, although the algorithm can be easily extended to any 0, 1, or even more than 2-input generic logic gates. In addition, all *PI*s of the circuit are assumed to be error-free.

Some of the common conventions used throughout the *CPEP* algorithm are listed below:

- Z^* - The ideal gate output (internal net)
- Z - The output of the entire unreliable gate (external net)
- x_0 - The logic value on net 'x' is "*0*"
- x_1 - The logic value on net 'x' is "*1*"
- x_ϵ - The logic value on net 'x' is incorrect
- x_c - The logic value on net 'x' is correct
- $P(X)$ - The probability of event X
- *SP*$_x$ - Static probability of net 'x', $P(x_1)$
- *EP*$_x$ - Error probability of net 'x', $P(x_\epsilon)$

3.1.2 The Independent Probabilistic Gate Model

Initially, the *PI*s (A and B) of an unreliable *AND* gate are assumed to be mutually independent, i.e., not sourced from the same fanout gate. For the unreliable *AND* gate shown in Figure 3.1, the output *static probability* (SP) is the joint probability of *SP*$_A$ and *SP*$_B$ as defined in Equation 3.1, when assuming the

input A and B are independent.

$$SP_{Z^*} = SP_A \times SP_B \tag{3.1}$$

A faulty state can present on the output Z of an unreliable gate due to one of the following two possible reasons:

- the propagated error is presenting on the gate input node, i.e., Z^*_ϵ; or
- the occurrence of a fault within the gate (injected fault), i.e., FPI_1.

For the first case, note that an input error does not necessarily result in a faulty output Z^*_ϵ. Consider, for example, an error-free "0" on one of the input pins of the *AND* gate, which would logically mask the error on the other input node from being propagated onto Z^*, and the gate always gives the correct output, "0". Similarly, one can consider the situation when the input pins are set to "0" and "1", but both are in an error state. For such a double error event, the inputs mutually negate each other, and they will still result in a correct output state.

Hence, due to the logic masking and double error events, there are no simple rules to predict the gate output state. Table 3.1 presents an exhaustive enumeration on all the possible cases with their associated output status.

To explain the table, consider when $A = 0$ and $B = 0$ in which case Z^* should ideally be "0". But, due to the fact that the gates in the circuit are unreliable, each of the inputs can possibly be in error (ϵ) or correct (c) state. The state of the inputs determines if $Z^* = 0$ is correct or not. It is clear that, for these input values, Z^* is in error state if and only if both of the inputs are in an error state. Therefore, it is evident from Table 3.1 that only six out of the sixteen possible cases may result in an error on Z^*.

The probabilities for each of these events to occur are presented in the last column in Table 3.1. Thus, the error probability of Z^* can be represented by

the sum of all the six terms in Table 3.1, as defined in Equation 3.2.

$$\begin{aligned} EP_{Z^*} =& P(A_0)P(A_\epsilon)P(B_0)P(B_\epsilon)+ \\ & P(A_0)P(A_\epsilon)SP_B P(B_c)+ \\ & SP_A P(A_c)P(B_0)P(B_\epsilon)+ \\ & [P(A_\epsilon)+P(B_\epsilon)-P(A_\epsilon)P(B_\epsilon)]SP_A SP_B \end{aligned} \tag{3.2}$$

By further detailed analysis of Table 3.1, it was observed that a correlation exists between the Z^* value and the status of Z^*.

Table 3.1: Truth table of unreliable *AND* gate

Actual/ State A	Actual/ State B	Actual/ State Z^*	Terms
0 / c	0 / c	0 / c	
0 / c	0 / ϵ	0 / c	
0 / c	1 / c	0 / c	
0 / c	1 / ϵ	0 / c	
0 / ϵ	0 / c	0 / c	
0 / ϵ	0 / ϵ	0 / ϵ	$t_1 = P[A_0, A_\epsilon, B_0, B_\epsilon]$
0 / ϵ	1 / c	0 / ϵ	$t_2 = P[A_0, A_\epsilon, B_1, B_c]$
0 / ϵ	1 / ϵ	0 / c	
1 / c	0 / c	0 / c	
1 / c	0 / ϵ	0 / ϵ	$t_3 = P[A_1, A_c, B_0, B_\epsilon]$
1 / c	1 / c	1 / c	$t_7 = P[A_1, A_c, B_1, B_c]$
1 / c	1 / ϵ	1 / ϵ	$t_4 = P[A_1, A_c, B_1, B_\epsilon]$
1 / ϵ	0 / c	0 / c	
1 / ϵ	0 / ϵ	0 / c	
1 / ϵ	1 / c	1 / ϵ	$t_5 = P[A_1, A_\epsilon, B_1, B_c]$
1 / ϵ	1 / ϵ	1 / ϵ	$t_6 = P[A_1, A_\epsilon, B_1, B_\epsilon]$

Therefore, equation 3.3 is used to identify the four different origins of the error terms. These equations will be used to propagate the error probability and reliability of the next level. Note that the equations also indicate that all errors have the identical probabilities at both logic states "*1*" and logic "*0*" in

current algorithm implementation.

$$\begin{aligned}
CP[Z_0^*, Z_c^*] &= P(Z_0^*) - (t_1 + t_2 + t_3) \\
CP[Z_0^*, Z_\epsilon^*] &= t_1 + t_2 + t_3 \\
CP[Z_1^*, Z_c^*] &= t_7 \\
CP[Z_1^*, Z_\epsilon^*] &= t_4 + t_5 + t_6
\end{aligned} \tag{3.3}$$

Next, the fault injection part can be considered as follows. The static probability of the fault injection *XOR* gate SP_Z follows Equation 3.4.

$$SP_Z = SP_{Z^*} \times P(F_0) + P(Z_0^*) \times SP_F \tag{3.4}$$

The calculation of EP_Z should be according to the faulty truth table of an *XOR* gate, as shown in Table 3.2, that presents the analysis of the possible configurations of the *XOR* gate. It is noticed that the *F PI* of the fault inject *XOR* gate has only two possibilities, whose value "1" means injecting a fault, and "0" means fault-free.

Table 3.2: Truth table of the error injection *XOR* gate

Actual/ State Z^*	Actual/ State F	Actual/ State Z
0 / c	0 / c	0 / c
0 / c	1 / ϵ	1 / ϵ
0 / ϵ	0 / c	0 / ϵ
0 / ϵ	1 / ϵ	1 / c
1 / c	0 / c	1 / c
1 / c	1 / ϵ	0 / ϵ
1 / ϵ	0 / c	1 / ϵ
1 / ϵ	1 / ϵ	0 / c

Similar to that of the *AND* gate, the error can propagate only if the output is an ϵ, in red colour, and there is a correlation between the value of Z and the status

of Z, as shown in Equation 3.5.

$$\begin{aligned}
CP[Z_0, Z_c] &= P[Z_0^*, Z_c^*, F_0] + P[Z_1^*, Z_\epsilon^*, F_1] \\
CP[Z_0, Z_\epsilon] &= P[Z_0^*, Z_\epsilon^*, F_0] + P[Z_1^*, Z_c^*, F_1] \\
CP[Z_1, Z_c] &= P[Z_0^*, Z_\epsilon^*, F_1] + P[Z_1^*, Z_c^*, F_0] \\
CP[Z_1, Z_\epsilon] &= P[Z_0^*, Z_c^*, F_1] + P[Z_1^*, Z_\epsilon^*, F_0]
\end{aligned} \tag{3.5}$$

Finally, the EP_Z and reliability are expressed by Equation 3.6 and Equation 3.7, respectively.

$$EP_Z = CP[Z_0, Z_\epsilon] + CP[Z_1, Z_\epsilon] \tag{3.6}$$

$$R_Z = 1 - EP_Z \tag{3.7}$$

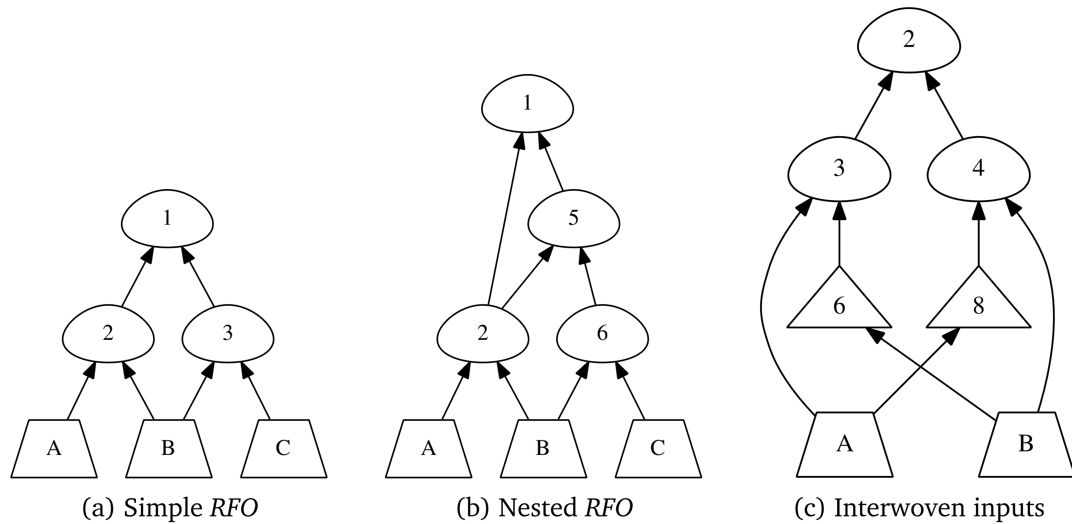
With the similar principle, the *NOT* gate model as well as other generic gates can be easily obtained.

3.1.3 Re-convergent Fanouts

A *fanout source* represents a gate output net driving more than one gate. A *Re-convergent Fanout* (RFO) node is defined as the gate that has more than one path to its inputs from one fanout source [84]. The methodology developed in the previous section does not account for the impact of the *RFO* node, which may lead to statistical dependencies between the two inputs, thereby affects the node output error probability estimation result.

As stated in [45], there are two types of *RFO*s, *Disjoint RFO*, and *Dependent RFO*, which can be further divided into *Nested RFO* and *Interwoven Branches of Inputs*. Some typical *RFO* scenarios are portrayed in Figure 3.2.

Computing the node error probability in the presence of *RFO* is complicated because the above equations for the independent model do not hold true. This is because each of the terms in Equation 3.2 cannot be factorized due to de-



(a) Simple *RFO* (b) Nested *RFO* (c) Interwoven inputs

Figure 3.2: *RFO* structures

pendencies, and there is no exact closed-form solution. Iterative approaches do exist but the algorithm implementation will be very complicated, and the algorithm complexity grows exponentially for each of the 6 terms, thus the run time can be easily out of the acceptable range. In our approach, all *RFO*s are treated as to be originated from the *PI*s, regardless of the type of *RFO*. In other words, when applying the *Theorem of total probability* (TTP) for a particular *RFO* node, sub-cases of its total probability are obtained by conditioning its corresponding fanout *PI*s rather than deriving closed-form propagation expressions for every single fanout node.

Specifically, two lists (ordered sets), namely *All PI* (API) list and *Common PI* (CPI) list, are employed to capture the details of all the *RFO*s and the corresponding fanout *PI*s.

Some operators should be defined for the list operations as follows:

- "$A \cup B$" – the list union operation, for which particularly $X \cup NULL = X$;
- "$A \cap B$" – the list intersection operation, for which particularly $X \cap NULL = NULL$;
- "$A + B$" – the appending operation, for which particularly we define $X + NULL = X$;

- "$A - B$" – the excluding operation, for which particularly we define $X -$ *NULL* $= X$ and *NULL* $- X =$ *NULL*

where *NULL* is an empty list. Note that the appending operator, "$+$", concatenates the list B to the end of list A, and the excluding operator, "$-$", removes the elements in the list B from list A.

For a two-input *AND* gate "Y", the left and right hand side child nodes are denoted by *LX* and *RX*, respectively. Then the *API* and *CPI* of Y can be derived using Equation 3.8 and Equation 3.9.

$$API(Y) = API(LX) \cup API(RX) \tag{3.8}$$

$$CPI(Y) = API(LX) \cap API(RX) \tag{3.9}$$

In the case of an *Inverter*, the right sub-node can be set to "*NULL*". Then the corresponding *API* and *CPI* can be derived using Equation 3.10 and Equation 3.11, respectively.

$$API(Y) = API(LX) \cup \textit{NULL} \tag{3.10}$$

$$CPI(Y) = API(RX) \cap \textit{NULL} \tag{3.11}$$

For a *PI*, the *API* and *CPI* are defined as itself and "*NULL*", respectively.

Consider the simple circuit in Figure 3.2a as an example, the lists are *API*$(1) = \{A, B, C\}$ and *CPI*$(1) = \{B\}$. Similarly, for the circuit in Figure 3.2b, an extra *PI*, A, has to be taken into consideration as well due to the fanout introduced by *node 2*. Then, the lists are *API*$(1) = \{A, B, C\}$ and *CPI*$(1) = \{A, B\}$.

The *CPI* list of a particular node represents its all fanout source *PI*s. Thus, the probabilities (for both of the *EP* and the *SP*) of this node can be expressed using the *TTP* in Equation 3.12.

$$P(Z) = \sum_{j=0,1} P(X_j)P(Z|X_j) \tag{3.12}$$

Specifically, when *CPI* list is not "NULL", then the first *PI* in *CPI* is set to "1" and "0", respectively, for conditioning. Once a certain value is assigned to the *PI*, a new sub-circuit can be obtained with the corresponding fanout been removed. Then, the same process will be performed for the second *PI* in *CPI*, and this process will be preformed iteratively until the circuit becomes the simplest circuit (the one without *RFO*s). After calculating the probabilities for the conditioned circuits with the independent models, the conditional probabilities will be obtained, which can be subsequently substituted into the Equation 3.12 to obtain the total probability.

It is obvious that the efficiency of this conditioning algorithm depends on the conditioning order of the *PI*s in the *CPI* list. If the common *PI*s in both *LX*s *CPI* and *RX*s *CPI* are conditioned first, the circuit will be conditioned to the simplest more rapidly. To achieve this improvement, another list called the *Order PI* (OPI) list to order the *PI*s, which in the *CPIs* of both *LX* and *RX* to be conditioned first, is introduced, as shown in Equation 3.13.

$$\begin{aligned} \textit{OPI}(Y) &= \textit{CPI}(LX) \cap \textit{CPI}(RX) \\ &+ [\textit{CPI}(Y) - \textit{CPI}(LX) \cap \textit{CPI}(RX)] \end{aligned} \tag{3.13}$$

For example, the *OPI* of *node 1* in the logic circuit in Figure 3.2b is $\{B, A\}$.

Algorithm 1 illustrates the process of resolving the *RFO*s in *CPEP* and will be discussed in detail in Section 3.2.

3.2 Reliability Evaluator Implementation

3.2.1 Consideration of Multiple Outputs

In the realistic combinational logic circuit in modern *VLSI*, most circuits have more than one or even larger number of outputs. Thus, the most straightforward method is employed, although there is a slight runtime penalty. Specifically, the complete circuit are split into several individual cones by each of the

Algorithm 1 Compute_EP Methodology

```
Require: OPI list
 1: procedure COMPUTE_EP(Z)                                ▷ Z is the root node
 2:     if Z is PI then
 3:         Read default values of SP, EP, CP of PI
 4:         return
 5:     end if
 6:     if Z is INV then
 7:         COMPUTE_EP(LX)
 8:         Calculate SP, EP, CP with independent model
 9:         return
10:     end if
11:     if OPI is NULL then                                ▷ Independent
12:         COMPUTE_EP(LX)
13:         COMPUTE_EP(RX)
14:         Calculate SP, EP, CP with independent model
15:     else                                               ▷ Reconvergent Fanout
16:         cond_PI = OPI.first
17:         New0=CONDITION(Z, cond_PI, 0)
18:         COMPUTE_EP(new0)
19:         New1=CONDITION(Z, cond_PI, 1)
20:         COMPUTE_EP(new1)
21:         Calculate total probability employing Equation 3.12
22:     end if
23: end procedure
```

outputs, and the CPEP algorithm for a single output circuit as described above is applied for each cone from each output node iteratively. This approach allows the cones under each of the outputs to maintain the original function, and the original fanouts and *RFO*s topologies. The detailed implementation is discussed in section 3.2.3.

3.2.2 Data Structure

As shown in Figure 3.1, the employed data structure that suits to implement the proposed model is composed of three layers, namely *probabilistic gate* (PGate), *AIG node* (AIGn), and *AIG*.

PGate A probabilistic gate is an extension of an ideal logic gate where each logic gate consists of these 3 parameters: *SP*, *EP*, and *CP*. The equations are

implemented depending on the gate functionality. The *'P'* part described in Figure 3.1 comprises the probabilistic gate. There are five main types of *PGate*, *pAND*, *pXOR*, *pINV*, *pPI* and *pLogic*, which represent the *AND*, *XOR*, *NOT*, *PI*, and logic constant 0 and 1, respectively.

AIGn An *AIG* node is a graph node composed of probabilistic gates and the *PI*s lists related to the *RFO* resolving algorithm.

AIG The complete graph is representing the circuit with all *AIG* nodes implemented using a hash table that enables us to easily propagate the probabilities, lists, and the logic values.

3.2.3 Computation Algorithm

Algorithm 2 presents the top-level procedure to compute circuit reliability employing the *CPEP* methodology.

Algorithm 2 Top Level Description of the *CPEP*

Require: Input Circuit Gate Level Netlist
1: Generate data structure according to the netlist file
2: Error injection procedure
3: $CurrentPO = PO.first$ ▷ First element in output_list
4: **while** $CurrentPO! = NULL$ **do**
5: UPDATE_LISTS($CurrentPO$)
6: COMPUTE_EP($CurrentPO$)
7: $CurrentPO = CurrentPO \rightarrow next$ ▷ Move to next output
8: **end while**

The algorithm is implemented utilizing *Depth-first Inorder Search* (DFS) method, due to its recursive definition of the graph. The input of this algorithm is a netlist file consisting of the gate level description of the circuit, gate error rate, and input *SP*s on the *PI*s. The first step is to convert the input netlist into *AIG* data structure in the internal format. All of the *PO* nodes are saved into a list called *output_list*. For each element within the *output_list*, a recursive procedure is employed to perform reliability calculations on all the gates within that cone.

The implementation details of the *EP* computation procedure are presented in Algorithm 1. It is rather simple to traverse a binary tree topology, which is in the circuits without *RFO*s (except for the *PI*s) such as the circuit in Figure 3.2a. However, when traversing an *AIG* such as the one in Figure 3.2b, a straightforward method slightly compromising on total runtime is employed, that is each time visiting node X from different parent nodes, the sub-tree under node X is traversed once again. For example, the visiting order of Figure 3.2b circuit is "$1 \rightarrow 2 \rightarrow A \rightarrow B \rightarrow 5 \rightarrow 2 \rightarrow A \rightarrow B \rightarrow 6 \rightarrow B \rightarrow C$".

Obviously, the computation complexity of the algorithm depends on the number of *RFO*s in the circuit.

3.3 Test Environment Setup and Comparison Results

To evaluate the performance of the CPEP algorithm, the Monte-Carlo (MC) simulation-based approach was developed as the reference. Simulation-based reliability estimation techniques can be employed at various levels of abstraction, namely, at the transistor, gate, and block-level with different accuracy levels. In this thesis, the gate-level simulation-based reliability estimation is considered, which involves simulating a Boolean network consisting of logic gates while keeping track of transitions in both error-free and error-prone conditions. During a simulation, the gate output value is determined according to the values at the gate input each time an input changes. Gate-level simulation for switching activity computation is a well-studied problem, and much effort has been placed on improving its speed [85, 86, 87].

Note that while the simulation-based methods are more accurate when compared to probabilistic methodologies, its effectiveness depends on the availability of significantly random input vectors. The traditional random function $rand$ from the 'C' library provides random numbers with a small period of length $2^{32} - 1$. Since the number of required simulation steps runs into millions, there

is a high possibility that patterns are repeated and highly correlated with each other. To overcome this problem, the *Mersenne twister* pseudo-random number generator is used. The name has derived from the concept of *Mersenne prime*, which in mathematics is a prime number of the form $M_n = 2^n - 1$ and has a very long period of $2^{19937} - 1$. Thus, it helps us in overcoming the limitation of the standard *rand* function.

The sample size used for reliability analysis is 10^6 *MC* simulations for 0.001, 0.01, and 0.05 gate error rates. Assume that the *PI*s are independent and set the switching activity on all the input pins to 0.5. The switching activity numbers are flexible and can be set to any other value.

Table 3.3: Accuracy and performance evaluation for different gate errors (ϵ)

Benchmark	PI Count	Gate Count		Avg Error Deviation on all outputs %			Runtime (s)	
		AND	*Inverter*	$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.05$	GLS	CPEP
apex6	135	659	544	0.384	1.079	0.243	2889.96	**0.012013**
b9	41	101	77	0.247	0.696	0.138	520	**1.515699**
cht	47	187	189	0.237	0.702	0.308	789.97	**0.003880**
cm138a	6	16	13	0.130	0.389	0.129	120.03	**0.000154**
cm42a	4	18	14	0.133	0.267	0.518	140.07	**0.000101**
cm82a	5	20	25	0.514	1.132	2.549	129.98	**0.015272**
cm85a	11	36	30	0.412	1.200	0.908	209.94	**0.991886**
cmb	16	47	29	0.153	0.710	0.853	220.08	**0.000597**
count	35	127	130	0.282	0.416	1.004	530.07	**0.015586**
cu	14	55	29	0.120	0.290	0.150	260.08	**0.096305**
decod	5	30	4	0.080	0.250	0.090	159.94	**0.000138**
des	256	4092	2650	0.738	2.148	1.703	16070.07	**3.038316**
frg2	143	1120	611	0.347	0.998	1.104	4889.96	**0.039363**
pair	173	1474	1198	0.385	0.743	1.065	6449.99	**0.387360**
x3	135	811	512	0.258	0.666	0.306	3739.96	**0.009839**

In Table 3.3, column 1 lists the benchmark circuit under consideration, and column 2 provides the gate count. Column 3 captures the accuracy of the method when compared with *MC* simulations for different gate error probabilities, while Column 4 highlights the significant time savings the proposed algorithms achieved when compared with MC simulations.

From the table, it is clear that the proposed algorithm maintains accuracy within $3\ \%$ of the *MC* simulations while significantly speeding up on the computation process. In addition, it is definite that the *MC* simulation runtime directly depends on the number of *PI*s and the gate count. Hence, *des* with the largest number of *PI*s has the highest runtime. On the Contrary, the *CPEP* algorithm is more independent from the number of gates and the *PI* count while more dependent on the complexity (number of *RFO* nodes) of the circuits.

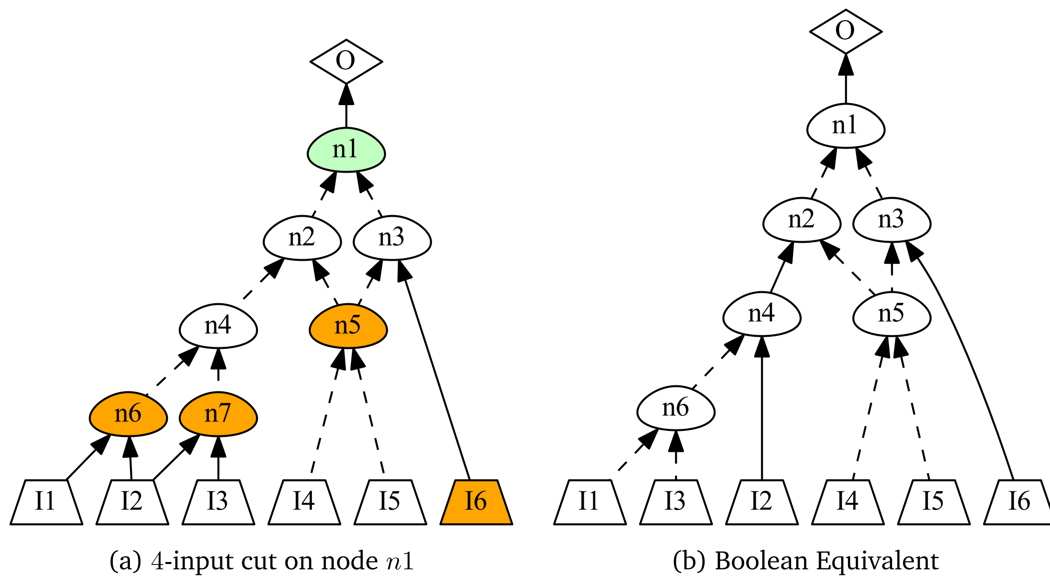
3.4 Synthesis for Reliability Enhancement based on Cut-rewriting Technique

A fast and accurate algorithm to compute the *Soft Error Rate* (SER) and the reliability of the circuit output is presented in the previous sections. In this section, a novel reliability-driven circuit optimisation and synthesis flow based on the standard *cut-rewriting* technique to improve the reliability of circuits built out of unreliable gates will be introduced. Again, the *AIG* representation is utilized to represent the logic circuits.

Firstly, some common terminologies encountered in the logic synthesis domain will be explained as follows.

Cut: A cut of a node A in a network is a set of nodes C such that any path from a *PI* to A must pass through at least one node in the set C [88]. Obviously, node A itself forms a trivial cut. The size of a cut refers to the number of leaves (i.e., the nodes in the cut set C), rather than the number of interior nodes. Cuts of up to a certain size k are called *k-feasible cuts* and particularly cuts with

size k are called *k-cuts*. An example is depicted in Figure 3.3a. The node-set $\{I6, n6, n7, n5\}$ defines a *4-cut* C on the node $n1$. C is a *4-cut* since it has four leaves: $n5$, $n6$, $n7$, and $I6$. Clearly, every path from a *PI* to $n1$ passes through at least one of them. Nodes $n2$, $n3$, and $n4$ are the internal nodes of the cut.



(a) 4-input cut on node $n1$ (b) Boolean Equivalent

Figure 3.3: Cut example and locally equivalent circuit

***Negation-Permutation-Negation* (NPN) equivalence:** Two Boolean functions, F and G, are *NPN* equivalent, i.e., belong to the same *NPN* equivalence class, if F can be transformed into G through the *negation of inputs* (N), the *permutation of inputs* (P), and *negation of the output* (N) [89]. For example, the sub-circuit comprising nodes $\{n6, n4, n2\}$ in Figure 3.3a and Figure 3.3b, evaluating the function G and F, are *NPN* equivalent. G can be transformed to F by employing the following procedure: (i) negate $I3$ and $I1$, (ii) swap the position of $I3$ and $I2$, and (iii) negate the output node.

Boolean matching: Boolean matching, a technique widely used in technology mapping [90], is the process of replacing a subgraph with another functionally equivalent subgraph. It is normally done by calculating the canonical form representation of functions by employing simple methods like negation and swapping of the nodes.

3.4.1 Reliability Optimisation: the Algorithm Implementation

In cut based rewriting, a library of a set of pre-computed best circuits (*cut forest*), which have the equivalent functions from a subset of all *NPN classes* of *4-cuts*, has to be created and saved first. Then, when rewriting a circuit, all possible cuts will be loaded from the library rather than being computed every time.

In this work, the basic pre-computed forest consists of 2004 nodes implementing 106 *NPN* function classes, while the *ABC*'s original area-based rewriting uses a forest of 1785 nodes implementing 139 function classes. In the *ABC*'s default rewriting algorithm for area optimisation, *XOR* nodes are also included. These *XOR* nodes are excluded from this reliability-driven rewriting algorithm since the *XOR* nodes can result in significant output error. To perform a cut rewriting, *cut-enumeration* and *Boolean matching* are performed to enumerate and identify the cuts rooted at a certain node. Then functionally equivalent alternative cuts from the pre-computed forest are evaluated as replacements for that node.

The main algorithm implemented in this work is called *Rewrite for Reliability* or *RWREL* for its command in *ABC*. This algorithm is based on the standard cut rewriting algorithm from *ABC*, adapted so that the goal is reliability instead of area. Modified versions of the data structures were created to include node error probability values. The key functions of the rewriting process were then rewritten to use reliability as a goal. All utility functions from the rewriting module were duplicated, just changing them to work with the expanded structures instead of the originals. The cut enumeration and decomposition modules are self-contained and are used directly. Equation 3.14 quantifies the "percent decrease in error" goal function that is used in guiding the optimisation algorithm to select the better cut.

$$R_{metric} = \frac{P_\epsilon(old_cut) - P_\epsilon(new_cut)}{P_\epsilon(old_cut)} \tag{3.14}$$

The algorithm is presented in Algorithm 3.

Algorithm 3 Cut-based Reliability Optimisation

Require: P_{ntk}, $Network$, PI_{sp}, G_{err} **return** PO_{err}

1: **for** each $node$ in $Network$, in topological order **do**
2: $cone \leftarrow cpySubCircuit(Network, node)$
3: **for** each 4-input cut u of $node$ in $cone$ **do**
4: $F \leftarrow getTT(u)$
5: **for** each possible equivalent S of F **do**
6: **if** getLevel(S)>MaxLevel **then**
7: Continue
8: **end if**
9: $replaceNodes(cone, u, S)$
10: $C \leftarrow computCost(cone)$
11: **if** $C < bestC$ **then**
12: $S_{best} = S$
13: **end if**
14: **end for**
15: Apply S_{best} to the $Network$
16: **end for**
17: $PO_{err} \leftarrow computeEP(Network, PI_{sp}, G_{err})$
18: **end for**

The aim is to find a sequence of transformations leading to an *AIG* network that minimizes the cost function. Specifically, a heuristic approach will be used to find an acceptable solution, i.e., an *AIG* providing higher reliability than the original circuit. The strategy chosen is to figure out all the functionality equivalent cuts for a given node and choose the best among them. Given a combinational circuit implementing the Boolean function and its *AIG* network, the algorithm, starting from a *PO*, traverses through the graph in topological order to see if any of the transformations are applicable to the given node. For every possible transformation, the new circuit reliability value is computed using the *CPEP* algorithm described in the previous sections. The configuration that yields the highest circuit reliability improvement is chosen, and the new topology is generated. This process is continued on every graph node until reaching the *PI*s where no more transformations are applicable.

3.4.2 Experimental Results

The total savings achieved in terms of circuit reliability is studied by performing a set of experiments using *MCNC* benchmark circuits [91] with more than 3000 node *AIG*s after *structural hashing* (strash). A unit gate error probability of 10^{-2} is assumed in all our simulations. The networks were loaded in *Berkeley Logic Interchange Format* (BLIF) and *strashed* to *AIG*s. Functional equivalence was extensively verified during development using the *ABC* equivalence check command *cec*. Then, map the *AIG* network to a gate-level *Verilog* netlist with the *TSMC 65 GP CMOS* standard cell library. *Synopsys Design Compiler* is employed for power, timing and area estimation on the two netlists. After established the algorithm correctness and performance, its effect on the implementation of *8051 microcontroller* basic blocks like adder, multiplier, and divider was also tested.

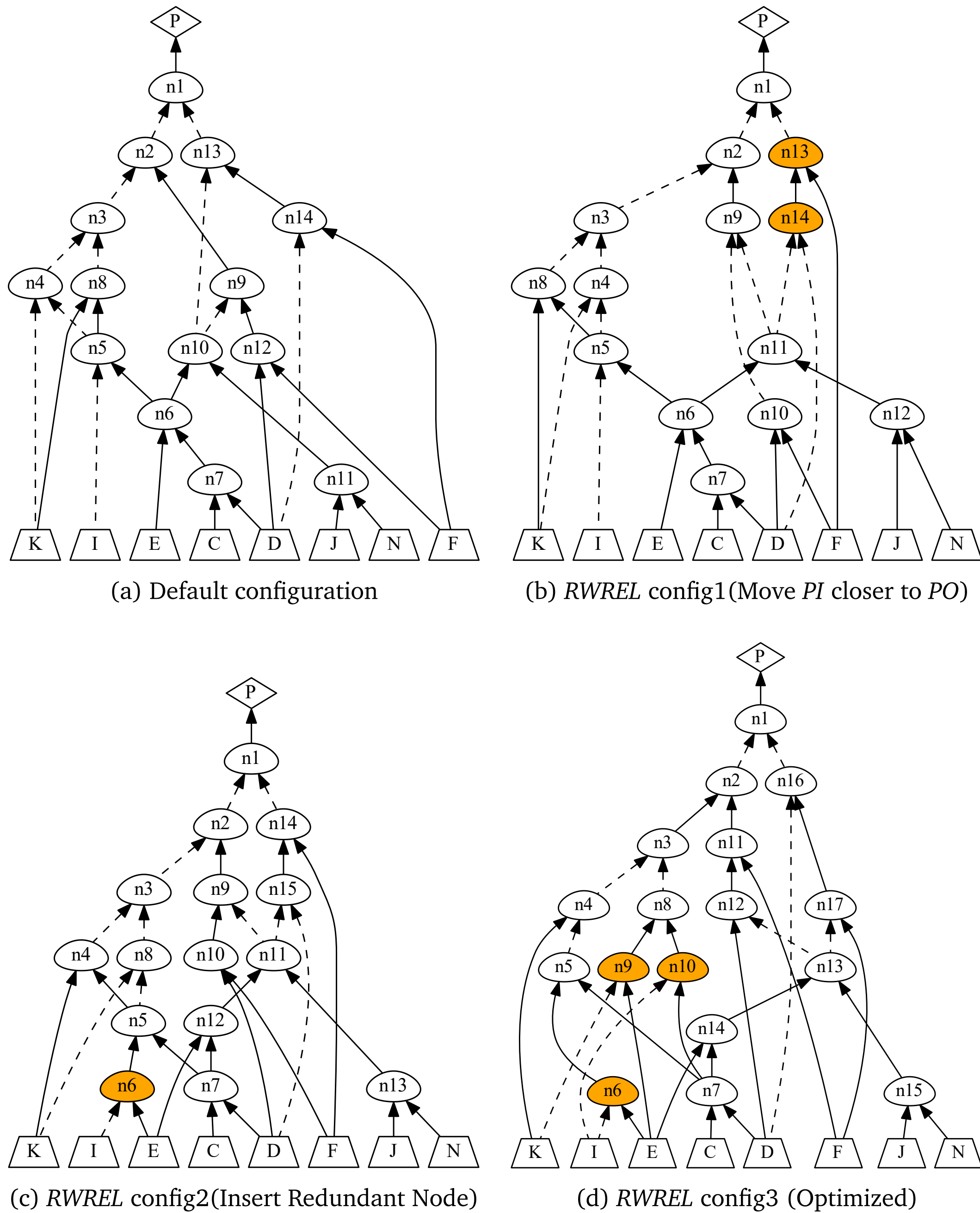
In the rest of the section, a case study will be presented highlighting the important differences in the way the Boolean matcher finds alternative cuts when reliability is the constraint. Then, the results obtained for various *MCNC* benchmark circuits and *8051* functional units will be discussed.

3.4.2.1 *cm162a* – A Case Study

The proposed reliability-aware synthesis algorithm is applied on the *MCNC* benchmark circuit *cm162a*, which implements a 14-input, 5-output logic function. The current case study considers the cone comprising of all the gates driving the output pin P (an 8-input sub-circuit).

Figure 3.4a represents the default *AIG* configuration. The total number of nodes in the default *AIG* representation of the circuit is 14. With a unit gate error probability of 10^{-2}, the output node P has an error probability $EP_{org} = 0.071750$.

Now, tree specific functionally equivalent logic configurations generated by the RWREL tool will be discussed. The idea is to emphasize how the Boolean matcher works when the constraint is set to the output error optimisation in-



(a) Default configuration

(b) *RWREL* config1(Move *PI* closer to *PO*)

(c) *RWREL* config2(Insert Redundant Node)

(d) *RWREL* config3 (Optimized)

Figure 3.4: A case study: *MCNC* benchmark *cm162a*

stead of the area.

Figure 3.4b depicts the first sub-case: Moving the *PIs* closer to the *POs* wherever possible. It is clear that the *PI* node F has been pushed closer to the *PO*. This helped in reducing the overall output error on the output node to $EP_{rwrel1} = 0.068002$.

Figure 3.4c depicts the second sub-case: intelligently insert redundant nodes. Extra nodes, if added pragmatically, can result in a significant error masking effect that reduces the overall error on the output nodes considerably. A redundant node $n6$ has been appended onto the logic. Thus total node count is increased to 15 and reduces the overall output error to $EP_{rwrel1} = 0.067668$.

After multiple iterations, the final version of the optimized circuit configuration in Figure 3.4d is obtained. The final output error value of the optimized circuit is $EP_{opt} = 0.063353$, and the total node count is 17. Two redundant nodes $n9$ and $n10$ are added into the configuration. Finally, some of the important differences is listed below, observed in the way Boolean matching is performed by our reliability-aware rewriting algorithm when compared to the area-driven approach in *ABC*:

- Cuts with *PI*s closer to the *PO*s are preferred. *PI*s have lower *EP* when compared to the internal nodes. Thus, moving a highly reliable node closer to the output can result in cancelling out some of the errors;
- Redundant Node insertion is commonly performed. This increases the total area but has a higher masking effect and thereby reducing the output node error probability;
- One other significant difference that has been observed is the matching of path lengths. Most of the time, the Boolean matcher selects the cuts that result in the circuit that is more balanced so that the depths of both of the *LX* and *RX* paths are similar.

3.4.2.2 Evaluation of Benchmark Circuits

More simulation results comparing the average circuit output errors of the original and the optimized configuration obtained from our tool are reported in Table 3.4. The first column presents the benchmark circuit name, while the second column provides the output pin count of each circuit. The following three columns list the node count of the original circuit, the optimized circuit, and the

optimisation induced node count change in percentage, respectively. Columns 6 (7) provides the circuit depth of the default (optimized) circuits. Columns 8 (9) lists the average output error value for the default (optimized) circuits. Columns 10 through 12 list the average, maximum, and the minimum relative output error improvement.

Table 3.5 reports the area, timing, and power analysis after applying *RWREL* on *MCNC* benchmark circuits. In the table, P_{org}, T_{org} and A_{org} are the power, timing, and area results from the original *MCNC* netlist. P_{opt}, T_{opt} and A_{opt} are the power, timing, and area results from the netlist obtained via *RWREL* optimisation.

From the tables, it is clear that significant *SER* reduction can be achieved at a very low area overhead of $6.57\ \%$ that results in $13.52\ \%$ higher power consumption by employing the optimisation algorithms reported in this paper. An average improvement of $14\ \%$ and a peak improvement of up to $75.5\ \%$ was observed. *I7* benchmark is a simple logic circuit consisting of 199 *PI*s, 67 *PO*s, and a total node count of 903. Output pin $V266(6)$ provides a peak error improvement of $75.5\ \%$. Another important fact worth mentioning is that benchmark circuits *I6*, *I7*, *I8*, and *too_large* report reduction in node count. This proves that our algorithm can decide when to add redundancy and when to remove unnecessary nodes that can result in better output reliability. Also, the benchmark circuit *too_large* reports a pretty high output error. This is because it has 824 nodes and just 3 output pins. It implies that if a large number of gates are placed within the cone to emulate a particular function, the *EP* on the output will be higher. Also, the circuit depth of the logic resembles no direct relation to the output error as circuit depth has reduced in certain cases, while in others, it has been increased.

Table 3.4: *RWREL* performance evaluation on different benchmark circuits (gate error $\epsilon = 0.001$)

Benchmark	N Outputs	Node Count			Circuit Depth		Averaged Absolute Error		Relative Improvement (%)		
		N_{org}	N_{opt}	Inc(%)	D_{org}	D_{opt}	E_{org}	E_{opt}	Avg	Max	Min
apex7	36	221	252	14.03	14	14	0.005895	0.005240	10.44	51.33	1.74
cm162a	5	36	44	22.22	9	8	0.005874	0.005128	13.88	15.66	9.74
comp	3	107	119	11.21	18	19	0.009510	0.007232	18.31	25.31	5.98
dalu	16	1371	1602	16.85	35	35	0.019144	0.016449	13.98	19.45	7.98
I6	67	692	523	-24.42	5	4	0.008381	0.007291	12.83	15.19	4.23
I7	67	903	702	-22.26	6	5	0.009235	0.008303	10.21	75.5	3.16
I8	81	3310	2187	-33.93	21	20	0.018626	0.015219	18.83	32.04	6.8
k2	45	1998	2152	7.71	23	19	0.031802	0.028530	12.94	25.65	4.52
unreg	16	112	111	-0.89	5	5	0.006506	0.005382	18.43	19.03	17.78
vda	39	924	1042	12.77	16	18	0.030577	0.027491	12.81	65.6	5.31
too_large	3	824	773	-6.19	30	27	0.053492	0.046468	13.87	17.43	11.33
8051_add	19	175	222	26.86	28	29	0.014149	0.013330	16.99	41.56	5.86
8051_mul	17	630	765	21.43	48	47	0.030946	0.029475	10.58	32.65	1.58
8051_div	17	1010	1181	16.93	198	181	0.023831	0.022888	12.31	31.75	2.18

Table 3.5: Area, delay and power analyses

Benchmark	Area			Delay			Dynamic Power			Leakage Power		
	A_{org}	A_{opt}	Inc(%)	T_{org}	T_{opt}	Inc(%)	P_{org}	P_{opt}	Inc(%)	P_{org}	P_{opt}	Inc(%)
apex7	705.24	779.04	10.46	1.10	1.16	5.45	46.25	53.32	15.28	3.12	3.47	11.28
cm162a	121.68	137.88	13.31	0.69	0.56	-18.84	7.91	8.97	13.47	0.54	0.61	13.29
comp	360.36	405.72	12.59	1.36	1.38	1.47	34.22	36.11	5.54	1.61	1.79	11.46
dalu	4232.52	4876.2	15.21	3.41	3.15	-7.62	310.43	403.97	30.13	18.66	21.93	17.52
I6	1861.92	1677.6	-9.9	1.37	0.89	-35.04	133.68	137.11	2.57	8.28	7.62	-7.97
I7	2386.8	2144.16	-10.17	1.48	1.38	-6.76	174.82	175.92	0.63	10.66	9.65	-9.47
I8	8147.52	5780.16	-29.06	4.59	3.11	-32.24	720.15	522.14	-27.5	37.7	26.66	-29.29
k2	4675.32	5343.84	14.3	1.96	1.87	-4.59	119.51	177.1	48.19	23.53	26.07	10.79
unreg	367.2	372.6	1.47	0.48	0.52	8.33	29.36	32.16	9.53	1.64	1.71	4.34
vda	2158.92	2638.08	22.19	1.08	1.31	21.3	88.61	129.34	45.97	11.18	13.09	17.08
too_large	2176.2	2087.64	-4.07	2.32	2.20	-5.17	118.6	121.57	2.5	9.45	9.16	-3.08
8051_add	595.08	723.6	21.6	2.49	2.60	4.42	74.19	84.25	13.55	2.67	3.24	21.34
8051_mul	2038.32	2432.16	19.32	3.93	3.82	-2.8	304.85	363.33	19.18	9.17	10.87	18.51
8051_div	3295.8	3781.44	14.74	16.96	15.43	-9.02	555.68	612.15	10.16	14.95	17.1	14.38

3.5 Chapter Summary

In this chapter, a novel soft error, reliability analysis algorithm *CPEP*, and a reliability enhancement framework for combinational digital *VLSI* circuits were proposed and explained.

CPEP, which stands for conditional probability-based soft error propagation algorithm, can accurately estimate the error probability, hence the reliability of a circuit output within $3\ \%$ of error and improve the speed by up to orders of magnitude, compared to the traditional *MC* simulation-based methodologies.

Although the *CPEP* method currently works only on the combinational circuits, it has the potential to be extended to sequential circuits by the employment of the *Markov Chain*. Hence the *state-based CPEP* can be implemented in the future. Also, more generic gates can be easily implemented to make the method works on more general circuit analysis circumstances.

The reliability-driven synthesis framework based on *cut-rewriting* techniques, inspired by the area-driven synthesis approach used in the *ABC* synthesis tool, makes use of the reliability evaluation results from the *CPEP* and improve the reliability by iteratively replacing the *4-input* cuts with more robust functional equivalent alternative cuts. The obtained results show that up to $75.5\ \%$ reliability improvement has been obtained with the acceptable area and power consumption overhead.

This chapter is based on the following papers:

- Bo Yang, Satish Grandhi, Christian Spagnol, Emanuel Popovici, Sorin Cotofana, "An approach for digital circuit error/reliability propagation analysis based on conditional probability," 27th IEEE Irish Signals and Systems Conference (ISSC), 2016, pp. 1-6
- Satish Grandhi, Bo Yang, Christian Spagnol, Emanuel Popovici, "An EDA framework for reliability estimation and optimization of combinational circuits," Journal of Low Power Electronics 12(3), 2016, pp. 242-258

Chapter 4

Voltage Scalable *SSTA* Gate Modelling Techniques

In the previous chapter, the soft error related reliability at the gate level has been discussed. To make power consumption and propagation delay optimisation possible, the timing reliability related to the device variabilities under various operating voltage (V_{dd}), load, and other environmental conditions has to be taken into consideration. Therefore, in this chapter, V_{dd} dependent *Statistical Static Timing Analysis* (SSTA) gate models will be presented. In the models, parameters are categorized into *device process parameters* and *circuit setup parameters*. As will be stated in Section 4.1, V_{th0} and L are two of the most interesting variation factors impacting the propagation delay. Thus, they will be the main process variations that our models target on. In terms of circuit setup parameters, the operating voltage V_{dd}, the effect of both current gate fanout and driver gate fanout, and the driver gate strength will be taken into consideration as well.

An initial approach is a pure analytical propagation delay estimation model based on a set of mathematical formulas. The approach works with only one process variation, V_{th0}, as it is impractical to derive the closed-form expressions of the τ_{pd} for more complex circumstances. However, the principle lays a foun-

dation for the improved *ANN-based* model. To eliminate these limitations, an improved model making use of the *ANN* function approximator is proposed on the foundation of the initial approach, which takes more factors into account.

This chapter will be organized as follows. Firstly, in Section 4.1, the general concepts and considerations of *CMOS* process variations and their impacts on timing will be introduced. Secondly, the conventional *SPICE Monte-Carlo* simulation-based *SSTA* approach will be presented in Section 4.2. Then, in Section 4.3 and Section 4.4, the proposed two new *SSTA* gate models, i.e., the simple analytical model and the *ANN-SSTA* models, will be described, respectively. Finally, the comparison and applications of the approaches will be summarized and discussed in Section 4.5.

4.1 Process Variations and their Impacts on Timing Analysis

SSTA methodologies are developed to predict propagation delay deviations due to the process parameter value fluctuations to the design parameters in the specification. To deal with the uncertainties, the parameter variations would commonly be modelled as *Random Variables* (RVs) following certain distributions, and then the delay distribution, P_{τ}, will be profiled accordingly either analytically or by simulation. Thus, in this section, the common process variation sources will be studied first.

The variations originate from several sources, including process variations due to manufacturing imperfections, environmental variations such as temperature and power supply variations, modelling variations due to limited accuracy of the device models, and other physical effects that cause process parameter deviation with device ageing [92].

For convenience, in most literature, process variations are commonly classified into two categories: *Global variation* and *Local variation* in terms of the level of

impact, as shown in Figure 4.1. Note that in the figure, the dashed-line arrows indicate the variations across it.

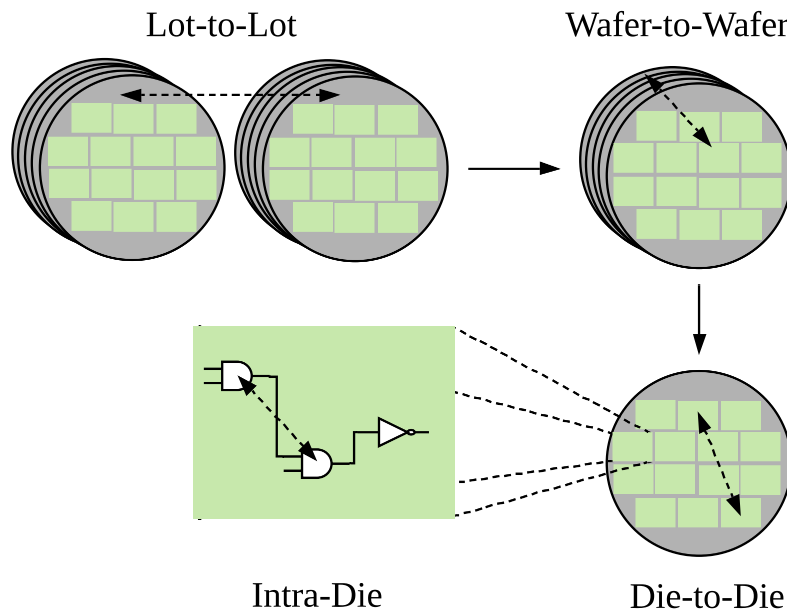


Figure 4.1: The variation source classifications

1. *Global variations:* Also referred to as "inter-die" or chip-level variations. The variations affect the performance identically across lot to lot, wafer to wafer or die to die, which can be modelled by a single *RV* (i.e., give a single random value to all parameters in a simulation sample). Global variations include the lot-to-lot, wafer-to-wafer, and die-to-die variations, as shown in Figure 4.1.

2. *Local variations:* Also called "intra-die" or transistor level variations. The variations affect the performance differently from device to device within a die or even a circuit, as the intra-die variation shown in Figure 4.1. Thus in the worst case, an extra *RV* has to be introduced for each transistor within a die. In addition to potentially higher *RV* dimensionality, what makes dealing with local variations more complicated is the existence of correlations either spatially or between parameters. Therefore, in terms of the existence of correlations, the local variation can be further sub-classed into *random variations* and *correlated variations*.

- *Random variations:* This is a term that may cause confusion, but by convention, in the context of *CMOS* process variations, the random variation is specifically referred to as the independent local variations.
- *Correlated variations:* Apart from the topological correlations induced by the *Re-convergent Fanouts* (RFOs), *Spatial Correlations*, and *dependencies between parameters* are the two correlation types of most importance.

Note that the global variation can be treated as a variation that is perfectly correlated for all devices at a certain level of impact. Also, some variations may have components of some types of the above variation classes simultaneously, for example, the V_{th0} and the gate-length variations have both global components and local components. Especially interesting, the V_{th0} has an intra-die component, such as channel doping concentration variation, which is independent of the gate-length variation, and also the global and correlated local components due to the gate length variations [92, 93].

Now that the process variations have been clearly categorized, the variance of a parameter can be written as Eq. 4.1

$$\Delta X = \Delta X_g + \Delta X_{lrandom} + \Delta X_{lcorr}, \tag{4.1}$$

where ΔX represents the total variance, ΔX_g denotes the *Global variation*, $\Delta X_{lrandom}$ is the *Random local variation*, and ΔX_{lcorr} is the *Correlated local variation*.

In different situations and circumstances, the dominant variations are different, as well. As the progressively scaling of the *CMOS* devices, the local variations are more and more critical components to be considered, while global variation is the more dominant one in previous lower technology nodes. Also, as the technology nodes progressing, the layout density of devices is growing as well, which makes the parameters more tightly spatially correlated. It is also

noticed that the global variations affect more on the mean of the final delay distributions, while the local variations contribute more to the variance of the distributions [92].

The increasing criticality of local variations makes the development of efficient *SSTA* gate models more critical, as the simulation-based *SSTA* methods such as *SPICE MC* simulation-based *SSTA* becomes more expensive when dealing with local variations, while the *corner-based STA* completely loses its prediction power [26].

4.2 *SPICE* Monte-Carlo based *SSTA*

The *SPICE Monte-Carlo* (MC) simulation is one of the standard methodologies for *SSTA*, which is treated as the reference approach in most of the literature, and the results are often treated as "golden truth" for *SSTA*, due to its high accuracy and theoretically universal modelling capability. Thus, the *SPICE MC-based SSTA* will be introduced in this section before the proposed models are presented.

SPICE, which stands for *Simulation Program with Integrated Circuit Emphasis*, is essentially a electrical simulator that simulates the physical behaviours of devices and circuits [94]. As a result, high estimation precision can be obtained as long as detailed device models are being used. On the other hand, as large numbers of equations need to be solved analytically or even numerically during each simulation, the runtime could be too long to be acceptable.

The process of the *MC-based* method is similar to the area estimation example described in Section 2.4 and is depicted in Figure 4.2.

The *MOSFET* models and the corresponding variation models are commonly included in a technology *Process Design Kit* (PDK) provided by the manufacturer. The *SPICE* netlist contains the simulation setup, circuit under simulation, and the interested performance measurements such as τ_{pd}, t_{rise} and t_{fall}. When the *MC* simulation starts, samples of parameter variances will be randomly gener-

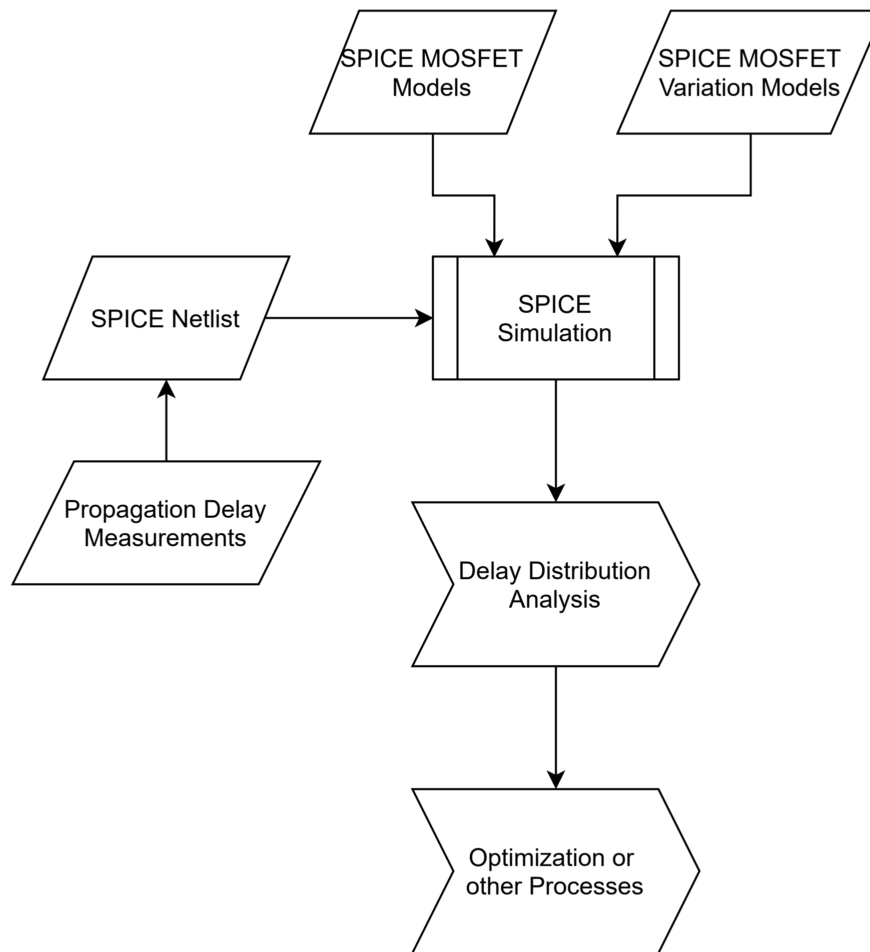


Figure 4.2: *MC-based SSTA* basic workflow

ated according to the *MOSFET* variation models. Once the simulation is completed, analyses on the resulting performance metrics can be performed, for instance, a histogram analysis.

HSPICE is a *Synopsys* version of *SPICE* simulator and is commonly used as a default simulation tool in the industries. In addition to the *SPICE* simulation engine, *HSPICE* provides many other features that make simulation and analyses more powerful. *Variation Block* is one of the latest features enabling flexible variation modelling and is used in all our *MC* simulations. A more detailed introduction of the *HSPICE Variation Block* is provided in Appendix C.

The *MC-based SSTA* flow is flexible and accurate and is a preferable *SSTA* method until simulating the large scale circuit becomes a requirement, although some methodologies were proposed in order to (partially) overcome the scala-

bility limitations of the *MC SSTA*. For example, [95, 96] introduced a *stratification+hybrid quasi Monte-Carlo* (SH-QMC) approach to reduce the *MC* sample size. Another stratification based method that selects the process variations according to their importance was proposed in [97]. In [98], *optimal low L2-discrepancy QMC* samples are employed to improve the precision as well as performance. Also, the authors of [99] improved the *MC SSTA* efficiency from another direction, where they handle the delays as vectors to eliminate the simulation iterations.

4.3 An Analytical Analysis of V_{dd} Dependent τ Estimation

In this section, an initial simplified analytical analysis of V_{dd} dependent propagation delay estimation model will be performed, which provides insight of how the propagation delay, τ_{pd}, responds to the process parameter V_{th} and the circuit setup parameter V_{dd}, although it may not fit for more complex analyses when more variation sources such as *channel length* L and circuit setup parameters such as load conditions are taken into consideration.

Some analytical delay estimation techniques have been developed in, e.g., [100, 101] and [102]. These models can accurately approximate the transient response of the *CMOS* gates and hence to estimate the propagation delay. However, none of them can model gate delay as a function of V_{dd} while taking V_{th} variation into account. In addition, [100, 101] can work only with a certain range of load capacitance and input slope rate values. But in reality, the load capacitances and input slope rates are varying as the delay being propagated through the circuit. Thus, the accuracy of delay propagation is limited in these models. [102] introduced a number of expensive numerical computations, which increases the computation complexity.

To illustrate the model, the output falling transition of an *Inverter* gate at $V_{dd} = 0.9V$ is taken as an example, and the *PTM 32 nm HP* technology node is used in

the *SPICE* simulation. Some of the naming conventions in the rest of the section are described as follows:

- t_r represents the rising time, and t_f the falling time.
- $t_{V_{th}} = \frac{V_{th}}{V_{dd}} \times t_r$ is the time when the output reaches the threshold voltage V_{th}.
- t_{50} is the switching threshold time (transition time at 50% of the output voltage).
- V_{max} is the maximum output voltage during the overshoot.
- V_{tr} represents the output voltage at t_r.

As it can be observed in Figure 4.3, the whole transition process can be divided into four regions *I*, *II*, *III*, and *IV* with boundary conditions as follows:

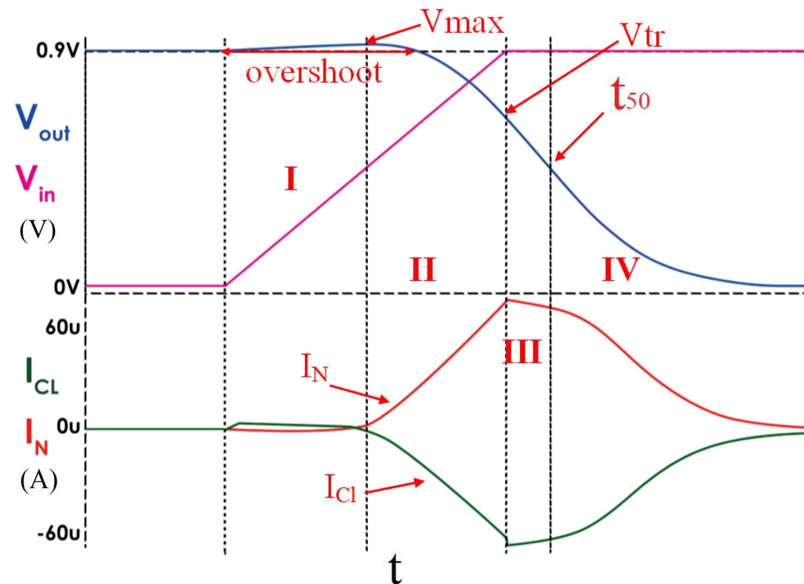


Figure 4.3: The falling transition of an *Inverter*

- Region *I*, $t_0 < t < t_{V_{th}} = \frac{V_{th}}{V_{dd}} \times t_r$. In this region, the overshoot occurs.
- Region *II*, $t_{V_{th}} < t < t_r$, I_{ds} is a function of both V_{ds} and V_{gs}.
- Region *III*, $t_r < t < t_{50}$, as $V_{gs} = V_{dd}$, I_{ds} is a function of the *NMOS* V_{ds} only, which can be approximated as a linear function with the coefficient

k_a as shown in Equation 4.2 [100].

- Region *IV*, $t > t_{50}$.

$$k_a = \frac{I_{ds0} - I_{ds1}}{V_{tr} - V_{50}}, \tag{4.2}$$

where I_{ds0} is the I_{ds} at $V_{out} = V_{tr}$, and I_{ds1} is the I_{ds} at $V_{out} = V_{50}$ (50% of V_{dd}).

The corresponding *NMOS* operation regions can be identified in the I_{ds}-V_{ds} plot, as shown in Figure 4.4.

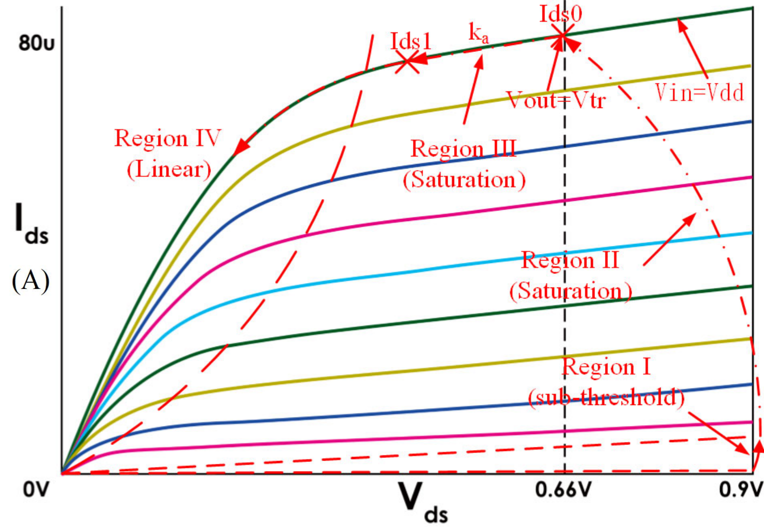


Figure 4.4: Operating regions of *NMOS* in a falling transition

4.3.1 Drain to Source Current Model

To accurately estimate the propagation delay at a given V_{dd}, the drain-source current has to be accurately modelled. The *EKV* model [103] is a *trans-regional* drain-source current model, which is shown in Equation 4.3:

$$I_{ds} = I_s \times \ln^2 \left[1 + \exp\left(\frac{V_{gs} - V_{th}}{2nU_T}\right)\right], \tag{4.3}$$

where $I_s = 2n\mu\frac{W}{L}C_{ox}U_T^2$ is the specific current, n is the weak inversion slope factor, and U_T is the thermal voltage.

The equation implies that I_{ds} is an exponential function at the deep sub-threshold regime and a quadratic function at a strong inversion regime. However, this model does not take mobility degradation and velocity saturation effects, which suppresses the quadratic relationship to an α power (typically $\alpha < 2$) relationship [104], into consideration. Thus, it is less precise to estimate I_{ds} in deep sub-micron technologies. To improve this model, the α-power-law fitting approach is integrated into the *EKV* model in this work:

$$I_{ds}(V_{gs}, V_{ds}) = I_s \times \ln^{\alpha}\left[1 + \exp\left(\frac{V_{gs} - V_{th}}{2nU_T}\right)\right], \tag{4.4}$$

In the above equation, I_s and $k = \frac{1}{2nU_T}$ can be treated as constants that do not depend on V_{dd} and V_{th}, thereby I_s, k, and α are treated as fitting constants. V_{th} is modelled as a linear function of V_{ds} due to *Drain-Induced Barrier Lowering* (DIBL), *Channel Length Modulation* and *body effects* [105, 106].

Due to the *DIBL* effect [107], V_{th} cannot be directly extracted from the *BSIM* parameter V_{th0}, which is called the *long channel threshold voltage*. However, the threshold voltage V_{th} can be modelled as a linear function against V_{ds} and can be measured using *SPICE* simulation, as shown in Figure 4.5.

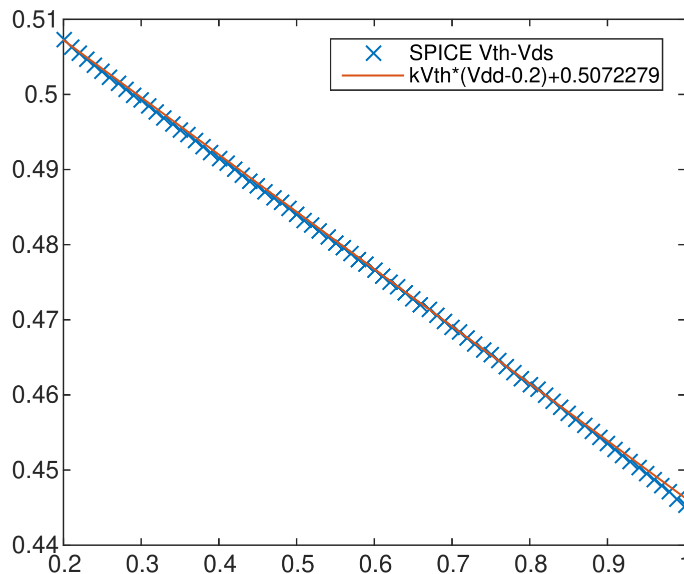


Figure 4.5: V_{th} vs. V_{ds} linear fitting

The V_{th} in the figure is extracted by measuring the constant current threshold voltage [108] at different V_{ds} values. Then V_{th} can be expressed as:

$$V_{th} = k_{V_{th}} \times (V_{ds} - V_{dsref}) + V_{thref}, \tag{4.5}$$

where $k_{V_{th}}$ is the coefficient fitted from the V_{th}-V_{ds} curve in Figure 4.5 and (V_{dsref}, V_{thref}) can be any point on the straight line in Figure 4.5.

By substituting the fitted parameters into Equations 4.4 and 4.5, we can get the I_{ds}-V_{gs} curves at different V_{ds} values, as presented in Figure 4.6.

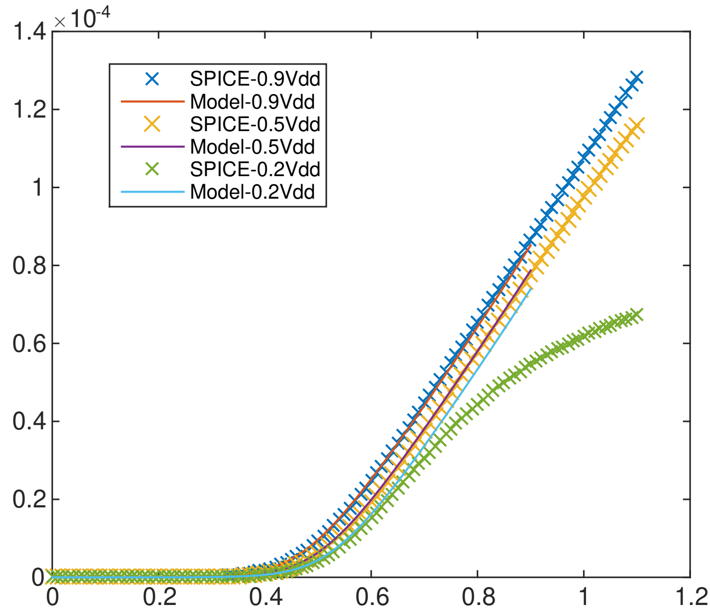


Figure 4.6: I_{ds} vs. V_{gs} curve fitting @$V_{ds} = 0.9, 0.5, 0.2$ V

The estimation results are in a very good match with *SPICE* results at $V_{ds} = 0.5$ V to $V_{ds} = 0.9$ V and when $V_{gs} < 0.6$ V at $V_{ds} = 0.2$ V, which ensures good accuracy in the saturation region while the poor matching happens in the linear region.

4.3.2 Consideration of Threshold Voltage Variation

As stated in [106], the variation of the threshold voltage is equivalent to the V_{th0} variation, therefore, by substituting the effective V_{gs} and V_{ds} with ΔV_{th0}

into Equation (4.4), we obtain:

$$I_{ds}(V_{gs}, V_{ds}, V_{th0}) = I_{ds}(V_{gs} + \Delta V_{th0}, V_{ds} + \Delta V_{th0}), \tag{4.6}$$

where $\Delta V_{th0} = V_{th0}(nominal) - V_{th0}$.

Figure 4.7 presents the I_{ds} estimation results when V_{th0} varies by $+/-10\ \%$, respectively.

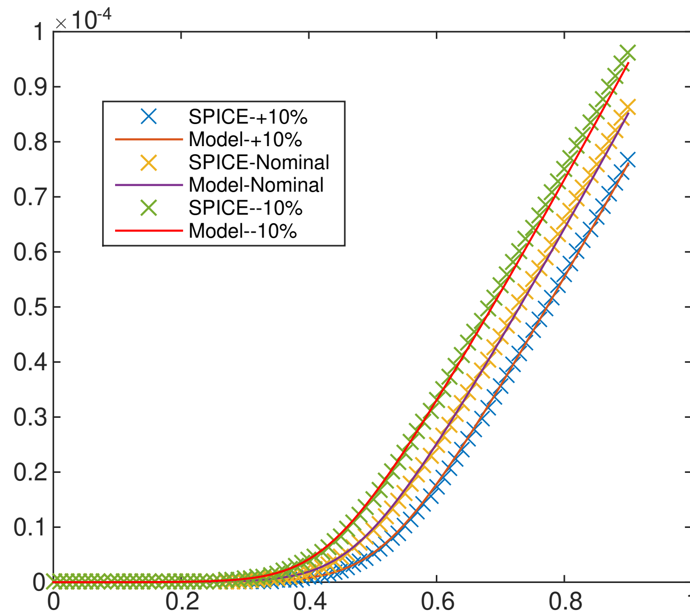


Figure 4.7: I_{ds} vs. V_{gs} with $+/-10\ \%\ V_{th0}$ variation

When compared with *SPICE* simulation, the estimation provides highly accurate results with $error_{(avg)} < 2\ \%$.

4.3.3 Propagation Delay Estimation

The overshoot effect results from the transistor gate-to-drain capacitance C_m [100, 101] indicated in Figure 4.8, where according to *Kirchhoff's Current Law*:

$$I_P + I_{C_m} = I_{Cl} + I_N, \tag{4.7}$$

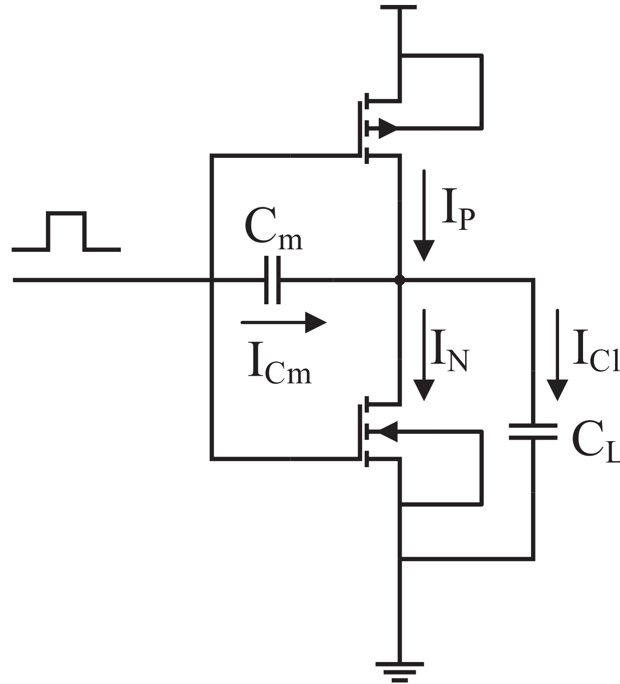


Figure 4.8: *Inverter* circuit topology

Equation (4.7) is used to calculate the V_{out} in different regions. The effect of *PMOS* inverse linear current is ignored as the V_{ds} is small all along the inverse linear region of *PMOS*. Thus, the I_P term can be omitted during the calculation.

The derivation of the propagation delay τ_{pd} is done by solving a set of *Ordinary Differential Equations* (ODEs) under each region.

Region I: In this region, the *NMOS* transistor is in the cut-off region and the *PMOS* transistor is in the inverse linear region, which means that the *NMOS* is discharging the load capacitance (C_L) with its sub-threshold current and *PMOS* is also helping to discharge the C_L [102]. At the same time, V_{in} is charging C_L through C_m. As *PMOS* inverse linear current and *NMOS* sub-threshold current are difficult to estimate, we start by investigating the charging of C_L. The total V_{out} increase can be expressed as Equation (4.8):

$$\Delta V_{out} = \frac{Q_c}{C_L}, \tag{4.8}$$

where Q_C is the integral of I_{Cl} from t_0 to t_{Vth}.

To estimate Q_C, the area under I_{Cl} between t_0 and $t_{V_{th}}$ can be treated as a rectangle, then we get Equation (4.9):

$$Q_C = I_{Cl(avg)} \times t_{V_{th}} (\text{taking } t_0 = 0), \tag{4.9}$$

Practically, $I_{Cl(avg)}$ should be varying with V_{dd} changes. However, as $I_{Cl(avg)}$ variation has a very small impact on Q_C, to keep simple, $I_{Cl(avg)}$ is treated as a constant (is not changing as a function of V_{dd}). Finally, V_{max} can be expressed as Equation (4.10):

$$V_{max} = V_{dd} + \Delta V_{out} \tag{4.10}$$

Region II: The *NMOS* is operating in the saturation region. Thus C_L starts to discharge through the *NMOS*, and meanwhile, V_{in} is continually contributing to charging current through C_m. As I_{ds} is a function of both V_{ds} (V_{out}) and V_{gs} (V_{in}), it is easy to predict that this should be a *first-order linear non-homogeneous differential equation*. However, the $I_{ds}(V_{gs}, V_{ds})$ equation is difficult to be used in this *ODE*, therefore in the case of $V_{gs} \gg V_{th}$, the drain-source current I_{ds} can be further approximated as Equation (4.11):

$$I_{ds}(V_{gs}, V_{ds}) = I_s \times k^{\alpha}(V_{gs} - V_{th}), \tag{4.11}$$

Then substitute Equation (4.11) into Equation (4.7) and re-arrange the equation to get the standard form of the first-order linear non-homogeneous *ODE* in Equation (4.12) [109].

$$\frac{dV_{out}}{dt} + P(t)V_{out} = Q(t), \tag{4.12}$$

$$P(t) = -I_s \times k^{\alpha} \times \frac{k_{V_{th}}}{C} \tag{4.13}$$

$$Q(t) = -\frac{I_s k^{\alpha} V_{dd}}{t_r C} t + \frac{C_m}{C} \times \frac{Vdd}{t_r} - \frac{I_s k^{\alpha}}{C}[\Delta V_{th0} - k_{V_{th}}(\Delta V_{th0} - V_{dsref}) - V_{thref}] \tag{4.14}$$

Solving the *ODE* by applying the initial condition with $t = t_{V_{th}}$, $V_{out} = V_{max}$ will

obtain Equation (4.15):

$$V_{out}(t) = e^{-mt} \times [q\frac{e^{mt}}{m^2}(mt-1) + \frac{n}{m}e^{mt} + V_0], \tag{4.15}$$

$$n = \frac{C_m V_{dd}}{C t_r} - u[\Delta V_{th0} - k_{V_{th}}(\Delta V_{th0} - V_{dsref}) - V_{thref}] \tag{4.16}$$

$$m = -u \times k_{V_{th}}, q = -u \times \frac{V_{dd}}{t_r}, u = -\frac{I_s k^{\alpha}}{C} \tag{4.17}$$

$$V_0 = e^{mt_{V_{th}}} V_{max} - q \times \frac{e^{mt_{V_{th}}}}{m^2}(mt_{V_{th}} - 1) - \frac{n}{m}e^{mt_{V_{th}}} \tag{4.18}$$

Region III: In this region, the *NMOS* is operating in the saturation region, and $V_{in} = V_{dd}$, thus V_{in} does not contribute current anymore. So, Equation (4.19) can be derived from the Equation (4.7).

$$C\frac{dV_{out}}{dt} + k_a V_{out} + I_{ds0} - k_a V_{dd} = 0, \tag{4.19}$$

To solve this first-order linear *ODE*, apply the initial condition with $t = t_r$ and $V_{out} = V_{tr}$, we get Equation (4.20):

$$V_{out}(t) = (V_{tr} - A_1)e^{A_2(t-t_r)} + A_1, \tag{4.20}$$

$$A_1 = V_{dd} - \frac{I_{ds0}}{k_a}, A_2 = -\frac{k_a}{C} \tag{4.21}$$

$$I_{ds0} = I_{ds}(V_{dd}, V_{tr}, V_{th0}) \tag{4.22}$$

$$I_{ds1} = I_{ds}(V_{dd}, \frac{1}{2}V_{dd}, V_{th0}) \tag{4.23}$$

The propagation delay is measured from $50\ \%$ of V_{in} to $50\ \%$ of V_{out}, which can be substituted into Equation (4.20) to get t_{50} as shown in Equation (4.24):

$$t_{50} = \frac{1}{A_2} \times \ln\left(\frac{\frac{1}{2}V_{dd} - A_1}{V_{tr} - A_1}\right) + t_r \tag{4.24}$$

Therefore the complete propagation delay τ_{pd} equation becomes:

$$\tau_{pd} = t_{50} - \frac{1}{2}t_r \tag{4.25}$$

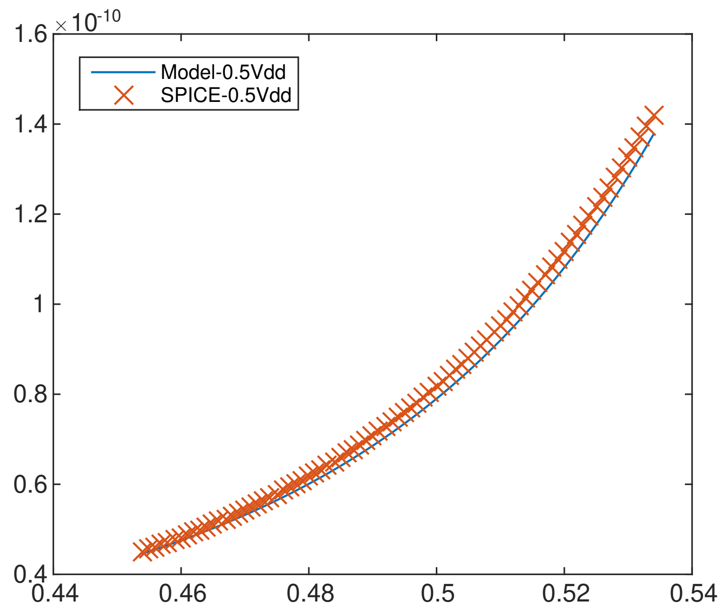
Table 4.1 presents the comparison of the model and *SPICE* simulation results at V_{dd} values from 0.5 V to 0.9 V and $V_{th0(nominal)} = 0.49396$ V, the nominal value under *PTM 32 nm HP* model card.

Table 4.1: Delay estimation results at different voltages with the nominal V_{th0}

$V_{dd}(V)$	*SPICE* (ps)	Model (ps)	Error (%)
0.5	74.83467	72.4337	3.21
0.6	30.57148	30.03257	1.76
0.7	19.86262	19.9565	0.47
0.8	15.54658	15.56202	0.01
0.9	13.17285	13.1904	0.13

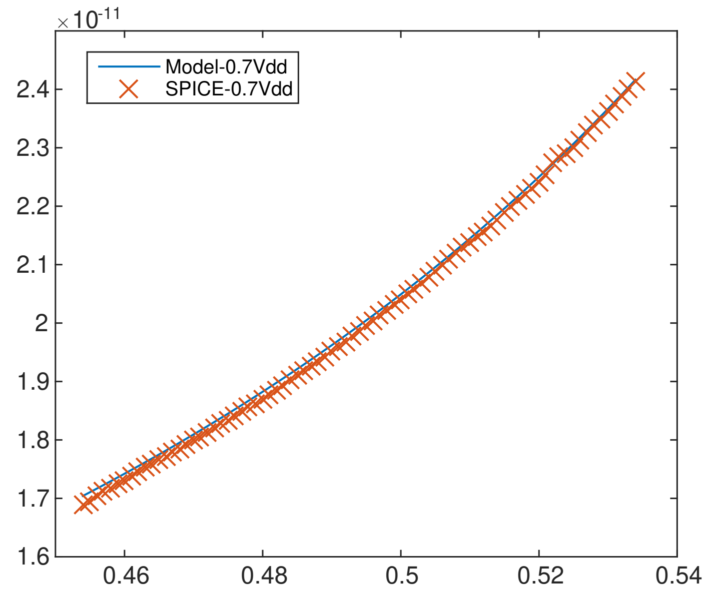
One can observe that the estimates are very accurate over the considered V_{dd} range with a maximum error of 3.3 % at 0.5 V.

Figure 4.9a, 4.9b and 4.9c depict τ_{pd} as a function of threshold V_{th0} at $V_{dd} = 0.5$ V, 0.7 V, and 0.9 V, respectively.

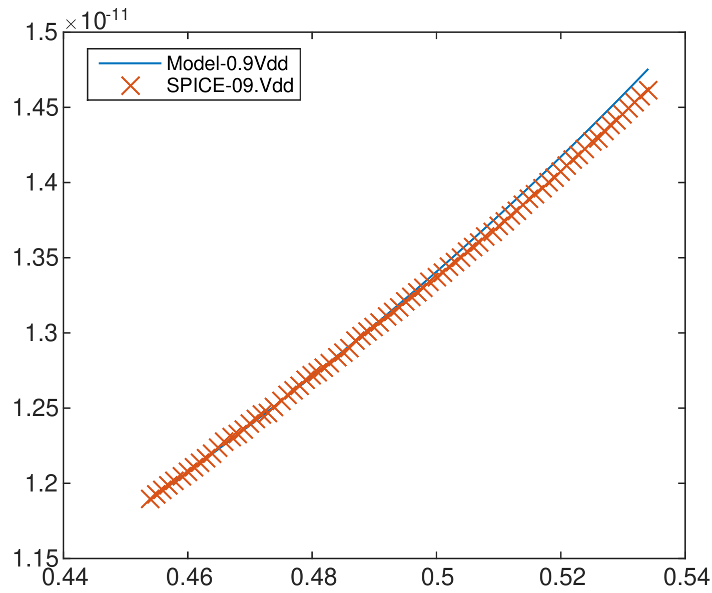


(a) $V_{dd} = 0.5$ V

Figure 4.9: Delay vs. V_{th0} @$V_{dd} = 0.5, 0.7, 0.9$ V, respectively



(b) $V_{dd} = 0.7$ V



(c) $V_{dd} = 0.9$ V

Figure 4.9: Delay vs. V_{th0} @$V_{dd} = 0.5, 0.7, 0.9$ V, respectively

From the figure, it can be observed that as V_{dd} increases, the delay-V_{th0} curve becomes linear, which results in different *PDF* curve shapes, as stated in [60].

4.3.4 *PDF* Generation Model for *CMOS* Gates

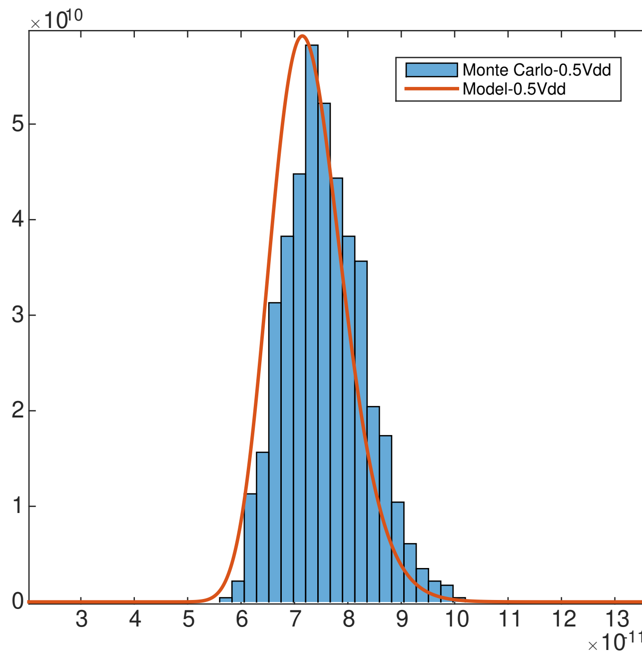
As V_{th0} is assumed to be the *RV* obeying Gaussian Distribution, the *PDF* of the propagation delay (P_τ) should be derived using the single variate *Change of Variables* theorem, as Equation (4.26).

$$P_\tau = P_{V_{th0}} \times \frac{dV_{th0}}{dt} = P_{V_{th0}} \times \frac{1}{\frac{dt}{dV_{th0}}}, \tag{4.26}$$

where $P_{V_{th0}}$ is the Gaussian distribution *PDF* of V_{th0}, and t represents the time.

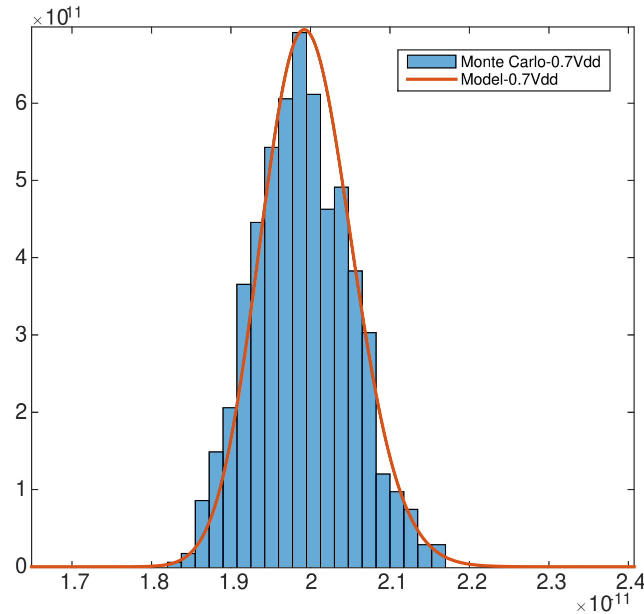
It is noticed that $V_{th0}(\tau_{pd})$ is difficult to obtain from $\tau_{pd}(V_{th0})$, so an alternative way to derive $\frac{dV_{th0}}{dt}$ is applied, as expressed by the latter expression in Equation (4.26).

Figure 4.10a, 4.10b and 4.10c present the model estimation and *SPICE MC* simulation results at different V_{dd} values with Gaussian parameters $\mu(V_{th0}) = 0.49396$ V and $6 \times \sigma(V_{th0}) = 40$ mV.

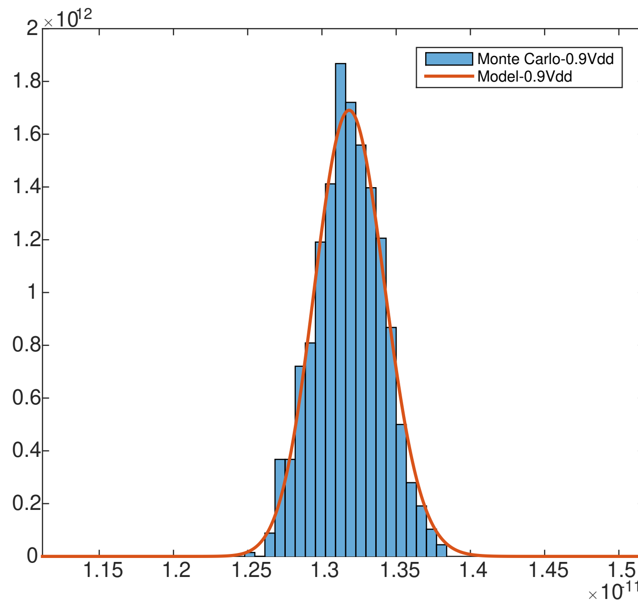


(a) $V_{dd} = 0.5$ V

Figure 4.10: Comparison of *PDF* curves with *SPICE MC* simulation



(b) $V_{dd} = 0.7$ V



(c) $V_{dd} = 0.9$ V

Figure 4.10: Comparison of *PDF* curves with *SPICE MC* simulation

The figure indicates that the model can accurately match *SPICE MC* simulation results at all V_{dd} values with respect to both the mean value and curve shape.

Although the results presented here are for the *Inverter*, it is easy to extend the principle to the other simple generic Boolean gates, such as the *AND* and *XOR*

gates, and comparable accuracy can be obtained.

4.4 *ANN-SSTA*: *ANN-based SSTA* Gate Model

In this section, the *ANN* function approximator will be employed to construct the *SSTA* gate models. In this model, both of the threshold voltage (V_{th0}) and channel length (L) variations will be considered for demonstration purposes, although more variations can be included, such as the gate oxide thickness (t_{ox}). The framework flow is depicted in Figure 4.11.

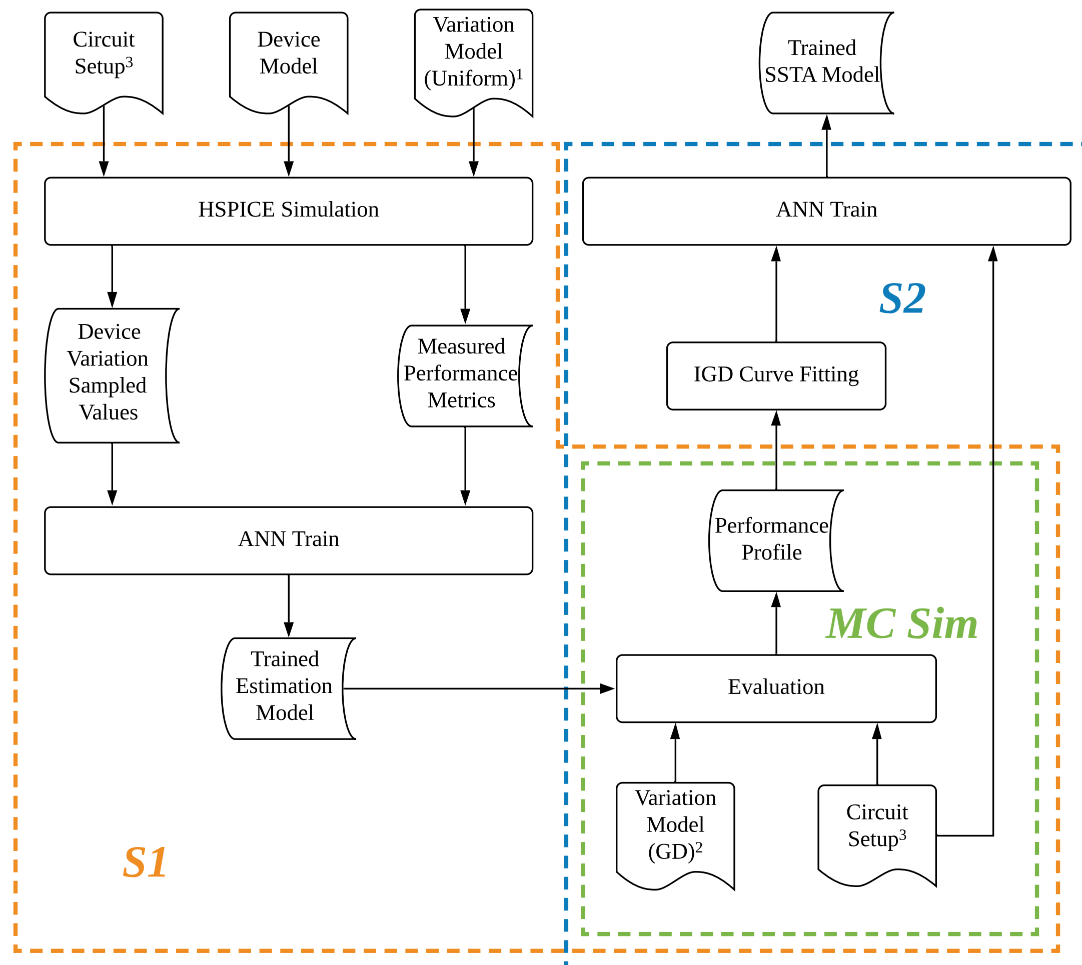


Figure 4.11: The *ANN* τ_{pd} estimation model framework flowchart

Two different modelling strategies will be introduced:

1. The flow inside the orange dashed frame labelled $S1$ represents the *Mod-*

elling Strategy 1, which is similar to the analytical analysis introduced in the previous section, but instead of being derived from the transistor operation formulas, the τ_{pd} will be predicted by a trained *ANN* model. Then, the *PDF* generation will be done with a traditional *MC* simulation.

2. The flow in the blue frame labelled $S2$ represents the *Modelling Strategy 2*, which makes use of the assumption that the response space P_τ is following the *IGD* when assuming the process parameters to be *RV*s following the *Gaussian Distribution* (GD). As mentioned in Section 1.3, *IGD* has better accuracy approximating the propagation delay *PDF*, especially in trans-regional (i.e., can be applied to circuits operating at both weak and strong inversion regions) applications. The *IGD* parameters, μ and λ, are a function of the process and circuit setup parameters, therefore by fitting the histogram of the propagation delays from *MC* simulation, a sample pair of μ and λ can be predicted for a specific circuit setup (e.g., a certain V_{dd} value) by an *ANN* function approximator.

The $S1$ flow starts with a normal gate or cell characterization with the *HSPICE MC* simulation. At the transistor level, the model is mapping the process parameters variances into τ_{pd}, as shown in Equation 4.27:

$$\tau_{pd} = f(\{\Delta V_{th0}^i\}, \{\Delta L^j\}), \tag{4.27}$$

where ΔV_{th0} and ΔL are random variables, i and j are integers represent each individual transistor parameter variance. τ_{pd} responds to every individual transistor parameters' variations in each *MC* trial, regardless of the type of the variation, as the global variations and local variations for a single transistor are additive as described in Section 4.1. Therefore, to extract the τ_{pd} model from *SPICE*, all device parameter variations used are local variations and uniformly distributed, as the superscript "*1*" in Figure 4.11.

The $S2$ flow starts from the *MC* simulation results of $S1$, *IGD* fitting will be performed on the resultant τ_{pd} histograms, and the corresponding (μ, λ) pairs can

be achieved. With the circuit setup parameters as inputs and the corresponding (μ, λ) pairs as targets, the *ANN* will be trained to predict the (μ, λ) pair, hence the final P_τ.

It is worth mentioning that unlike $S1$, $S2$ does not capture the characteristics of the process parameters. Thus there is no need to define the (μ, λ) function with respect to the process parameters.

In the abstraction level of $S2$, in order to achieve sufficient accuracy in the targeting μ and λ, millions of samples to simulate is very common, hence the modelling of a single *CMOS* gate may cost days with traditional *HSPICE* simulation. This is what the $S1$ is designed for, as it provides a very fast and accurate framework based on the *MC* simulation, which improves the modelling time of one *CMOS* gate to the order of seconds on an average performance laptop PC.

4.4.1 Circuit Model for *ANN* Training Dataset Generation

At the circuit level, the model is mapping a certain circuit setup parameters combination into τ_{pd} or P_τ as shown in Equation 4.28 and Equation (4.29), respectively.

$$\tau_{pd} = f(\Delta V_{dd}, \Delta N_{FO}, \Delta N_{FOD}, \dots) \tag{4.28}$$

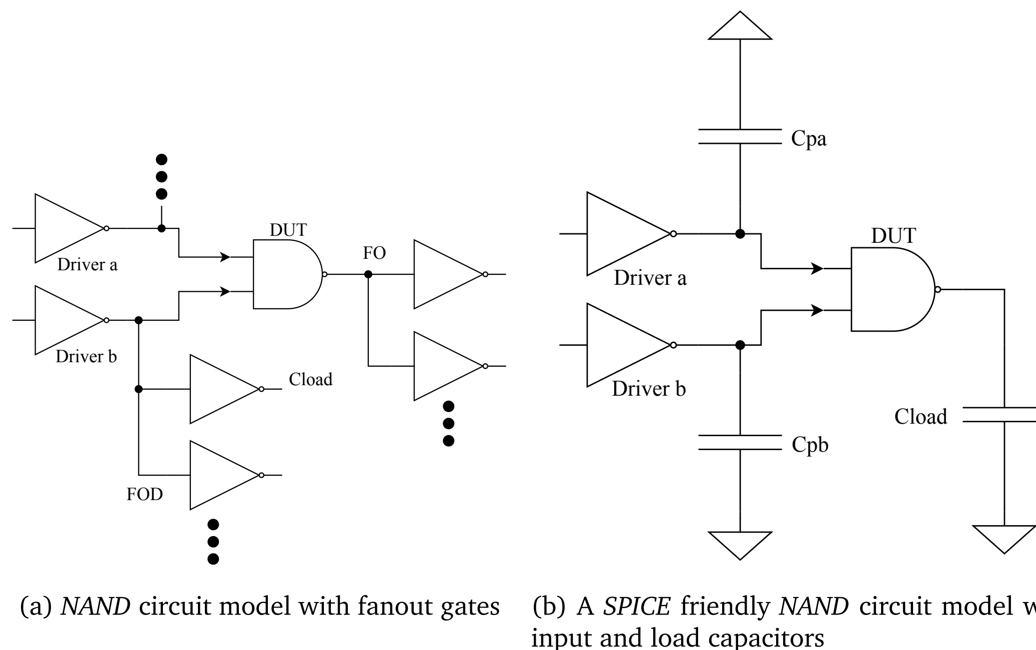
$$P_\tau = f(\mu, \lambda; \Delta V_{dd}, \Delta N_{FO}, \Delta N_{FOD}, \dots), \tag{4.29}$$

where $\Delta V_{dd}, \Delta N_{FO}$, and ΔN_{FOD} are the random variables, the ellipsis shorts for the parameter variances in Equation 4.27, N_{FO} is the number of *fanout gates* (FO) of the *Design Under Test* (DUT), and N_{FOD} is the number of *fanout gates of the driver gate* (FOD). Contrary to parameter variations, circuit setup variations affect all devices identically in one *MC* trial. Thus circuit setup variations used are global variations and uniformly distributed, as the superscript "3" in Figure 4.11.

When extracting the estimation model from *HSPICE* simulation, a circuit model supposed to have the following features has to be constructed:

- Is able to capture all proposed parameters, including the circuit setup parameters (i.e., V_{dd}, fanout, etc.) and device design parameters (i.e., V_{th0} and L).
- Easy to be implemented in a *SPICE* netlist.
- Fast to converge when running the *SPICE MC* simulation, especially when a large number of samples are to be simulated.

A most straightforward model is shown in Figure 4.12a, the fanout can be represented by a number of different loading gates. To calculate the total load, a unit capacitance gate (e.g., an *Inverter*) can be defined as the reference load L_0, and each gate is rational to the unit gate, then the total load can be obtained by adding all unit gates' coefficients ρ, $L_{tot} = (\sum \rho) \times L_0$.



(a) *NAND* circuit model with fanout gates (b) A *SPICE* friendly *NAND* circuit model with input and load capacitors

Figure 4.12: The *SPICE* simulation circuit model for gate or cell characterisation

This model can capture the driver fanout, current gate fanout, and driver gate strength. However, the model will introduce extra complexity and performance degradation when being simulated in *SPICE* as the netlist has to be altered to simulate different numbers of fanout gates to extract the effects of the fanout. Another disadvantage of this model is the load value discontinuity, which may

affect the efficiency of training the *ANN* and may introduce the amount of duplicated values for loading value in the dataset. Therefore, a more efficient equivalent capacitor approximation circuit model is proposed and depicted in Figure 4.12b.

Each of the drivers' fanout loads was approximated by a constant load capacitor in parallel with the *DUT* input capacitance, and a load of *DUT* is also approximated by a constant capacitor. This model perfectly solved the issues of the model proposed in Figure 4.12a, although a slight error may be introduced due to the use of constant capacitors to approximate the gate capacitances, which are dynamic during the circuit switching [106].

Both of the two modelling strategies share the same circuit model, and that is the reason why $S2$ can be constructed based on the results of $S1$. In the rest of this chapter, the two modelling strategies will be introduced successively.

4.4.2 Modelling Strategy 1

The *Predictive Technology Model* (PTM) *45nm SPICE MOSFET* model [110] provided as part of the FreePDK45 [111] will be used in the *HSPICE* simulation. All standard gate or cell used is from *Nangate Open Cell Library* [112].

As will be detailed in Appendix C, the advanced sampling methods can reduce the number of samples needed; thus, *Latin Hypercube Sampling* (LHS) is used here. A 10^5-sample *MC* simulation was run on a quad-core Intel Core i5 laptop PC, which costs around 2500 seconds to conclude for a single *NAND* gate testing circuit in Figure 4.12b. A summary of the *HSPICE MC* simulation environment setup is depicted in Table 4.2.

It is noticed that due to the strength mismatch, the equivalent capacitance differences between Pull-up and Pull-down networks of *CMOS* circuits as well as the different fanout configurations between the two inputs will result in non-identical rising and falling edge propagation delays. Thus, the propagation delay profile of a logic gate cannot be represented by only one *PDF*, which will

Table 4.2: Training dataset generation *SPICE MC* simulation setup and parameters

parameters and options	setup
V_{dd}	$[0.4, 0.9]$ V uniformly distributed
C_{load}, C_{pa}, C_{pb}	$[0, 10]$ fF uniformly distributed
V_{th0n}, V_{th0p}	± 10 % uniformly distributed
L	± 30 % uniformly distributed
Sampling method	*LHS*
MC samples	10^5
DUT gate	NAND2_X1
Driving gates	INV_X1
Input vectors	10001011 00101001

result in over- or under-estimation. For example, for a *NAND* gate, the different input transitions (e.g., $01 \rightarrow 11$ and $11 \rightarrow 10$) will result in different propagation delay profiles, and the differences are not negligible even if the *PMOS* and *NMOS* ratio is carefully adjusted.

Therefore, during the simulation, 18 values were measured, including:

- 8 τ_{pd} values representing the effective transitions, $\{01, 10, 00a, 00b\} \rightarrow 11$ and $11 \rightarrow \{01, 11, 00a, 00b\}$, respectively;
- 6 output t_{rise}/t_{fall} values with each for one of the 6 effective transitions;
- 4 input t_{rise}/t_{fall} values.

The reason why t_{rise} and t_{fall} are measured will be discussed in the later paragraphs.

Figure 4.13 shows some sample measurements for transitions $01 \rightarrow 11$ and $11 \rightarrow 00$. By convention, t_{rise} is measured from 10 % to 90 % of V_{dd}. Similarly, t_{fall} is measured from 90 % to 10 % of V_{dd}. τ_{pd} is measured from 50 % of the trigger signal to 50 % of the target signal [113].

The reported parameter variances data in *.mc0* file [114] may look similar to Table 4.3. The resulting measurement file *.mt0* contains measured timing values, which will be used as target label when training the *ANN* function approxima-

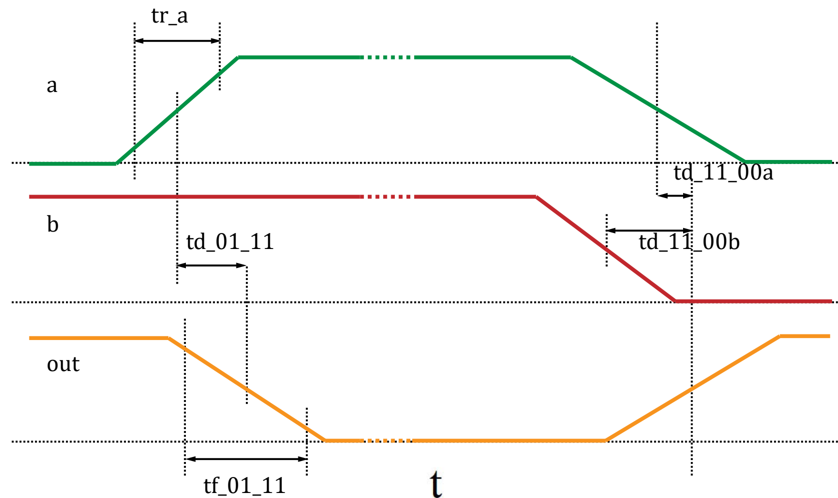


Figure 4.13: Performance measurement demonstration

tor. Note that for ease of documenting, the data matrix has been transposed to $M \times N$, i.e., each column in Table 4.3 represents a sample. Each row in this table represents a uniformly distributed random variable in the range $[-0.5, 0.5]$, which will be linearly transformed into the desired range then added to the nominal value of the corresponding parameter internally in *HSPICE* when doing the simulation. As all transformations are linear, the *ANN* can be trained directly using the raw variances. More details for *HSPICE* variation analysis and the *.mt0* file can be found in Appendix C.

Table 4.3: *HSPICE* generated *.mc0* file contents

	parameters	1 [1]	2	3	4	...	100k
	vdd	0.	0.2023510572	-0.4488469296	0.1676261753		-0.4112228853
	cload	0.	-2.137398286e-02 [2]	-0.1984278652	6.180707007e-02		0.4049748591
	cpa	0.	-0.3870780276	0.1022240096	-0.2904834930		6.143584373e-02
	cpb	0.	-0.2935470745	-0.1366238586	-0.1264847123		0.2892990319
NMOS	xina.m0:vth [3]	0.	-0.2667102345	0.2485355339	0.1026946604		-0.4529972709
	xina.m0:l	0.	0.3115190725	0.1683437424	0.1583698373		0.2621251652
	xinb.m0:vth	0.	-0.4692071011	0.1129886504	0.1162120047		0.4526021262
	xinb.m0:l	0.	5.608925964e-02	-0.3268394160	-0.4125052107		0.2443484570
	xdut.m1:vth	0.	-0.1249298198	-5.839675539e-02	-0.3235137060		-0.3602246470
	xdut.m1:l	0.	0.4071649862	0.1972641692	0.4829383198	...	0.2660199211
	xdut.m0:vth	0.	-0.4070512796	0.3017251486	0.1193054185		3.398277231e-02
	xdut.m0:l	0.	0.2237755085	-0.3795613591	0.3501212173		0.4320408238
PMOS	xina.m1:vth	0.	0.2950965538	-4.538605735e-02	-0.4656191632		0.2672850504
	xina.m1:l	0.	0.4857183650	-0.3502920159	-0.3197167544		0.3665694821
	xinb.m1:vth	0.	-8.822687064e-02	3.145546635e-02	-0.2622012619		0.3580840247
	xinb.m1:l	0.	-0.4078403762	-6.309100763e-02	-0.2210902685		-0.1606862565
	xdut.m3:vth	0.	-0.2034734905	0.4179550357	-0.2739983003		0.2573089282
	xdut.m3:l	0.	0.2811667637	9.992049287e-02	-0.4992261433		2.931601019e-02
	xdut.m2:vth	0.	0.3218585473	-0.2423866270	0.2522292848		-0.2105270758
	xdut.m2:l	0.	-0.4907818319	0.1431688274	0.2261734682		0.3982023437

Figure 4.14 presents the *Pearson Correlation Matrix* showing the correlations between the performance space and parameter space. These coefficients provide a clear view of how timing is varying against each of the parameters.

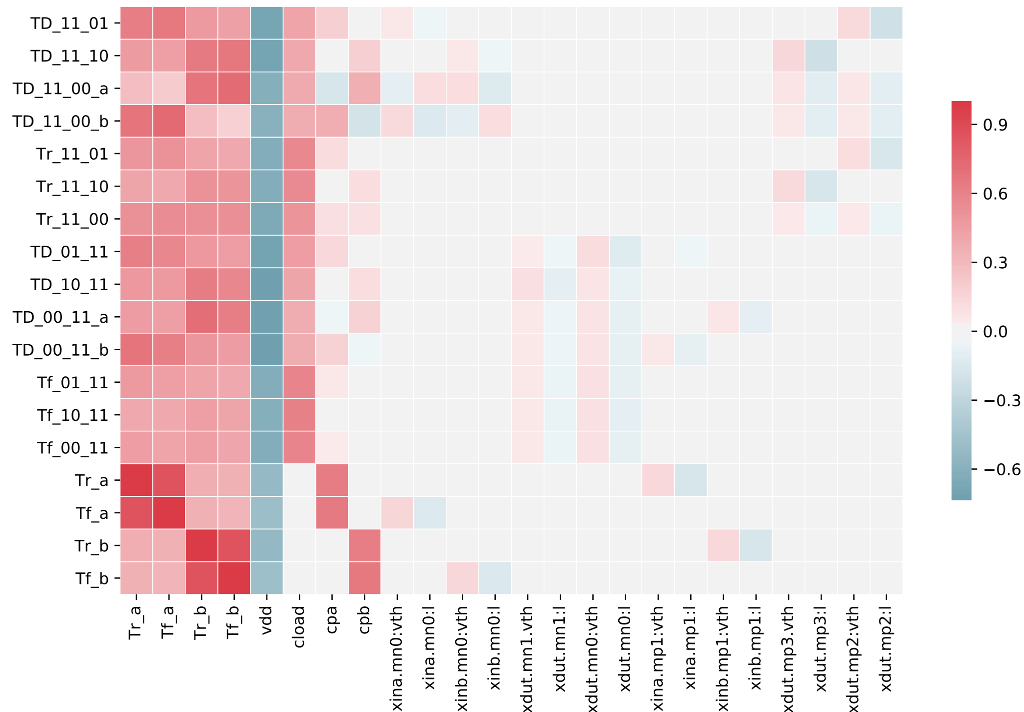


Figure 4.14: Heatmap of Pearson correlation coefficients between performance and parameter variances

As can be seen from the heatmap, the timing is a strong function of V_{dd} and load capacitance. The falling (rising) time is a function of the *NMOS* (*PMOS*) parameters of the current gate. Additionally, the timing is also a function of the driving gates device parameters, which makes the problem complicated as the independent prediction capability of the gate model will be broken. Non-negligible information loss will be introduced if the variances associated with the driving gates are simply discarded, hence lead to a large error when training the *ANN* model, while more sophisticated methods such as a look-up-table (LUT) has to be applied to include the driving characteristics of different kinds

[1]The first *MC* sample in *HSPICE* will always be the nominal simulation by default, thus all parameter variances are 0 in this sample

[2]Negative values may exist for the τ_{pd} measurements due to very slow input signals

[3]This is a hierarchical presentation of the parameters, where *xin1* is the subcircuit instance name, *.m0* indicates the component name and *:vth* is the parameter being varied

of the driving gate. On the other hand, as is found in the heatmap, the driving gates parameter variances along with the *fanout gates of the driver gate* (FOD) equivalent capacitors C_{pa} and C_{pb} variances can be captured by their t_{rise} and t_{fall}. In other words, the variances of C_{pa}, C_{pb} and parameters of *xina*, *xinb* will be directly reflected by variances of relevant $t_{r(f)a}$ and $t_{r(f)b}$. As a result, $T_{r(f)a}$ and $T_{r(f)b}$ also appear in the x-axis of the heatmap to indicate the corresponding rising and falling time of the gate inputs.

For efficient training of the *ANN*, two *ANN*s will be separately trained to predict the falling and rising timing, respectively, although a single *ANN* with more neurons in the hidden layer can achieve the same performance yet using longer training time.

Summarizing the above discussions, the complete τ_{pd} estimation model will be expressed as Equation 4.30.

$$\begin{aligned}(\tau_{pd_{fall}}, t_{fall}) &= f_f(\Delta V_{dd}, \Delta C_{load}, \{\Delta V^i_{th0N}\}, \{\Delta L^i_N\}, t_{r(f)a}, t_{r(f)b}) \\ (\tau_{pd_{rise}}, t_{rise}) &= f_r(\Delta V_{dd}, \Delta C_{load}, \{\Delta V^i_{th0P}\}, \{\Delta L^i_P\}, t_{r(f)a}, t_{r(f)b}),\end{aligned} \tag{4.30}$$

where i represents each *NMOS* or *PMOS* transistor.

Table 4.4: *ANN* architecture and training setup

Input	*ANN* Config	Output
10 features $\{\Delta V_{dd}, \Delta C_{load}, \Delta V^{(i)}_{th0N(P)}, \Delta L^{(i)}_{N(P)}, t^{(j)}_{r(f)}\}$, where $i \in \{1,2\}$, $j \in \{a,b\}$	$[16, 6]$ 16 neurons in the hidden layer with *Sigmoid* activation functions; 6 neurons in the output layer with linear activation functions	6 outputs $\{\tau_{pd}\|_{X(Y)}, t\|_{X(Y)}\}$, where $X \in \{11 \rightarrow 01, 11 \rightarrow 10, 11 \rightarrow 00\}$, $Y \in \{01 \rightarrow 11, 10 \rightarrow 11, 00 \rightarrow 11\}$

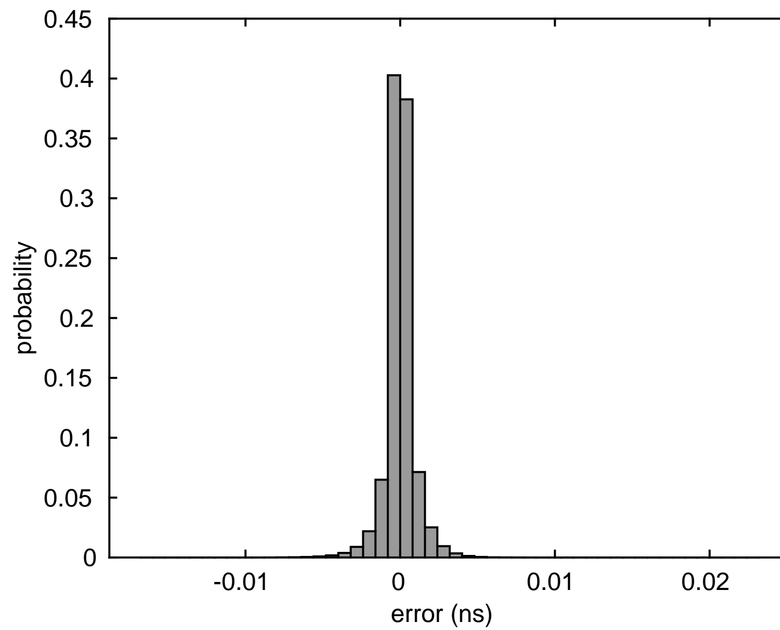
By the guidance of Equation 4.30, the training dataset for *NAND* gate model contains $N = 10^5$ samples of $M = 10$ input vectors and $N = 10^5$ samples of $K = 6$ target labels. The detailed *ANN* training setup is described in Table 4.4.

The training of the *ANN*s was converged in around 600 seconds, Table 4.5 summarizes the training performance.

Table 4.5: Training results of *NAND ANN* model

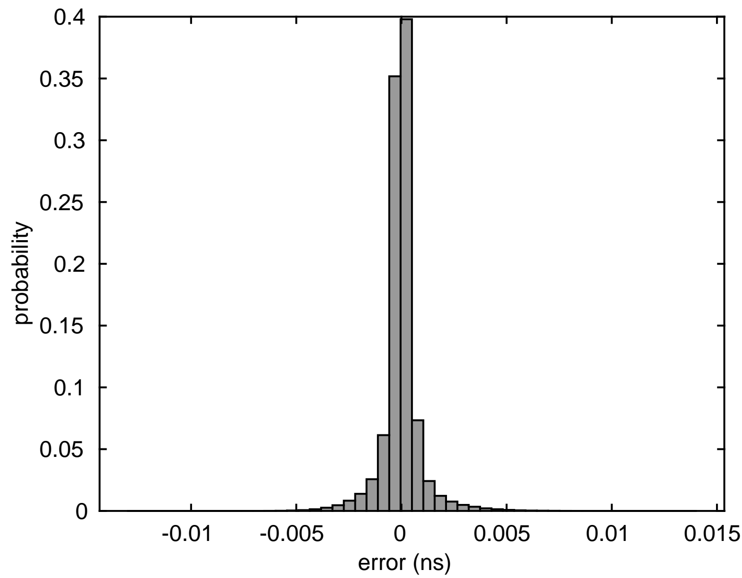
	falling	rising
Train algorithm	Levenberg-Marquardt	
Train/Verification/Test	$70\ \%/10\ \%/20\ \%$	
Train time	≈ 600 s	
Test MSE	3.04×10^{-7}	5.94×10^{-7}
Test ME	$< 1\ \%$	$< 2\ \%$

The error histograms are shown in Figure 4.15, as can be seen from the figure, over $99\ \%$ of the measured errors are within ± 2 ps, which indicates very high accuracy of the prediction model.



(a) *NAND* rising edge error

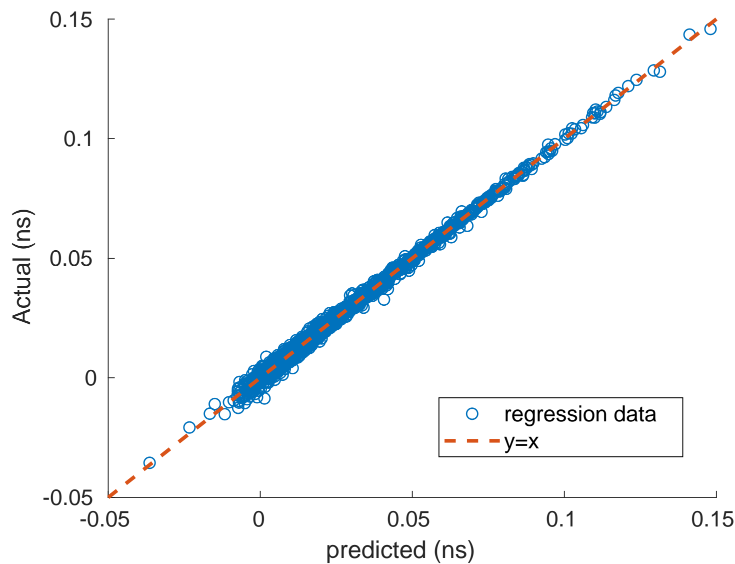
Figure 4.15: Error histogram of the trained *NAND* model



(b) *NAND* falling edge error

Figure 4.15: Error histogram of the trained *NAND* model

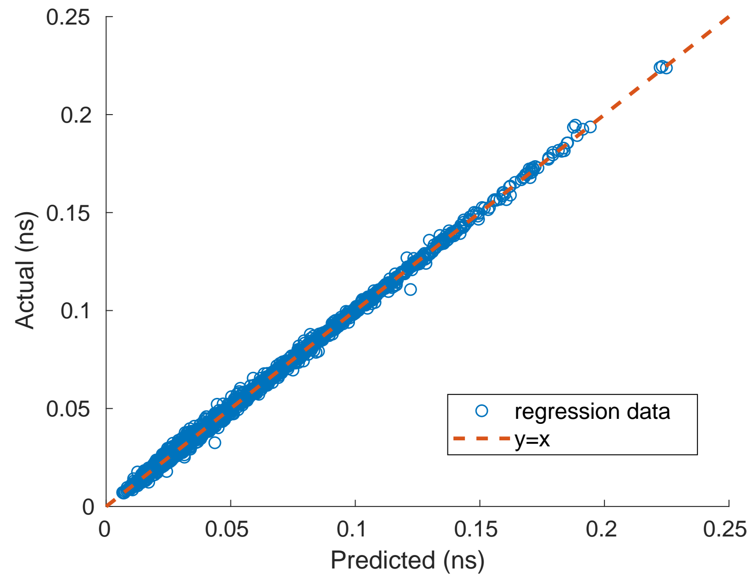
Figure 4.16 shows the regression diagram of a 10^4-sample test run with the same condition with the model training summarized in Table 4.2. The *ANN* model prediction takes around 0.0044 seconds while the *HSPICE* concludes in around 300 seconds, i.e., over 60000 times performance boost was obtained.



(a) *NAND* rising edge regression

Figure 4.16: Regression diagram of 10^4-sample test of the trained *NAND* model

The dataset generation and training process of a *NAND* gate model are relatively



(b) *NAND* falling edge regression

Figure 4.16: Regression diagram of 10^4-sample test of the trained *NAND* model

simple as the rising, and falling transitions are only driven by a certain *MOSFET* type, either *PMOS* or *NMOS*. When modelling more complex gates, such as an *XOR* gate, the process will be slightly more complex, yet the basic concept remains the same, i.e., selecting only the relevant parameters indicated by the *Pearson correlation coefficients* as the inputs.

With this *ANN* function approximator based τ_{pd} estimation framework, other *CMOS* gates or standard cell prediction models can also be easily constructed. Table 4.6 summarizes the training and prediction results for some other commonly used *CMOS* gates. Note that column 2 shows the *SPICE* simulation run time and the corresponding random samples simulated, while column 3 depicts the *ANN* training time in seconds, and columns 5 and 6 present the 10^4-sample tests run time taken by the *ANN-based* model and the *SPICE* simulation, respectively.

As can be seen from the table, the *ANN-based* model is drastically faster than its *SPICE* counterpart. What is more attractive, the *ANN-based* model performance has a relatively weak relationship with the circuit complexity as the performance is mainly determined by the scale of the *ANN*, in contrast to the *SPICE*

Table 4.6: Training results of some commonly used *CMOS* gates or cells

Cell	*SPICE MC* (s)	Training (s)	*MSE* (ns^2)	Predict 10^4-sample test (s)	*SPICE* 10^4-sample test (s)
NAND2	$2500/10^5$	650	$\approx 4.0 \times 10^{-7}$	0.0044	300
INV2	$450/5 \times 10^4$	53	$\approx 9.0 \times 10^{-8}$	0.0013	96
AND2	$5000/1.2 \times 10^5$	780	$\approx 8.0 \times 10^{-7}$	0.0048	520
NOR2	$3000/10^5$	660	$\approx 8.0 \times 10^{-8}$	0.0045	310
OR2	$4500/1.2 \times 10^5$	1200	$\approx 8.0 \times 10^{-7}$	0.0044	510
XOR2	$7200/1.2 \times 10^5$	530	$\approx 7.0 \times 10^{-7}$	0.0067	980

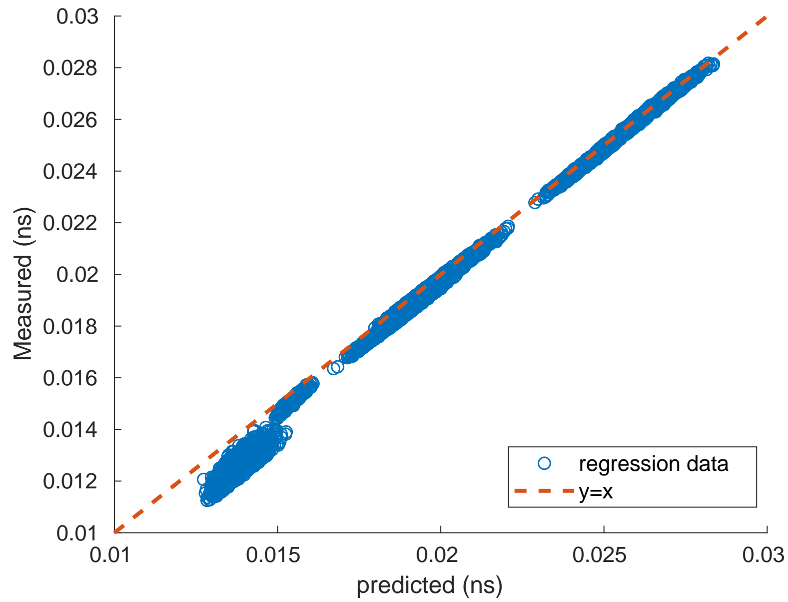
simulation.

The *ANN* τ_{pd} estimation model can be used in cooperating with the *MC-based SSTA* to implement the *SSTA* gate model without having to assume certain distributions of the V_{th0} and L. In practice, this method can work virtually in any scenarios where the *SPICE MC* simulation works, which makes this *SSTA* gate model really flexible but significantly faster than the *SPICE* based method.

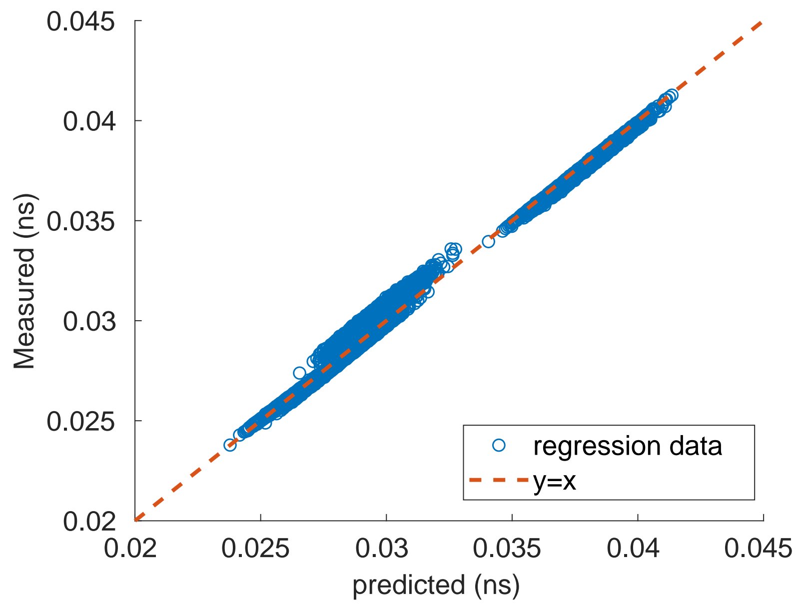
To demonstrate the *SSTA* framework, the V_{th0} and L were simply assumed to be independent *Gaussian RV*s, and having both global and local variations. To be specific,

- *Global Variation:* V_{th0} and L are both varied by $4\text{-}\sigma = 5\ \%$.
- *Local Variation:* V_{th0} and L are varied by $4\text{-}\sigma = 5\ \%$ and $4\text{-}\sigma = 10\ \%$, respectively.

The circuit model in Figure 4.12b operating at $V_{dd} = 0.65$ V, $C_{load} = 5$ fF, $C_{pa} = 5$ fF and $C_{pb} = 5$ fF with 10^4 samples is used to demonstrate the framework. The resulting *MSE* of the prediction is 3.73×10^{-7} and 1.23×10^{-7} for rising and falling transitions, respectively. The regression diagrams of the prediction results are shown in Figure 4.17.



(a) Rising edge regression



(b) Falling edge regression

Figure 4.17: Regression diagram of 10^4-sample Gaussian variation prediction $@V_{dd} = 0.65$ V, $C_{load} = 5$ fF, $C_{pa} = 5$ fF and $C_{pb} = 5$ fF

Table 4.7 shows the prediction results for the *NAND* gate. In the table, μ and λ are the *IGD* fitting parameters, where the upper values of each voltage are the predicted relative errors for the rising edge, and the lower ones are those for the falling edge compared to the *SPICE* simulation.

From the results, it can be noted that the estimation of most transition types can achieve very high accuracy for both μ and λ. The prediction power for μ is high in almost all circumstances and achieved errors of less than $1\ \%$. It can also be observed that the λ estimation is relatively weaker. Also, it makes sense that the prediction power for cases where both inputs are switching is weaker because only the maximum τ is selected as a target in this case.

With the *ANN-based* τ_{pd} models for all common gates achieved, the circuit level *SSTA* can be performed with *MC* simulation. To demonstrate the *MC-based SSTA* on a circuit, a case study on a simple Majority Voter (MJV) circuit will be presented.

Table 4.7: Relative estimation errors (%) of the predicted P_τ for *NAND* gate

V_{dd} (V)	τ_{pda}		τ_{pdb}		τ_{pdab}		t_a		t_b		t_{ab}	
	μ	λ	μ	λ	μ	λ	μ	λ	μ	λ	μ	λ
0.45	0.52	1.17	0.56	1.95	0.60	17.93	0.19	1.60	0.02	1.34	0.52	5.29
	0.14	1.58	0.28	1.47	1.17	0.34	0.16	4.70	0.18	4.19	0.32	7.14
0.55	0.62	8.97	0.20	1.33	2.61	19.47	0.38	1.76	0.22	2.90	1.37	17.51
	0.15	6.51	0.39	3.34	1.78	5.34	0.51	1.97	0.61	0.61	0.91	0.04
0.65	2.10	3.64	1.01	3.11	5.72	27.69	0.63	6.40	0.34	2.51	2.52	17.42
	0.19	8.14	0.90	4.79	2.32	3.94	0.39	7.08	0.34	1.97	0.78	2.52
0.75	1.14	1.41	0.02	2.84	6.40	28.51	0.69	0.91	0.69	3.91	2.36	18.27
	0.08	6.08	0.34	8.19	3.48	0.43	0.49	7.28	0.43	3.73	0.95	6.77
0.85	0.12	7.76	0.42	2.69	6.82	23.36	0.55	11.92	0.69	6.65	1.34	30.36
	0.55	2.01	0.16	7.08	4.43	1.53	0.29	6.10	0.77	0.60	0.66	2.94

Before simulating a real circuit, the equivalent capacitances must be measured. Table 4.8 summarizes the approximated capacitances of the *CMOS* gates.

Table 4.8: Summary of the capacitances for various cells

Cell	Capacitance (fF)
NAND2_X1	1.1
INV2	0.42
AND2_X1	0.55
NOR2	0.96
OR2_X1	0.54
XOR2_X1	1.55

As mentioned above, the different types of transitions may result in a non-negligible difference in the corresponding P_τs. Thus, a transition-by-transition style *SSTA* on the active paths will be performed for demonstration purpose.

Consider the 3-input Majority Voter circuit composed of *AND*, *XOR* and *OR* gates in Figure 4.18.

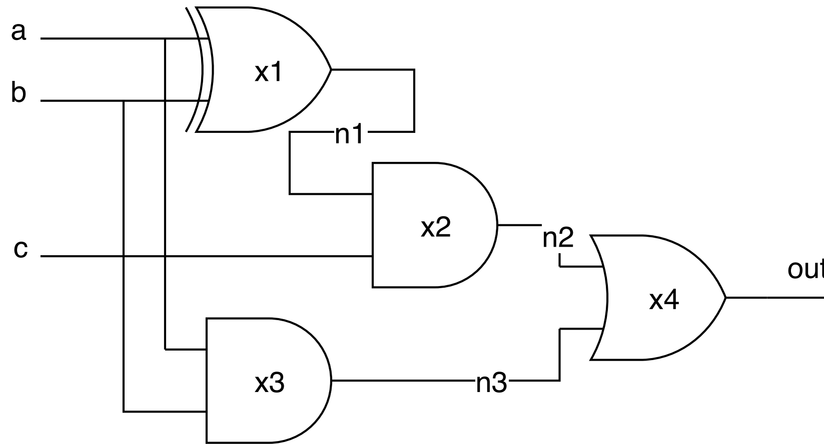
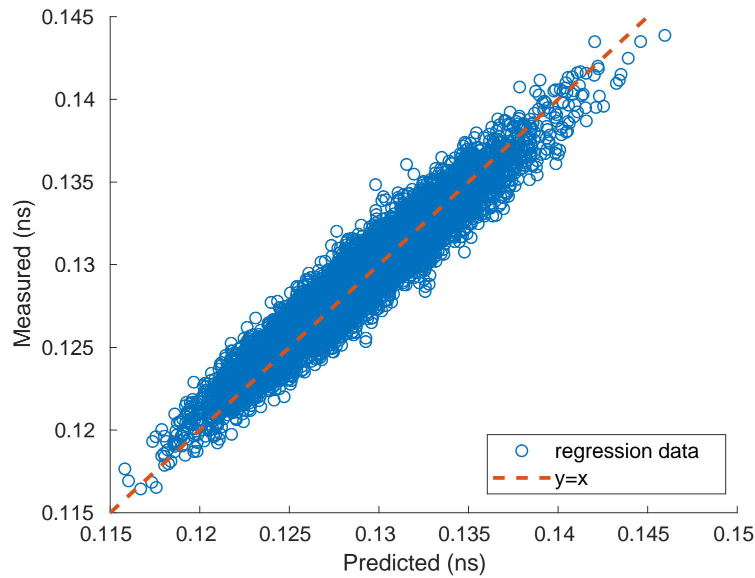


Figure 4.18: A Majority Voter circuit composed of basic *CMOS* gates

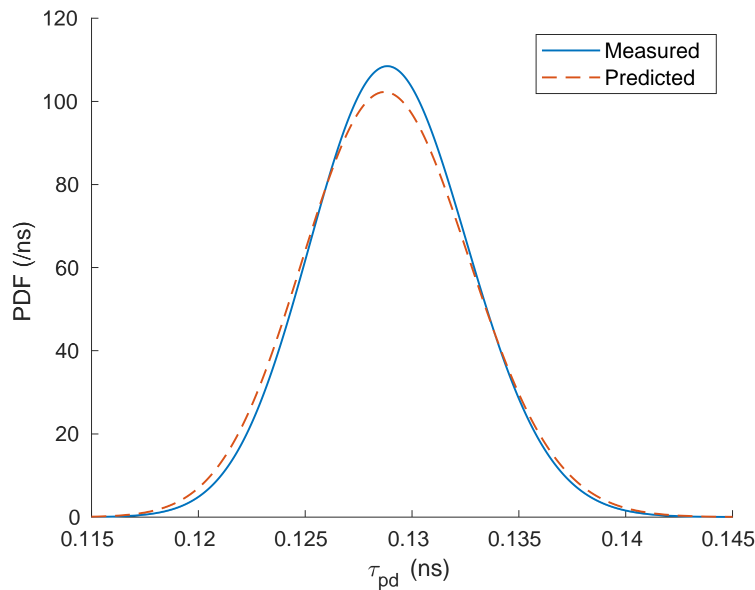
A certain input transition $001 \rightarrow 011$ yields the switching of gates $x1, x2, x4$, thus the corresponding path delay is simply the sum of the corresponding τ_{pd}s, i.e. $\tau_{pd}(out) = \tau_{pd}(x1)|_{00\rightarrow 01} + \tau_{pd}(x2)|_{01\rightarrow 11} + \tau_{pd}(x4)|_{00\rightarrow 10}$. Hence, the $P_\tau(out)$ can be obtained from deriving predicted τ_{pd}s for each sample accordingly.

Figure 4.19 shows the path delay from the model prediction and that from

SPICE simulation when operating at $V_{dd} = 0.5$ V.



(a) Regression diagram



(b) *PDF*

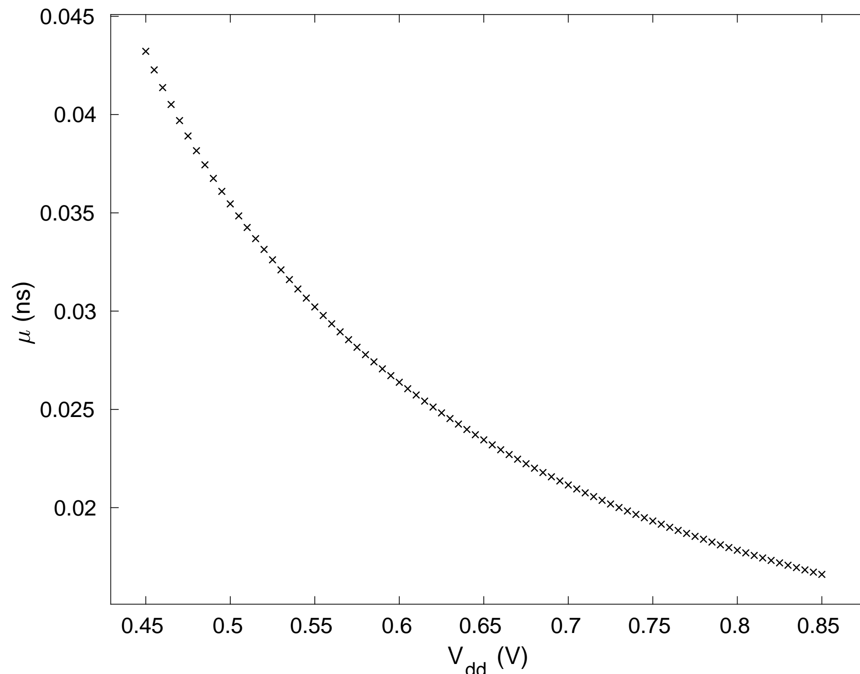
Figure 4.19: Regression diagram and *PDF* plots of a Majority Voter $@V_{dd} = 0.5V$

The μ and λ errors are $\epsilon_{\mu} = 0.11$ ps $(0.02\ \%)$ and $\epsilon_{\lambda} = 18$ $(2.79\ \%)$, respectively. The prediction model completes in less than 0.01 second compared to over 400 seconds of *SPICE* simulation, i.e. over 40000 times performance boost was obtained.

4.4.3 Modelling Strategy 2

Figure 4.20a and Figure 4.20b show that μ and λ as functions of V_{dd}, assuming the same variation conditions and circuit setup (i.e. $C_{load} = C_{pa} = C_{pb} = 5$ fF) from the 10^4-sample *MC* simulations with $S1$:

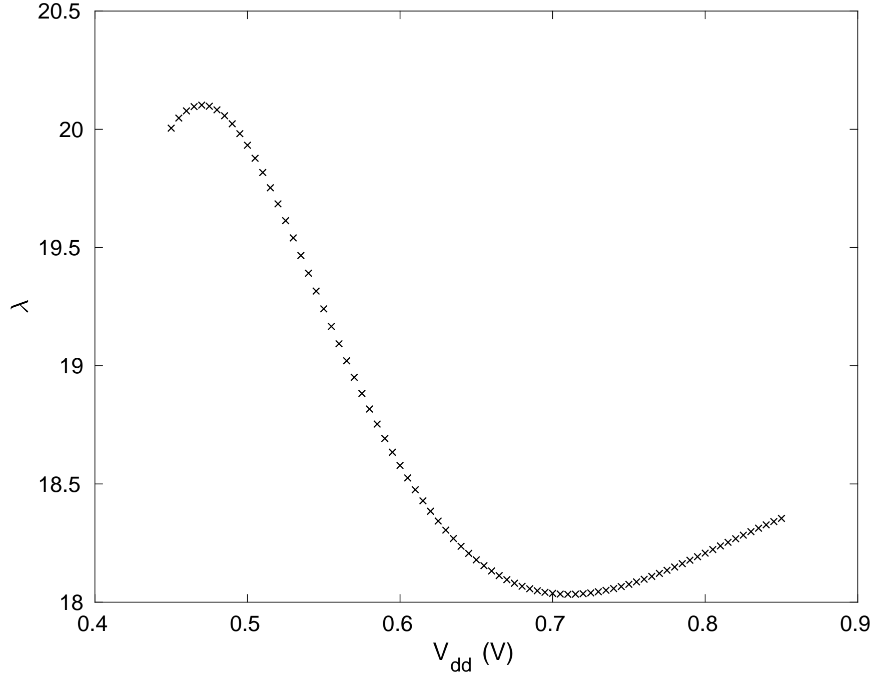
- *Global Variation:* V_{th0} and L are both varied by $4\text{-}\sigma = 5\ \%$.
- *Local Variation:* V_{th0} and L are varied by $4\text{-}\sigma = 5\ \%$ and $4\text{-}\sigma = 10\ \%$, respectively.



(a) μ vs. V_{dd}

Figure 4.20: Function of μ and λ against V_{dd}

As observed from the figures, the relationship between μ and V_{dd} is straightforward as μ is mathematically indicating the mean value of the distribution, which should be monotonically decreasing with V_{dd} increasing, while the λ curve is not well defined as λ is a shape adjustment parameter.



(b) λ vs. V_{dd}

Figure 4.20: Function of μ and λ against V_{dd}

For $S1$, the *SSTA* on an *MJV* circuit with a transition-by-transition fashion has been demonstrated, which is not performance optimal. For this higher level modelling strategy $S2$, the mixture-distribution [115] of all possible transition P_τs of a gate is introduced, as shown in Equation 4.31:

$$\begin{aligned} \textit{MP}_\tau = \sum P(X) \times P_\tau|_X + \sum P(Y) \times P_\tau|_Y, \\ X \in \{11 \rightarrow 01, 11 \rightarrow 10, 11 \rightarrow 00\}, \\ Y \in \{01 \rightarrow 11, 10 \rightarrow 11, 00 \rightarrow 11\} \end{aligned} \tag{4.31}$$

For a sufficiently large input vector size (i.e., workloads, 500-bit vectors are simulated in this work), the final observed distribution should be the *weighted combination* (*finite mixture-distribution*) [115, 116] of the *PDF* of each *effective transition*, i.e., the transitions make output switch, using the corresponding transition probability as the weight.

Specifically, considering the *NAND* gate, all possible effective transitions, i.e., $P_\tau|_{00\rightarrow11}$, $P_\tau|_{01\rightarrow11}$, $P_\tau|_{10\rightarrow11}$, $P_\tau|_{11\rightarrow00}$, $P_\tau|_{11\rightarrow01}$, $P_\tau|_{11\rightarrow10}$, of one random circuit

setup should be profiled. Thus by simulating a number of different circuit setup combinations, we will get a dataset with $N \times M \times K$ values, where N is the number of simulated V_{dd} values, M the number of *MC* samples, and K the number of possible effective transitions.

Equation 4.31 reveals the fact that the mixture P_τ, *MP*$_\tau$, is a weighted combination of all individual transition P_τ, where the weights are the occurrence probabilities of each respective transitions. The *MP*$_\tau$ is composed of two parts, i.e., the first part is the measurements of the transitions producing the rising edge at the output, while the second part is measurements of those producing the falling edge.

From the *ANN* prediction model implementation perspective, the *ANN* inputs contain only the circuit setup parameters, i.e., V_{dd}, C_{load}, $\mu_{t_{r(f)a}}$, and $\mu_{t_{r(f)b}}$. Note that differing from $S1$, to capture the strength of the driving gate, the two statistical characteristics of the rising and falling transitions $\mu_{t_{r(f)a}}$ and $\mu_{t_{r(f)b}}$ are used and propagated. Therefore,the *ANN* model has 6 inputs and 7 outputs. The detailed *ANN* configuration is depicted in Table 4.9.

Table 4.9: *ANN* architecture and training setup

Input	*ANN* Config	Output
6 features $\{\Delta V_{dd}, \Delta C_{load}, \mu_{t_{r(f)a}}, \mu_{t_{r(f)b}}\}$	$[64, 7]$ 64 neurons in the hidden layer with *Sigmoid* activation functions; 7 neurons in the output layer with linear activation functions	7 outputs $(\mu, \lambda)\|_{X(Y)}$, where $X \in \{11 \rightarrow 01, 11 \rightarrow 10, 11 \rightarrow 00\}$, $Y \in \{01 \rightarrow 11, 10 \rightarrow 11, 00 \rightarrow 11\}$; $\mu_{t_{r(f)}}$

The training data is obtained by simulating on uniformly distributed $V_{dd} \in [0.4\text{ V}, 1.0\text{ V}], C_{load}, C_{ina}, C_{inb} \in [0, 2.0\text{ fF}]$ combinations with V_{th} and L variations using $S1$.

Figure 4.21 depicts the regression diagram of the training results. The training and testing *MSE* of the model are 8.3×10^{-6} and 1.8×10^{-5}, respectively.

The mixture P_τ technique allows a more representative propagation delay dis-

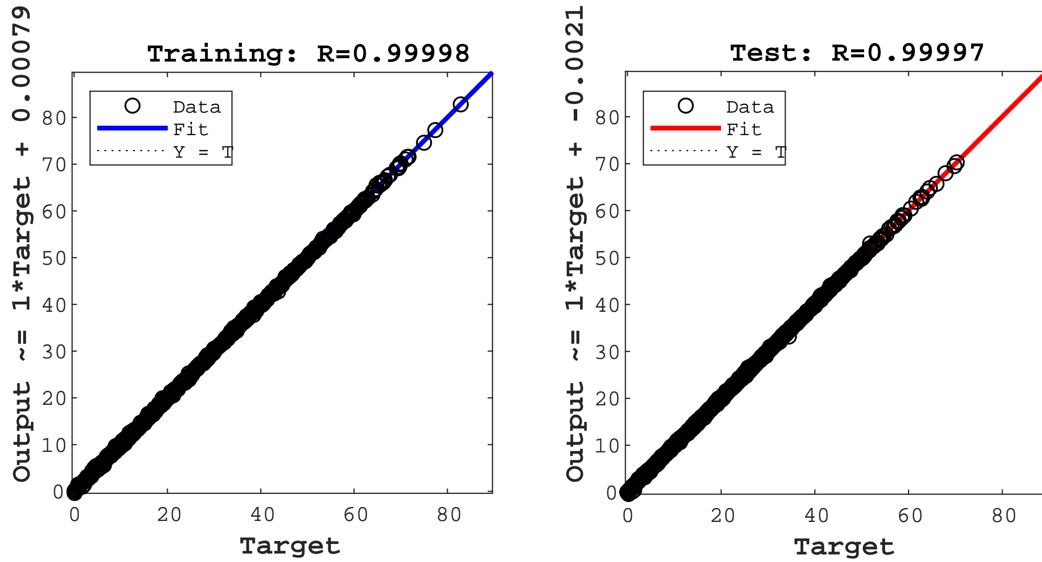


Figure 4.21: The regression diagrams for $S2$ *ANN* prediction models

tribution to be predicted. However, the calculation and propagation of the transition probabilities are quite an intensive task if the circuit of workloads being simulated is on a large scale. In order to fast derive the transition probabilities, a neural network method is again introduced.

As implemented in [117], a neural network can be trained to estimate the transition probabilities of a logic gate from its input vector statistics, which is much cheaper to propagate in the circuits. To train the network, an input vector sequence with $L = 500$ bits were randomly generated, which results in an $L \times N = 500 \times 2$ matrix, where N is the number of inputs. Then five most relevant statistical inferences were calculated for each input vector; hence 10 probability values were obtained. Once the values are obtained, a Principle Component Analysis (PCA) [118] is applied to further reduce the input dimension of the network before training.

Table 4.10 shows the detailed data preparation procedure and the network setup.

Table 4.10: *ANN* training setup

Input stats	PCA	*ANN*	Output
10 probabilities – 5 for each input – $P(x_i = 1)$, $P(X_i\|_{0\to0})$, $P(X_i\|_{0\to1})$, $P(X_i\|_{1\to0})$, $P(X_i\|_{1\to1})$, $i \in \{in_a, in_b\}$	10 input statistical inferences transform into the first 4 principle components representing 99.9 % variations	4 inputs, 128 neurons in the hidden layer, 11 outputs, 500 samples in the dataset	11 probabilities – 5 output statistics, $P(y = 1)$, $P(Y\|_{0\to0})$, $P(Y\|_{0\to1})$, $P(Y\|_{1\to0})$, $P(Y\|_{1\to1})$ and 6 resulting transition events – $P(Z_j)$, Z_j is one of the 6 transition events

Figure 4.22 illustrates the prediction power of the trained network, the training and testing *MSE* are 2.87×10^{-4} and 3.40×10^{-4}.

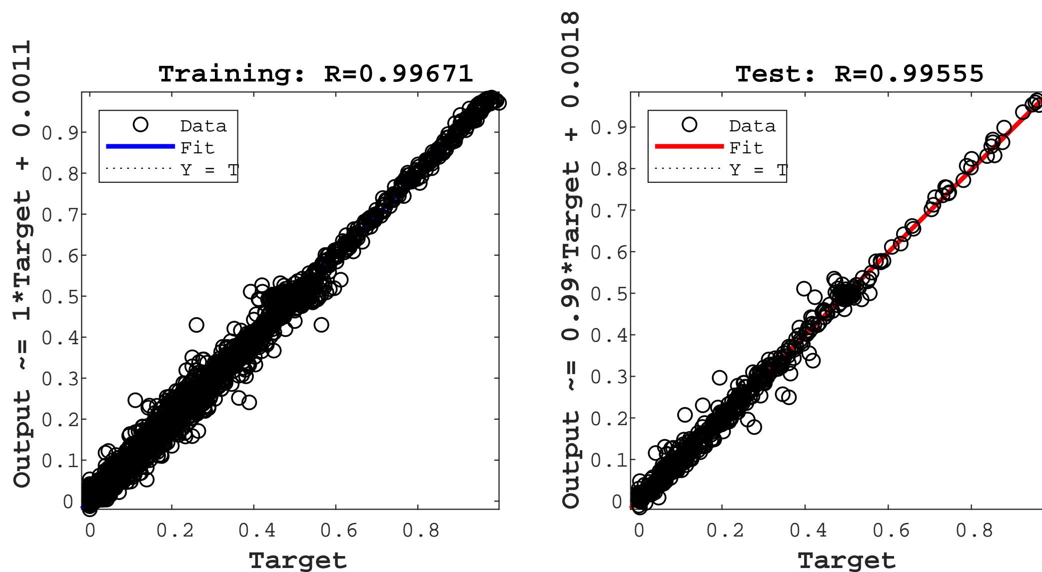
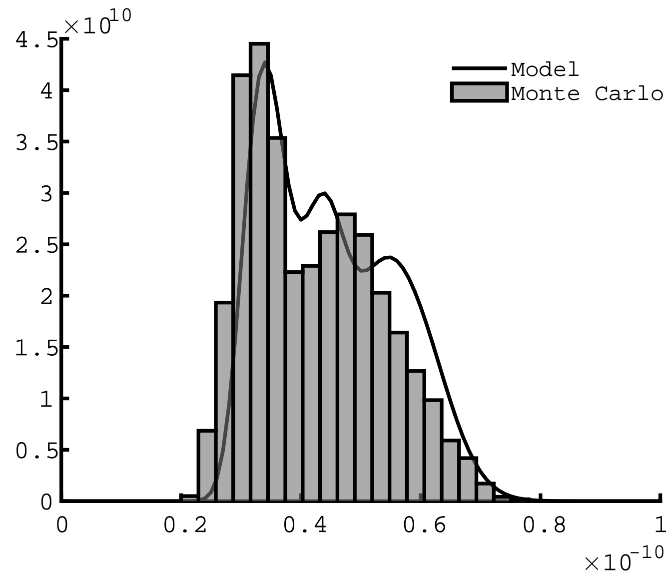
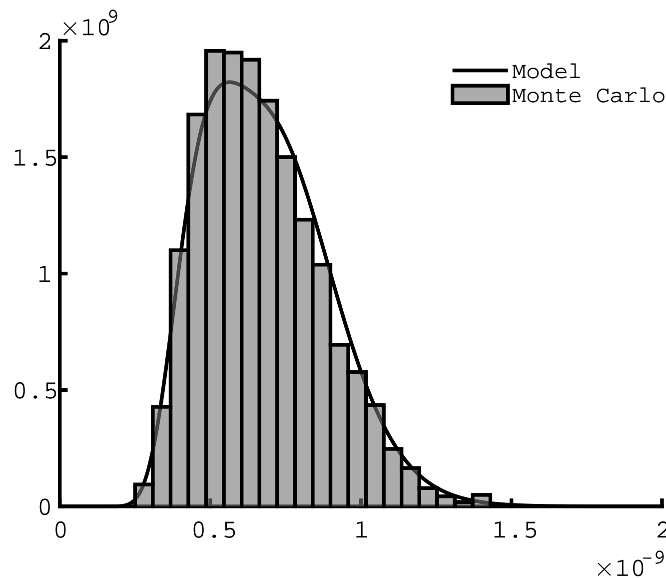


Figure 4.22: Regression diagrams of the switch event estimation trained model

With the two *ANN* models, the whole model of *S2* can be obtained. Figure 4.23 depicts some predicted and simulated propagation profiles for a single *NAND* gate test on a randomly generated 500-bit input vector.



(a) $V_{dd} = 0.70$ V, $C_{load} = 0.65$ fF, $C_{ina} = 1.82$ fF and $C_{inb} = 1.04$ fF



(b) $V_{dd} = 0.42$ V, $C_{load} = 1.50$ fF, $C_{ina} = 1.27$ fF and $C_{inb} = 0.92$ fF

Figure 4.23: Single *NAND* test, two different setups, 500 bits uniformly distributed random input vectors were simulated for each setup

The results show the great prediction accuracy (*MSE* $= 0.2554$ and *R-Square* $= 0.844$ for Figure 4.23a and *MSE* $= 0.0152$ and *R-Square* $= 0.952$ for Figure 4.23b) of the model operating in both near-threshold (weak inversion) and strong inversion regions ($V_{th} \approx 0.43$ V). As can be seen from the figures, the impact of V_{th0} variation is bigger when operating in the sub-threshold region

as the propagation delay deviation is much larger (wider) in Figure 4.23b. It is also worth mentioning that for a single *NAND* gate, a *SPICE* simulation takes around 30 minutes to complete, while our model uses less than 1 second on an average performance laptop PC.

For demonstration simplicity, the propagation of the P_τ model through the circuit, a modified *block-based SSTA* algorithm is applied with the proposed gate model on a 2-*NAND* chain circuit shown in Figure 4.24.

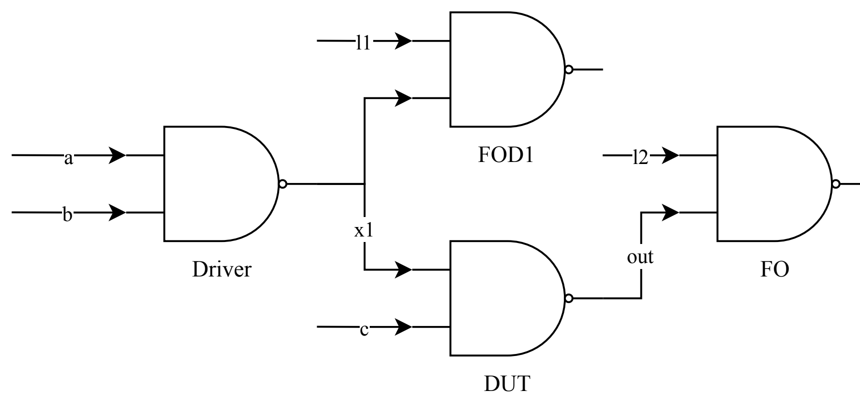


Figure 4.24: The simulated *NAND-chain* circuit

The circuit has 3 inputs a, b and c, which are fed with a 500-bit randomly generated 3-dimensional vector sequence. The setup parameters of the circuit are described in Table 4.11.

Table 4.11: *NAND* chain circuit setup

	V_{dd}	C_{load}	C_{in1}	C_{in2}
x1	0.45 V	0.3 fF	0	0
out	0.45 V	0.15 fF	0.15 fF	0

Figure 4.25 depicts the prediction results and comparison with those of *MC* simulation. The resulting P_τ of the first *NAND* gate (*MP*$_\tau$(*x1*), input *a*, *b*) is straightforward and is similar to the results in the previous section, while the statistical sum (discrete convolution in this case) must be performed when calculating the P_τ of the second *NAND* gate (*MP*$_\tau$(*out*), input *x1*, *c*).

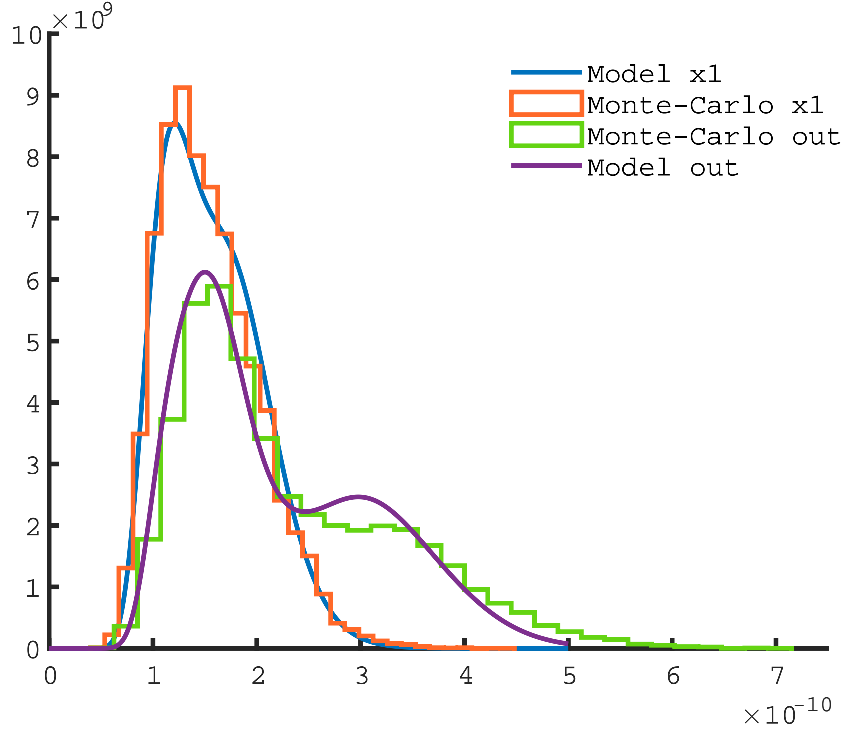
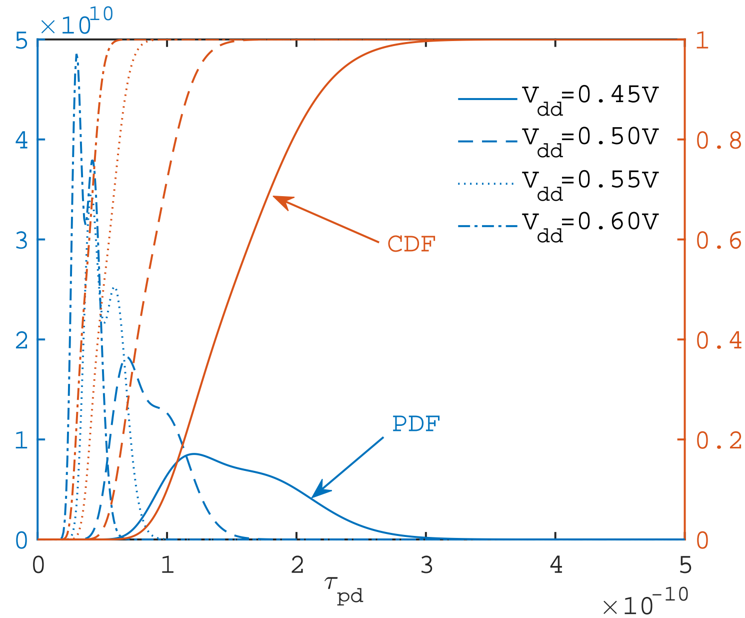


Figure 4.25: Model prediction vs. *MC* @ $V_{dd} = 0.45$ V, simulated on 500 randomly generated input vectors, *SPICE* runtime: > 1 h, Model runtime: ~ 1 s

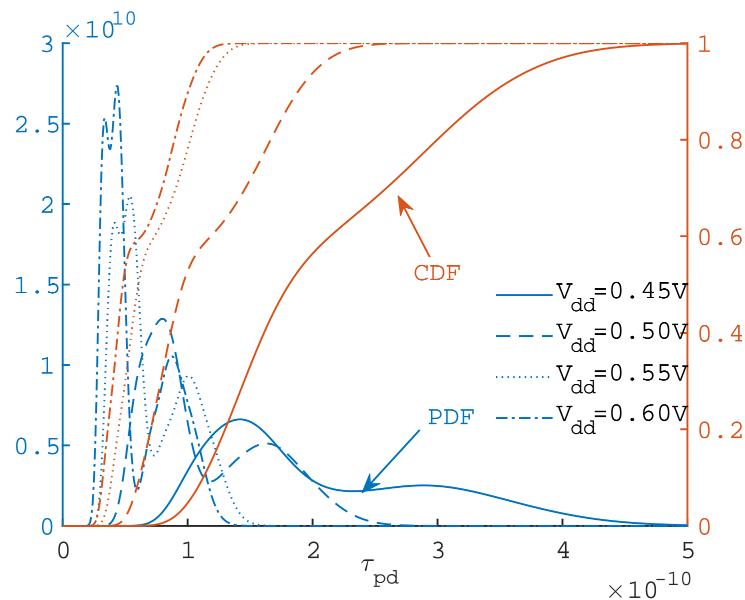
Equation 4.32 describes the algorithm of propagating the P_τ from the first *NAND* gate to the second. Similar to Equation 4.31 $MP_\tau(out)$ is composed of two parts. The first part is measurements of the propagated arrival time from *PI* *a* or *b* to *out*, while the second part is measurements of the arrival time directly from *PI* *c* to *out*. As can be seen from the equation, the traditional statistical *MAX* operation in a typical block-based SSTA algorithm is replaced by the mixture distribution, which makes the estimation less pessimistic.

$$\begin{aligned} MP_\tau(out) = \sum P(X) \cdot CONV(MP_\tau(x1), P_\tau(out)|_X) \qquad (4.32) \\ + \sum P(Y) \times P_\tau(out)|_Y, \\ X \in \{11 \to 01, 11 \to 10, 11 \to 00\}, \\ Y \in \{01 \to 11, 10 \to 11, 00 \to 11\} \qquad (4.33) \end{aligned}$$

Figure 4.26a and Figure 4.26b show the predicted *PDF* and *CDF* curves of the two outputs *x1* and *out*, respectively at varying V_{dd} values.



(a) The predicted *x1 PDF* and *CDF* curves for varying V_{dd} values



(b) The predicted *out PDF* and *CDF* curves for varying V_{dd} values

Figure 4.26: The results of the *NAND-chain* simulation

The *CDF* curves are the significant tool for the *Multi-objective Optimization* during the circuit optimization phase, as the timing related reliability can be observed on a *CDF* curve at certain time called "cut-off delay".

4.5 Chapter Summary

In this chapter, two *ANN-based SSTA* gate modelling strategies were introduced. The first strategy has a lower abstraction level, based on propagation delay τ_{pd} estimation and *MC* simulation, while the second strategy has a higher abstraction level, which directly extracts the *IGD* parameters from the τ_{pd} profile simulated using strategy $S1$.

Strategy $S1$ is more flexible, due to the nature of the flexibility and maturity of the *MC-based SSTA* approaches, while dramatically improving the current *SPICE-based MC* simulation, by order of 10^5 times. Taking advantage of this flexibility, this approach can be used in almost any case where the traditional *MC* simulation is suited. In addition, it is not limited to a certain type of distribution. It can be easily extended to include more variables, to deal with the correlated variation sources and spatially correlated devices for moderate scale circuit *SSTA*.

Strategy $S2$, on the other hand, is more limited in the use cases, as it can only capture the profile which is generated by the *MC* simulations. Also, it is more complicated and trickier to employ the existing *SSTA* approaches, such as the *path-based* and *block-based SSTA* algorithms, especially when dealing with the cases considering the variation dependencies and spatial correlations. However, when these limitations can be safely neglected in some fast estimate circumstances, this strategy can perform well and further improve the performance of $S1$ by several orders of magnitude, which makes the approach very suitable for very large scale circuits *SSTA*.

This chapter is based on the following papers:

- Bo Yang, E. Popovici, M. A. Quille, A. Amann and S. Cotofana, "A supply voltage-dependent variation aware reliability evaluation model," 2016 IEEE/ACM International Symposium on Nanoscale Architectures, Beijing, 2016, pp. 79-84
- Bo Yang, Sorin Cotofana, Emanuel Popovici, "A novel ANN-based depen-

dent statistical static timing analysis combinational gate model," submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020

Chapter 5

Synthesizing an *MLP-based* Combinational Logic Circuit

In the previous two chapters, circuit reliability issues at different levels of abstractions in the design flow have been discussed. Specifically, the fast evaluation and estimation of the effects of variabilities and faults is investigated, which is the direction of designing efficient circuits under uncertainties. Another avenue of inquiry can be the investigation of circuit architectures that inherently have the tolerance to the variabilities and faults or more predictable performance variations while maintaining efficiency in other parameters (e.g., area or performance). To follow the later line of inquiry, the possibility of implementing a new circuit design scheme is investigated, which implements the logic circuit model with the *Multilayer Perceptron* (MLP). As the *MLP-based* logic circuits may have welcomed properties such as the built-in ability of fault tolerance, smaller in size, and smaller logic depths, etc.

While the *MLP*s capability of approximating some simple Boolean functions has already been reported in [66, 67], etc., to the best of our knowledge, there is no previous work attempting to discuss the implementation of the complex logic cells from an *EDA* perspective. Thus, in this chapter, many aspects of the *EDA* considerations for implementing the *MLP-based* combinational logic circuits will

be discussed.

The concept to implement complex combinatorial logic cells using *MLP VLSI* circuits will be investigated in this chapter, as the *MLP*s can approximate much more complex Boolean functions and feature high computational efficiency in terms of area and power consumption once efficiently implemented on hardware. To prove the concept, the toolset written in *C++* and *Python* was created.

Figure 5.1 shows the proposed circuit synthesis flow. The underlying circuit synthesis process is an *MLP-ANN* model training process. In order to train the network to approximate a Boolean function, a training dataset containing the samples of the input-output vector pair is needed. As shown in the figure, the circuit *Register Transfer Level* (RTL) description (e.g., a ".v" file for *Verilog*) will be taken as input to the flow. The circuit will be functionally simulated, based on the given input vectors, either randomly generated or sequentially enumerated, where the target output vectors will be derived, the training dataset will be written to a python archive file (".npz"). Then the *MLP* architecture in terms of the number of layers and layer configuration will be predicted before training the network. The training and optimisation process could be an iterative procedure which may perform several trials so that the simplest network can be achieved. Along with the training algorithm, techniques such as *Regularization*, *Pruning*, and *Quantization* techniques will be applied to optimize the network, so that the trained network model is more suitable for hardware implementation.

Note that, as was described in Section 1.3, the *Sigmoid* function is used as the activation when training the MLP models in this work. Normally, a *Sigmoid*-like function can be achieved with an amplifier circuit, such as the one proposed in [119].

The contents of this chapter will be organized as follows. Some typical *ANN VLSI* architectures, particularly those suitable for this work, will be reviewed, and some general hardware implementation considerations will be reviewed in Section 5.1. The *MLP* Boolean function approximation and the efficient learning

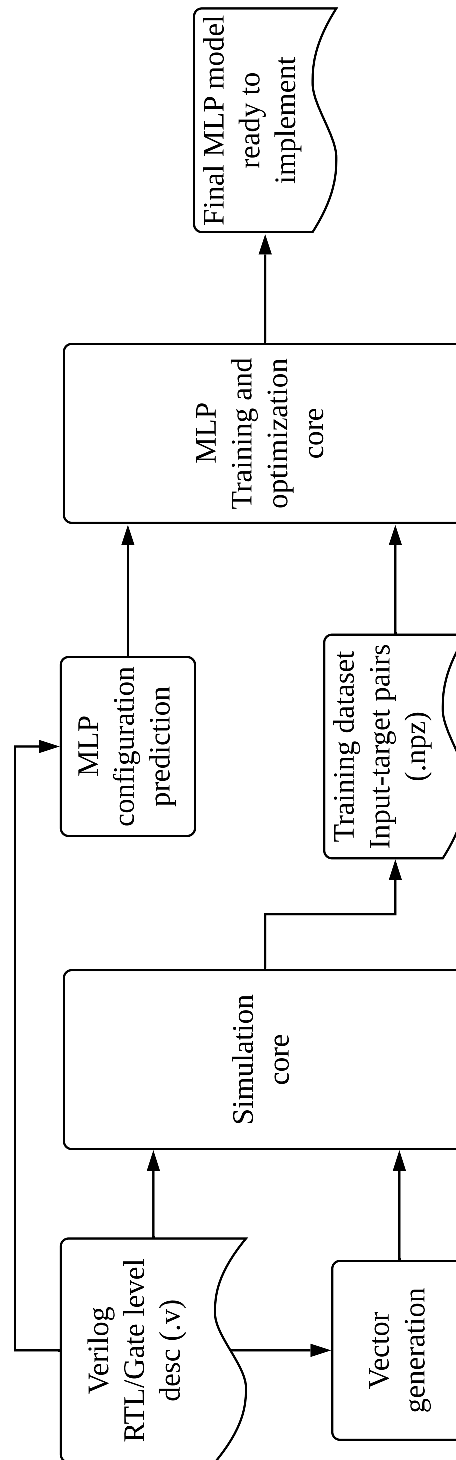


Figure 5.1: The *EDA* flow for synthesizing an *MLP-based* logic gate

algorithms will be discussed in Section 5.2. In Section 5.3, the issues that should be considered when transferring the trained model to silicon implementation will be discussed. Finally, Section 5.4 summarizes the work by indicating the contributions and the future work of this framework.

5.1 *ANN VLSI* Implementations

The *ANN*s are commonly implemented on digital *VLSI* circuits (*GPU*s, *ASIC*s, and *FPGA*s) or analogue circuits [120, 121]. However, a digital *VLSI* circuit based *ANN* for implementing combinatorial *VLSI* circuits cannot avail of large computational power but is constrained by latency as well as the area of implementation.

In most of the analogue *ANN* architectures, the fundamental properties of the devices or circuits are used to accomplish the large scale parallel *ANN* system computation, the synaptic and neuronal operations are encoded by the current or voltage signals. For example, the *ANN* architectures based on resistive or capacitive components may employ the *Ohm's Law* or *Charge Conservation Law* to perform the *Multiply-Accumulate* (MAC) operations, and the transfer function of a simple *CMOS Inverter* can be used as a *Heaviside* activation function in an artificial neuron.

While analogue circuit based *ANN*s have great advantageous over the digital counterparts, the following challenges still exist [121]:

- Poor noise margin, a compromise between robustness and circuit complexity;
- The fabrication can hardly reproduce the desired design parameter precision;
- Strong variability sensitivity, in terms of power, voltage, and temperature (PVT);
- Most artificial neuron implementations have static power consumption;

- Some architectures rely on new advanced devices or technologies, which are more expensive and commercially immature.

More efforts on the hardware design are indeed necessary to overcome the above challenges, and some improvements can be achieved from optimizing the *ANN* models, which will be the main topics of discussion in this thesis. Some particular promising neuromorphic computing architectures and devices that are suitable for this work will also be introduced.

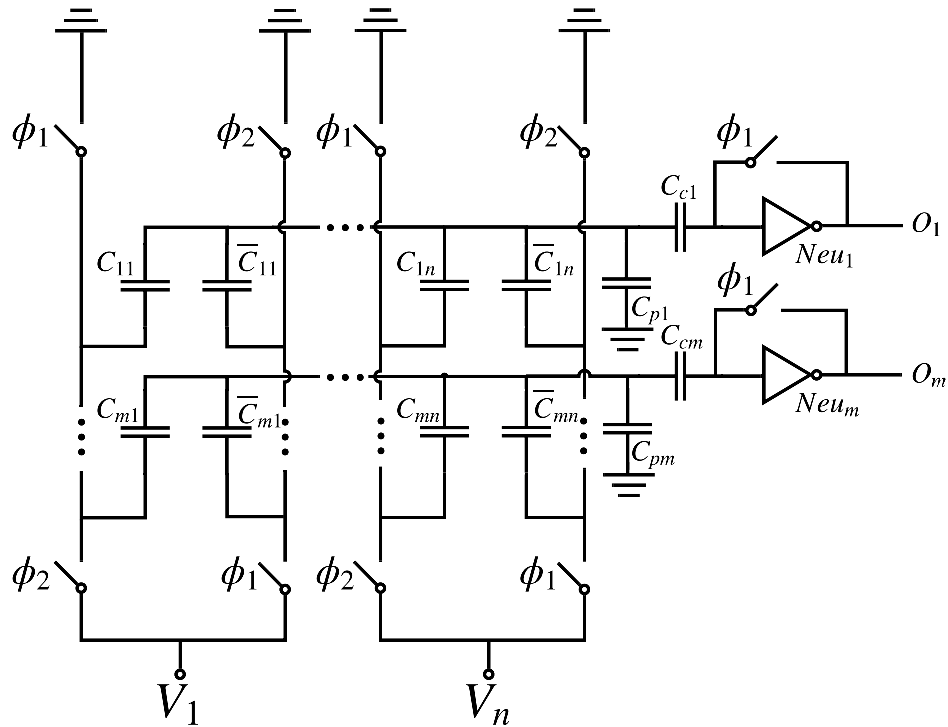
As the most significant factor affecting the efficiency of an *ANN* hardware architecture is the synaptic computing system (matrix *MAC* operation), when discussing different architectures, we focus only on the synaptic computation circuit while assuming a properly implemented neuron circuit is available.

5.1.1 Capacitive Network

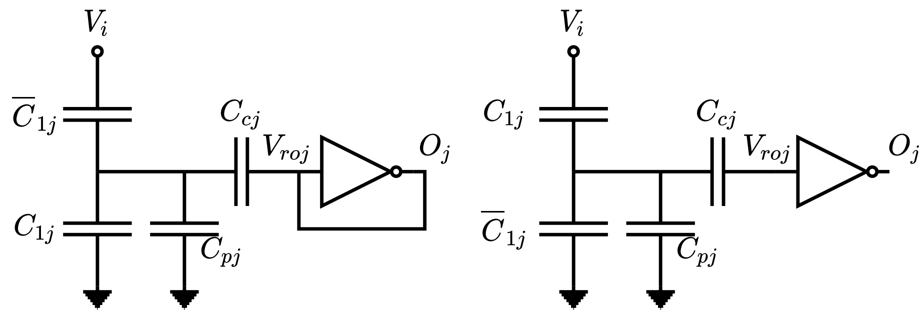
A pure capacitive synaptic computing network was proposed in [122]. The simplified capacitor network-based *ANN* layer architecture is shown in Figure 5.2a.

In this architecture, a capacitive network is used to perform the synaptic computation, and an *Inverter*-based comparator is used as a *Hardlim* neuron. The network forms an array of $M \times N$ cells, where each cell contains a pair of capacitors C_{ij} and $\overline{C}_{ij}$.

The system is controlled with a two-phase clock signal, where ϕ_1 and ϕ_2 denoting the precharge phase and the charge redistribution/evaluation phase, respectively. Shown in the left Figure 5.2b is the extracted single cell operating in phase ϕ_1, and in the right Figure 5.2b the phase ϕ_2. During phase ϕ_1, the synapse output V_{ro} is clamped at the switching threshold of the *Inverter* (ideally, $\frac{V_{dd}}{2}$, due to the negative feedback), and the capacitors are charged. Once switched to phase ϕ_2, the positions of C_{1j} and $\overline{C}_{1j}$ are swapped. Hence, the charge in the system is redistributed, resulting in the value of V_{ro} becoming that of Equation 5.1. Consequently, the *Inverter* output will reach the final value



(a) Simplified circuit



(b) Two-phase operation

Figure 5.2: Capacitive network *ANN* architecture [122]

depending on the value of V_{ro}.

$$V_{roj} = \frac{(C_{ij} - \overline{C}_{ij})x_i}{K_j}, \tag{5.1}$$

where K_j is a positive scale factor for coupling capacitors.

The main drawbacks of this implementation are the static power consumption and that the size of the circuit is dependent on the capacitances. But the advantages are also clear with its capability to calculate both positive and negative

weights inherently without additional devices as well as the acceptable rational accuracy of integrated capacitors. In addition, this architecture can be implemented completely using the current design flows for *CMOS* technology.

While this architecture is based on the traditional *CMOS* circuit, the following technologies make use of the novel analogue non-volatile memory devices as the basic synaptic computing elements.

5.1.2 Charge Trapping Devices

The *Floating Gate Transistor* (FGT) [125] concept was originally used for *FLASH* memory and later found to be also useful in the analogue domain [126]. Recently, similar devices, *Silicon Oxide Nitride Oxide Silicon* (SONOS) [127, 123] and *Charge Trap Transistors* (CTT) [128, 129], are attracting increasingly more interest, particularly for *ANN*s, due to their simpler fabrication processes. Both devices have similar principles of operation, that makes use of the so-called *Charge Trapping Mechanism*, which is typically an unwanted phenomenon in the standard *MOS* technology. From the $I - V$ characteristics of an *NMOS* operating in the sub-threshold region:

$$I_{DS} = I_s \frac{W}{L} \text{exp} \left(\frac{V_{GS} - V_{th}}{nU_T} \right), \tag{5.2}$$

where I_s is the characteristic current, W and L are the width and length of the transistors, respectively, n is the subthreshold slope factor, and U_T is the thermal voltage. When a charge Q_t was trapped at the gate surface, and given the gate capacitance to be C_t, Equation 5.2 becomes Equation 5.3

$$I_{DS} = I_s \frac{W}{L} \text{exp} \left(\frac{Q_t - C_t V_{th}}{nU_T} \right) \text{exp} \left(\frac{C_t V_{GS}}{nU_T} \right) \tag{5.3}$$

Let $w = I_s \frac{W}{L} \text{exp} \left(\frac{Q_t - C_t V_{th}}{nU_T} \right)$ be a weight term that can be programmed by tuning the amount of charge trapped at the gate surface.

Figure 5.3 depicts a typical simplified charge trapping device based *ANN* layer

architecture. Again the synaptic matrix contains $M \times N$ cells, each being a single charge trapping *MOSFET*. The input is the voltage signal taken from the gate of the transistor, while the weight has been internally stored in the transistor, the multiplication operation $w_{ij} \times x_i$ encodes the signal to current, and the accumulation can naturally be done by the *Kirchhoff's Current Law* (KCL).

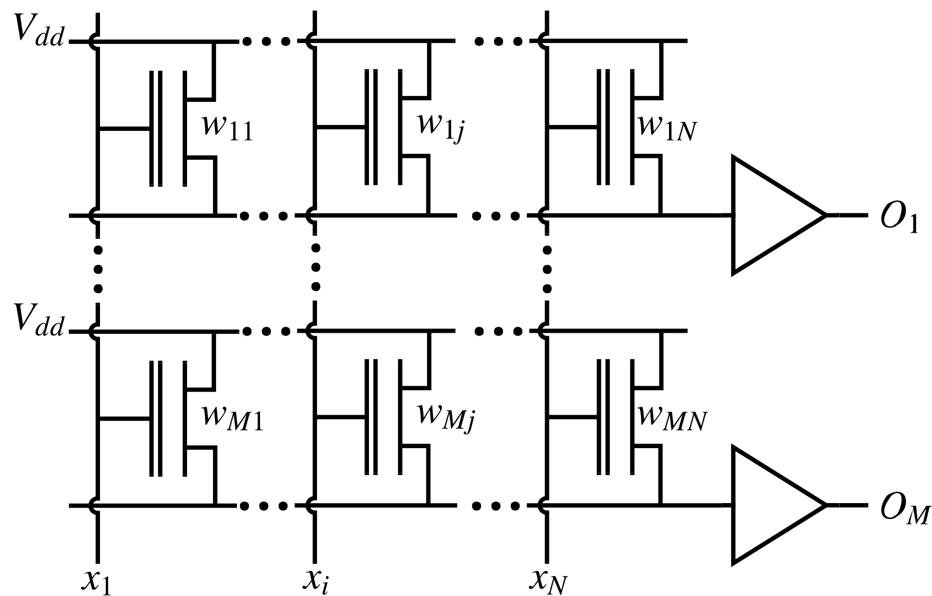


Figure 5.3: Charge trapping device *ANN* architecture [123]

The low power consumption feature of their implementation with *SONOS* devices was reported by [123], and [129] claims that this implementation has a very high density of integration and is *CMOS* process compatible. These valuable features of charge trapping devices based *ANN* make the device very attractive for our work.

In addition to the *charge trapping device*, there are more non-volatile memory devices under sustained research, and the representative ones are *Memristor* or more generally *Resistive Random Access Memory (RRAM)* and *Phase-Change Memory (PCM)*, etc. Among them, *Memristor* is attracting increasingly more interests and is known as one of the most promising types of devices for neuromorphic computing.

5.1.3 Memristors

Memristor is a non-linear device, of which the $V-I$ characteristics is following the equation:

$$v = M(q) \cdot i, \tag{5.4}$$

where $M(q)$ is the charge-controlled memristor given by

$$M(q) = \frac{d\phi(q)}{dq}, \tag{5.5}$$

where ϕ is the magnetic flux. As indicated by the equations, the memristor resistance is varied depending on the historical charge passing through the device, and this characteristic is kept even in power-down mode.

The synaptic operation using memristors can be implemented as a resistive network, as shown in Figure 5.4.

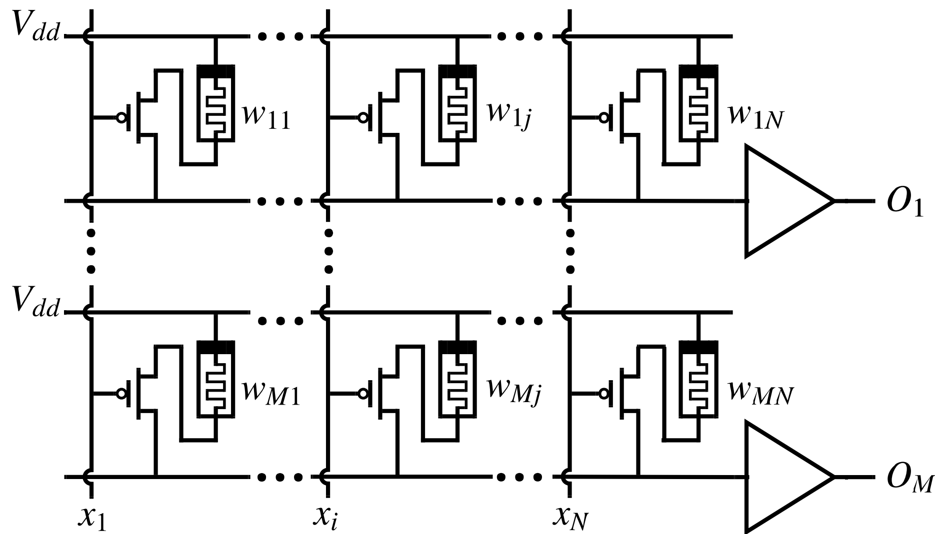


Figure 5.4: Memristive network *ANN* architecture [124]

The synaptic architecture proposed by [124] is similar to the charge trapping device based architecture shown in Figure 5.3, with the charge trapping device in each cell replaced by a *1-Transistor-1-Memristor* (1T1M) element. The weights are stored in the memristor, and each memristor is gated by a *P-type*

MOSFET controlled by the corresponding input. Then, the *MAC* operation is performed, making use of *Ohm's Law* and the *KCL*. Although the memristor is a type of device that has great potential to have better integration density than the *CMOS* technology, as an emerging device, the memristor is not mature enough for fabrication and mostly non-*CMOS* compatible.

Table 5.1 summarizes the features of the three architectures. Notably, the programmable precisions of all three types of devices are sufficient for the precision requirement of the networks after quantization, which will be detailed in Section 5.3.2.

Table 5.1: Comparison between the architectures

	Capacitive network	Charge trapping device	Memristor
Technology	Conventional Circuit	NVM based	RRAM
Density	Low	Medium	High
Speed	Low	High	High
CMOS compatibility	Standard *CMOS* & Design flow compatible	*CMOS* NVM compatible	Mostly non-*CMOS* compatible
Training algorithm	Normal	Special	Normal
Programmable precision	8 bits [122]	10 bits [126]	8 bits [130]

5.2 *MLP* Universal Boolean Function Approximator

5.2.1 Boolean Function Approximation using *MLP*

As has been described in section 2.6, a single layer *ANN* can be used as a universal function approximator for any "well-behaved" continuous function. Similarly, the *MLP* is also a universal Boolean function approximator [66]. It has been reported in the literature that the basic logic gates *NOT*, *AND/NAND*, *OR/NOR* can be approximated by a single-neuron perceptron, while the *XOR/ XNOR* gate must be approximated with at least a 2-layer perceptron [67].

Table 5.2 lists five simple logic gates, their Boolean expressions, and their *Hasse Diagram*[131]. In a Hasse Diagram, the white circles represent the output "*0*" of the circuit, and the black circles represent the output "*1*".

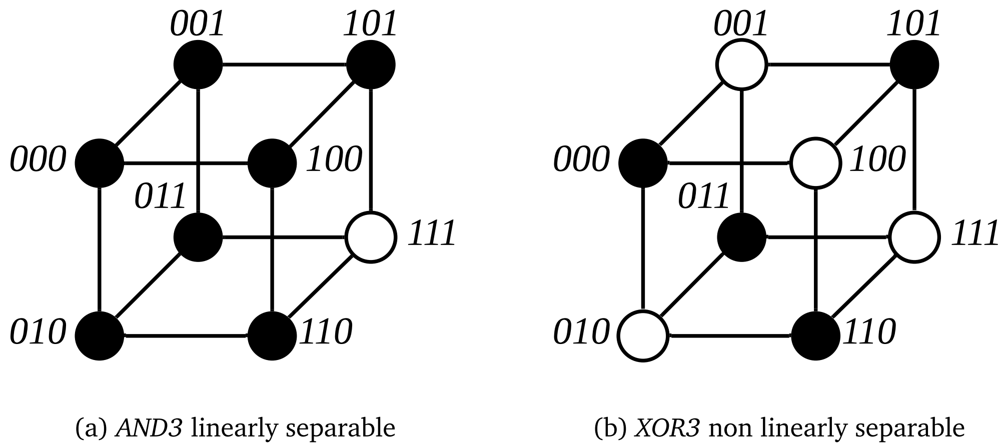
Note that in the last column of the table, the outputs of the first four functions have a common property that is the "1"s and "0"s are linearly separable which means that the output values can be divided into two groups with a single straight line, while this is not possible for the *XOR* gate. In general, a single perceptron can approximate the linearly separable Boolean functions (i.e., *monotonic Boolean functions*[132]) with an arbitrary number of inputs, while non-monotonic Boolean functions, i.e., *XOR*, needs at least a 2-layer *MLP* constructed from 2 neurons in the hidden layer and 1 neuron in the output layer.

Table 5.2: Basic logic gates and their representations

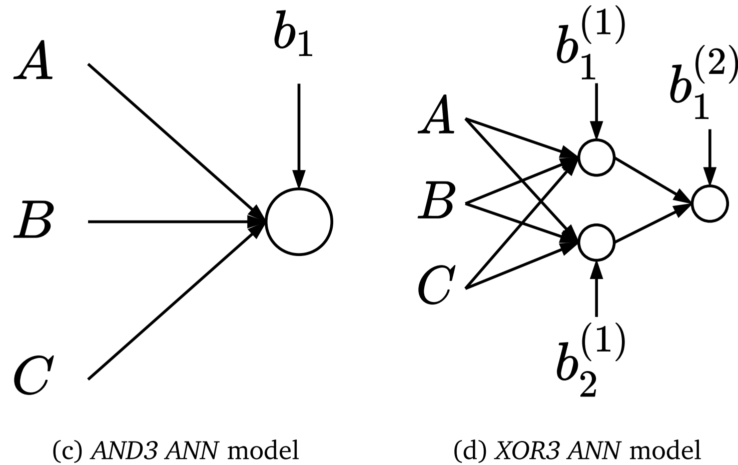
Logic	Boolean expression	Hasse diagram
NOT	$Y = \bar{A}$	0, 1
AND	$Y = A \cdot B$	00, 01, 10, 11
OR	$Y = A + B$	00, 01, 10, 11
NAND	$Y = \overline{A \cdot B}$	00, 01, 10, 11
XOR	$Y = A \cdot \bar{B} + \bar{A} \cdot B$	00, 01, 10, 11

This principle holds for logics that have more than 2 inputs, as shown in Figure 5.5, there exists no such a plane that can linearly cut the black and white nodes into two groups for the *XOR3* gate.

In order to predict the structure of the *MLP* that can perform a specific Boolean function, the worst-case usage of layers and neurons is investigated. The worst case occurs when the function is completely constructed by *XOR* gates (can not be further reduced), if arranged in a 1-hidden and 1-output layer *MLP*, $O(2^{N-1})$ neurons hence $O(N2^{N-1})$ parameters are needed. The exponential relationship between the number of network parameters and the number of inputs makes



(a) *AND3* linearly separable (b) *XOR3* non linearly separable



(c) *AND3 ANN* model (d) *XOR3 ANN* model

Figure 5.5: The diagram graph of *AND3* and *XOR3*

approximating larger Boolean functions using the *MLP* impractical. Fortunately, by stacking more layers, the number of neurons and network parameters can be drastically reduced.

Table 5.3 depicts the trained *ANN* models for *XOR* gates with varying numbers

of inputs under 2-layer and/or 3-layer configurations. Interestingly, the trained *XOR* models show that the neurons needed in a 1-hidden layer network are much less than the theoretical analysis given by the $O(2^{N-1})$. Table 5.4 indicates that the hidden layer neurons can automatically categorize the inputs in terms of the number of 1s, which leads to a best-case of only $O(log_2^N)$ neurons being needed in the hidden layer to implement an N-input *XOR* logic with a "1-hidden-

Table 5.3: Trained *XOR* models with 2, 3, 4, 6, 8 and 10 inputs under 2- and/or 3- layer configurations

N inputs	Layer setup			
	Two layers	N params	Three layers	N params
2	<2, 1>	9	-	
3	<2, 1>	11	-	
4	<3, 1>	19	-	
6	<5, 1>	41	<3, 2, 1>	32
8	<16, 1>	161	<6, 4, 1>	87
10	-		<10, 6, 1>	183

1-output" MLP, although the best case is getting more difficult to reach as N is growing.

Table 5.4: *XOR3* layer outputs

Input			Hidden		Output
a	b	c	n1	n2	o
0	0	0	0.27	0.26	0
0	0	0.03	0.20	0.006	0.3
0	0.03	0	0.20	0.006	0.3
0.03	0	0	0.20	0.006	0.3
0	0.03	0.03	0.1	0.29	0
0.03	0	0.03	0.1	0.29	0
0.03	0.03	0	0.1	0.29	0
0.03	0.03	0.03	0.03	0.3	0.3

It is also obvious from Table 5.3 that by adding sufficient layers, the size of the *MLP* can be effectively reduced, at the cost of increased propagation delay. In other words, the trade-off is exhibited between the size of the *MLP* and the propagation delay when implemented as hardware.

5.2.2 Learning Algorithm for Boolean Logic Function

There are many learning algorithms exist. In this work, some representative training algorithms were investigated to train the *MLP*, and their performance in terms of the possibility of convergence, convergence speed, the resultant network configuration, and accuracy are compared.

Table 5.5 lists the training results on tested *MCNC* benchmark circuits with some typical optimisation algorithms. As the training performance may vary depending on the network parameter initial values, thus in order to compare the possibility of convergence, the third column presents the number of convergence out of 20 experiments. All the listed results are trained on an average performance laptop PC.

Table 5.5: Training algorithms comparison on benchmark circuits

Circuit	Algorithm	Convergence (/20)	Layer Config	Speed	*MSE*
AND2	*LM*	20	<1>	1.0	4.33×10^{-16}
	BFGS	17	<1>	1.0	$< 10^{-16}$
	Adam	20	<1>	8.0	1.11×10^{-9}
AND3	*LM*	20	<1>	1.0	8.26×10^{-16}
	BFGS	11	<1>	1.0	$< 10^{-16}$
	Adam	20	<1>	8.2	1.52×10^{-9}
XOR2	*LM*	18	<2, 1>	1.9	2.58×10^{-16}
	BFGS	11	<2, 1>	1.3	1.03×10^{-11}
	Adam	12	<2, 1>	8.5	6.27×10^{-14}
XOR3	*LM*	18	<2, 1>	18.3	8.18×10^{-16}
	BFGS	9	<3, 1>	12	5.14×10^{-13}
	Adam	11	<2, 1>	9.5	1.18×10^{-13}
Full Adder	*LM*	17	<2, 2>	28.6	4.76×10^{-16}
	BFGS	8	<3, 2>	9.7	$< 10^{-16}$
	Adam	18	<2, 2>	15	1.04×10^{-8}
b1	*LM*	16	<3, 3>	1.5	4.43×10^{-16}
	BFGS	7	<6, 3>	2.2	1.82×10^{-14}
	Adam	11	<5, 3>	18	2.33×10^{-9}
t481	*LM*	16	<10, 6, 1>	963.7	7.87×10^{-16}
	BFGS	-	-	-	-
	Adam	12	<20, 15, 1>	246	5.53×10^{-9}
cm151a	*LM*	12	<6, 4, 2>	428.7	9.2×10^{-16}
	BFGS	-	-	-	-
	Adam	14	<10, 8, 2>	363.8	1.86×10^{-9}
cu	*LM*	15	<4, 8, 11>	1802.4	4.46×10^{-16}
	BFGS	-	-	-	-
	Adam	8	<6, 10, 11>	531.5	5.38×10^{-7}

It is obvious that the *Levenberg-Marquardt* (LM) algorithm has greater capability in terms of learning Boolean logic, while the typical first-order algorithm, *Adam*, although faster, can hardly achieve the comparable accuracy on more complex Boolean functions. The biggest logic circuit tested is the *MCNC* benchmark com-

binatorial circuit *t481* with 16 inputs and 1 output, composed by approximately 1842 *CMOS* gates (synthesized by *ABC*). A 2-hidden layer network is trained to approximate the logic, totalling 243 parameters, and 17 artificial neurons.

It is worth mentioning that the above networks are trained using datasets that are the complete set of input vectors, meaning that all 2^N samples for an N-input circuit are used to train the network. One of the useful functionalities of an *ANN* approximator is its capability of predicting the "unseen" data. This holds true for the *MLP* Boolean function approximator. Experiments show that most of the tested complex functions can be trained using only $\approx 45\ \%$ samples without degradation in testing accuracy.

5.3 Model Transfer Considerations

The proposed *EDA* flow synthesizes the logic circuit by training the *MLP* "off-silicon", in other words, the *ANN* is trained based on the *ANN* circuit physical or mathematical model before the trained network parameters are transferred to the real silicon implementation.

As discussed in the previous section, the network parameters are denoted by one or more of the circuit component design parameters. Therefore, the design and implementation mismatch severely affects the efficiency, reliability, or even usability of the final integrated circuit.

The main considerations are as follows:

Firstly, it is beneficial to have a network with a simpler architecture and smaller parameters, which lead to minimized model overfitting as well as less energy consumed and/or less area taken by the implemented circuits. For example, for the circuit architecture in Figure 5.2a, a simpler network translates to simpler switch capacitor network and a smaller amount of capacitors used, and smaller network parameters values mean smaller sized capacitors and less charge to be transferred in the circuit.

Secondly, while the 32-bit floating-point numbers are commonly used in the *ANN* training algorithms, however, the network implementation can only work with much lower precision numbers due to the programming precision limitation of the devices. This induces the most mismatch between design and implementation. Thus the network parameters precision has to be properly reduced to meet the device's programming precision specifications.

Last but not least, with the inevitable variations in the fabrication process, voltage, and temperature (PVT), the exact value of design parameters can hardly be met precisely. In addition, the device and circuit models for "off-chip" training can hardly be exactly accurate, leading to a model-silicon mismatch.

5.3.1 Regularization and Pruning

Regularization is a technique widely used to overcome the so-called overfitting when training an *ANN*. Overfitting is commonly caused by one or more of the following:

- The amount of training samples is too small;
- The network is too complex, i.e., the size of the network is so big that unnecessary details will be learned by the network;
- Large network parameters exist, which make the network oversensitive to small network input changes;
- The training is over-performed.

Accordingly, many regularization techniques have been used to constraint the network training process. *Bayesian Regularization* (BR) [133] is a technique based on the Bayesian prediction, which is able to reduce the effective complexity of the *ANN*. *Dropout* [134] is another methodology that prevents the network from being overly dependent on some specific network parameters. *L1 Regularization* works as a feature selector, which is attempting to screen the most relevant features, and *L2 Regularization* is trying to confine the parameter's value by penalizing the large ones [135]. *Early stopping* [136], i.e., stop

the training process once the verification loss is greater than the training loss, is commonly used to prevent the overtraining.

BR is applied with the *LM* training algorithm to help reducing the complexity of the network. The terminology *effective parameter* is used to indicate the number of parameters that effectively affects the capability of the network, i.e., the fewer the effective parameters exist, the simpler the network is. The third column of Table 5.6 shows the effective number of parameters trained with *BR*.

It is obvious from Table 5.6 that the trained network has a relatively smaller number of effective parameters than the total number of parameters trained with the raw *LM* in Table 5.5. Notably, the circuit *b1* was trained using two-layer configurations with a different number of total parameters. However, the number of effective parameters is relatively constant. In other words, the number of effective parameters may also suggest the simplest network configurations.

In addition, *Pruned Neural Network* has been investigated in order to reduce the parameters and connections in the *ANN*. Many *ANN* pruning methodologies have been proposed in the literature [137, 138, 139, 140]. For demonstration purposes, the passive pruning approach introduced in [137] that directly zero out the parameters that are much smaller than others in each layer while keeping the accuracy from degrading significantly, was investigated in this work. The number of parameters in the network before and after pruning is shown in the fourth column of Table 5.6. Results show that the pruned networks have $10\ \% \sim 20\ \%$ reductions in the number of parameters compared to their original counterparts with negligible accuracy degradation.

Moreover, the estimated number of components used by *MLP* architectures (Section 5.1) and their standard *CMOS* counterparts are shown in the fifth and sixth columns of Table 5.6. Note that in the table, the number of components used by *MLP* architectures is estimated by assuming that 10 transistors are needed by each artificial neuron and that each standard *CMOS* equivalent logic gate contains 4 transistors.

Table 5.6: Training results using the *BR* and Pruning and the estimated number of components

Circuit	Total params	Effective params	Network Pruning	N components (Est.)	
				MLP (this work)	*CMOS* (standard)
b1	24	14	16	76	52
	61	17	42	-	-
	73	16	51	-	-
z9sym	20	20	18	50	1000
cm151a	116	103	102	219	132
t481	243	224	225	395	7368
cu	199	188	172	402	192
clip	183	174	171	391	816
misex3	1284	1167	1034	1774	6868

It can be seen from the estimations that the *MLP* architecture of logic circuits such as *z9sym*, *t481*, and *misex3* achieved the best efficiency in terms of the number of components (hence area and energy consumption) because the complex logic circuits were implemented by relatively small *MLP*s. These results show a promising way to dramatically improve the cost-effectiveness of the combinatorial logic circuits.

5.3.2 Parameter Quantization

The *Quantized Neural Network* (QNN), i.e., reduced precision *ANN*s, became a very attractive area of study in recent years as a consequence of the development of the *ANN* in hardware. This technique is originally introduced to conduct efficient neuromorphic computation on digital hardware in terms of memory size, latency, and energy consumption. The main benefit of using the quantized neural network in this work is that the process technology precision specifications can be significantly reduced. Hence, more robust analogue *ANN* hardware can be created.

Methodologies that quantize the parameters of the trained (full precision) *ANN* when inferencing, are reported in [141, 142, 143, 144, 145]. On the other hand, methodologies such as [146] and [147] attempt to fully train the *QNN*, which may achieve competitive prediction performance compared to its full precision counterpart. However, the quantization leads to discontinuities between successive training iterations. As a result, new training algorithms working with quantized parameters, activations, or even gradients must be invented. Otherwise, the quantization may induce convergence difficulties. Among the methods, the evolutionary methodology proposed in [144] was investigated and turned out to be effective for quantizing a trained *MLP* Boolean approximator. With this method, the network is trained with full precision parameters, then a small portion of parameters are rounded before reperforming the training algorithm (i.e., *LM* here) to let the network "recover" in each iteration until all parameters are quantized.

To demonstrate the possibility of obtaining a low bit quantized network, the network is trained with the lowest precision that can converge the algorithm before further rounding the parameters to lower precision.

Table 5.7 depicts the experimental results on a number of combinational circuits.

Table 5.7: Training *ANN* with reduced precision

Circuit	Train decimal digits	Train MSE	Test decimal digits	Test MSE
b1	2	7.82×10^{-16}	1	2.58×10^{-13}
cm151a	2	3.34×10^{-15}	1	7.32×10^{-11}
cu	3	5.21×10^{-15}	2	1.5×10^{-11}
t481	3	9.36×10^{-16}	2	8.49×10^{-11}

The second and fourth columns in the table are the number of decimal digits used in training and testing, respectively. The results indicate that the method can effectively create networks with 1 or 2 decimal digit(s) (< 7 bits) at the cost of acceptable loss of accuracy, which easily meets the devices programmable precision requirements.

5.3.3 Parameter Variations

As described in Section 5.1, the neurons are assumed to be reliable or easily protected. Then the only source of uncertainty to be considered in this work is the network parameters. In our analysis, local variations are considered, i.e., each parameter is assumed to be a *Gaussian Random Variable* (RV), varying around its ideal designated value. In order to measure how bad the outputs of the circuits are affected by the variations of the parameters, the *Soft Error Rate* (SER) can be employed. The *SER* is defined as the ratio of error events to total simulated samples. Thus, the procedure to obtain the *SER* is as follows: (a) Randomly generate the varied network parameters following a Gaussian distribution; (b) Simulate the network with the varied parameters for N input vectors; (c) Repeat (a) and (b) for K sample sets and calculate the *SER* by:

$$r = \frac{n(\epsilon = 1)}{N \times K}, \tag{5.6}$$

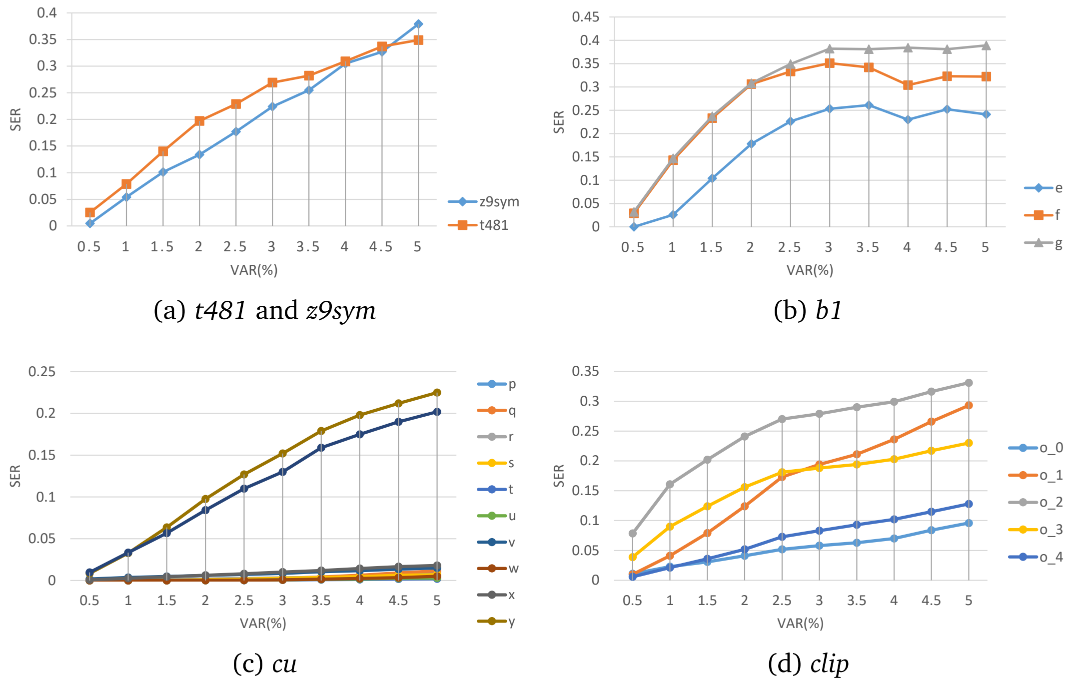
where $n(\epsilon = 1)$ denotes the number of output errors, and N denotes the total number of input vectors tested on the trained model in each random parameter set sample, and K represents the number of random parameter sets generated and tested. Note that the error event $\epsilon = 1$ is defined in Equation 5.7, as 0.3 and 0 are treated as a logic high and low, respectively.

$$\epsilon = \begin{cases} 1 & \text{if } abs(t - \hat{y}) > 0.1 \\ 0 & \text{else} \end{cases}, \tag{5.7}$$

where t and $\hat{y}$ denote the target and predicted output values, respectively. In other words, the output is identified as an error when its actual value is higher or lower than the target value by a threshold, 0.1.

Figure 5.6 depicts the *SER* testing results against the variances of the parameters from $\sigma = 0.5\ \%$ to $\sigma = 5\ \%$ for some trained benchmark circuit models listed in Table 5.7. The number of random parameter sets simulated was $K = 1000$. Figure 5.6a depicts the outputs of two one-output circuits *t481* and *z9sym* and Figures 5.6b, 5.6d, and 5.6c show the multiple-output circuits *b1*, *clip*, and *cu*, respectively and each line represents one output of the corresponding circuit.

As shown in the figures, the *SER*s are increasing with the increase of the variances of the parameters. However, the worst *SER*s (i.e., when $\sigma = 5\ \%$) of each circuit, regardless of the complexity, are all likely to saturate around the values of $0.3 \sim 0.4$. This may because that the stochastic behaviour of the parameter variations compensates each other. Also, for multi-output circuits like $b1$, cu and $clip$, some outputs are more robust to variations than the others. It is reported that it is possible to train the *ANN* to be more robust to variability in [148], where they invented a regularization technique that can improve the performance by up to $70\ \%$. In addition, from a system-level perspective, the outputs



(a) *t481* and *z9sym* (b) *b1*

(c) *cu* (d) *clip*

Figure 5.6: The variation analyses

of a circuit can be protected by an *error correction code* proposed in [149, 150].

5.4 Chapter Summary

In this work, an *EDA* framework for synthesizing the *MLP-based* combinatorial logic gates was proposed. The *MLP-based* logic gates can implement more complex Boolean functions than *TLG*s. However, the circuit synthesis *EDA* framework needs more considerations to be made to achieve efficient circuits.

Specifically, an architecture that efficiently synthesizes the *MLP-based* logic gates was proposed. The logic synthesis process in this context becomes *ANN* training as a Boolean logic approximator. To the best of our knowledge, there is no literature reporting the implementation of Boolean logics that are more complex than simple gates like *XOR* using the *MLP*, while in this work, the training results for large and complex logic circuits were reported. Also, different training algorithms were compared, the *Levenberg-Marquardt* algorithm is identified to be the most efficient algorithm in this specific training task.

Secondly, three *ANN* hardware architectures and devices that are suitable for this framework were compared and discussed. A memristive network could be the most promising architecture due to its high integration density and ease of operation, while *CMOS* capacitive network is the most mature and low-cost architecture; The programming precision of the three devices is able to meet the *MLP* model precision specifications (> 8 bits).

Thirdly, *ANN* hardware implementation relative training and optimisation techniques were reviewed and employed to build more energy, area and reliability efficient network: *Bayesian Regularization* was applied to effectively simplify the network, simple pruning method was applied, showing the possibility to reduce the size of the network up to $20\ \%$. Low precision training and quantization to reduce the precision requirement (< 7 bits).

Finally, parameter variability impacts on the output soft error rates were analyzed, the worst *SER*s (when the variances of the parameters $\sigma = 5\ \%$) are around $0.3 \sim 0.4$ regardless of the complexity of the circuit.

The major concern is the device *PVT* variations. Fortunately, the *ANN* has the ability to tolerate variations to some extent, especially when trained to be variation insensitive [148]. On the other hand, from the system perspective of view, the integrated error correction code block can be employed to protect the erroneous outputs, such as the *Code Prediction Encoding* (CPE) method proposed in [149, 150], as will be detailed in the next chapter. Another critical limitation of using this framework is the exponential relationship between training dataset size and number of the circuit inputs, making the *MLP* learning of Boolean functions with more than 20 inputs a computationally infeasible task for a normal PC. From the *ANN* perspective, *Modular Neural Network* [151] is worth to be investigated, where a complex function with a large number of inputs are decomposed to simpler ones, trained separately before being merged to perform the complete function. From the learning algorithm perspective of view, on the one hand, more aggressive pruning methodologies can be helpful to further reduce the network complexity, resulting in reduced learning time. On the

other hand, on-chip training with specific training hardware will drastically increase training efficiency. For example, [152] implemented the *LM* algorithm on the *FPGA* based hardware, which can be used to train an *ANN* model more efficiently.

This chapter is based on the following paper:

- Bo Yang, Sorin Cotofana, Emanuel Popovici, "Synthesizing a multilayer perceptron based combinational logic circuit," submitted to Integration, the VLSI Journal, 2020

Chapter 6

Circuit Linearization with Bi-decomposition

In Chapter 5, the concept of implementing combinational logic circuits using the *MLP* Boolean function approximators has been discussed. It was mentioned that adding more layers will help the *MLP* reducing the total number of neurons compared to the basic perceptron (single layer fully connected feedforward network). It is also well known that the *MLP* has the difficulty in implementing the *XOR-based* logic functions efficiently. Thus, a *Circuit Linearization* methodology was developed to extract and separate the *XOR-based* sub-circuit, which could be treated outside the general framework of the *MLP-based* logic, from the original circuit.

One should distinguish the terms of *(non-)linear logic function* and *(non-)linearly separable function*, that the linearly separable Boolean functions refer to those Boolean functions with the output space that can be divided into two subspaces with a straight line or a plane, while the linearity of a Boolean function is mathematically defined as:

Definition 6.1 ***(Linearity)*** *A Boolean function $f(x_i, \boldsymbol{x}_0)$ is linear with regard to the variable x_i if and only if*

$$\frac{\partial f(x_i, \boldsymbol{x}_0)}{\partial x_i} = 1 \tag{6.1}$$

With the above definition, we can derive that the *XOR* function is linear while the *AND* function is non-linear.

The recently published ideas of the *Vectorial Bi-decomposition* [68] extending the known approaches of *Strong* and *Weak Bi-decompositions* [69, 153, 154, 155] can lead to a partition of a circuit into parts that contain either only non-linear *AND* gates or only linear *XOR* gates.

Additionally, in the project of *EU FP7 FET-Open i-RISC*, a system-level reliability enhancement methodology called *Codeword Prediction Encoding* (CPE) [149, 156] was proposed, which employs the error correction coding to protect the erroneous *XOR* network.

In this chapter, the circuit decomposition method will be introduced and evaluated. Also, the complete reliability enhancement system framework implementation, as shown in Figure 6.1, will be briefly discussed and will be provisioned in future work.

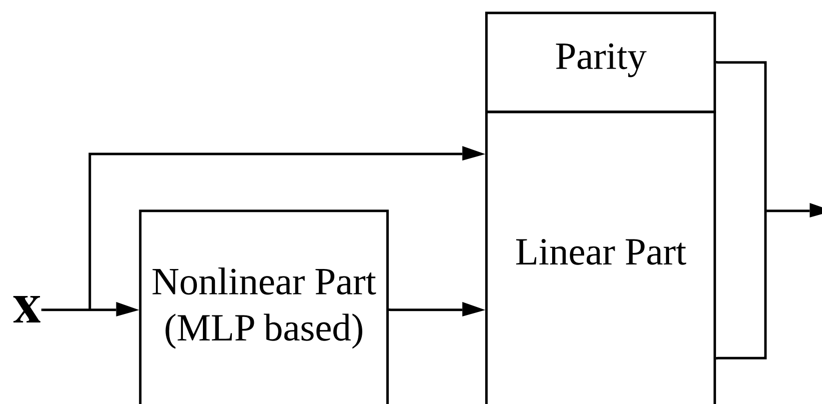


Figure 6.1: Reliability enhancement system framework architecture

The chapter will be organized as follows. Firstly, the basic concepts of Boolean function linearity, such as *Boolean Differential Calculus*, *Independence* and *Degree of Linearity* etc. will be introduced in Section 6.1. Then, based on these concepts, the principles of *linear conversion*, *Bi-decomposition*, and *Vectorial Bi-decomposition* will be explained in Section 6.2. Next, the decomposition of

Adder circuits will be demonstrated and evaluated in Section 6.3 to show the working principle of the approach. Finally, the future work to integrate the linear decomposition method with the *MLP-based* logic gates will be clarified in Section 6.4.

6.1 Boolean Function Linearity Basic Concepts

A Boolean function can be expressed as a function of a vector of n Boolean variables, $\boldsymbol{x}$, i.e., $f(\mathbf{x}) = f(x_i, \mathbf{x}_0)$, then we define: [157, 158]

Definition 6.2 ***(Single Derivative)*** *The single derivative of Boolean function $f(\boldsymbol{x})$ with respect to Boolean variable x_i is*

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} = f(x_i, \boldsymbol{x}_0) \oplus f(\overline{x_i}, \boldsymbol{x}_0) \tag{6.2}$$

Definition 6.3 ***(Single Minimum)*** *The single minimum of Boolean function $f(\boldsymbol{x})$ with respect to Boolean variable x_i is*

$$\min_{x_i} = f(x_i, \boldsymbol{x}_0) \cdot f(\overline{x_i}, \boldsymbol{x}_0) \tag{6.3}$$

Definition 6.4 ***(Single Maximum)*** *The single maximum of Boolean function $f(\boldsymbol{x})$ with respect to Boolean variable x_i is*

$$\max_{x_i} = f(x_i, \boldsymbol{x}_0) + f(\overline{x_i}, \boldsymbol{x}_0) \tag{6.4}$$

Theorem 6.1 *If the Boolean function $f(\boldsymbol{x})$ is linear, the function has the property that satisfies:*

$$f(x_i, \boldsymbol{x}_0) = x_i \oplus g(\boldsymbol{x}_0), \tag{6.5}$$

where

$$g(\boldsymbol{x}_0) = \max_{x_i}(x_i \oplus f(x_i, \boldsymbol{x}_0)) \tag{6.6}$$

Definition 6.5 ***(Independence)*** *The Boolean function $f(\boldsymbol{x})$ is said to be independent of the Boolean variable x_i if and only if*

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} = 0 \tag{6.7}$$

From Definition 6.2, it can be seen that the *single derivative* of Boolean function $f(\mathbf{x})$ itself is a Boolean function, which is evaluated value "1" when f is linear with respect to Boolean variable x_i and value "0" when f is independent with regard to x_i. Thus a *Degree of Linearity* (DOL) can be defined as below, by the number of function values "1" of the *single derivative* of $f(\mathbf{x})$ with regard to x_i:

Definition 6.6 ***(Degree of Linearity)*** *The Boolean function $f(\boldsymbol{x})$ of n Boolean variables has a degree of linearity with regard to the variable x_i in range of $[0, 1]$ defined as:*

$$d_{x_i} f(x_i, \boldsymbol{x}_0) = \frac{1}{2^{n-1}} * \rho \left(\frac{\partial f(\boldsymbol{x}_0)}{\partial x_i} \right), \tag{6.8}$$

where $\rho(t)$ is the number of values 1 of the evaluated function t.

In other words, a *DOL* of 1 indicates the function is purely linear, which is an *XOR*-only function, while a *DOL* smaller than 1 but greater than 0 suggests that the function contains some non-linear components.

6.2 Bi-Decomposition and Vectorial Bi-Decomposition

According to the Definition 6.1, a *Parity* function or *Sum* function $f_s(x, y, c) = \bar{x}\bar{y}c + \bar{x}y\bar{c} + x\bar{y}\bar{c} + xyc$ is a linear function, as can be derived as follows:

$$f_s(x, [y, c]) = x \oplus g(y, c), \tag{6.9}$$

where

$$\begin{aligned} g(y,c) &= \max_x (x \oplus f_s(x,[y,c])) \\ &= \bar{y}c + y\bar{c} \\ &= y \oplus c \end{aligned} \tag{6.10}$$

Thus, it is obvious that the function $f_s(x,y,c)$ is linearly converted into an *XOR* network, as the shown procedure from upper to lower in Figure 6.2, and this process will be called *Linear conversion* of a function in the rest of the thesis.

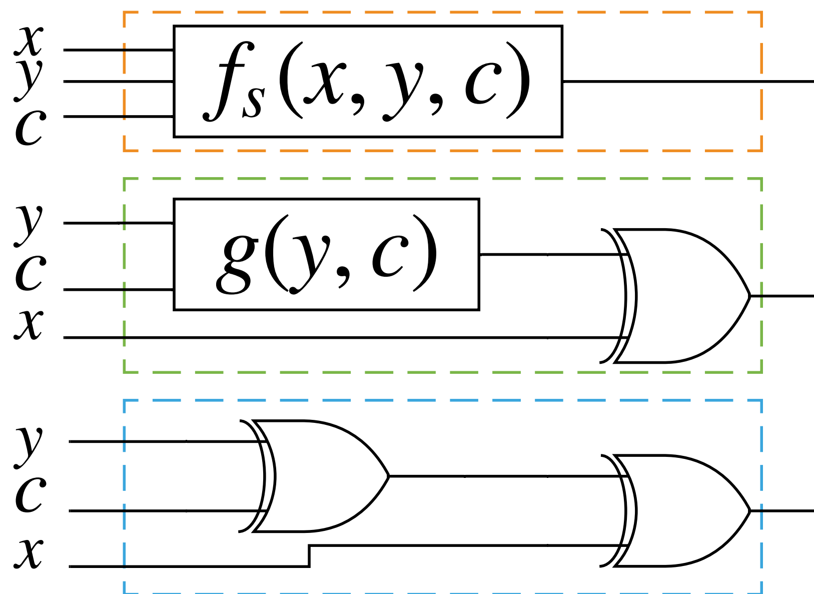
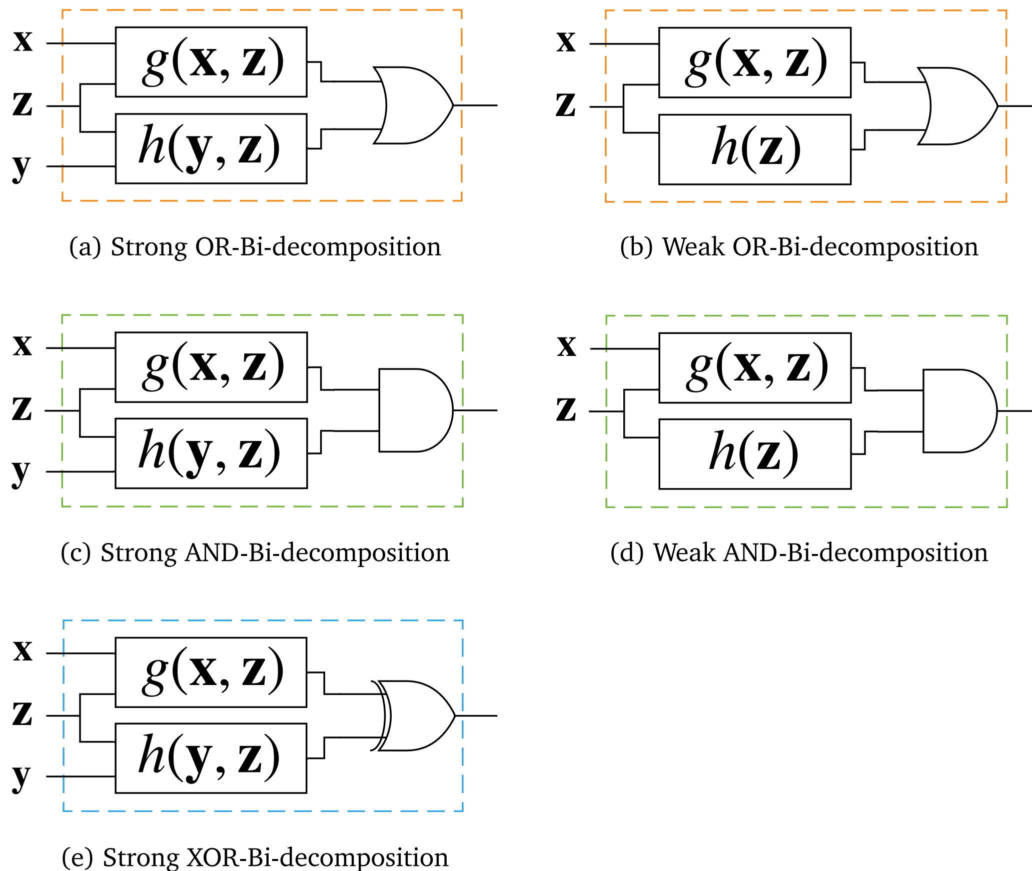


Figure 6.2: Linear separation on *Parity* function

However, it may not be possible to linearly covert the function, which is not purely linear. Therefore, the concept of *Bi-decomposition* was proposed to decompose the original function to two sub-functions, which are a linear only part and a non-linear part. Based on the decomposed function architectures, the Bi-decomposition is classed into *Strong Bi-decomposition* and *Weak Bi-decomposition* [154].

The *Strong OR-Bi-decomposition*, *AND-Bi-decomposition* and *XOR-Bi-decomposition* are described by Figure 6.3a, 6.3c, and 6.3e, respectively. Meanwhile, the

Weak OR-Bi-decomposition and *AND-Bi-decomposition* are described by Figure 6.3b and 6.3d, respectively.



(a) Strong OR-Bi-decomposition
(b) Weak OR-Bi-decomposition
(c) Strong AND-Bi-decomposition
(d) Weak AND-Bi-decomposition
(e) Strong XOR-Bi-decomposition

Figure 6.3: Strong and Weak Bi-decompositions

Note that although a *Weak XOR-Bi-decomposition* exists for each function, their decomposition functions can be even more complex than the original function. Thus, this type of Bi-decomposition is normally avoided.

In addition to the single Bi-decomposition, *Vectorial Bi-decomposition* was recently proposed in [68]. The *Vectorial Bi-decompositions* exist even if there is no *Strong Bi-decomposition* for a given function exists.

Definition 6.7 ***(Vectorial Bi-Decomposition)*** *A Boolean function $f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)$ is OR, AND, or* XOR *bi-decomposable with regard to the subsets of variables $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ if:*

1. *$f(\boldsymbol{x})$ can be expressed by*

$$f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) = g(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) + h(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \tag{6.11}$$

$$f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) = g(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \cdot h(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \tag{6.12}$$

$$f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) = g(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \oplus h(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c), \tag{6.13}$$

   respectively.

2. *The decomposition functions $g(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)$ and $h(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)$ satisfy:*

$$\frac{\partial g(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)}{\partial \boldsymbol{x}_b} = 0 \tag{6.14}$$

$$\frac{\partial h(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)}{\partial \boldsymbol{x}_a} = 0 \tag{6.15}$$

From the conditions indicated in the definition, the following theorem can be derived:

Theorem 6.2 ***(Conditions of Vectorial Bi-decomposition)*** *A Boolean function $f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)$ is* OR, AND *or* XOR *bi-decomposable with regard to the dedicate sets of variables $\mathbf{x}_a$ and $\mathbf{x}_b$ if:*

$$f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \cdot \max_{\boldsymbol{x}_a} \overline{f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)} \cdot \max_{\boldsymbol{x}_b} \overline{f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)} = 0 \tag{6.16}$$

$$\overline{f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)} \cdot \max_{\boldsymbol{x}_a} f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \cdot \max_{\boldsymbol{x}_b} f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) = 0 \tag{6.17}$$

$$\frac{\partial}{\partial \boldsymbol{x}_b}\left(\frac{\partial f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)}{\partial \boldsymbol{x}_a}\right) = 0 \tag{6.18}$$

Also, the decomposition functions can be derived as:

Theorem 6.3 ***(Vectorial decomposition function)***

- *The vectorial OR-Bi-decomposition:*

$$g(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) = \min_{\boldsymbol{x}_b} f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \quad (6.19)$$

$$h(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) = \min_{\boldsymbol{x}_a} f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \quad (6.20)$$

- *The vector AND-Bi-decomposition:*

$$g(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) = \max_{\boldsymbol{x}_b} f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \quad (6.21)$$

$$h(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) = \max_{\boldsymbol{x}_a} f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \quad (6.22)$$

- *The vectorial XOR-Bi-decomposition:*

$$g(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) = \frac{\partial[\bar{x}_{b_i} \cdot f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)]}{\partial \boldsymbol{x}_b} \quad (6.23)$$

$$h(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) = f(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c) \oplus g(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c), \quad (6.24)$$

where $x_{b_i} \in \boldsymbol{x}_b$.

6.3 Linear Decomposition of Adders

In Section 6.2, the linear separation of an *Adder* sum function has been explained. As the *Sum* function is purely linear, the decomposition function can be implemented using just *XOR* gates. However, the situation is different for the *Adder carry function*, $f_c(x, y, c) = \bar{x}yc + x\bar{y}c + xy\bar{c} + xyc$, as no *Strong Bi-decomposition* exists for it. Thus, *Vectorial Bi-decomposition* for it will be explained here.

First of all, the carry function is Vectorial XOR-Bi-decomposable as the condition in Equation 6.18 in Theorem 6.2 is satisfied. The decomposition functions are:

$$g_c(x, y, c) = \frac{\partial[\bar{c} \cdot (\bar{x}yc + x\bar{y}c + xy\bar{c} + xyc)]}{\partial c} = xy \quad (6.25)$$

$$h_c(x, y, c) = (\bar{x}yc + x\bar{y}c + xy\bar{c} + xyc) \oplus xy = yc \oplus xc \quad (6.26)$$

Therefore, the XOR-Bi-decomposed carry function can be implemented using three *AND* gates and two *XOR* gates, as shown the Figure 6.4.

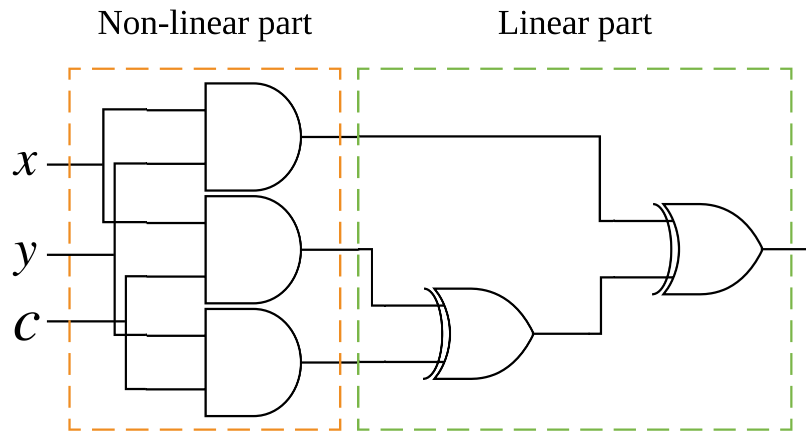


Figure 6.4: Vectorial XOR-Bi-decomposition on *Carry* function

A 1-bit *Adder* is composed of a *Sum* function described in Section 6.2 and a *Carry* function described above, and the decomposition of it has been discussed. Furthermore, for an N-bit *Adder*, a straightforward approach is to cascade the carries of the 1-bit Adders one after another, which resembles the structure of a *Ripple Carry Adder* (RCA). As this kind of decomposition is done "locally" inside each of the *Adder* bits, this decomposition architecture is called the *Locally* decomposed *Adder*. On the other hand, instead of decomposing by individual 1-bit *Adder*, the decomposition can also be performed on the whole N-bit *Adder* circuit directly, which results in an architecture similar to the *Carry Look-ahead Adder* (CLA), and is thereby called *Globally* decomposed *Adder*.

The local decomposition generates circuits with much fewer components than

its global counterparts. However, the global decomposition is a preferred approach for the *CPE* method, as the resulting circuit is composed of a single linear block (*XOR* network) that can be protected by the *CPE* and a single non-linear part that can be implemented with the *MLP-based* logic.

Table 6.1 shows the results of the gate count comparison between globally decomposed adders with the different number of bits and the standard *RCA*.

Table 6.1: Global decomposition experimental results

N bits	Globally decomposed			Standard *RCA*
	Linear part	non-linear part	Total	Total
4	32	17	49	34
5	65	39	105	43
6	132	86	218	53
7	266	180	446	62
8	534	371	904	72
10	2138	1520	3658	91

As can be seen from the figure, the gate count is exponentially increasing with the number of bits due to the process of the internal carry generation.

6.4 Chapter Summary

In this chapter, the Bi-decomposition based circuit linearization approach was proposed and explained. The Bi-decomposition allows a circuit to be divided into a linear part and a non-linear part. Two decomposition architectures of *Adder*s, i.e., local and global decomposition, were described and experimented to demonstrate the methodology.

The globally decomposed architecture is very useful for the *CPE* reliability enhancement architecture, as the *CPE* can protect the output of an *XOR* network from being faulty. Additionally, the global decomposition indicates the methodology works not only on *Adder*s but also on any arbitrary combinational circuits.

However, the global decomposition functions were identified to be large in gate counts. Thus, in the future, the *MLP-based* logic gates might be used to reduce

the circuit size of the non-linear part. This is because that the *MLP-based* logic gates can be implemented very efficiently, especially for the non-*XOR* network, as explained in Chapter 5. The *MLP-based* logic on decomposed non-linear function will be evaluated to study the efficiency of the *MLP-based* non-linear logic functions. Hence, an architecture of the complete reliability enhancement system framework with *CPE*, and *MLP-based* logic gates with circuit linearization, as depicted in Figure 6.1 can be explored in future work.

This chapter is based on the following book chapter:

- B. Steinbach, Emanuel Popovici, Daniel Rodas Bautista, Bo Yang, "Synthesis techniques for reliability using Bi-decompositions," Book chapter in "Further Improvements in the Boolean Domain," Cambridge Scholars Publishing, 2019, pp. 321-335

Chapter 7

Conclusions and Future Work

The thesis is part of a big project, *EU FP7 FET-Open i-RICS*, which seeks approaches to allow the implementation of reliable computation system on unreliable ultra-low-power *VLSI* circuits. In order to create optimal *VLSI* circuit design that has balanced performance, power consumption, area, and reliability, advanced design flow, methodologies, and *EDA* tools are increasingly demanded. In this thesis, many aspects of modern low-voltage *CMOS VLSI* design *EDA* considerations have been discussed. The main purpose of this work is to create the advanced *EDA* algorithms to help to evaluate, and optimize the circuit in an early stage of the design flow, i.e., design and synthesis, emphasizing the reliable operation of the digital circuit. My contributions are in the area of efficient methodologies for circuit analyses emphasizing the reliability, and new circuit design paradigms based on the recent development of *Artificial Neural Network* (ANN).

The studied reliability was clarified to include two aspects, i.e., the soft error propagation based reliability and the device variations related timing reliability. Then, methodologies encompass these two aspects were proposed.

Firstly, an efficient analytical gate-level soft error propagation algorithm, *CPEP*, was developed, which can accurately estimate the error probability, hence the reliability of combinational digital circuits. The methodology makes use of the

AIG alike circuit representation and propagates the soft error through the circuit under the presence of the *Logic Masking* mechanism. The *Conditional Probability* and *Theorem of Total Probability* is employed to resolve the *Re-convergent Fanout* problem during the soft error propagation.

The identified results on a set of *MCNC* benchmark circuit tests are more than 10^5 times faster within $3\ \%$ accuracy degradation compared to *Monte-Carlo* (MC) simulation based methods.

To demonstrate the concept, the algorithm for the circuits that are composed of *AND*, and *Inverter* gates were implemented and explained. Nevertheless, the algorithm can be very straightforwardly extended to any generic logic gates (e.g., *XOR*, *NOR*, *MUX*, etc.), in the future. Besides, the current implementation is for the combinational circuits only, but in the future, the algorithm can be extended to a *stated-based CPEP* that works on synchronous sequential circuits as well.

Then, a *Cut Rewriting-based EDA* framework for reliability-driven synthesis was proposed and described. The algorithm utilizes the *AIG* as the data structure and rewrites the *4-cut* sub-circuits with the pre-computed more robust library cuts iteratively during the traversal, hence improves the reliability of the complete circuit.

The framework was tested on a set of *MCNC* benchmark circuits and *8051 MCU* building blocks. The results show that our algorithm can improve the circuit reliability up to $75\ \%$ against the original synthesized circuit with the acceptable area and power consumption overhead.

Secondly, the *ANN* function approximator is employed to predict the propagation delay distribution of a circuit, hence the timing-related reliability taking the circuit setup parameters (e.g., V_{dd} and load conditions) into account. Initially, an analytical model was developed based on mathematical models of the *MOSFET* charging and discharging currents. The model is precise but difficult to be extended to include more parameters other than the only threshold volt-

age, V_{th}. Thus, the *ANN-based* methodology with two modelling strategies was subsequently introduced:

- *ANN-τMC-SSTA:* The *ANN* approximator is used to predict the propagation delay, τ_{pd}, based on the circuit setup and device parameter variabilities, which is used later with an *MC* simulation to perform the *SSTA*. This strategy can achieve 10^5 times higher performance, while as flexible as the *SPICE MC* simulation.
- *ANN-IGD-SSTA:* The τ_{pd} distribution, P_τ, is approximated by an *IGD*, where the *ANN* approximator is employed to predict the *IGD* parameters μ and λ depending on the circuit setup. The performance can be further improved with respect to the *ANN-τMC-SSTA* strategy at the cost of flexibility.

The gate model can be employed as the *SSTA* tool for conventional *VLSI* circuit design flow. However, the more interesting area of usage in the future will be when integrated into a *Multi-objective Optimisation* framework, where the V_{dd} can be adjusted continuously to find the optimal circuit operation condition that encompasses the proper area, power consumption, performance, and reliability for the specific application.

Another interesting research goal will be the timing-related reliability estimation on faulty circuits, which may combine the two aspects of the reliability studied in this thesis. Specifically, the *CPEP* algorithm deals with the *Logic Masking* mechanism, and the *ANN-SSTA* models can be used to analyze the *Electrical Masking*, as the electrical masking mechanism is related to the widths of glitches (i.e., circuit timing related). The combined methodologies can be a more comprehensive framework for reliability analysis.

Thirdly, inspired by the *CPE* fault-tolerant computation system design framework and the *Multilayer Perceptron* (MLP) Boolean function approximation capability, the *MLP-based* Boolean logic was proposed and investigated, along with a proposal of circuit linearization framework based on the Boolean function *Bi-decomposition* methodology.

- A synthesis framework for the *MLP-based* logic gate was first introduced. The Boolean function approximation capability of *MLP*s was investigated, and the suitable learning algorithms of the multilayer perceptron were discussed; Some promising *ANN* circuit architectures for the practical implementation of the *MLP-based* logic gates were reviewed and explained; And the significant aspects of considerations and techniques for the practical hardware model transfer of the trained *ANN* models, such as *Regularization*, *Quantization*, *pruning* and *variability analysis*, were clarified, experimented and reported. The benchmark circuits as complex as *t481* can be implemented by an *MLP* of 17 neurons and 243 parameters in 3 layers.

- The circuit linearization framework was proposed, which can partition an arbitrary circuit into a linear part (*XOR* network) and a non-linear part. The methodology employs the concepts of *Boolean Derivative* and *Vectorial Bi-decomposition*. The *Adder* circuits with a different number of bits were used to demonstrate the working principle of the linearization algorithm. Specifically, two approaches for decomposing an *Adder* circuit were explained, i.e., the *Local Decomposition* and *Global Decomposition*. *Global Decomposition* is identified to be useful in our context, as the whole *Adder* circuit can be decomposed comprehensively to be a non-linear part and a linear part.

As for future work, the implementation of *MLP-based* logic gates can be further investigated in many aspects, such as:

- The practical *ANN* circuit implementations, emphasizing the sub-threshold region to further improve the power consumption profile of the designs;

- The implementations of more efficient quantization and pruning methodologies, which leads to more efficient *ANN* model hardware transfer;

- The variability-insensitive training algorithms, which can improve the variation resilience of the circuits;

- Also, the identification of critical nodes (the synapses or neurons having the highest impact on the output reliability) in the network will be investigated. The identified critical nodes can then be specifically protected.

Besides, the *MLP-based* logic gates for non-linear components only circuits will be implemented and evaluated. Hence the whole framework of *MLP-based* logic gates with *Circuit Linearization* and *CPE* fault-tolerant system will be implemented, which will improve the development of the ultra-low-power efficient computation system.

Finally, in the future, a complete *EDA* framework that encompasses all the above methodologies and algorithms needs to be investigated in the context of *Multi-objective Optimisation* (reliability-power-area-performance), in order to create more efficient *VLSI* digital systems.

References

[1] "Research report on global and China's integrated circuit industries, 2019 to 2023." https://www.researchandmarkets.com/reports/4774098/research-report-on-global-and-chinas-integrated, 06 2019 (last accessed on Apr. 2020).

[2] F. Wright and T. M. Conte, "Standards: Roadmapping computer technology trends enlightens industry," *Computer*, vol. 51, pp. 100–103, jun 2018.

[3] B. Höfflinger, *ITRS 2028—International Roadmap of Semiconductors*, pp. 143–148. Cham: Springer International Publishing, 2016.

[4] B. Hoefflinger, *Chips 2020: A Guide to the Future of Nanoelectronics*. The Frontiers Collection, Springer Berlin Heidelberg, 2012.

[5] "This giant AI chip is the size of an iPad and holds 1.2 trillion transistors." https://singularityhub.com/2019/08/26/this-giant-ai-chip-is-the-size-of-an-ipad-and-holds-1-2-trillion-transistors, 08 2019 (last accessed on May 2020).

[6] T. N. Theis and H. S. P. Wong, "The end of Moore's law: A new beginning for information technology," *Computing in Science and Engg.*, vol. 19, p. 41–50, Mar. 2017.

[7] R. S. Williams, "What's next? the end of Moore's law," *Computing in Science Engineering*, vol. 19, no. 2, pp. 7–13, 2017.

[8] F. Peper, “The end of Moore’s law: Opportunities for natural computing?,” *New Generation Computing*, vol. 35, no. 3, pp. 253–269, 2017.

[9] H. Khan, D. Hounshell, and E. Fuchs, “Science and research policy at the end of Moore’s law,” *Nature Electronics*, vol. 1, 01 2018.

[10] M. A. Kastner, “The single-electron transistor,” *Rev. Mod. Phys.*, vol. 64, pp. 849–858, Jul 1992.

[11] V. S. Zharinov, T. Picot, J. E. Scheerder, E. Janssens, and J. Van de Vondel, “Room temperature single electron transistor based on a size-selected aluminium cluster,” *Nanoscale*, vol. 12, pp. 1164–1170, 2020.

[12] L. Chua, “Memristor-the missing circuit element,” *IEEE Transactions on Circuit Theory*, vol. 18, pp. 507–519, Sep. 1971.

[13] S. Banerjee, A. Chaudhuri, and K. Chakrabarty, “Analysis of the impact of process variations and manufacturing defects on the performance of carbon-nanotube FETs,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–14, 2020.

[14] M. Krüger, M. R. Buitelaar, T. Nussbaumer, C. Schönenberger, and L. Forró, “Electrochemical carbon nanotube field-effect transistor,” *Applied Physics Letters*, vol. 78, no. 9, pp. 1291–1293, 2001.

[15] P. K. Mukku, S. Naidu, D. Mokara, P. Pydi Reddy, and K. Sunil Kumar, “Recent trends and challenges on low-power FinFET devices,” in *Smart Intelligent Computing and Applications* (S. C. Satapathy, V. Bhateja, J. R. Mohanty, and S. K. Udgata, eds.), (Singapore), pp. 499–510, Springer Singapore, 2020.

[16] M. Liu, S. Scholz, A. Hardtdegen, J. H. Bae, J. Hartmann, J. Knoch, D. Grützmacher, D. Buca, and Q. Zhao, “Vertical Ge gate-all-around nanowire PMOSFETs with a diameter down to 20 nm,” *IEEE Electron Device Letters*, vol. 41, no. 4, pp. 533–536, 2020.

[17] R. Mehrotra, T. English, M. Schellekens, S. Hollands, and E. Popovici, "Timing-driven power optimisation and power-driven timing optimisation of combinational circuits," *Journal of Low Power Electronics*, vol. 7, no. 3, pp. 364–380, 2011.

[18] "IEEE standard definitions for use in reporting electric generating unit reliability, availability, and productivity," *ANSI/IEEE Std 762-1987*, pp. 1–24, 1987.

[19] R. Xiao and C. Chen, "Gate-level circuit reliability analysis: A survey," *VLSI Design*, vol. 2014, pp. 1–12, 07 2014.

[20] A. Amaricai, S. Cotofana, E. Popovici, and B. Vasic, "Circuit level fault models for sub-powered CMOS circuits for uncorrelated and correlated errors." `http://www.i-risc.eu/Lists/StaticFiles/delivrables/i-RISC_D2.1.pdf`, February 2014 (last accessed on Jan. 2020).

[21] A. Amaricai, S. Cotofana, E. Popovici, and B. Vasic, "Higher abstraction fault models and their simulation methodology." `http://www.i-risc.eu/Lists/StaticFiles/delivrables/i-RISC_D2.2_v2.pdf`, November 2014 (last accessed on Jan. 2020).

[22] A. Bhanu, M. S. K. Lau, K. Ling, V. J. Mooney III, and A. Singh, "A more precise model of noise based PCMOS errors," in *2010 Fifth IEEE International Symposium on Electronic Design, Test Applications*, pp. 99–102, 2010.

[23] T. Karnik and P. Hazucha, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 128–143, 2004.

[24] N. Miskov-Zivanov and D. Marculescu, "Circuit reliability analysis using symbolic techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 12, pp. 2638–2649, 2006.

[25] M. Hwang, "Supply-voltage scaling close to the fundamental limit under process variations in nanometer technologies," *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2808–2813, 2011.

[26] A. M. Baker and Y. Jiang, "Modeling and architectural simulations of the statistical static timing analysis of the non-Gaussian variation sources for VLSI circuits," 2013.

[27] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, "Statistical timing analysis: From basic principles to state of the art," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 4, pp. 589–607, 2008.

[28] M. Merrett, P. Asenov, Y. Wang, M. Zwolinski, D. Reid, C. Millar, S. Roy, Z. Liu, S. Furber, and A. Asenov, "Modelling circuit performance variations due to statistical variability: Monte Carlo static timing analysis," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pp. 1–4, IEEE, 2011.

[29] J. J. Zasio, K. C. Choy, and D. R. Parham, "Static timing analysis of semiconductor digital circuits," May 8 1990. US Patent 4,924,430.

[30] Chenming Hu, Simon C. Tam, Fu-Chieh Hsu, Ping-Keung Ko, Tung-Yi Chan, and K. W. Terrill, "Hot-electron-induced MOSFET degradation - model, monitor, and improvement," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 1, pp. 295–305, 1985.

[31] M. Kamal, Q. Xie, M. Pedram, A. Afzali-Kusha, and S. Safari, "An efficient reliability simulation flow for evaluating the hot carrier injection effect in CMOS VLSI circuits," pp. 352–357, 09 2012.

[32] W. Wang, V. Reddy, A. Krishnan, R. Vattikonda, S. Krishnan, and Y. Cao, "Compact modeling and simulation of circuit reliability for 65-nm CMOS technology," *Device and Materials Reliability, IEEE Transactions on*, vol. 7, pp. 509 – 517, 01 2008.

[33] P. Rani, G. Singh, and M. Kaur, "Impact of negative bias temperature instability on 6T cmos sram cell performance," *International Journal of Computer Applications*, vol. 128, pp. 1–6, 10 2015.

[34] H. Al Ayubi, N. Rizvi, A. Yadav, and Rahul, "Performance & reliability analysis for vlsi circuits using 45 nm technology," *IEEE - ICEEOT - SAP-MVD*, 05 2016.

[35] V. Savin, D. Declercq, S. Cotofana, E. P. A. Amaricai, G. Djordjevic, and B. Vasic, "Innovative reliable chip designs from low-powered unreliable components." `http://www.i-risc.eu`, 2013 (last accessed on Jan. 2020).

[36] J. von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components," in *Automata Studies* (C. Shannon and J. McCarthy, eds.), pp. 43–98, Princeton University Press, 1956.

[37] C. Zhao, X. Bai, and S. Dey, "Evaluating transient error effects in digital nanometer circuits," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 381–391, 2007.

[38] S. Nimara, A. Amaricai, and M. Popa, "Analysis of transient error propagation in sub-powered cmos circuits," in *2014 29th International Conference on Microelectronics Proceedings - MIEL 2014*, pp. 375–378, 2014.

[39] M. Eslami, B. Ghavami, M. Raji, and A. Mahani, "A survey on fault injection methods of digital integrated circuits," *Integration*, vol. 71, pp. 154 – 163, 2020.

[40] E. Popovici, S. Grandhi, C. Spagnol, V. Savin, S. Cotofana, and A. Amaricai, "Data structures and design flow for fault tolerant circuit synthesis." `http://www.i-risc.eu/Lists/StaticFiles/delivrables/i-RISC_D5.1.pdf`, February 2014 (last accessed on Jan. 2020).

[41] D. Gil, L. J. Saiz, J. Gracia, J. C. Baraza, and P. J. Gil, "Injecting intermittent faults for the dependability validation of commercial microcon-

trollers," in *2008 IEEE International High Level Design Validation and Test Workshop*, pp. 177–184, 2008.

[42] S. R. Seward and P. K. Lala, "Fault injection for verifying testability at the VHDL level," in *International Test Conference, 2003. Proceedings. ITC 2003.*, vol. 1, pp. 131–137, 2003.

[43] W. Ibrahim, V. Beiu, and H. Amer, "How much input vectors affect nano-circuit's reliability estimates," in *Nanotechnology, 2009. IEEE-NANO 2009. 9th IEEE Conference on*, pp. 699–702, July 2009.

[44] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Probabilistic transfer matrices in symbolic reliability analysis of logic circuits," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, pp. 8:1–8:35, Feb. 2008.

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

[46] A. Abdollahi, "Probabilistic decision diagrams for exact probabilistic analysis," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pp. 266–272, Nov 2007.

[47] S. Bhanja and N. Ranganathan, "Switching activity estimation of VLSI circuits using Bayesian networks," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 11, no. 4, pp. 558–567, 2003.

[48] M. Choudhury and K. Mohanram, "Reliability analysis of logic circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 3, pp. 392–405, 2009.

[49] G. Asadi and M. B. Tahoori, "An analytical approach for soft error rate estimation in digital circuits," in *2005 IEEE International Symposium on Circuits and Systems*, pp. 2991–2994, IEEE, 2005.

[50] A. Biere, "The AIGER: And-inverter graph (aig) format, version 20070427," *Johannes Kepler University*.

[51] A. Mishchenko *et al.*, "ABC: A system for sequential synthesis and verification," 2007.

[52] S. Almukhaizim, Y. Makris, Y. Yang, and A. Veneris, "Seamless integration of SER in rewiring-based design space exploration," in *2006 IEEE International Test Conference*, pp. 1–9, 2006.

[53] S. Krishnaswamy, S. M. Plaza, I. L. Markov, and J. P. Hayes, "Enhancing design robustness with reliability-aware resynthesis and logic simulation," in *2007 IEEE/ACM International Conference on Computer-Aided Design*, pp. 149–154, 2007.

[54] K. Wu and D. Marculescu, "A low-cost, systematic methodology for soft error robustness of logic circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 2, pp. 367–379, 2013.

[55] M. R. Choudhury and K. Mohanram, "Low cost concurrent error masking using approximate logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 8, pp. 1163–1176, 2013.

[56] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification* (T. Touili, B. Cook, and P. Jackson, eds.), (Berlin, Heidelberg), pp. 24–40, Springer Berlin Heidelberg, 2010.

[57] S. Zaynoun, M. S. Khairy, A. M. Eltawil, F. J. Kurdahi, and A. Khajeh, "Fast error aware model for arithmetic and logic circuits," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pp. 322–328, IEEE, 2012.

[58] R. Rithe, J. Gu, A. Wang, S. Datla, G. Gammie, D. Buss, and A. Chandrakasan, "Non-linear operating point statistical analysis for local variations in logic timing at low voltage," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*, pp. 965–968, IEEE, 2010.

[59] S. Keller, D. M. Harris, and A. J. Martin, "A compact transregional model for digital CMOS circuits operating near threshold," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 10, pp. 2041–2053, 2014.

[60] J. Chen, C. Spagnol, S. Grandhi, E. Popovici, S. Cotofana, and A. Amaricai, "Linear compositional delay model for the timing analysis of sub-powered combinational circuits," in *VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on*, pp. 380–385, IEEE, 2014.

[61] J. Chen, S. Cotofana, S. Grandhi, C. Spagnol, and E. Popovici, "Inverse gaussian distribution based timing analysis of sub-threshold CMOS circuits," *Microelectronics Reliability*, vol. 55, no. 12, Part B, pp. 2754 – 2761, 2015.

[62] A. A. Mullin, "Threshold logic: A synthesis approach (Michael L. Dertouzos)," *SIAM Review*, vol. 8, no. 3, pp. 405–406, 1966.

[63] V. Beiu, J. M. Quintana, and M. J. Avedillo, "VLSI implementations of threshold logic-A comprehensive survey," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1217–1243, 2003.

[64] L. Gao, F. Alibart, and D. B. Strukov, "Programmable CMOS/memristor threshold logic," *IEEE Transactions on Nanotechnology*, vol. 12, no. 2, pp. 115–119, 2013.

[65] A. P. James, D. S. Kumar, and A. Ajayan, "Threshold logic computing: Memristive-CMOS circuits for fast fourier transform and vedic multiplication," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 23, no. 11, pp. 2690–2694, 2015.

[66] B. C. Csáji *et al.*, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, no. 48, p. 7, 2001.

[67] B. Steinbach and R. Kohut, "Neural networks–a model of boolean functions," in *Boolean Problems, Proceedings of the 5th International Workshop*

on Boolean Problems, pp. 223–240, 2002.

[68] B. Steinbach, "Vectorial bi-decompositions of logic functions," 05 2015.

[69] D. Bochmann, F. Dresig, and B. Steinbach, "A new decomposition method for multilevel circuit design," in *Proceedings of the European Conference on Design Automation.*, pp. 374–377, 1991.

[70] J. A. Bondy, U. S. R. Murty, *et al.*, *Graph theory with applications*, vol. 290. Macmillan London, 1976.

[71] S. M. Ross, "Chapter 3 - elements of probability," in *Introduction to Probability and Statistics for Engineers and Scientists (Fifth Edition)* (S. M. Ross, ed.), pp. 53 – 87, Boston: Academic Press, fifth edition ed., 2014.

[72] P.-L. Hsu and H. Robbins, "Complete convergence and the law of large numbers," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 33, no. 2, p. 25, 1947.

[73] J. Benesty, J. Chen, Y. Huang, and I. Cohen, *Pearson Correlation Coefficient*, pp. 1–4. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[74] S. Amari *et al.*, *The handbook of brain theory and neural networks*. MIT press, 2003.

[75] S. Haykin, *Neural Networks: A Comprehensive Foundation*. USA: Prentice Hall PTR, 1st ed., 1994.

[76] T. Chen, H. Chen, and R.-w. Liu, "Approximation capability in by multilayer feedforward networks and related problems," *Neural Networks, IEEE Transactions on*, vol. 6, pp. 25 – 30, 02 1995.

[77] H. H. Tan and K. H. Lim, "Review of second-order optimization techniques in artificial neural networks backpropagation," *IOP Conference Series: Materials Science and Engineering*, vol. 495, p. 012003, jun 2019.

[78] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[79] R. Fletcher, *Practical methods of optimization*. No. v. 1 in Wiley-Interscience publication, Wiley, 1987.

[80] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, pp. 989–993, Nov 1994.

[81] H. Yu and B. Wilamowski, *Levenberg–Marquardt Training*, pp. 1–16. 02 2011.

[82] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *ICML*, pp. 265–272, 2011.

[83] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pp. 389–398, 2002.

[84] S. Xu and E. Edirisuriya, "A new way of detecting reconvergent fanout branch pairs in logic circuits," in *Test Symposium, 2004. 13th Asian*, pp. 354–357, Nov 2004.

[85] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for deep-submicron FPGAs*. Kluwer Academic Publishers, 1999.

[86] E. Todorovich, M. Gilabert, G. Sutter, S. López-Buedo, and E. Boemo, "A tool for activity estimation in FPGAs," in *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, pp. 340–349, Springer, 2002.

[87] R. Burch, F. Najm, B. Ping Yang, and T. N. Trick, "A Monte Carlo approach for power estimation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 1, pp. 63–71, March 1993.

[88] N. Li and E. Dubrova, "AIG rewriting using 5-input cuts," in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pp. 429–430, Oct 2011.

[89] S. L. Hurst, J. C. Muzio, and D. M. Miller, *Spectral Techniques in Digital Logic*. Orlando, FL, USA: Academic Press, Inc., 1985.

[90] F. Mailhot and G. De Micheli, "Technology mapping using boolean matching and don't care sets," in *Design Automation Conference, 1990., EDAC. Proceedings of the European*, pp. 212–216, Mar 1990.

[91] S. Yang, *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.

[92] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical Analysis and Optimization for VLSI: Timing and Power*. Integrated Circuits and Systems, Springer US, 2006.

[93] C. Forzan and D. Pandini, "Statistical static timing analysis: A survey," *Integration*, vol. 42, no. 3, pp. 409 – 435, 2009. Special Section on DCIS2006.

[94] G. Massobrio and P. Antognetti, *Semiconductor device modeling with SPICE*, vol. 21. McGraw-Hill New York, 1993.

[95] V. Veetil, D. Sylvester, and D. Blaauw, "Efficient Monte Carlo based incremental statistical timing analysis," in *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, (New York, NY, USA), pp. 676–681, ACM, 2008.

[96] V. Veetil, K. Chopra, D. Blaauw, and D. Sylvester, "Fast statistical static timing analysis using smart Monte-Carlo techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 6, pp. 852–865, 2011.

[97] Z. Yang, J. Yang, K. Xing, and G. Yang, "Monte carlo based statistical timing analysis using an efficient sampling method," *IEICE Electronics Express*, pp. 13–20160735, 2016.

[98] J. Jaffari and M. Anis, "On efficient Monte Carlo-based statistical static timing analysis of digital circuits," in *2008 IEEE/ACM International Con-*

ference on Computer-Aided Design, pp. 196–203, 2008.

[99] M. Imai, T. Sato, N. Nakayama, and K. Masu, "Non-parametric statistical static timing analysis: An SSTA framework for arbitrary distribution," in *2008 45th ACM/IEEE Design Automation Conference*, pp. 698–701, 2008.

[100] Y. Wang and M. Zwolinski, "Analytical transient response and propagation delay model for nanoscale CMOS inverter," in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pp. 2998–3001, May 2009.

[101] Z. Huang, A. Kurokawa, M. Hashimoto, T. Sato, M. Jiang, and Y. Inoue, "Modeling the overshooting effect for CMOS inverter delay analysis in nanometer technologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, pp. 250–260, Feb 2010.

[102] L. Bisdounis, S. Nikolaidis, O. Koufopavlou, and C. Goutis, "Switching response modeling of the CMOS inverter for sub-micron devices," in *Design, Automation and Test in Europe, 1998., Proceedings*, pp. 729–735, Feb 1998.

[103] D. Binkley, *Tradeoffs and Optimization in Analog CMOS Design*. Wiley, 2008.

[104] T. Sakurai and A. R. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 584–594, Apr 1990.

[105] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE*, vol. 91, pp. 305–327, Feb 2003.

[106] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson Education, 2011.

[107] T. H. Morshed, D. D. Lu, W. M. Yang, M. V. Dunga, X. J. Xi, J. He, W. Liu, M. C. Kanyu, X. Jin, J. J. Ou, M. Chan, A. M. Niknejad, and C. Hu,

"BSIM4 v4.7 MOSFET model - user's manual," 2011.

[108] A. Loke, Z.-Y. Wu, R. Moallemi, D. Cabler, C. Lackey, T. T. Wee, and B. Doyle, "Constant-current threshold voltage extraction in hspice for nanoscale cmos analog design," 03 2010.

[109] W. T. Ang and Y. S. Park, *Ordinary Differential Equations: Methods and Applications*. Universal Publishers, 2008.

[110] C. Yu, Z. Wei, and E. Wang, "Predictive technology model." `http://ptm.asu.edu/`, 2008 (last accessed on June 2019).

[111] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An open-source variation-aware design kit," in *Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, MSE '07, (Washington, DC, USA), pp. 173–174, IEEE Computer Society, 2007.

[112] J. Knudsen, "Nangate 45nm open cell library," *CDNLive, EMEA*, 2008.

[113] H. Kaeslin, *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. Cambridge University Press, 2008.

[114] Synopsys Ltd., "HSPICE 2010 user guide: Simulation and analysis." `https://www.ele.uri.edu/Courses/ele448/HspiceRef/hspice_sa.pdf`, 2010 (last accessed on Oct. 2019).

[115] B. S. Everitt, *Finite Mixture Distributions*. American Cancer Society, 2014.

[116] B. Liu, "Signal probability based statistical timing analysis," in *2008 Design, Automation and Test in Europe*, pp. 562–567, 2008.

[117] N. C. Laurenciu and S. D. Cotofana, "Fast and accurate workload-level neural network based IC energy consumption estimation," in *2017 14th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pp. 1–4, June 2017.

[118] I. Jolliffe, *Principal Component Analysis*, pp. 1094–1096. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[119] H. Roy and M. Sharad, “Current mode neuron for the memristor based synapse,” 2019.

[120] R. Rojas, *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer-Verlag, 1996.

[121] S. Yu, “Neuro-inspired computing with emerging nonvolatile memorys,” *Proceedings of the IEEE*, vol. 106, pp. 260–285, Feb 2018.

[122] U. Cilingiroglu, “A purely capacitive synaptic matrix for fixed-weight neural networks,” *IEEE Transactions on Circuits and Systems*, vol. 38, pp. 210–217, Feb 1991.

[123] L. Fick, D. Blaauw, D. Sylvester, S. Skrzyniarz, M. Parikh, and D. Fick, “Analog in-memory subthreshold deep neural network accelerator,” in *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4, April 2017.

[124] G. Papandroulidakis, A. Serb, A. Khiat, G. V. Merrett, and T. Prodromakis, “Practical implementation of memristor-based threshold logic gates,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, pp. 3041–3051, Aug 2019.

[125] D. Kahng and S. M. Sze, “A floating gate and its application to memory devices,” *The Bell System Technical Journal*, vol. 46, pp. 1288–1295, July 1967.

[126] V. Srinivasan, D. W. Graham, and P. Hasler, “Floating-gates transistors for precision analog circuit design: an overview,” in *48th Midwest Symposium on Circuits and Systems, 2005.*, pp. 71–74 Vol. 1, Aug 2005.

[127] P. C. Y. Chen, “Threshold-alterable Si-gate MOS devices,” *IEEE Transactions on Electron Devices*, vol. 24, pp. 584–586, May 1977.

[128] F. Khan, E. Cartier, J. C. S. Woo, and S. S. Iyer, "Charge trap transistor (CTT): An embedded fully logic-compatible multiple-time programmable non-volatile memory element for high- k -metal-gate CMOS technologies," *IEEE Electron Device Letters*, vol. 38, pp. 44–47, Jan 2017.

[129] Y. Du, L. Du, X. Gu, J. Du, X. S. Wang, B. Hu, M. Jiang, X. Chen, J. Su, S. S. Iyer, and M.-C. F. Chang, "An analog neural network computing engine using CMOS-compatible charge-trap-transistor (CTT)," 2017.

[130] R. Berdan, T. Prodromakis, and C. Toumazou, "High precision analogue memristor state tuning," *Electronics Letters*, vol. 48, pp. 1105–1107, August 2012.

[131] B. A. Davey and H. A. Priestley, *Introduction to lattices and order*. Cambridge university press, 2002.

[132] Y. Crama, "Dualization of regular boolean functions," *Discrete Applied Mathematics*, vol. 16, no. 1, pp. 79 – 85, 1987.

[133] F. Dan Foresee and M. T. Hagan, "Gauss-Newton approximation to Bayesian learning," in *Proceedings of International Conference on Neural Networks (ICNN'97)*, vol. 3, pp. 1930–1935 vol.3, June 1997.

[134] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[135] M. Schmidt, G. Fung, and R. Rosales, "Fast optimization methods for L1 regularization: A comparative study and two new approaches," in *Machine Learning: ECML 2007* (J. N. Kok, J. Koronacki, R. L. d. Mantaras, S. Matwin, D. Mladenič, and A. Skowron, eds.), (Berlin, Heidelberg), pp. 286–297, Springer Berlin Heidelberg, 2007.

[136] L. Prechelt, *Early Stopping - But When?*, pp. 55–69. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998.

[137] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transactions on Neural Networks*, vol. 1, pp. 239–242, June 1990.

[138] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 1135–1143, Curran Associates, Inc., 2015.

[139] R. Setiono and W. K. Leow, "Pruned neural networks for regression," in *PRICAI 2000 Topics in Artificial Intelligence* (R. Mizoguchi and J. Slaney, eds.), (Berlin, Heidelberg), pp. 500–509, Springer Berlin Heidelberg, 2000.

[140] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *ArXiv*, vol. abs/1611.06440, 2016.

[141] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[142] R. Alvarez, R. Prabhavalkar, and A. Bakhtin, "On the efficient representation and execution of deep acoustic models," 2016.

[143] G. Dundar and K. Rose, "The effects of quantization on multilayer neural networks," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1446–1451, Nov 1995.

[144] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," 2017.

[145] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," 2019.

[146] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision

weights and activations," 2016.

[147] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016.

[148] M. Conti, S. Orcioni, and C. Turchetti, "Training neural networks to be insensitive to weight random variations," *Neural Networks*, vol. 13, no. 1, pp. 125 – 132, 2000.

[149] S. Grandhi, E. Dupraz, C. Spagnol, V. Savin, and E. Popovici, "CPE: Codeword prediction encoder," *accepted IEEE European Test Symposium*, May 2016.

[150] E. Popovici, S. Grandhi, C. Spagnol, V. Savin, and S. Cotofana, "Fault tolerant synthesis through error correcting codes driven graph augmentationg." `http://www.i-risc.eu/Lists/StaticFiles/delivrables/i-RISC_D5.2_v2.pdf`, February 2015 (last accessed on Jan. 2020).

[151] B. L. Happel and J. M. Murre, "Design and evolution of modular neural network architectures," *Neural Networks*, vol. 7, no. 6, pp. 985 – 1004, 1994. Models of Neurodynamics and Behavior.

[152] M. A. Cavuslu and S. Sahin, "FPGA implementation of ann training using levenberg and marquardt algorithms," *Neural Network World*, vol. 28, pp. 161–178, 01 2018.

[153] A. Mishchenko, B. Steinbach, and M. Perkowski, "An algorithm for bi-decomposition of logic functions," pp. 103– 108, 02 2001.

[154] B. Steinbach and C. Posthoff, *Logic Functions and Equations – Binary Models for Computer Science*. 01 2004.

[155] B. Steinbach and C. Lang, "Exploiting functional properties of boolean functions for optimal multi-level design by bi-decomposition," *Artif. Intell. Rev.*, vol. 20, pp. 319–360, 12 2003.

[156] S. Grandhi, D. McCarthy, C. Spagnol, E. Popovici, and S. Cotofana, "ROST-C: Reliability driven optimisation and synthesis techniques for combinational circuits," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pp. 431–434, Oct 2015.

[157] B. Steinbach and C. Posthoff, "Boolean differential calculus—theory and applications," *Journal of computational and theoretical nanoscience*, vol. 7, no. 6, pp. 933–981, 2010.

[158] B. Steinbach and C. Posthoff, "Boolean differential equations: A common model for classes, lattices, and arbitrary sets of boolean functions," *Facta universitatis-series: Electronics and Energetics*, vol. 28, no. 1, pp. 51–76, 2015.

Appendix A

Derivation of Back-propagation Algorithm

The basic idea of the *BP* algorithm is that the prediction is propagating forward while the error is propagating backwards through the network. Figure A.1 depicts a compact representation of an *MLP* similar to the one shown in Figure 2.10.

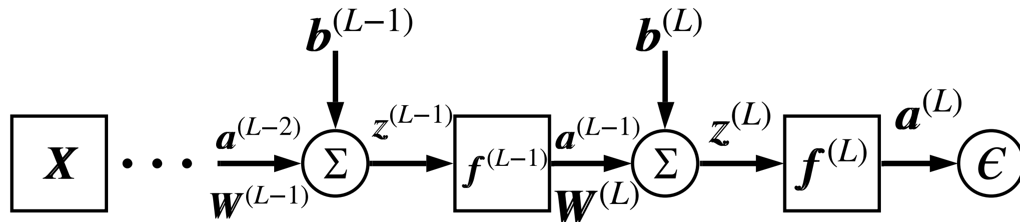


Figure A.1: Compact *ANN* model for *BP* demonstration

Note that the shown *MLP* has L layers and the output of each layer is denoted by $\boldsymbol{a}^{(l)}$. Specially, the input $\boldsymbol{X} = \boldsymbol{a}^{(0)}$, the output $\boldsymbol{o} = \boldsymbol{a}^{(L)}$ and the input of layer l is $\boldsymbol{a}^{(l-1)}$. The weighted sum of inputs of each layer with $\boldsymbol{w}^{(l)}$ is equal to $\boldsymbol{z}^{(l)}$ and will be fed into the activation function of each layer denoted by $\boldsymbol{f}^{(l)}$.

Assuming *Sum Squared Error* (SSE) to be the cost function ϵ:

$$\epsilon = \frac{1}{2}\sum_{p=1}^{P} \|\boldsymbol{t}_p - \boldsymbol{o}_p\|^2, \tag{A.1}$$

where P means the number of samples in a batch.

Then an intermediate term *error derivative vector* $\boldsymbol{\delta}^{(L)}$ is defined as the first-order partial derivative of the total error function with respect to the weighted sum output $\boldsymbol{z}^{(L)}$ can be obtained using *Chain Rule* of the multivariate calculus:

$$\boldsymbol{\delta}^{(L)} = \frac{\partial \epsilon}{\partial \boldsymbol{a}^{(L)}} \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} = \nabla_{a^{(L)}} \odot f'^{(L)}(\boldsymbol{z}^{(L)}), \tag{A.2}$$

where $\odot$ denotes a vector *element-wise product* operator. Specially, for the case of *SSE* function $\nabla_{a^{(L)}} = \boldsymbol{t}_p - \boldsymbol{o}_p$. Note that the activations of the neurons are assumed to be identical in the above descriptions.

Now the *error gradient vector*, $\nabla_{w^{(L)}}$ and $\nabla_{b^{(L)}}$ can be defined as the first-order partial derivative of the total error function with respect the L-th layer network parameters $\boldsymbol{w}^{(L)}$ and $\boldsymbol{b}^{(L)}$, respectively:

$$\begin{aligned} \nabla_{w^{(L)}} &= \frac{\partial \epsilon}{\partial \boldsymbol{w}^{(L)}} = \frac{\partial \epsilon}{\partial \boldsymbol{a}^{(L)}} \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} \frac{\partial \boldsymbol{z}^{(L)}}{\partial \boldsymbol{w}^{(L)}} = \boldsymbol{\delta}^{(L)} \boldsymbol{a}^{(L-1)} \\ \nabla_{b^{(L)}} &= \frac{\partial \epsilon}{\partial \boldsymbol{b}^{(L)}} = \frac{\partial \epsilon}{\partial \boldsymbol{a}^{(L)}} \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} \frac{\partial \boldsymbol{z}^{(L)}}{\partial \boldsymbol{b}^{(L)}} = \boldsymbol{\delta}^{(L)} \end{aligned} \tag{A.3}$$

Note that Equation A.4 can be generalized to any hidden layer as:

$$\begin{aligned} \nabla_{w^{(l)}} &= \boldsymbol{\delta}^{(l)} \boldsymbol{a}^{(l-1)} \\ \nabla_{b^{(l)}} &= \boldsymbol{\delta}^{(l)} \end{aligned} \tag{A.4}$$

Now, the core of the *BP* technique is the derivation of the error derivative of

layer (l) from that of next layer $(l+1)$:

$$\boldsymbol{\delta}^{(l)} = ((\boldsymbol{w}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}) \odot f'^{(l)}(\boldsymbol{z}^{(l)}) \tag{A.5}$$

In summary, the backwards pass of a training process starts from Equation A.2, then backwards propagates the error derivative through each layer with Equation A.5, where in each layer, the error gradient vectors can be calculated using Equation A.4.

Appendix B

Levenberg-Marquardt Algorithm

The *LM* algorithm is evolved from the *Newton's Method* and *Gauss-Newton Method*.

Newton's method using the parameters update rule like $\boldsymbol{w}_{j+1} = \boldsymbol{w}_j - \boldsymbol{H}_j^{-1}\boldsymbol{g}_j$, where $\boldsymbol{H}$ is the *Hessian Matrix*, which is the second-order derivatives of total error function and j is the index of iteration.

However, the computation of the Hessian Matrix is a very computationally intensive task. Thus, the Gauss-Newton method was invented, which approximates the Hessian Matrix using the Jacobian Matrix as $\boldsymbol{H} \approx \boldsymbol{J}^T\boldsymbol{J}$. Therefore, the update rule of Gauss-Newton method is like $\boldsymbol{w}_{j+1} = \boldsymbol{w}_j - (\boldsymbol{J}_j^T\boldsymbol{J}_j)^{-1}\boldsymbol{J}_j\boldsymbol{e}_j$.

Although the approximation works, the matrix $\boldsymbol{J}^T\boldsymbol{J}$ may not always be invertible, which may lead to convergence issue. Thus, the *LM* algorithm is introduced by including a learning hyper-parameter, μ, as:

$$\boldsymbol{H} \approx \boldsymbol{J}^T\boldsymbol{J} + \mu\boldsymbol{I}, \tag{B.1}$$

where $\boldsymbol{I}$ is the *Identity Matrix*.

Hence, the update rule of the *LM* algorithm becomes:

$$\boldsymbol{w}_{j+1} = \boldsymbol{w}_j - (\boldsymbol{J}_j^T\boldsymbol{J}_j + \mu\boldsymbol{J})^{-1}\boldsymbol{J}_j\boldsymbol{e}_j \tag{B.2}$$

The equation indicates that the *LM* algorithm works as switching between the *Steepest Decent Algorithm* and the Gauss-Newton method, by adjusting the hyper-parameter μ. Specifically, the *LM* is approaching to Gauss-Newton when μ is small and Steepest Descent when μ is large.

The pseudo-code of the algorithm implementation is depicted in Algorithm 4.

Algorithm 4 The *LM* Algorithm

Require: Training Samples $\{t, in\}$

```
 1: net.networkInit()                                  ▷ Initialing the parameters
 2: while cost > E_obj do                              ▷ Train until the objective reached
 3:     cost, jac, err = net.evaluate(t, in)           ▷ Evaluate the net, get the cost
 4:                                                    ▷ (sse), Jacobian and error
 5:     while True do                                  ▷ Loop until cost decreased
 6:         new_cost = net.learn(jac, err, μ)          ▷ Update the params with
 7:                                                    ▷ Equation B.2, and
 8:                                                    ▷ compute the new cost
 9:         if new_cost ≤ cost then
10:             μ ← μ × 0.1                            ▷ cost decreased, reduce
11:             cost ← new_cost                        ▷ μ and go to next eporch
12:             break
13:         else
14:             μ ← μ × 10                             ▷ cost increased, increase
15:                                                    ▷ μ, discard this update
16:                                                    ▷ and try another update
17:             restore network params to the state before this update
18:         end if
19:     end while
20: end while
```

Appendix C

HSPICE variation analyses

The basic usage of *HSPICE* variation analysis will be briefly introduced in this Appendix.

HSPICE delivered a convenient way to present the variations including both global and local variations called *Variation Block* (VB style). In a *VB*, most of the parameter variations, e.g., device model parameter, element parameter and top-level environment parameter variations, are supported. Also, various distributions such as *Uniform Distribution* and *Gaussian Distribution* for either independent or dependent random variables are provided. In addition, apart from the traditional *Simple Random Sampling* (SRS) method, some advanced more efficient sampling methods such as *Latin Hypercube Sampling* (LHS) and *Low-Discrepancy Sequence* (LDS) based methodologies are implemented.

- *SRS method*: The a sample value of the random variable obeying certain distribution is randomly generated simply without constraints using a pseudo-random number generator and the *Inverse Transform Sampling*. This method is mostly used to generate reference samples for verifying the goodness of a prediction model.
- *LHS method*: A hyperplane divided into N hyperplanes with equal probability will be filled by sample points which satisfies the Latin Hypercube, i.e., only one sample point can exit at a certain axis-aligned position in

the hyperplane. This sampling technique can generate evenly distributed random numbers and is effective when dealing with high dimensional random variables.

- *LDS method*: Low-Discrepancy geometrically represents that the gaps between samples in a sequence is small. *LDS* is a quasi-random sampling technique which is neither random nor pseudo-random, instead, the samples are constructed mathematically and later points in the sequence will try to fill the gaps between the former generated points. This technique can also effectively generate evenly distributed samples.

A typical *VB* structure is illustrated by List. C.1

Listing C.1: Variation block typical structure

```
.VARIATION
  OPTION ...
  .GLOBAL_VARIATION
    device model parameter global variations ...
    top parameter variations ...
    ...
  .END_GLOBAL_VATIATION
  .LOCAL_VARIATION
    device model parameter local variations ...
    element parameter local variations ...
    subcircuit parameter local variations ...
    ...
  .END_LOCAL_VATIATION
.END_VARIATION
```

In the *Option* section, sampling methods, random seeds, and other options and switches can be specified to control the variation simulation and analyses. In the *Global Variation* sub-block, a single value is generated for all devices in each simulation while in the *Local Variation* sub-block, a value is generated for each

individual device specified by model name, element name or subcircuit name in each simulation.

Once an *HSPICE* simulation job is concluded, some useful report files will be generated, including:

- *.mc#* file(s) that contains a $N \times M$ data matrix, where N is the number of samples and M the number of captured device parameters;
- *.mt#* file listing $N \times K$ measured observations, where N is the number of samples and K the number of measured performance metrics.

Appendix D

SPICE Simulation Code Snippets

Listing D.1: NAND gate training dataset generation SPICE variation block

```
.VARIATION
Option  Sampling_Method=LHS
.GLOBAL_VARIATION
   Parameter  univV=U()
   Parameter  Cloadvar=U()
   Parameter  Cavar=U()
   Parameter  Cbvar=U()
   Top  nominal=Perturb('0.5*univV')
   Top  load=Perturb('10f*Cloadvar')
   Top  c_prl_a=Perturb('10f*Cavar')
   Top  c_prl_b=Perturb('10f*Cbvar')
.END_GLOBAL_VATIATION
.LOCAL_VARIATION
   Parameter  VthnVar=U()
   Parameter  VthpVar=U()
   Parameter  LeffnVar=U()
   Parameter  LeffpVar=U()
   NMOS  NMOS_VTL  vth0=Perturb('20*VthnVar')%
   PMOS  PMOS_VTL  vth0=Perturb('20*VthpVar')%
```

```
    NMOS  NMOS_VTL  XL=Perturb('60*LeffnVar')%
    PMOS  PMOS_VTL  XL=Perturb('60*LeffpVar')%
  .END_LOCAL_VATIATION
.END_VARIATION
```

Listing D.2: NAND gate training dataset generation SPICE variation block

```
  .VARIATION
  Option  Sampling_Method=LHS
  .GLOBAL_VARIATION
    NMOS  NMOS_VTL  vth0=1.25%
    PMOS  PMOS_VTL  vth0=1.25%
    NMOS  NMOS_VTL  XL=1.25%
    PMOS  PMOS_VTL  XL=1.25%
  .END_GLOBAL_VATIATION
  .LOCAL_VARIATION
    NMOS  NMOS_VTL  vth0=1.25%
    PMOS  PMOS_VTL  vth0=1.25%
    NMOS  NMOS_VTL  XL=2.5%
    PMOS  PMOS_VTL  XL=2.5%
  .END_LOCAL_VATIATION
.END_VARIATION
```