A Study on Unintentional and Intentional Sources of Variability in Nanometer Scale Digital Circuits

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Deepashree Sengupta

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Sachin S. Sapatnekar, Advisor

April, 2017

© Deepashree Sengupta 2017
ALL RIGHTS RESERVED

Acknowledgements

First of all, I want to thank my advisor, Prof. Sachin Sapatnekar for his support throughout my graduate studies. It has been an honor working with a great researcher like him. This thesis is an amalgamation of his vast acumen, intellect, and dedication, as well as his encouragement that boosted my efforts all along my graduate studies. Over the past five years, he has been an incredible mentor, guiding me through challenging problems, helping me organize my thoughts, and teaching me to pay attention to detail. I also want to thank him for helping me develop the skill of technical writing which I lacked completely at the beginning of my graduate studies. The time I have worked with him has been the most productive five years of my life, and I am grateful for that.

I want to thank my committee members at the University of Minnesota: Prof. Chris Kim, Prof. Pen-Chung Yew, and Prof. John Sartori, for their constructive feedback about my research. I am grateful to the Doctoral Dissertation Fellowship and the ECE Department Fellowship selection committee at the University of Minnesota, Semiconductor Research Corporation (SRC), and the National Science Foundation (NSF), for the resource and financial support towards my thesis projects. I want to express my gratitude to the ECE staff, especially Carlos and Chimai, for their help and support.

I am grateful to my teachers at IIT Kharagpur, especially, Prof. Siddhhartha Sen, Prof. Aurobindo Routray, and Prof. Siddhartha Mukhopadhyay for encouraging me to pursue a PhD. I am also thankful to my IIT friends for making life at IIT a memorable experience and supporting me through various ups and downs in my life.

I want to thank Dr. Vivek Mishra, Dr. Saket Gupta, Dr. Sravan Marella, and Sri Harsha Vadlamani for their help during the starting years of graduate school with respect to navigating through the various CAD tools needed for my projects, and insightful inputs about my research. I would also like to thank Brandon, Meghna, Zhaoxin,

Farhana, Tengtao, Qianqian, and Masoud for being wonderful labmates.

I would not be at this stage of my life without the support of my family. I want to thank my parents, Sujata Sengupta and Ashoke Sengupta for their belief in my capabilities. I also want to thank my ever optimistic husband and best friend, Arja Ray, without whose support and constant encouragement, I could not have finished my PhD. He deserves almost as much credit as my advisor in terms of providing constructive criticisms on my work, and dealing with failures and rejections. He has also been a tremendous support ensuring that I eat and sleep regularly while writing this thesis for the last few months. I also want to thank my brother-in-law, Dr. Baidurja Ray, and sister-in-law, Dr. Ishita Chakraborty for helping me realize that PhD is really the best time of one's life, and it should always be cherished.

Last but not the least, I want to thank my friends in Minneapolis, and I feel lucky to have a group to hang out with, outside work. Thanks to Dr. Avijit Barik, Dr. Suvankar Biswas, Dr. Nilanjana Banerji, Dr. Amit Mitra, Dr. Taraswi Mitra, Shubhabrata, Abhirup, and Shriya, for the get-togethers and happy hours, and the opportunity to discuss about various interesting topics, thus making graduate school a joyful experience.

Dedication

To my parents and my wonderful husband.

Abstract

As technology has scaled aggressively over the past 50 years, device reliability issues and escalated power dissipation have become growing concerns in digital very large scale integrated (VLSI) circuits. Today's integrated circuits, which implement digital systems with billions of transistors in about a square centimeter, are tremendously susceptible to inadvertent errors, and circuit modifications required to ensure error resilience incur significant power overheads that are growing alarmingly as transistor dimensions shrink. Technology scaling has thus caused a resounding effect on the performance of digital circuits due to process, voltage, temperature, and usage-related variations, and increased power dissipation, resulting in lower battery life and increased system costs.

Unintentional reliability failures due to circuit aging pose serious threat for safety-critical and security applications, and must be mitigated. Intentional errors, on the other hand, can be introduced to a limited extent in circuits pertaining to error-tolerant applications, to reduce the system power without significantly affecting user experience. In this thesis, we study these two aspects of circuit reliability, i.e., the *unintentional reliability* failures arising out of circuit aging, and the *intentional unreliability* introduced in circuits implementing error-tolerant applications to reduce system power.

Temporal variations are injected into circuits due to aging during the normal operation of a chip. The predominant aging effects that cause circuit delay shifts over time are bias temperature instability (BTI) and hot carrier injection (HCI), both of which cause long-term degradations in transistor performance, and result in unintentional reliability issues in circuits. These effects are exacerbated as transistor sizes reduce, causing temporal delay degradations at the circuit level, thus introducing inadvertent errors during computations. Typically, these errors are mitigated by guardbanding through circuit overdesign or by increasing the supply voltage to speed up the systems. However, both result in wasted system power, and there is a need for effective aging estimation, so that just enough adaptive techniques can be applied, for error-free operation of a circuit.

On the other hand, numerous applications related to recognition, mining, and synthesis, especially those from image and audio processing domains, are error tolerant, since they pertain to the inherently limited human perception. A new design paradigm

called approximate computing, leverages this error-tolerance to implement arithmetic operations through approximate circuits. In other words, circuits implementing these applications can be intentionally designed to be unreliable, to achieve significant speed-up and system power savings through simplified hardware to perform complex arithmetic operations. The desired accuracy is often a user-specified input, and a there is a need to quantify the error injected into a computation as a function of the extent of approximation to systematically obtain the tradeoff between accuracy and power savings achieved in approximate circuits to aid their design.

For applications where aging-related errors are a critical problem, the first half of the thesis proposes efficient aging sensor schemes that enable system adaptation for error-free operation. On-chip ring-oscillator-based (ROSC-based) structures are chosen as the surrogate aging sensors and this thesis presents a method for inferring circuit delay shifts due to BTI and HCI aging from these sensors. The proposed method efficiently computes calibration factors that translate delay shifts in the ROSCs to those in the monitored circuits within 1% of the true values. These factors are shown to be independent of temperature and supply voltage variations in practice. Further, a refinement strategy is proposed where the sensor measurements are amalgamated with infrequent online delay measurements on the monitored circuit to partially capture its true workloads, leading to 8% lower delay guardbanding overheads compared to the conventional methods.

For error-tolerant applications, the second half of the thesis proposes algorithms for error analysis, and design of approximate circuits. The proposed analysis algorithms generate the distribution of the error injected into a computation when implemented using approximate arithmetic circuits. These algorithms are based on the Fourier and the Mellin transform to efficiently compute the total error accumulated at the output of an approximate circuit abstracted as a directed acyclic graph, through its topological traversal. The resulting error distribution is obtained much faster than Monte Carlo simulations, with the error statistics being within 2% of their true values. The proposed design algorithm uses the second moment of this distribution as a guideline to construct approximate arithmetic circuits through an optimization problem which maximizes their power savings while constrained by a user-specified error budget. Fast heuristics have been proposed to solve the integer non-linear optimization problem, and over 30% improvement in power savings is achieved compared to the conventional methods.

Contents

Acknowledgements **i**

Dedication **iii**

Abstract **iv**

List of Tables **ix**

List of Figures **xi**

1 Introduction **1**

1.1 Unintentional Reliability 1

1.2 Intentional Unreliability 4

1.3 Thesis Organization 6

2 Background on BTI- and HCI-induced Aging **8**

2.1 Aging Models for BTI and HCI 8

2.1.1 BTI-induced Aging 9

2.1.2 HCI-induced Aging 10

2.2 Effect of BTI and HCI on Delay Degradation 11

2.3 Conclusion 12

3 Aging Estimation using the UofM Delay Model **13**

3.1 Delay Estimation and Aging Prediction 16

3.1.1 Presilicon Circuit Characterization 17

3.1.2 Post-silicon Circuit Aging Estimation from ROSC Sensors 19

3.1.3 Effects of Operating Conditions on Degradation Ratios 20
3.2 Experimental Setup and Results 23
3.2.1 Aging Estimation from ROSCs using the UofM Bound 24
3.2.2 V_{dd} and T Independence of the UofM Bound 25
3.3 Conclusion 26

4 Aging Estimation through ReSCALE 27
4.1 Sensor Recalibration and Aging Estimation 29
4.1.1 Delay Bounds based on Post-silicon Circuit Measurements 30
4.1.2 Post-recalibration Aging Estimation in a Circuit 33
4.2 Experimental Setup and Results 34
4.2.1 Speed Wastage Factor 35
4.2.2 Choice of Measurement Instants 36
4.3 Conclusion 40

5 Error Analysis in Unsigned Multipliers 41
5.1 Characterizing the PMF of Full Adders 44
5.2 Overview of the FEMTO Algorithm 46
5.2.1 Convolution using the Z-transformed Domain 49
5.2.2 Inverse Fast Fourier Transform to infer the Error PMF of Multipliers 50
5.3 Enhancing Efficiency of FEMTO 51
5.4 Experimental Setup and Results 53
5.5 Conclusion 58

6 Generalized Error Analysis in Approximate Circuits 59
6.1 A DAG Model for an Approximate Circuit 61
6.2 Output Distribution of Full Adders 63
6.3 Distribution of Generated Error in DAG Nodes 66
6.3.1 PMF of Generated Errors in Approximate Adders 68
6.3.2 PMF of Generated Errors in Approximate Multipliers 73
6.4 Distribution of Output Error in a DAG 77
6.4.1 Error Propagation through Adders 78
6.4.2 Error Propagation through Multipliers 79

6.4.3 Error PMF at Output Nodes of a DAG 81
6.5 Experimental Setup and Results 81
6.5.1 PMF of Generated Error in Approximate Nodes 82
6.5.2 Output Distribution in DAGs 84
6.6 Conclusion 87

7 Approximate Circuit Design through SABER 88
7.1 Error Characterization 90
7.1.1 Error Precharacterization for an Adder 90
7.1.2 Error Computation of a DAG 93
7.2 Optimization through SABER 94
7.2.1 The Optimization Problem 95
7.2.2 Heuristics to Solve the Original Problem 97
7.3 Experimental Setup and Results 99
7.3.1 Optimization Results on an Example DAG 99
7.3.2 Optimization Results on FIR Filters 101
7.4 Conclusion 105

8 Thesis Conclusion 106

References 107

Appendix A. Proof of Theorems 119
Proof of Theorem 3.1: 119
Proof of Theorem 3.2: 121
Proof of Theorem 4.1: 123
Proof of Theorem 7.1: 124
Proof of Theorem 7.2: 125

Appendix B. Miscellaneous Derivations 126
Derivation of E in Eq. (5.12): 126
Derivation of $\mathcal{E}$ in Eq. (6.15): 127
Derivation of $\mathcal{F}$ in Eq. (6.30): 127
Derivation of PMF of $A_t\Delta B$ and $B_t\Delta A$ in Eq. (6.39): 128

List of Tables

1.1 Demonstration of the inherent error-tolerance of human perception. . . 4

3.1 Degradation ratios from presilicon analysis. 24

4.1 Statistics of $\boldsymbol{E_\Delta}$ with sensor recalibration, expressed as percentage, for various sets of measurement instants, T_M, in years. 39

5.1 PMF of the FA output. 46

5.2 PMF of the errors in FA output. 46

5.3 Five versions of approximate adders from [1]. 53

5.4 Normalized percentage error in estimated mean ($\Delta\mu_{norm}$) of PMF obtained by FEMTO compared against Monte Carlo simulation. 55

5.5 Normalized percentage error in estimated standard deviation ($\Delta\sigma_{norm}$) of PMF obtained by FEMTO compared against Monte Carlo simulation. 55

5.6 Runtime comparison to obtain the error PMFs. 57

6.1 PMFs associated with of the FA output (Sum) and output errors (Δs and ΔSum). 65

6.2 The values of $\Delta\sigma_{norm}$ for approximate adders and multipliers 84

6.3 Normalized percentage errors in estimated standard deviations ($\Delta\sigma_{norm}$ defined in Eq. (6.40)) for five DAGs from [2]. 86

6.4 Runtime of our algorithm and 6000 Monte Carlo simulations. 87

7.1 Fitting parameters for adder error variance. 92

7.2 Total number of approximate FAs by SABER and the uniform approximation case for different error budgets. 101

7.3 Power dissipation of the approximate filters by SABER and uniform approximation, for three values of error budget, m, compared to the accurate filter. 104

7.4 SNR degradation (in dB) between the accurately filtered signal and those from the approximate filters constructed for different error budgets, m, using SABER. 104

List of Figures

3.1 ROSC-based aging sensors [3] interspersed within multiple circuits along with LUT of degradation ratios. 14
3.2 CUT delay as maximum of path delays under aging. 16
3.3 Estimated delays across different (V_{dd}, T) from degradation ratios at a fixed (V_{dd}, T), indicating V_{dd} and T independence the UofM scheme. . . 25
4.1 ROSCs interspersed within multiple circuits along with LUT of degradation ratio and test pattern storage block for direct CUT measurement. . 28
4.2 Upper-bounding the CUT delay with intermediate CUT measurements. 30
4.3 Example of the recalibration method for aging estimation in two circuits: mem_ctrl follows Case I (left), while i2c follows Case II (right). 32
4.4 Effect of the choice of t_{m_1} on average SWF. 37
4.5 Reduction in average SWF with number of measurement instants. . . . 37
5.1 Schematic of the FEMTO algorithm on an unsigned multiplier. 43
5.2 Full adder (FA) with the associated truth table (*appx1* from [1]). 45
5.3 Output signal and error distribution for the *appx1* adder from [1]. 45
5.4 Structure of a 4-bit×4-bit array multiplier. 47
5.5 An N-bit×N-bit multiplier from N/K-bit×N/K-bit multipliers. 52
5.6 An error PMF (not to scale) with unit-granularity (left) and its binned version into five windows (right). 53
5.7 CDF of the normalized error for 8-bit×8-bit multipliers with different percentages (45%, 50% and 55%) of approximate LSBs in the product. The adders in (a)-(e) correspond to those from Table 5.3. 56

5.8 Hellinger distance between Monte Carlo-generated PMF, and PMFs obtained by FEMTO and our implementation of MIA for (a) 6-bit×6-bit, and (b) 8-bit×8-bit multipliers. 58
6.1 Representation of a DAG with approximate operations, producing erroneous outputs which can be characterized by PMFs. 62
6.2 Representation of y approximate LSBs, where the subscript, i, refers to the values at the ith bit position. 62
6.3 Full adder (FA) with the associated truth table (*appx1* from [1]). 64
6.4 Output signal and error distribution for the *appx1* adder from [1]. 64
6.5 Approximate node of a DAG with exact inputs and erroneous output, where the error, ΔR, is generated at this node. 66
6.6 A signed adder where some of the LSB FAs can be approximated to introduce approximation in the circuit. 67
6.7 A signed array multiplier where some of the LSB FAs can be approximated to introduce approximation in the circuit. 67
6.8 (a) Approximate adder, and (b) multiplier with inputs, A and B, both of which can be erroneous, producing approximate output, C. 78
6.9 PMF of the product, $C_1 = A_t \Delta B$, as a sum of four PMFs depending on the sign of the two operands, A_t and ΔB. 81
6.10 Distribution of error in an 10-bit signed adder (top row) and 10-bit×10-bit signed multiplier (botom row) implemented with the appx1-appx5 versions of the FAs from [1]. 83
6.11 Distribution of error in the deepest output node of the DAGs consisting of 10-bit adders and multipliers implemented with the appx1-appx5 versions of the FAs from [1]. 84
6.12 Hellinger distance between PMFs obtained by our approach and Gaussian approximation, as compared to the Monte Carlo PMFs. Our method is often well below the threshold, and our Hellinger distance is generally better than or comparable to the Gaussian approximation. 85
7.1 Error variance due to the uncorrelated and correlated adder inputs as a function of y. 93

7.2 (a) A DAG consisting of adders and multipliers, (b) Representation of multipliers as shift and add operations. 93
7.3 Example of an optimization problem similar to (7.5). 97
7.4 Structure of DAG10 with ten nodes. 100
7.5 Distribution of the number of approximate LSBs over the ten nodes of DAG10. 100
7.6 An FIR filter with symmetric coefficients [4]. 102
7.7 Tradeoff plot of SNR degradation with error variance in the FIR filter used to select the error budget. 103
7.8 Distribution of the number of approximate LSBs over the 63 nodes of the FIR filter. 103
A.1 (a) Maximum among the $x_i(t)$ functions, denoted by $x_M(t)$, (b) Smooth upper-bounded estimate of $x_M(t)$, denoted by $y(t)$. 119

Chapter 1

Introduction

Integrated circuits (ICs) constitute the core of almost all modern electronic machines that influence our day-to-day lives, from laptops, smartphones, medical devices, to automotive and defense applications. The demand for systems with high performance has increased tremendously, driving the shift of CMOS technology towards more deeply scaled nanometer feature sizes according to Moore's Law. While greater on-chip device integration within the same chip area offers higher computing capabilities, the aggressive scaling has also resulted in a concomitant increase in performance unreliability in circuits due to process, voltage, temperature, and aging induced variations, accompanied by a steep increase in power consumption and dissipation to overcome such variations.

While performance reliability of circuits is imperative in safety-critical and security applications, deliberate unreliability is permissible in certain error-tolerant applications, thus allowing relaxation of design effort and reduction of system power. In this thesis, we study these two aspects of circuit reliability, i.e., the *unintentional reliability* arising out of circuit aging and how it can be estimated to ensure the desired chip performance, as well as the *intentional unreliability* introduced in circuits to reduce system power by leveraging the error-tolerance of certain applications.

1.1 Unintentional Reliability

With continued scaling, the susceptibility of nanometer-scale transistors to aging-related wear-out phenomena has increased significantly [5]. Aging causes a circuit to fail, either

catastrophically in being unable to achieve correct logic functionality, or parametrically, in being able to achieve correct logic functionality but not at the correct specifications (e.g., the timing specifications may be violated). If the extent of circuit degradation due to aging can be correctly sensed, appropriate compensation techniques can be applied to ensure reliable circuit operation. In this thesis we focus on two major degradation mechanisms, bias temperature instability (BTI) [6,7] and hot carrier injection (HCI) [8], which result in parametric failures in circuits, and propose novel techniques using surrogate sensors to estimate circuit aging due to these two mechanisms.

Both BTI and HCI affect the transistor drive current by degrading the threshold voltage and mobility, respectively, under voltage and temperature stress experienced during circuit operation. While BTI is partially reversible on the removal of stress, HCI is an irreversible effect. The long-term degradation due to BTI depends on the average duty cycle of the stressing signal, or the signal probability (SP), which is the probability that the signal is at the logic high level. Degradation due to HCI, on the other hand, occurs only when a transistor switches and hence, HCI degradation depends on the time spent in switching, which is proportional to the signal activity factor (AF), i.e., the ratio of average number of signal transitions to clock transitions, and the clock frequency. In practice, most transistors in a circuit tend to switch infrequently, and therefore HCI is often dominated by BTI [9]. However, over long periods, HCI grows at a faster rate than BTI, and therefore its effects can be noticeable for long-lifetime parts [10].

At a fixed supply voltage, since the degradation in each transistor adversely affects its delay, circuit aging is manifested as reduction in its maximum operating frequency, $\mathcal{F}_{Max}$, with time. At the presilicon design phase, the foreknowledge of the average workload of a circuit in the field is often unavailable. Hence, the schemes deployed at this phase, provide protective, albeit pessimistic, guardbands over $\mathcal{F}_{Max}$ of the circuit so that it is guaranteed to work under all operating conditions throughout its lifetime. Since both BTI and HCI aging depend on the average signal probability and activity factor (SPAF) of the stressing signals, it is common practice to choose pessimistic SPAF values for every transistor within the circuit to mimic the worst-case workload [11].

At the post-silicon stage, to ensure that a chip meets its timing requirements over its lifetime, various compensation techniques are employed during its field operation, such as clock frequency adjustment, supply voltage scaling, and body bias modification [12–

14]. These techniques typically use data from surrogate sensors, built in at the presilicon phase and tested at the post-silicon stage, to adaptively provide on-the-fly compensation to mitigate the effects of aging. These sensors range from simple inverter-chain-based circuits [3, 15–18] to circuits based on representative critical paths [19, 20].

To a limited extent, surrogate sensors may successfully capture the environment faced by the circuit, e.g., if they are placed close to the circuit and have a similar connection to the power grid, they can capture the thermal and supply voltage environment, and undergo similar shifts due to systematic or spatially-correlated process variations. However, since these sensors are mere surrogates, they are unable to reflect aging in the circuit with complete accuracy. This inability arises due to the structural differences between the near-critical paths of the circuit under test (CUT) and the sensor. At the device level, the transistors in the sensor experience different stressing input patterns as compared to the CUT, and also have different delay sensitivities to aging shifts, due to which they age differently. At the circuit level, the number of near-critical paths in the CUT, which could potentially become critical under aging, is typically much larger than those in the sensor. A ring oscillator sensor has just one path, and although it is possible to build surrogate sensors based on representative critical path circuits (RCPs) [19, 20], their design overhead is significant. Additionally, the cost of constructing RCP circuits that could cover a sufficient number of critical paths could be onerous.

In this thesis, we aim to infer delay shifts due to BTI- and HCI-induced aging in a CUT based on delay shifts measured from ring-oscillator-based (ROSC-based) sensors. These sensors are widely used because they are cheap, compact, and can be easily replicated many times within a chip. Specifically, we use the sensors in [3], which can separately measure the contribution of BTI and HCI to their delay shifts.

We propose two post-silicon schemes to estimate the delay degradation of a CUT, in the first part of our thesis that deals with the unintentional unreliability. The first scheme translates measurement from surrogate aging sensors to circuit delay degradations using a look-up table (LUT). The second scheme called, ReSCALE (Recalibration of Sensor Circuits for Aging and Lifetime Estimation), amalgamates these sensor measurements with very infrequent measurements performed directly on the CUT, and uses both to update the LUT. The updated values are then used in conjunction with inexpensive sensor measurements to infer the delay of the CUT.

There is a trade-off between the simplicity of implementation versus the accuracy of aging estimates obtained by the two proposed techniques. While the first one is very easy to implement without the need of any additional circuitry other than the ROSC-based sensors, the second one is more accurate, at the cost of higher design effort and overheads due to the additional CUT delay measurement and the LUT update.

1.2 Intentional Unreliability

Errors in a circuit operation are not always undesirable, and circuit design can, in fact, be greatly simplified if errors are allowed in the system. These errors, when manifested in applications pertaining to human perception, are permissible to a certain extent due to the inherent limitation of human senses. For example, Table 1.1, lists some of the limits of human perception pertaining to visual and auditory senses, as a result of which it is often difficult to differentiate between various accurate and approximate results of a computation, up to a certain extent. The psychological quantities listed in Table 1.1 are not exhaustive, but are mainly related to the various aspects of digital signal processing (DSP), and hence, recognition, mining, and synthesis applications pertaining to image, video, and audio processing [21] have significant levels of error-tolerance.

Table 1.1: Demonstration of the inherent error-tolerance of human perception.

Psychological quantities	**Human perception limit**
Angular resolution	1 arcminute
Threshold of hearing	-9 dB (sound pressure level)
Flicker fusion threshold	60-90 Hz

Approximate computing [22] is a new design paradigm that leverages the inherent error-tolerance of human senses, and can potentially achieve large efficiencies in the design by deliberately introducing errors in computation by either simplifying a circuit logic or architecture, or by modifying its operating conditions (supply voltage and frequency). Hence, such intentional unreliability in computation can reduce hardware costs, in terms of energy, power, and area.

One of the vital ingredients of any methodology based on approximate design is a fast and accurate procedure that can quantify the distribution of error injected into

a computation by an approximation scheme. The most common building blocks in DSP applications, specifically for error-resilient computations, are adders [1,23–31] and multipliers [32–35]. Hence, we first propose an **analysis algorithm** to obtain the probability mass function (PMF) of the errors injected into a computation, when it is performed on such approximate circuits. Once a method to analytically quantify these errors is in place, in keeping with the spirit of approximate computing, the logical next step is to design an approximate circuit with fixed error constraints, while trying to maximize its power savings. Hence, we next propose a **design algorithm** that precisely achieves this for the design of approximate circuits, with user-specified error budgets.

For the **analysis algorithm**, we follow a bottom-up approach, where we first quantify the errors in the fundamental block of an approximate circuit. Since we consider the transistor level approximations where errors are introduced by simplifying the logic of full adders (FAs) [1, 28], the fundamental block in our analysis is an FA. Next, we proceed to approximate adders and multipliers which are simply arrays of these FAs, constructed suitably, based on their respective architectures. We consider the ripple carry and the modified Bough-Wooley two's complement architectures for the adder and the multiplier [36], respectively, for our analysis. Finally, we analyze an entire circuit that constitutes these approximate adders and multipliers, and propose an algorithm to quantify the error PMF at the primary outputs of such a circuit. The circuit is abstracted as a directed acyclic graph (DAG) [2], where each node is an approximate adder or a multiplier, and we perform a topological traversal of this DAG to compute the PMF of the error accumulated at each node. We use the Fourier transform to compute the error PMF at the output of the adder nodes, while we implement a combination of the Mellin transform [37] along with the Fourier transform, to compute the error PMFs at the output of the multiplier nodes within the DAG. We implement our algorithm at each hierarchical level using standard benchmarks, to demonstrate its effectiveness with respect to both accuracy and runtime.

Next, for the **design algorithm**, we abstract an approximate circuit as a DAG of adder nodes, while representing the multipliers by an equivalent network of adders and arithmetic shifters, and use the total error variance of this DAG as the error metric that is to be constrained by a user-specified budget. Our objective is to maximize the power savings of this DAG, without compromising the user experience which is actually

the specified error budget. We formulate the design problem through an integer non-linear optimization framework, and propose fast heuristics to solve it efficiently. Our algorithm has been implemented for the design of finite impulse response (FIR) filters, and the excellent performance of these approximate filters have been demonstrated for benchmark audio signals [38], thus proving the effectiveness of our approach.

1.3 Thesis Organization

The thesis is broadly divided into two parts. While Chapters 2, 3, and 4 relate to the unintentional reliability failures due to aging, Chapters 5, 6, and 7 pertain to the intentional unreliability issues through approximate computing. Specifically,

- Chapter 2 explains the two aging mechanisms, BTI and HCI, in further detail, and lays the foundation of the circuit aging model used in the next two chapters.
- Chapter 3 describes the circuit delay model (UofM) developed for aging estimation, and outlines the methodology to translate delay degradation of on-chip ROSC sensors measured at the post-silicon phase to that of the monitored circuit.
- Chapter 4 discusses the ReSCALE algorithm to refine the aging estimate by amalgamating post-silicon ROSC measurements with infrequent online delay measurements on the monitored circuit, and sensor recalibration.
- Chapter 5 presents the foundational algorithm, FEMTO, for obtaining error PMFs in approximate unsigned multipliers using the error PMF of the constituent FAs.
- Chapter 6 first extends the algorithm from Chapter 5 for signed adders and multipliers, and then presents a technique of propagating error PMFs through each node of an approximate DAG, using the concepts of transform calculus, to obtain the output error PMF of the DAG.
- Chapter 7 outlines the the SABER algorithm for the design of an approximate circuit through formulating and efficiently solving an optimization framework to maximize its power savings, with user-specified error budgets.
- Chapter 8 concludes the thesis.

The bibliography is provided at the end of Chapter 8, followed by two appendices. The proofs of the various theorems developed in this thesis are outlined in the first appendix. The second appendix provides the derivation of miscellaneous terms, used in Chapters 5 and 6, for better readability.

Chapter 2

Background on BTI- and HCI-induced Aging

Bias Temperature Instability (BTI) and Hot Carrier Injection (HCI) are predominant aging effects in transistors under normal operating conditions that result in parametric failure in circuits. While BTI degrades the threshold voltage of transistors under voltage and temperature stress, HCI is dependent on the switching activity of the transistors, and degrades their drive current. Together, they result temporal delay degradations at the circuit level. In this chapter, we describe models for BTI and HCI aging, their impact on delay degradation of individual transistors, and on larger circuits. This lays the foundation for the subsequent chapters for estimating the BTI- and HCI-induced aging using surrogate sensors.

2.1 Aging Models for BTI and HCI

Under BTI, a PMOS (NMOS) device is stressed when its gate voltage is negative (positive), leading to negative (positive) BTI, or NBTI (PBTI), while under HCI, a transistor is stressed only while it switches.

In the following discussion, we will assume that the observation time for the circuit starts at time, t_0, and continues until the lifetime of the circuit, t_f.

2.1.1 BTI-induced Aging

The precise mechanism of BTI is a matter of debate within the research community. Two candidates have emerged: the reaction-diffusion (RD) model [6] and the charge trapping (CT) model [39]. The threshold voltage shift is a cumulative effect of multiple cycles of stress and recovery. BTI is independent of the frequency of the stressing signal for frequencies higher than 1Hz, and only depends on its average duty cycle [7]. In general, at time, t, the threshold voltage shift, $\Delta V_{th_x}(t)$, $x \in \{n, p\}$, due to BTI in an NMOS or PMOS device (denoted by n and p) can be modeled as:

$$\Delta V_{th_x}(t) = C_x(f(t_{\text{st}}) - f(t_{\text{st},0})) = \psi_{B_x}(f(t) - f(t_0)) \tag{2.1}$$

where C_x is a constant dependent on the process, voltage and temperature (PVT) conditions of the device, $f(.)$ is a function that represents the temporal dependence of BTI aging, and the terms, $t_{\text{st},0}$ and t_{st}, refer to the effective stressing time after an elapsed time of t_0 and t, respectively. The effective stressing times are given by $t_{\text{st}} = \alpha t$ (and $t_{\text{st},0} = \alpha t_0$), where α is the stress probability for the device. Since a PMOS device is stressed when its input is at logic low level, $\alpha = 1 - s$, where s is the SP of the input signal. Similarly, for an NMOS device, $\alpha = s$, since it is stressed when its input is at logic high level. The $f(t_{\text{st},0})$ term thus relates to the aging of the chip that is built in at time, t_0. We can absorb the effect of the signal probability into ψ_{B_x} along with other PVT dependent parameters in C_x, and we note that $f(t)$ is purely dependent on the age of the circuit, t. In principle, $f(.)$ could be different for PMOS and NMOS devices, but these are experimentally observed to be similar, as documented in design manuals and the published literature [40]. Typically, $f(t)$ can assume either of the following forms based on the two models of BTI:

$$f(t) = \begin{cases} t^{n_1} & \text{, under the RD model} \\ a + b \log t & \text{, under the CT model} \end{cases} \tag{2.2}$$

where $n_1 \sim 0.16$ [41], and a and b are positive constants defined in [39]. Our analysis is designed to be general enough to be applicable on either form of $f(t)$. Although the V_{th}-shifts through multiple stress-recovery cycles are not monotonic, Eq. (2.1) captures the envelope of the delay function, including BTI recovery effects under AC stress.

2.1.2 HCI-induced Aging

At contemporary technology nodes, HCI affects NMOS devices more severely than PMOS [8, 42]. HCI occurs when carriers in the channel, subjected to a lateral electric field, gain sufficient energy and momentum to break the barriers of surrounding dielectric, such as the gate and sidewall oxides. A recent energy-driven framework for HCI stress proposes that carriers with sufficient energy can result in interface-state generation by impact ionization at the Si-SiO_2 interface directly without being injected into the gate oxide [8]. This leads to a gradual degradation in various electrical parameters of the transistors, thus affecting the circuit performance.

Based on [16, 43], we model HCI aging by expressing the drive current reduction as equivalent threshold voltage degradation, $\Delta V_{th_n}(t)$, of an NMOS after time, t, as:

$$\Delta V_{th_n}(t) = C_H \exp\left(\frac{E_{ox}}{E_0} - \frac{\phi_{it}}{q\lambda E_m}\right)(g(t_{\rm st}) - g(t_{\rm st,0})) \tag{2.3}$$

where C_H and E_0 are process dependent parameters, E_{ox} is the vertical field, ϕ_{it} is the trap generation energy, q is the electronic charge, λ is the hot electron mean free path, $t_{\rm st}$ and $t_{\rm st,0}$ are the effective stressing times, $g(.)$ is a function that encapsulates the temporal dependence of HCI aging, and E_m is the lateral electric field, given by:

$$E_m = \frac{V_{ds} - V_{dsat}}{L_{\mathit{eff}}} \tag{2.4}$$

$$\text{where } V_{dsat} = \frac{(V_{gs} - V_{th} + \frac{2k_BT}{q})L_{\mathit{eff}}E_{sat}}{V_{gs} - V_{th} + \frac{2k_BT}{q} + A_{bulk}L_{\mathit{eff}}E_{sat}} \tag{2.5}$$

The parameter, L_{eff}, is the effective channel length, T is the temperature, k_B is Boltzmann's constant, and A_{bulk} and E_{sat} are process-dependent constants defined in [44].

The effective stressing time, t_{st}, corresponding to HCI aging depends on the number of switching events experienced by the transistor, given by $(\mathrm{AF} \cdot \mathcal{F}_{clk} \cdot t)$, where AF is the activity factor for the transistor, $\mathcal{F}_{clk}$ is the clock frequency, and t is the elapsed time. During each of these switching events, the transistor is stressed during the time that the input signal makes its transition, given by the slew, t_{slew}. Hence, we can write $t_{\rm st} = (\mathrm{AF} \cdot \mathcal{F}_{clk} \cdot t)t_{slew}$, and this relation converts $g(t_{\rm st})$ to a function $g(t)$ of the elapsed time. Similarly, $g(t_{\rm st,0})$ can be converted to $g(t_0)$ as well. We thus absorb the effect of the switching activity and the time-independent parameters into ψ_{H_n} to rewrite

$\Delta V_{th_n}(t)$ as:

$$\Delta V_{th_n}(t) = \psi_{H_n}(g(t) - g(t_0)) \tag{2.6}$$

From experimental models [8], $g(t)$ is typically of the form:

$$g(t) = t^{n_2} \tag{2.7}$$

where $n_2 \sim 0.5$.

2.2 Effect of BTI and HCI on Delay Degradation

We denote the change in the underlying trend functions for BTI and HCI as:

$$\Delta f(t) = f(t) - f(t_0) \tag{2.8}$$

$$\Delta g(t) = g(t) - g(t_0) \tag{2.9}$$

Between time, t_0, to t, we can represent the shift in the delay, $D(t)$, of a logic gate as:

$$\Delta D(t) = \sum_{i \in \text{NMOS}} S_{n,i} \Delta V_{th_{n,i}}(t) + \sum_{i \in \text{PMOS}} S_{p,i} \Delta V_{th_{p,i}}(t) \tag{2.10}$$

where the two summations are taken over all NMOS and PMOS transistors in a gate. Here, the prefix, Δ, denotes a change in the quantity that succeeds it, $V_{th_{n,i}}(t)$ ($V_{th_{p,i}}(t)$) is the threshold voltage of the ith NMOS (PMOS) transistor in the gate, and $S_{x,i} = \left.\frac{\partial D}{\partial V_{th_{x,i}}}\right|_{t_0}$ for $x \in \{n, p\}$ is the sensitivity of transistor, i, to threshold voltage shifts. From Eqs. (2.1) and (2.6), we have:

$$\begin{aligned} \Delta V_{th_{n,i}}(t) &= \psi_{B_{n,i}} \Delta f(t) + \psi_{H_{n,i}} \Delta g(t) \\ \Delta V_{th_{p,i}}(t) &= \psi_{B_{p,i}} \Delta f(t) \end{aligned} \tag{2.11}$$

Therefore, we can rewrite Eq. (2.10) as:

$$\Delta D(t) = K_B \; \Delta f(t) + K_H \; \Delta g(t) \tag{2.12}$$

where $K_B = \left(\sum_{i \in \text{NMOS}} S_{n,i} \psi_{B_{n,i}} + \sum_{i \in \text{PMOS}} S_{p,i} \psi_{B_{p,i}}\right)$ and $K_H = \sum_{i \in \text{NMOS}} S_{n,i} \psi_{H_{n,i}}$. Thus, under fixed stress conditions of temperature, V_{dd}, SP, and AF, the delay is a function of time, and is easily computed if all the sensitivity values, $S_{n,i}$ and $S_{p,i}$, have been characterized for each gate.

2.3 Conclusion

While BTI and HCI are complex phenomena, the empirical models proposed in this chapter have been shown to fit the experimental data on aging. Since these models are dependent on various process parameters, the delay sensitivities to aging can be precharacterized for each gate in a gate library, similar to characterizing its nominal delay and slew values. Using the aging models, delay sensitivities of gates, operating conditions, and elapsed time a circuit has been in operation, the extent of its aging can be estimated in terms of its delay degradation.

Chapter 3

Aging Estimation using the UofM Delay Model

The overall effect of BTI and HCI is to reduce the maximum operating frequency, $\mathcal{F}_{Max}$, of a circuit over its lifetime. To ensure that a chip meets its timing requirements over its lifetime, compensation techniques need to be applied based on its extent of degradation. In the presilicon design, appropriate delay guardbands may be added [45, 46], while the postsilicon phase may adapt the circuit during operation in the field [12], using sensors built in at the presilicon phase, by adjusting its clock frequency, supply voltage, or body bias. However, by definition, presilicon techniques are unaware of the runtime operating environment experienced by a chip and must consider worst-case scenarios by assuming pessimistic stress conditions for the circuits. Postsilicon techniques limit pessimism and deploy just enough adaptive compensation, based on monitors that periodically evaluate $\mathcal{F}_{Max}$ in the circuit under test (CUT). Two classes of monitors may be employed:

- *Surrogate circuit monitors*: These are test circuits used to estimate $\mathcal{F}_{Max}$ degradation in the CUT by trying to emulate the operating conditions/functionalities of the CUT. These range from simple ring oscillators (ROSCs) [15], [18] to more complex representative critical path (RCP) circuits [19], [20], [47].

- *CUT monitors*: In these methods, delay tests are directly performed on the CUT at predetermined intervals to measure its performance in terms of $\mathcal{F}_{Max}$ degradation [48], [49].

CUT monitors are accurate since they directly monitor the CUT, but may suffer from large hardware and test time overheads. Although such tests are required infrequently and their runtime can be reduced [50], the overheads of testing the entire chip may still be onerous.

In this thesis, we use ROSC-based sensors as surrogate circuit monitors to characterize aging in a CUT. These sensors are widely used in industry, since they are compact and uniform, and require the design and layout of only a single repeatable macrocell. Specifically, we use the ROSC-based silicon odometer [3] as the aging sensor here, since they can separately monitor aging due to BTI and HCI, to produce an output proportional to the frequency degradation of the aged ROSC. Although there are two pairs of stressed and reference ROSCs in this sensor, with each pair sensing BTI and HCI aging separately, a simplified schematic to represent the idea is highlighted in Fig. 3.1 with just one pair. This sensor has been demonstrated to provide high resolution and remove common-mode disturbances.

Within a larger circuit, ROSCs can be placed close to the CUT as illustrated in Fig 3.1: since ROSCs are cheap and compact, many copies can be replicated within the chip. Small circuit blocks may share a ROSC, while a very large block could contain several ROSCs.

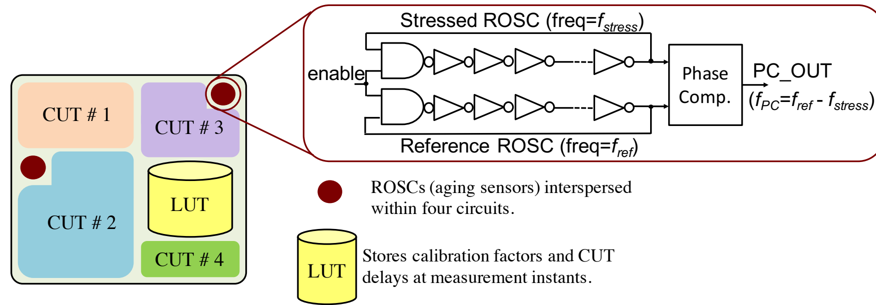


Figure 3.1: ROSC-based aging sensors [3] interspersed within multiple circuits along with LUT of degradation ratios.

However, ROSCs are mere surrogates for the CUT. Therefore, measuring frequency degradation in ROSC is not equivalent to measuring degradation in the CUT, for several reasons. First, since the gate types on a CUT path are not the same as those on the ROSC, the path delay sensitivity to V_{th}-shifts under aging is different from the ROSC

delay sensitivity. Second, the ROSC has a single path that ages along a constant profile through its lifetime; in contrast, the delay of a CUT is the maximum of all path delays. Since a set of near-critical paths may have different aging sensitivities, the critical path may change over the lifetime of the CUT, causing it to age at different rates at different times. To overcome these issues, we first propose a new *Upperbound on* $\mathcal{F}_{Max}$ (UofM) model for the delay of an aged CUT to estimate a safe $\mathcal{F}_{Max}$ that the CUT can operate at. This model accounts for the possibility that critical paths may change over the lifetime of a chip due to nonuniform delay degradation on various circuit paths, by finding an envelope for the CUT delay. Next, we propose a technique to compute calibration factors that translate ROSC degradation to that of the CUT, using the UofM model. These factors, called the *degradation ratios*, ξ_B^{CUT} and ξ_H^{CUT}, can translate the sensor measurement data to CUT delay degradation under BTI and HCI, respectively, and can be stored in an on-chip look-up table (LUT). The UofM model-based scheme thus obtains the degradation ratios from presilicon characterization of a CUT, and uses them to translate the ROSC measurements to CUT delay degradation under aging at the post-silicon stage when the circuit is deployed in the field.
The overall flow of the proposed framework can be summarized as follows:

- We obtain an envelope of the CUT delay under aging by performing aging-aware static timing analysis (STA) assuming worst-case workload on the CUT. This is the UofM delay model of the CUT.

- Using the above envelope, and aging-aware STA on the ROSC sensors, we obtain the degradation ratios, and store them in the on-chip LUT.

- After the chip has been deployed in field operation, the ROSC sensor can be probed from time to time to observe its delay degradation separately due to BTI and HCI aging.

- Multiplying the ROSC delay degradation with the corresponding degradation ratios from the LUT produces the CUT delay degradation, which is an indicator of its aging due to BTI and HCI.

The implementation details of our methodology and the experimental results are provided in the next two sections of this chapter.

3.1 Delay Estimation and Aging Prediction

Our goal is to estimate a safe value of the maximum frequency of operation, $\mathcal{F}_{Max}$, of the CUT under aging, using data from on-chip ROSCs. We obtain this estimate by determining degradation ratios, ξ_B^{CUT} and ξ_H^{CUT}, that multiply the delay degradation of nearby ROSC test structures to estimate the delay shift in the CUT. Our initial analysis, described in this section, performs presilicon analysis to determine the values for these degradation ratios, which are held constant over the lifetime of the circuit and stored in the LUT in Fig. 3.1.

We begin by observing that the rate at which a path ages depends on how it is stressed and on its sensitivity to stress. Due to this, the critical path of a CUT may change over its lifetime owing to the nonuniform delay degradation on its near-critical paths. Fig. 3.2 depicts possible aging trajectories for several near-critical paths of a CUT, C. The delay, $D^C(t)$, of the CUT is the maximum among the delays of the near-critical paths, and this is seen to be a piecewise-smooth curve. In contrast, the ROSC has a single path that ages along a constant profile through its lifetime and has a smooth trajectory, $D^R(t)$, similar to any of the path delays in Fig. 3.2.

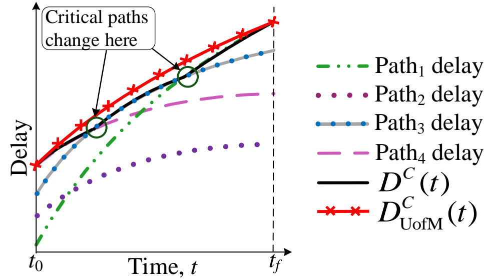


Figure 3.2: CUT delay as maximum of path delays under aging.

The above example indicates a primary difficulty in using a ROSC-based sensor to predict the delay degradation of the CUT, since it is nontrivial to develop a simple one-to-one functional relationship between a smooth trajectory (corresponding to the ROSC delay, $D^R(t)$) and a nondifferentiable function (which characterizes the delay, $D^C(t)$, of the CUT).

To overcome this, we first obtain a pessimistic and continuously differentiable presilicon bound, $D^C_{\text{UofM}}(t)$, of the CUT delay, as illustrated in Fig. 3.2. We refer to this

as the *Upper bound on $\mathcal{F}_{Max}$* (UofM) model. To ensure pessimism, $D^C_{\text{UofM}}(t)$, must lie above $D^C(t)$, $\forall t \in [t_0, t_f]$, so that if $D^C_{\text{UofM}}(t)$ meets the timing requirements throughout the lifetime, then so does $D^C(t)$. Next, we find a relation between the ROSC delay and the UofM delay to obtain the degradation ratios to estimate CUT delay degradation from the ROSC data.

In Sec. 3.1.1, we discuss the presilicon characterization of the CUT and the ROSC to compute the degradation ratios. Next, in Sec. 3.1.2, we outline the methodology of post-silicon aging estimation in the CUT from the ROSC measurements using these ratios, and in Sec. 3.1.3, we examine the validity of using degradation ratios characterized at the presilicon stage, since post-silicon operating conditions can be different due to PVT variations, dynamic voltage scaling, and supply gating.

3.1.1 Presilicon Circuit Characterization

UofM model: We first present a theorem to obtain an expression for a differentiable function that is an upper bound for the maximum of n functions, each of the form similar to Eq. (2.12).

Theorem 3.1. *In the interval, $[t_0, t_f]$, consider a set of monotonically increasing functions, $x_1(t), \cdots, x_n(t)$, such that $x_i(t) = x_i(t_0) + \theta_1^i \Delta f(t) + \theta_2^i \Delta g(t)$, with $\theta_1^i,\ \theta_2^i \geq 0$, where $\Delta f(t) = f(t) - f(t_0)$ and $\Delta g(t) = g(t) - g(t_0)$. Then for $f(t) = t^{n_1}$ or $a + b \log t$, and $g(t) = t^{n_2}$ with $a, b > 0$, and $1 > n_2 > n_1 > 0$, an upper bound on the maximum of these functions is given by another function, $y(t)$, of similar form:*

$$y(t) = x_M(t_0) + \theta_1^M \Delta f(t) + \theta_2^M \Delta g(t) \tag{3.1}$$

where $x_M(t)$ is the maximum envelope of the $x_i(t)$ functions, such that $x_M(t_0) = \max_{i \in \{1,\cdots,n\}}(x_i(t_0))$.

The coefficients, θ_1^M and θ_2^M, are obtained from the θ_1^i values, and by evaluating $x_M(t)$ at time instants, $t = t_0$ and t_f, and are defined as:

$$\theta_1^M = \max_i(\theta_1^i) \tag{3.2}$$

$$\theta_2^M = \frac{\Delta x_M(t_f) - \theta_1^M \Delta f(t_f)}{\Delta g(t_f)} \tag{3.3}$$

where $\Delta x_M(t) = x_M(t) - x_M(t_0)$.

A brief outline of the proof is that the UofM model is constructed as a curve of the form of Eq. (2.12) that matches the piecewise-smooth maximum function, $x_M(t)$, at $t = t_0$ and $t = t_f$, and lies above it at all other points, $t \in (t_0, t_f)$. The detailed proof is deferred to Appendix A for better readability.

To map the results of this theorem to our problem, we represent the delay of each near-critical path, p_i, of the CUT in the form of $x_i(t)$, and use Theorem 3.1 to determine the UofM delay bound. Note that the restrictions on a, b, n_1, and n_2 are easily satisfied by typical BTI/HCI models. The evaluation of the upper bound in Theorem 3.1 requires the values of $x_M(t_0)$, $x_M(t_f)$, and θ_1^i to be determined[1] . For the problem at hand, evaluating $x_M(t)$ at time instants, $t = t_0$ and $t = t_f$, in the theorem is equivalent to obtaining the presilicon circuit delay, $D_{pre}^C(t)$, at these two instants by performing STA on C. Obtaining θ_1^i is equivalent to computing the K_B value of the delay trajectory of p_i. To obtain K_B (K_H) value of p_i, we simply evaluate the delay of p_i at $t = t_0$ and t_f under BTI (HCI) aging alone for specific workload conditions, and compute $K_B^{p_i}$ and $K_H^{p_i}$ as:

$$K_B^{p_i} = \frac{D_{pre,B}^{p_i}(t_f) - D_{pre}^{p_i}(t_0)}{\Delta f(t_f)} \tag{3.4}$$

$$K_H^{p_i} = \frac{D_{pre,H}^{p_i}(t_f) - D_{pre}^{p_i}(t_0)}{\Delta g(t_f)} \tag{3.5}$$

where $D_{pre,B}^{p_i}(t)$ $\left(D_{pre,H}^{p_i}(t)\right)$ is the evaluated delay of p_i at t due to BTI (HCI) aging alone, and $D_{pre}^{p_i}(t_0)$ is the presilicon delay at time t_0.

To summarize, we obtain the smooth upper bound, $D_{\text{UofM}}^C(t)$, on $D^C(t)$ by using any timing analysis tool (home-grown or commercial) to perform two STA evaluations, at times, t_0 and t_f, on the CUT using the aging models for BTI and HCI from Sec. 2. Since we must perform these presilicon STA runs to account for all parts in the field excited with any SP or AF value, the specific workload conditions correspond to choosing these values pessimistically. For BTI analysis, we assume the worst-case SP of 1 for each gate input; similarly, for HCI analysis, we assume an AF of 1. For each near-critical path, p_i, a pair of timing evaluations under BTI aging alone can be used to compute the $K_B^{p_i}$ value for the path using Eq. (3.4). Finally, we use Theorem 3.1 to characterize

[1] Superficially, it may appear that Eqs. (3.2) and (3.3) are independent of θ_2^i, but the influence of this parameter is hidden from view in $\Delta x_M(t_f)$.

the constants, K_B^C and K_H^C, in $D^C_{\text{UofM}}(t)$, similar to θ_1^M and θ_2^M in $y(t)$ in the theorem, to obtain:

$$\Delta D^C_{\text{UofM}}(t) = D^C_{\text{UofM}}(t) - D^C_{\text{UofM}}(t_0) = K_B^C\ \Delta f(t) + K_H^C\ \Delta g(t) \tag{3.6}$$

where $K_B^C \Delta f(t)$ and $K_H^C \Delta g(t)$ are the aging contributions due to BTI and HCI, respectively, to the total delay degradation from $t = t_0$.

Degradation ratios from ROSC and CUT delay analysis: A ROSC is a chain of an odd number, $2l+1$, of inverters connected in a closed loop. Assuming, for simplicity, that each inverter has a rise delay of $d_r(t)$ and a fall delay of $d_f(t)$, the period of the ROSC is well known to be $(2l+1)(d_r(t)+d_f(t))$. We refer to the period of a ROSC, R, as its delay, $D^R(t)$.

Since the ROSC has 50% signal probability and toggles on every clock transition, the SP values at each of its gate inputs is 0.5 and its AF is 1. At the presilicon stage, the aging trends for the ROSC can be characterized to separately evaluate K_B^R and K_H^R. This requires an aging-aware timing analysis of the single critical path of the ROSC, first assuming only BTI, and then only HCI aging, each at $t = t_0$ and $t = t_f$, similar to Eqs. (3.4) and (3.5). The presilicon estimate of the delay degradation, $\Delta D^R_{pre}(t)$, of the ROSC is obtained in the form of Eq. (2.12), as:

$$\Delta D^R_{pre}(t) = D^R_{pre}(t) - D^R_{pre}(t_0) = K_B^R \Delta f(t) + K_H^R \Delta g(t) \tag{3.7}$$

Based on the K_B and K_H values computed to characterize Eqs. (3.6) and (3.7), we compute the degradation ratios, ξ_B^C and ξ_H^C, of the CUT, C, corresponding to BTI and HCI aging, respectively, as:

$$\xi_B^C = \frac{K_B^C}{K_B^R} \qquad \xi_H^C = \frac{K_H^C}{K_H^R} \tag{3.8}$$

Note that the degradation ratios above depend purely on the ratios of the K_B and K_H values and are independent of time.

3.1.2 Post-silicon Circuit Aging Estimation from ROSC Sensors

The values of the degradation ratios for each CUT, with respect to its associated ROSC, are stored in the on-chip LUT depicted in Fig. 3.1. To use this LUT in the post-silicon

context at time t, we will separately measure the values of delay shifts, $\Delta D_B^R(t)$ and $\Delta D_H^R(t)$, due to BTI and HCI, respectively, of the silicon odometer ROSC [3]. We infer $K_{post,B}^R = \Delta D_B^R(t)/\Delta f(t)$ and $K_{post,H}^R = \Delta D_H^R(t)/\Delta g(t)$ from the silicon odometer ROSC; recall that this sensor can separately measure its own BTI- and HCI-induced degradation. Based on this measurement, we estimate the post-silicon CUT delay degradation, $\Delta D_{post}^C(t)$, in a manner similar to Eq. (3.6) as:

$$\begin{aligned}\Delta D_{post}^C(t) &= K_{post,B}^C\ \Delta f(t) + K_{post,H}^C\ \Delta g(t) \\ &= K_{post,B}^R\ \xi_B^C \Delta f(t) + K_{post,H}^R\ \xi_H^C \Delta g(t)\end{aligned} \tag{3.9}$$

where $K_{post,B}^C$ and $K_{post,H}^C$ correspond to the unknown K_B and K_H values, respectively, of the CUT delay trajectory at the post-silicon stage, which we want to infer from the ROSC measurements. The above equation seems to imply that ξ_B^C and ξ_H^C, defined in Eq. (3.8), can also be formulated as:

$$\xi_B^C = \frac{K_{post,B}^C}{K_{post,B}^R} \qquad \xi_H^C = \frac{K_{post,H}^C}{K_{post,H}^R}$$

In other words, the degradation ratios from the presilicon stage are the same at the post-silicon stage, in spite of both being characterized by different operating conditions. This assumption is correct for all practical purposes, as will be explained in the next section.

3.1.3 Effects of Operating Conditions on Degradation Ratios

During circuit operation in the field, when the above ROSC measurements are taken and the CUT delay shift is estimated, both the CUT and the ROSC age under conditions that are different from those during presilicon estimation. We now critically examine the validity of using presilicon degradation ratios under these conditions, considering the effect of each factor that differs between the presilicon and post-silicon phases: specifically, the SP and AF for the circuit; systematic process variations; temperature, T, and supply voltage, V_{dd}, including supply gating and dynamic voltage scaling.

Circuit SP and AF: The precharacterized relation in Eq. (3.8) uses a worst-case SP and AF scenario for aging, and therefore provides a pessimistic estimate of the CUT

delay.

Process variations: The proximity of the ROSC and the CUT ensures that both have similar systematic variations in the process parameters (length, width, oxide thickness, and other critical dimensions). The dominant systematic variations [51] [52] are thus, very similar for both the CUT and the ROSC. As a result, the process dependence of K_B and K_H values in Eq. (2.12) are also similar for both the CUT and the ROSC, and hence, the degradation ratios, ξ_B^C and ξ_H^C, which are the ratios of K_B and K_H of the CUT to that of the ROSC, are practically independent of the process variations. The effect of the random variations is also minimized for a ROSC and CUT with multiple stages [15] [53], considered in this work.

Voltage and temperature variations: To investigate the impact of V_{dd} and T on these ratios, it is necessary to analyze K_B and K_H, as given in Sec. 2.2. We can rewrite the (V_{dd}, T)-dependent parts in K_B and K_H from [54] and Eq. (2.3) in Sec. 2.1.2, respectively, as:

$$K_B = \sum_{x\in\{n,p\}} S_x \psi_{B_x} = \Gamma_B P_B \exp\left(\frac{2E_{ox}}{E_0} - \frac{E_a}{k_B T}\right) \mathcal{S}(V_{dd}, T) \tag{3.10}$$

$$K_H = \sum_{x\in\{n\}} S_x \psi_H = \Gamma_H P_H \exp\left(\frac{E_{ox}}{E_0} - \frac{\phi_{it}}{q\lambda E_m}\right) \mathcal{S}(V_{dd}, T) \tag{3.11}$$

where P_B and P_H are the combined process-dependent terms, Γ_B and Γ_H indicate the effect of average SPAF conditions on BTI and HCI aging, and $\mathcal{S}(V_{dd}, T)$ is a general function representing the dependence of the delay sensitivities to the operating conditions, which can be assumed to be similar for both PMOS and NMOS. The exact form of $\mathcal{S}$ is not necessary for our analysis.

For BTI, the dependence on V_{dd} is embedded within the terms, $\mathcal{S}(V_{dd}, T)$ and $E_{ox} = \frac{V_{dd}-V_{th}}{T_{ox}}$, while the relationship with T is explicitly visible in the term, $\exp\left(-\frac{E_a}{k_B T}\right)$, in Eq. (3.10). For HCI degradation, the dependence on V_{dd} is visible in $\mathcal{S}(V_{dd}, T)$ as well as E_m in Eq. (2.4), while the trend with T is implicitly hidden in the V_{dsat} term in Eq. (2.5) that is referenced in the equation for E_m.

The gate delay shifts can be obtained by combining Eqs. (2.12), (3.10), and (3.11). For any path, X, which could be a near-critical path, p_i, of the CUT or the single path,

r, of the ROSC, R, we can represent its actual delay degradation from t_0 due to BTI and HCI, by $\Delta D_B^X(t)$ and $\Delta D_H^X(t)$, respectively, as:

$$\Delta D_B^X(t) = K_B^X \Delta f(t) = \mathcal{K}_B^X F_B(V_{dd}, T)\Delta f(t) \tag{3.12}$$

$$\Delta D_H^X(t) = K_H^X \Delta g(t) = \mathcal{K}_H^X F_H(V_{dd}, T)\Delta g(t) \tag{3.13}$$

where K_B^X and K_H^X represent the K_B and K_H values of the path, X, respectively, and $\mathcal{K}_B^X$ and $\mathcal{K}_H^X$ are the (V_{dd}, T)-independent effects of adding the gate delays along X under BTI and HCI aging, respectively. The functions, $F_B(.)$ and $F_H(.)$, represent the (V_{dd}, T)-dependent terms for BTI and HCI, respectively, as:

$$F_B(V_{dd}, T) = \exp\left(\frac{2E_{ox}}{E_0} - \frac{E_a}{k_B T}\right)\mathcal{S}(V_{dd}, T) \tag{3.14}$$

$$F_H(V_{dd}, T) = \exp\left(\frac{E_{ox}}{E_0} - \frac{\phi_{it}}{q\lambda E_m}\right)\mathcal{S}(V_{dd}, T) \tag{3.15}$$

Theorem 3.2. *For a given CUT, C, let p_f and p_0 be the paths that are critical at times, t_f and t_0, respectively, and r be the single path in the ROSC. Among all near-critical paths of C, let p_m be the path with the maximum value of K_B. Then,*

1. *The degradation ratio, ξ_B^C, of the CUT, C, is independent of the supply voltage, V_{dd}, and temperature, T.*

2. *The degradation ratio, ξ_H^C, has the following dependence:*

$$\xi_H^C = \frac{\Delta D^{p_f,p_0}(t_0) + \Delta\mathcal{K}_B^{p_f,p_m} F_B(V_{dd}, T)\Delta f(t_f)}{\mathcal{K}_H^r F_H(V_{dd}, T)\Delta g(t_f)} + \frac{\mathcal{K}_H^{p_f}}{\mathcal{K}_H^r} \tag{3.16}$$

where $\Delta D^{p_f,p_0}(t_0) = D^{p_f}(t_0) - D^{p_0}(t_0)$ is the difference in the delays of paths, p_f and p_0, at $t = t_0$, $\Delta\mathcal{K}_B^{p_f,p_m} = \mathcal{K}_B^{p_f} - \mathcal{K}_B^{p_m}$.

The factors, $\mathcal{K}_B^X$ and $\mathcal{K}_H^X$, for $X \in \{p_f, p_m, r\}$, are as defined in Eqs. (3.12) *and* (3.13), *and the functions, $F_B(V_{dd}, T)$ and $F_H(V_{dd}, T)$, are defined in Eqs.* (3.14) *and* (3.15), *respectively.*

The formal proof of the theorem is deferred to Appendix A for better readability.

Based on Theorem 3.2, ξ_B^C is independent of V_{dd} and T, while ξ_H^C is not. Hence, assuming (V_{dd}, T)-independence of ξ_H^C will incur some error in the aging estimate due

to HCI, which can be bounded by a maximum value with the knowledge of the operating conditions according to Eq. (3.16). However, in reality, the near-critical paths have similar delay profiles (i.e., $D^{p_0}(t_0) \approx D^{p_f}(t_0)$), as well as similar aging trends (i.e., $\mathcal{K}_B^{p_f} \approx \mathcal{K}_B^{p_m}$), due to which $\xi_H^C \approx \frac{\mathcal{K}_H^{p_f}}{\mathcal{K}_H^r}$ from Eq. (3.16) in Theorem 3.2. Hence, the error is negligible while considering (V_{dd}, T)-independence of ξ_H^C as well. In the rest of the chapter, we thus assume both ξ_B^C and ξ_H^C to be independent of V_{dd} and T.

Dynamic voltage scaling and V_{dd} gating: It is important to note that the above analysis is valid for any variation in V_{dd}, as long as the ROSC and the CUT are subjected to the same variation. Therefore, it is valid not only for fluctuations in the supply noise, which are identical due to spatial locality of a CUT and its nearby ROSC, but also for V_{dd} changes due to dynamic voltage scaling or V_{dd} gating.

Hence, the degradation ratios are practically independent of PVT variations, deliberate supply voltage changes, and time, and require characterization only once for the entire lifetime of a circuit to obtain its pessimistic delay estimate under the assumption of a worst-case SPAF workload, at the post-silicon stage from ROSC measurements.

3.2 Experimental Setup and Results

The ideas presented in this chapter were exercised on a set of representative circuits from the ISCAS'89 [55], ITC'99 [56], and the IWLS'05 [57] benchmark suites. Since the various model parameters and the V_{th} degradation equations are well documented in the public domain for 45nm, the circuits were synthesized using the NanGate 45nm Open Cell Library [58]. Each gate within the library was characterized for nominal delay, output slew and delay sensitivities to V_{th} variation of NMOS and PMOS devices (for both rise and fall transitions) by transistor level HSPICE simulations, and the circuits were synthesized using Synopsys Design Compiler [59]. The simulations were carried out at 125°C and 1.2V. Although we use the RD model for BTI aging in our experiments to show the applicability of the proposed methodology, it is also valid when BTI aging follows a logarithmic function of time, as described by the CT model. The lifetime, t_f, of each CUT has been assumed to be 10 years when both BTI and HCI are significant [42], and we consider $t_0 = 0$.

The aging-aware STA engine was developed in C++, and the experiments were performed on a 64-bit Ubuntu server with a 3GHz Intel® Core™2 Duo CPU E8400 processor.

3.2.1 Aging Estimation from ROSCs using the UofM Bound

We obtain the K_B and K_H values for benchmark circuits, whose names and gate counts, $|G|$, are listed in the first two columns of Table 3.1, by performing aging-aware STA on them under the worst-case workload assumption and using Theorem 3.1. We also obtain the K_B and K_H values corresponding to 33-stage ROSC sensors [3] using techniques described in Sec. 3.1.1. The degradation ratios, ξ_B^C and ξ_H^C, are listed in the third and fourth columns, respectively, of Table 3.1. The runtime in seconds, τ, required to generate degradation ratios for each circuit is listed in the fifth column. The estimated post-silicon delay, $D_{post}^C(t)$, of the aged CUT is obtained by multiplying the degradation ratios with the delay degradation of the ROSC, and adding the result with the nominal delay, $D_{pre}^C(t_0)$, of the CUT.

Table 3.1: Degradation ratios from presilicon analysis.

CUT, C	$\|G\|$	ξ_B^C	ξ_H^C	τ (s)	ΔE_{rms}
mem_ctrl	6086	1.98	2.35	30	0.001%
wb_dma	2313	1.19	1.61	15	0.096%
ac97_ctrl	8422	1.01	1.49	41	0.003%
i2c	550	1.07	1.28	7	0.028%
aes_core	23104	1.19	1.79	82	0.505%
b15	5581	2.90	3.51	28	0.091%
b17	16531	2.84	3.40	75	0.001%
b20	21625	5.92	9.73	85	0.706%
b21	21661	6.25	8.28	86	0.501%
b22	32513	6.49	8.57	125	0.033%
s5378	692	0.72	0.99	9	0.492%
s13207	594	0.83	0.86	8	0.001%
s15850	340	0.91	0.85	6	0.001%
s38417	4615	1.12	1.86	28	0.558%
s38584	4633	1.21	1.25	26	0.001%

The accuracy of our scheme is evaluated by the root mean square error between

the CUT delay and the corresponding estimated post-silicon delay over n time instants, $t_j \in [t_0, t_f]$, and is represented by ΔE_{rms}, as:

$$\Delta E_{rms} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(\frac{D^C_{post}(t_j) - D^C(t_j)}{D^C(t_j)}\right)^2}, \quad t_j \in [t_0, t_f] \tag{3.17}$$

where $D^C(t)$ is the true delay of the CUT under worst-case workload, and the error is sampled every half-year interval. The last column in Table 3.1 lists ΔE_{rms} for each circuit, expressed in percentage.

Clearly, the estimated delays match very well with the actual delays, since the ΔE_{rms} values are negligible. Additionally, the modest runtimes for the large benchmark circuits indicate that our method is fast, and hence, scalable to real circuits.

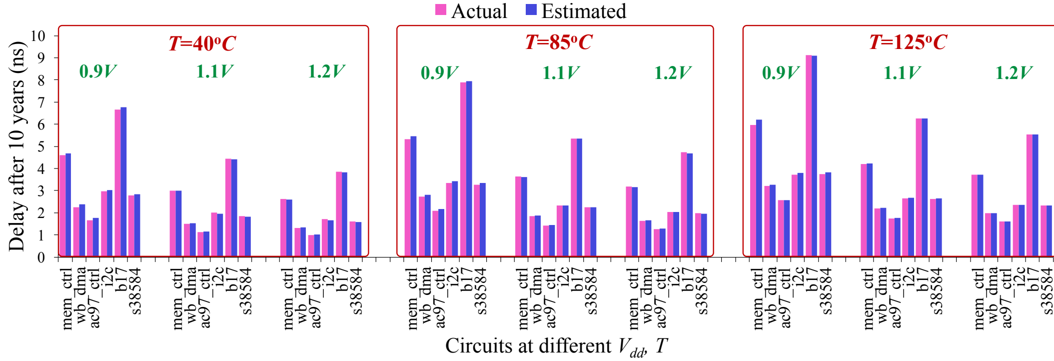


Figure 3.3: Estimated delays across different (V_{dd}, T) from degradation ratios at a fixed (V_{dd}, T), indicating V_{dd} and T independence the UofM scheme.

3.2.2 V_{dd} and T Independence of the UofM Bound

Here we study the effect of the (V_{dd}, T)-independence assumptions of the degradation ratios, on the aging estimate as described in Sec. 3.1.3. For this, we use the ξ^C_B and ξ^C_H of each circuit, C, from Table 3.1 which were computed at a particular (V_{dd}, T) condition, (1.2V, 125°C), and multiply them with the delay degradation of the ROSC obtained at different (V_{dd}, T) conditions. As before, we add the result to the nominal delay of the CUT to estimate its post-silicon delay, $D^C_{post}(t)$, at these new operating conditions. For a set of representative benchmark circuits, we plot the corresponding estimated

delays, $D^C_{post}(t)$, at the end of lifetime, $t_f = 10$ years, along with the actual delay values under the worst-case workload, obtained at several (V_{dd}, T) values in Fig. 3.3. The estimated delays show an excellent match with the true values across all nine combinations. Therefore, the degradation ratios are practically independent of V_{dd} and T, and it suffices to compute them at a single (V_{dd}, T) for each circuit.

To summarize, our method based on the UofM bound provides a workable indication of circuit aging with practically no extra effort on part of the designer and a small power/area overhead. A one-time presilicon circuit characterization was used to determine the degradation ratios, and subsequently ROSC measurements were translated to the circuit delay using these ratios.

3.3 Conclusion

In this chapter, we have presented a technique to estimate delay degradation of a circuit using nearby ROSC-based aging sensors. We quantitatively determine how the data from the ROSC can be used to find the change in the circuit delay. Experimental results show that we can use the UofM metric to distill the relation between the CUT delay trend and the ROSC delay trend into a single degradation ratio, which can accurately predict the CUT delay degradation based on inexpensive measurements on the ROSC.

Chapter 4

Aging Estimation through ReSCALE

It is crucial to estimate the extent of aging in a circuit so that remedial techniques can be applied to ensure its reliable operation over its specified lifetime. These schemes may be deployed at the presilicon as well as at the post-silicon stage of design. Of necessity, presilicon design must be predicated on the worst-case workload for the circuit so that it is guaranteed to work under all operating conditions. This involves the application of pessimistic guardbands whose power overheads may be excessive and unnecessary for a large fraction of parts in the field. Hence, in this chapter we propose a technique called ReSCALE (Recalibration of Sensor Circuits for Aging and Lifetime Estimation) in this chapter which amalgamates the sensor measurements based on the UofM model from the previous chapter, with very infrequent measurements performed directly on the CUT, and uses the results of these measurements to update the LUT. These updated LUT values are then used in conjunction with inexpensive sensor measurements to infer the delay of the CUT.

The circuit modifications associated with ReSCALE are depicted in Fig. 4.1, which shows multiple ROSC sensors interspersed within four circuits. Our scheme uses the same ROSC-based silicon odometer [3] from the previous chapter, which uses the notion of beat frequencies to measure delay variations in the ROSC to a very high degree of precision.

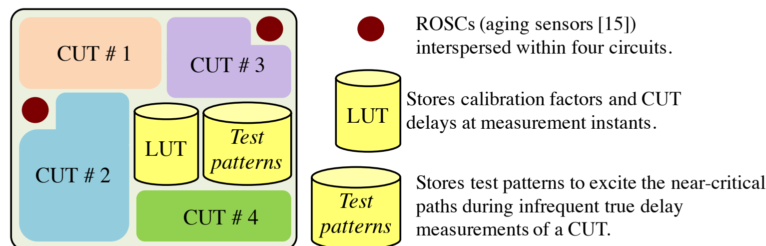


Figure 4.1: ROSCs interspersed within multiple circuits along with LUT of degradation ratio and test pattern storage block for direct CUT measurement.

This scheme permits the change in the period due to BTI and HCI to be measured easily and on-the-fly. The number and location of the ROSCs within a chip is a user input, and the granularity at which they are deployed reflects a trade-off between area overhead and accuracy. The degradation ratios of the CUT, and their true delays at certain instants during the CUT lifetime, are stored in the on-chip LUT. The true delay measurement circuitry is symbolically depicted by the *Test patterns* block in Fig. 4.1, which stores vectors/patterns to perform infrequent delay tests on the CUT. Under the applied test patterns, one of several existing schemes can be used here to measure the runtime delay of a circuit such as the Path-RO [47], delay shift circuits [48] [49], or by the techniques described in [13]. The degradation ratios can be recalibrated offline on the processor itself (in software) when the circuit is tested for its true delay, assuming that the processor has an arithmetic and logic unit.

The ReSCALE methodology is built upon the UofM model-based scheme, where the ROSC measurements are still used to infer CUT degradation, however, the degradation ratios are now updated in the LUT (although infrequently) after obtaining the CUT delay at prespecified time instants, called the *measurement instants*. The overall flow can be summarized as follows:

- The degradation ratios stored in the LUT from presilicon analysis and post-silicon ROSC delay degradation data are used to obtain CUT delay till the first measurement instant, using our first scheme.

- At each measurement instant, the true delay of the CUT is measured using a separate dedicated circuit, and stored in the LUT, along with updating the degradation ratios.

- Multiplying the ROSC delay degradation with the most recent degradation ratios from the LUT produces the CUT delay degradation, and adding that to the true delay of the CUT at the previous measurement instant produces a more accurate estimate of aging in the CUT.

Our approach blends the simplicity and low measurement overhead of surrogate sensors with the accuracy of direct measurement. These measurements are performed only occasionally, thus controlling the high overhead of run-time measurements. Furthermore, we develop a new theory that maps the results of direct measurement to the aging of the surrogate sensor, and propose a framework to recalibrate the surrogate sensors based on measurement data.

The implementation details of the methodology and the experimental results are provided in the next two sections of this chapter.

4.1 Sensor Recalibration and Aging Estimation

While the ROSC sensor can track PVT variations, voltage scaling, and V_{dd} gating in the CUT as described in Sec. 3.1, it is inherently incapable of tracking changes in the SPAF of the CUT. Hence, the interpretation of the ROSC data must assume SPAF settings that correspond to worst-case aging in the CUT. This may result in pessimistic estimates of the circuit performance, and may underestimate circuit delays by over 10%, as illustrated experimentally in Sec. 4.2.

The scheme presented in this section supplements data from the ROSC through infrequent direct measurements of the CUT, measuring its true delay in the field. The information gathered through these measurements is used to recalibrate the relationship between ROSC aging and CUT aging. In short, we combine infrequent delay measurements on the CUT with cheap and more frequent ROSC measurements, to obtain more accurate estimates of the CUT delay. This allows ROSC measurements to be personalized to individual chips in the field, accounting for the way they age, based on the specific stressing environment that the part is subjected to.

Our approach proceeds by recalibrating the degradation ratios, K_B and K_H, based on data from CUT measurements, so that ROSC measurements can be mapped more accurately on to CUT delay estimates. The modified scheme uses the LUT depicted in

Fig. 4.1 to store the degradation ratios and the *Test Patterns* block to store the test patterns required to determine the delay degradation of the circuit.

4.1.1 Delay Bounds based on Post-silicon Circuit Measurements

We illustrate the ReSCALE scheme through Fig. 4.2. The pessimistic delay degradation trajectory over all possible workloads is given by the UofM bound, $D^C_{\text{UofM}}(t)$. However, for a specific chip running a particular workload, the actual delay degradation follows the curve shown by $D^C_a(t)$; by definition, this must lie below the UofM bound. To correct this difference, the degradation ratios are modified at a set of measurement instants, corresponding to $t = t_{m_0}, \cdots, t_{m_3}$, by performing direct measurements on the circuit and appropriately recalibrating K^C_B and K^C_H to achieve a better prediction, shown by $D^C_{re}(t)$.

Up to the first measurement instant, t_{m_1}, $D^C_{re}(t)$ exactly follows the UofM bound. At this point, the bound is brought down to the measured delay, and the delay trajectory beyond this point must be predicted. Any such projection of future activity must be made without foreknowledge of the workload and therefore, the $D^C_{re}(t)$ curve must necessarily assume worst-case aging beyond t_{m_1}. The actual aging curve will lie below this bound, and at the next measurement instant, t_{m_2}, a recalibration is made to match the measured value, and so on. As a result, the $D^C_{re}(t)$ curve matches the actual aging curve, $D^C_a(t)$, more closely than the UoM curve, $D^C_{\text{UofM}}(t)$.

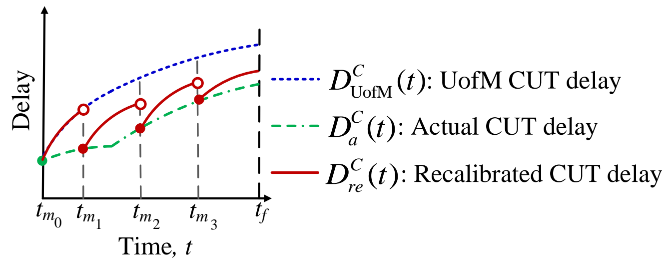


Figure 4.2: Upper-bounding the CUT delay with intermediate CUT measurements.

The delay measurements may be performed on the CUT using one of several existing schemes, such as the Path-RO [47], delay shift circuits [48] [49], or by the techniques described in [13]. These techniques typically use input vectors stored in an on-chip memory, with a test controller, which fetches a pair of these vectors to perform the

delay tests. The number of such patterns to excite near-critical paths of a CUT to measure the worst-case delay, is sufficiently small, and the hardware overhead of the associated test controller is less than 0.01% as reported in [49]. The *Test patterns* block in Fig. 4.1, thus abstracts the entire circuitry that is appended to the circuit block for its true delay measurement.

The following result provides two upper bounds to obtain $D_{re}^{C}(t)$ for $t \in [t_{m_j}, t_{m_{j+1}}]$.

Theorem 4.1. *Let $\{t_{m_0}, \cdots, t_{m_{N-1}}\}$ be the N measurement instants at which the degradation ratios are recalibrated, and $t_{m_N} = t_f$, and let*

$$\Delta f_j(t) = f(t) - f(t_{m_j})$$
$$\Delta g_j(t) = g(t) - g(t_{m_j})$$

After each measurement instant, the recalibrated upper bound on the delay, or, $D_{re}^{C}(t)$, is obtained as follows:
For $0 \leq t < t_{m_1}$,

$$D_{re}^{C}(t) = D_{\mathrm{UofM}}^{C}(t) \tag{4.1}$$

For $t_{m_j} \leq t \leq t_{m_{j+1}}, j > 1$, two upper bounds on $D_a^C(t)$ are:

$$\textbf{(I)}\ D_{re}^{C,I}(t) = D_a^C(t_{m_j}) + K_B^I \Delta f_j(t) + K_H^{II} \Delta g_j(t), \tag{4.2}$$

If $p_x = \{p_i \in S_{NC} | D_{pre}^{p_i}(t_{m_{j+1}}) - D_{pre}^{p_i}(t_{m_j})$ is maximized$\}$, $K_B^I = K_B^{p_x}$, $K_H^I = K_H^{p_x}$, where S_{NC} is the set of near-critical paths of C, and $D_{pre}^{p_i}(t) = D_{pre}^{p_i}(t_0) + K_B^{p_i} \Delta f(t) + K_H^{p_i} \Delta g(t)$ is the worst-case presilicon delay estimate of path $p_i \in S_{NC}$.

$$\begin{aligned} \textbf{(II)}\ D_{re}^{C,II}(t) &= D_a^C(t_{m_j}) + K_B^{II} \Delta f_j(t) + K_H^{II} \Delta g_j(t) \\ K_B^{II} &= \max_{p_i \in S_{NC}} \left(K_B^{p_i}\right), \\ K_H^{II} &= \frac{\left[D_{\mathrm{UofM}}^{C}(t_{m_{j+1}}) - D_a^C(t_{m_j})\right] - K_B^{II} \Delta f_j(t_{m_{j+1}})}{\Delta g_j(t_{m_{j+1}})}, \end{aligned} \tag{4.3}$$

where $K_B^{p_i}$ is the K_B value of the path p_i.

To use Theorem 4.1 for $j > 1$, we choose the bound that is tighter at time $t_{m_{j+1}}$. We select $K_{j,B}^C$ and $K_{j,H}^C$ as follows:

- If $D_{re}^{C,I}(t_{m_{j+1}}) \geq D_{re}^{C,II}(t_{m_{j+1}})$, then

$$K_{j,B}^{C} = K_{B}^{I} \qquad K_{j,H}^{C} = K_{H}^{I} \tag{4.4}$$

- Otherwise

$$K_{j,B}^{C} = K_{B}^{II} \qquad K_{j,H}^{C} = K_{H}^{II} \tag{4.5}$$

The proof of the upper bounds in Theorem 4.1 is presented in Appendix A. Hence, for $t \in [t_{m_j}, t_{m_{j+1}}]$, $j \geq 1$,

$$D_{re}^{C}(t) = D_{a}^{C}(t_{m_j}) + K_{j,B}^{C}\Delta f_j(t) + K_{j,H}^{C}\Delta g_j(t) \tag{4.6}$$

Intuitively, Case I provides one candidate for an upper bound, but if that bound exceeds the UofM prediction, Case II provides a tighter estimate.

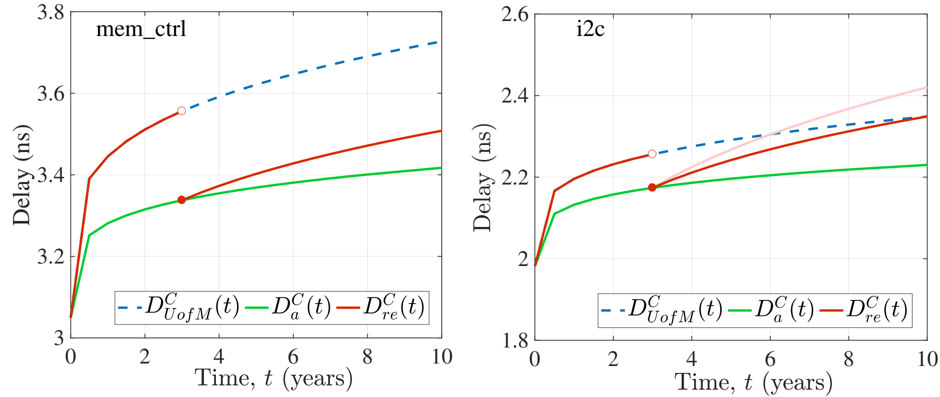


Figure 4.3: Example of the recalibration method for aging estimation in two circuits: mem_ctrl follows Case I (left), while i2c follows Case II (right).

An example of delay estimation using the current scheme for a single measurement instant (excluding t_{m_0}) is shown in Fig. 4.3. The two circuits, mem_ctrl and i2c, are from the IWLS'05 benchmark suite [57], and are stressed at 125°C and 1.2V. The real delay curve is obtained by using a set of simulated runtime SPAF values and the aging model described in Sec. 2. For mem_ctrl, Case I is applicable, but for i2c the bound from Case I exceeds the UofM bound. Therefore, Case II is applied to obtain a better bound, as shown in Fig. 4.3.

4.1.2 Post-recalibration Aging Estimation in a Circuit

In this section, we show how the aging estimation scheme in Sec. 3.1.2 is modified to incorporate recalibration. At each measurement instant, t_{m_j}, we measure the CUT and the ROSC delays, denoted by $D_a^C(t_{m_j})$ and $D_a^R(t_{m_j})$, respectively. The K_B and K_H values of the ROSC are recalibrated to $K_{j,B}^R$ and $K_{j,H}^R$, respectively, based on the methodology described in Case I of the previous section (Case II does not arise for the ROSC since it has only one path). Similarly, by measuring the circuit, we apply Theorem 4.1 to determine its $K_{j,B}^C$ and $K_{j,H}^C$ values in the interval $[t_{m_j}, t_{m_{j+1}})$. We then modify Eq. (3.8) after each t_{m_j} as:

$$\xi_{j,B}^C = \frac{K_{j,B}^C}{K_{j,B}^R} \qquad \xi_{j,H}^C = \frac{K_{j,H}^C}{K_{j,H}^R} \tag{4.7}$$

where $\xi_{j,B}^C$ and $\xi_{j,H}^C$ are the recalibrated degradation ratios which need to be updated in the LUT after each measurement instant, along with the actual CUT delay, $D_a^C(t_{m_j})$ at t_{m_j}.

The above procedure updates the parameters used for the upper bound in $[t_{m_j}, t_{m_{j+1}})$. Recall that this procedure is to be applied infrequently through the lifetime of the circuit. A more frequent operation is to measure the ROSC only, and to use the parameters of the bound to estimate the CUT delay. We will now explain how this is performed.

Based on the ROSC measurement and the stored ROSC delay at time, t_{m_j}, we first obtain the delay degradation of the ROSC in the interval between t and t_{m_j} as $\Delta D_{B,j}^R(t)$ and $\Delta D_{H,j}^R(t)$ due to BTI and HCI aging, respectively, using the silicon odometer [3]. We then infer $K_{post,B}^R = \frac{\Delta D_{B,j}^R(t)}{\Delta f_j(t)}$ and $K_{post,H}^R = \frac{\Delta D_{H,j}^R(t)}{\Delta g_j(t)}$ from these measurements. We then obtain the CUT delay degradation as:

$$\begin{aligned}\Delta D_{post,j}^C(t) &= K_{post,B}^C \Delta f_j(t) + K_{post,H}^C \Delta g_j(t) \\ &= \xi_{j,B}^C K_{post,B}^R \Delta f_j(t) + \xi_{j,H}^C K_{post,H}^R \Delta g_j(t)\end{aligned} \tag{4.8}$$

where $K_{post,B}^C$ and $K_{post,H}^C$ correspond to, respectively, the K_B and K_H values of the CUT delay trajectory after each measurement instant, which we infer from the frequent ROSC measurements.

Note that these operations are similar to those in Sec. 3.1.2, except that the degradation is relative to the delay at t_{m_j}, not t_0. Hence, the estimated CUT delay with

recalibration, $D^C_{post}(t)$, can be obtained by adding $D^C_a(t_{m_j})$, the measured delay at time, t_{m_j}, that is stored in the LUT, to $\Delta D^C_{post,j}(t)$.

4.2 Experimental Setup and Results

The ideas presented in this chapter were exercised on a set of representative circuits from the ISCAS'89 [55], ITC'99 [56], and the IWLS'05 [57] benchmark suites. Similar to the setup in the previous chapter, the benchmark circuits were synthesized using the NanGate 45nm Open Cell Library [58]. Each gate within the library was characterized for nominal delay, output slew and delay sensitivities to V_{th} variation of NMOS and PMOS devices (for both rise and fall transitions) by transistor level HSPICE simulations, and the circuits were synthesized using Synopsys Design Compiler [59]. The simulations were carried out at 125°C and 1.2V. Although we use the RD model for BTI aging in our experiments to show the applicability of the proposed methodology, it is also valid when BTI aging follows a logarithmic function of time, as described by the CT model. The lifetime, t_f, of each CUT has been assumed to be 10 years when both BTI and HCI are significant [42], and we consider $t_0 = 0$.

The difference between various manufactured circuits lies primarily in the variations that they experience due to process and environment effects, and due to the different SPAF values associated with their usage. As explained in Sec. 3.1.3, the impact of process and environment variations is minimal, and hence, the primary difference lies in the SPAF values. Some circuits may exercise a CUT frequently and correspond to active SPAFs, while others may be used less often and may correspond to inactive SPAFs. Therefore, we can use the SPAF value as a way to model how a circuit is used in the field. In our experiments, the SP and AF of each gate input of the CUT were assumed to be unity to emulate the worst-case workload. In a real workload, the input SPs are typically biased towards 0 or 1; hence, to emulate such a workload, we generated SPs from a bimodal distribution with peaks at SP = 0.1 and SP = 0.9, in consistence with [60], and set the input AFs to $2s(1-s)$, where s is the SP of that input. To generate a sample of true delay values for the CUT over time, we generated a sample of these input SPs and AFs, propagated them to internal nodes of the circuit, and performed aging-aware STA on it using these SPAF values to simulate the aging

of the circuit under an actual workload. The aging-aware STA engine was developed in C++, and the experiments were performed on a 64-bit Ubuntu server with a 3GHz Intel® Core™2 Duo CPU E8400 processor.

While we did not implement the method in silicon, our aging models are based on approaches that are widely accepted in the reliability community, and have been validated by other researchers using experimental silicon measurements.

4.2.1 Speed Wastage Factor

To observe the advantage of recalibrating the sensors, we first define a metric based on the inherent pessimism of the worst-case workload assumption. For this, we perform Monte Carlo simulations on the benchmark circuits with 500 sets of realistic SP and AF values to obtain the temporal trends of their delays under 500 different workloads. Each simulation corresponds to a sample of the realistic input SPs and AFs, propagated throughout a circuit to generate SPs and AFs at internal nodes, and translated into a delay degradation number for each gate. For the ith Monte Carlo run of a circuit, we define its *speed wastage factor* (SWF), or, $SWF(i,t)$, at time, t, expressed in percentage, as:

$$SWF(i,t) = \frac{\mathcal{F}_i(t) - \mathcal{F}_{pre}(t)}{\mathcal{F}_i(t)} \tag{4.9}$$

where $\mathcal{F}_{pre}(t)$ is the operating frequency set at the presilicon stage with a worst-case workload assumption, and $\mathcal{F}_i(t)$ is the maximum frequency at which the circuit can function correctly at time, t, without any timing violation, corresponding to the workload characterized by the ith Monte Carlo sample. If the exact workload of the CUT was known, it could have been operated at $\mathcal{F}_i(t)$ which is greater than $\mathcal{F}_{pre}(t)$. However, since the workload is unknown, the operating frequency is set to $\mathcal{F}_{pre}(t)$, considering a worst-case aging scenario, so that the CUT is guaranteed to function correctly during its lifetime under aging. The SWF is thus an indication of the performance margin left on the table while assuming the worst-case workload on the CUT, and the *lower* the SWF, the *better* is the performance of the CUT.

To observe the cumulative wastage over the entire lifetime of the CUT, we further define a vector, $\overline{\boldsymbol{SWF}}$, whose i^{th} element, $\overline{SWF}(i)$, is the sampled average of $SWF(i,t)$

at time instants, $t = t_j$, during the CUT lifetime, as:

$$\overline{SWF}(i) = \frac{1}{n}\sum_{j=1}^{n} SWF(i, t_j), \quad t_j \in [t_0, t_f] \tag{4.10}$$

where $i = 1, \cdots, 500$ corresponds to the 500 simulated workload scenarios.

The mean and range (minimum to maximum) of $\overline{\boldsymbol{SWF}}$ of a set of benchmark circuits without sensor recalibration, is depicted by the first set of bars in Fig. 4.4. This set of bars correspond to a case where a single measurement is performed at the beginning of lifetime of the CUT, i.e., the operating frequency is determined at the presilicon stage. The average height of this set of bars is 8.78%. In other words, if the aging sensor is calibrated assuming a pessimistic worst-case circuit workload, the circuit is operated at a frequency that is, on average, 8.78% slower than its true capability, consuming unnecessary power/area overheads. We aim to reduce this pessimism using the concept of sensor recalibration by infrequent CUT delay measurements (Sec. 4.1).

4.2.2 Choice of Measurement Instants

The choice of the measurement instants during the lifetime of a circuit is crucial in determining the extent of pessimism reduction. For various choices of a single measurement instant, t_{m_1}, over the lifetime of the circuit, we plot the statistics of $\overline{\boldsymbol{SWF}}$ over our Monte Carlo simulations for various benchmark circuits in Fig. 4.4, assuming the same $t_f = 10$ years for all circuits. In other words, we choose to perform recalibration at a single instant, t_{m_1}, through the entire 10-year lifetime, and observe the impact of this choice on the average SWF in Fig. 4.4. For example, year=0 in the figure corresponds to a (redundant) recalibration at the beginning of life, which means that no recalibration is performed during the lifetime of the CUT, and only presilicon analysis is used to set its operating frequency.

Although $\overline{\boldsymbol{SWF}}$ is reduced for any $t_{m_1} > 0$, there exists a global minimum in $\overline{\boldsymbol{SWF}}$ for a certain choice of the single measurement instant, which is at around 2 years in Fig. 4.4.

We thus attempt to optimize the choice of the N measurement instants which will minimize the SWF. Since we consider circuits with lifetime of 10 years, where both BTI and HCI are prevalent, we heuristically choose the interval between the measurement instants linearly in $f(t) + g(t)$. For N measurement instants in (t_0, t_f), the ith

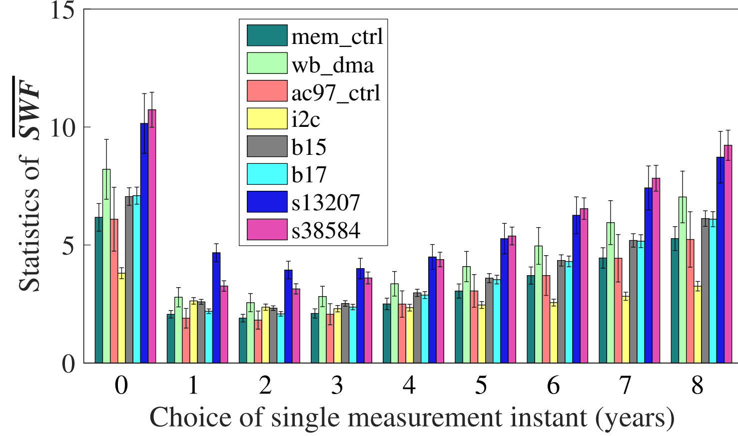


Figure 4.4: Effect of the choice of t_{m_1} on average SWF.

measurement instant, t_{m_i}, is thus obtained by solving the equation:

$$f(t_{m_i}) + g(t_{m_i}) = \left(\frac{i}{N+1}\right)(f(t_f) + g(t_f)) \tag{4.11}$$

This nonlinear equation does not have a closed-form solution but can easily be solved numerically using Mathematica [61]. For convenience, each solution, t_{m_i}, is rounded off to the nearest half-year.

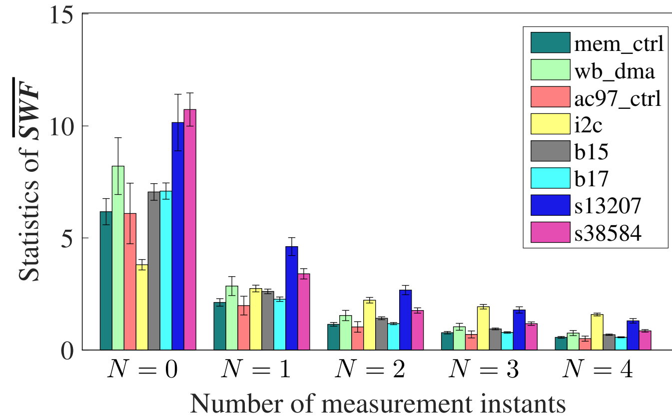


Figure 4.5: Reduction in average SWF with number of measurement instants.

Using Eq. (4.11) to obtain the N measurement instants for $N = 1, 2, 3, 4$, and assuming $t_f = 10$ years for each circuit, we show the mean and range of $\overline{\boldsymbol{SWF}}$ in Fig. 4.5. The SWF is guaranteed to reduce monotonically in all cases, due to the intermediate

true delay measurements of the CUT that refine the aging estimate, and for $N = 4$, the average SWF is less than 1%. In almost all cases, a steep reduction is seen at $N = 1$. The exception is the circuit, i2c, for which the reduction is less than other circuits. This occurs because the recalibration scheme for i2c follows Case II in Theorem 4.1, i.e., it fits the estimated delay curve between the actual delay at the current measurement instant and the worst-case delay at the next. This case was depicted in Fig. 4.3 (right) for $N = 1$ under one SPAF sample, corresponding to a specific workload. In general, Case II is seen to rarely arise, except for small circuits such as i2c (with ~ 500 gates): in particular, for large circuits that have tens of thousands of gates, this scenario was not observed.

Finally, N may be chosen depending on the desired amount of SWF reduction for a particular CUT lifetime, t_f. For example, in Fig. 4.5, the average SWF is already reduced to below 1% for $N = 4$. Hence, increasing N beyond four does not provide significant improvements in the SWF for the conditions assumed here.

We now examine the data from Fig. 4.5 in greater detail and focus on the error in $\Delta D_{post}^{C}(t)$ instead of $D_{post}^{C}(t)$ (which was incorporated in the SWF metric). This error is quantified by the vector, $\boldsymbol{E_\Delta}$, whose i^{th} element, $E_\Delta(i)$, corresponds to the ith Monte Carlo run using a specific workload, averaged over time instants, $t = t_j$, and expressed as:

$$E_\Delta(i) = \frac{1}{n}\sum_{j=1}^{n}\left[\frac{\Delta D_{post}^{C,i}(t_j) - \Delta D_{a}^{C,i}(t_j)}{\Delta D_{a}^{C,i}(t_j)}\right], \quad t_j \in (t_0, t_f] \tag{4.12}$$

where $\Delta D_{post}^{C,i}(t_j)$ and $\Delta D_{a}^{C,i}(t_j)$ are, respectively, the estimated post-silicon and actual delay degradation of C, from $t = t_0$ under the ith workload.

Table 4.1 reports the statistics of $\boldsymbol{E_\Delta}$ for various values of N. For each circuit, we show the mean and standard deviation of $\boldsymbol{E_\Delta}$ for $N = 0, 1, \cdots, 4$. The $N = 0$ case corresponds to the UofM model-based estimate where the sensors were calibrated based on presilicon analysis only. The columns also show the vector of measurement instants, T_M, expressed in years (excluding t_0), for each N based on Eq. (4.11) with $t_f = 10$ years.

Consistent with prior observations, there is a significant error for the $N = 0$ case, and this error is reduced as N is increased. However, the pessimism in the upper bound is never completely removed, which is desired since this guarantees timing closure

Table 4.1: Statistics of $\boldsymbol{E_\Delta}$ with sensor recalibration, expressed as percentage, for various sets of measurement instants, T_M, in years.

CUT	$N=0$ $T_M=[\,]$		$N=1$ $T_M=[1.5]$		$N=2$ $T_M=[0.5,3.5]$		$N=3$ $T_M=[0.5,1.5,5]$		$N=4$ $T_M=[0.5,1.5,2.5,6]$	
	μ_{E_Δ}	σ_{E_Δ}	μ_{E_Δ}	σ_{E_Δ}	μ_{E_Δ}	σ_{E_Δ}	μ_{E_Δ}	σ_{E_Δ}	μ_{E_Δ}	σ_{E_Δ}
mem_ctrl	70.21	10.83	23.91	3.30	12.25	1.56	8.12	1.02	5.89	0.74
wb_dma	89.62	24.23	30.44	7.78	15.53	3.87	10.27	2.55	7.42	1.85
ac97_ctrl	48.43	15.88	15.85	4.89	7.62	2.44	5.04	1.58	3.64	1.14
i2c	42.19	3.69	29.64	2.38	23.60	1.92	20.34	1.58	16.65	1.09
b15	103.51	11.00	37.89	3.34	19.50	1.58	12.71	0.99	9.13	0.69
b17	95.67	9.29	30.56	2.69	14.86	1.18	9.69	0.74	6.96	0.52
s13207	256.25	96.14	110.44	37.52	61.46	20.09	40.19	13.12	28.92	9.44
s38584	341.49	94.36	105.53	28.62	50.68	13.63	33.13	8.89	23.83	6.39

throughout the lifetime of the CUT.

To summarize, our first method based on the UofM bound provides a workable indication of circuit aging with practically no extra effort on part of the designer and a small power/area overhead. A one-time presilicon circuit characterization was used to determine the degradation ratios, and subsequently ROSC measurements were translated to the circuit delay using these ratios. The second proposed method provides a more accurate indication of aging, but requires extra overhead for the intermediate CUT delay measurement and for updating the LUT, and could introduce minor delay overheads ($\sim 1\%$ if the scheme in [47] is employed) in the near-critical paths of the CUT due to the increased fanout load. Although this technique needs higher design effort, the aging estimates obtained are more accurate compared to the first method. However once the test infrastructure is in place, the runtime overhead is negligible for this method since a couple of CUT measurements over 10 years are adequate to bridge the gap in the SWF left by the UofM method.

4.3 Conclusion

In this chapter we have presented a technique to estimate aging in circuits due to BTI and HCI, using on-chip ROSC-based sensors, and infrequent delay tests on the monitored circuit. Since the presilicon analysis is built on the premise of worst-case workload on a CUT, its aging estimate using the techniques from the previous chapter can be further refined using infrequent post-silicon measurements performed directly on the CUT to update the sensor calibration. The updated calibration factors, used in conjunction with the post-silicon measurements on the sensor, can partially capture the real workload of the CUT to yield more accurate aging estimates.

Chapter 5

Error Analysis in Unsigned Multipliers

So far we have discussed techniques to estimate the unintentional errors arising due to aging-related parametric failures in circuits to ensure reliability in safety-critical applications. This chapter onwards, we will switch gear, and study the intentional unreliability in circuits introduced through approximate computing for error tolerant applications.

A vital ingredient of any methodology based on approximate design is a fast and accurate procedure that can quantify the distribution of error injected into a computation by an approximation scheme. The most common building blocks that are used to build hardware for error-resilient computations are adders and multipliers. While existing methods have made some progress in analyzing errors in adders [29–31, 62], design of the approximate multipliers [32–35] still relies on error metrics from Monte Carlo simulations for performance evaluation since there are no known analytical methods that can scalably and accurately analyze the error in multipliers.

In this chapter, we propose a novel algorithm, **FEMTO**: **F**ast **E**rror Analysis in **M**ultipliers through **TO**pological Traversal, to efficiently quantify the errors in the output of an approximate multiplier by determining their probability of occurrence. The errors in approximate circuits which follow discrete asymmetric distributions [63], are propagated through networks using a topological traversal, and FEMTO uses the

frequency domain to reduce computation.

At the gate level of approximate circuit design, the error of a logic function can be quantified by comparing the truth table of the approximate and exact implementations. However, this is not scalable beyond a small number of inputs because the size of the truth table grows exponentially with the number of inputs. Prior approaches that attempt to overcome the computational bottleneck of error estimation can be classified into two categories:

- **Methods that estimate the range of error**: These methods capture the range of the error in approximate computation in terms of its minimum and maximum value, and are primarily based on interval and affine arithmetic [29, 63–65]. The runtime for such interval-based approaches increases exponentially when more intervals are required for large ranges of signals, and they clamp the errors to maximum/minimum values in the range, thus often overestimating/underestimating the actual erros.

- **Methods that capture the statistics of the error distribution**: These methods use the computationally intensive Monte Carlo simulations using millions of random input vectors to obtain various quality metrics in an approximate computation such as the error rate, error significance, average error, and mean square error, to quantify the error in approximate systems [31, 32, 34, 62, 66, 67].

In several scenarios, it is essential to determine the entire probability distribution of error rather than certain quality metrics, e.g. for hypothesis testing in stochastic sensor circuits [68] and for accuracy evaluation [34, 35] of approximate circuits (which is currently performed by exhaustive/Monte Carlo simulations). Our algorithm captures the entire probability distribution of error, and is significantly faster than Monte Carlo simulations. The advantage of obtaining an analytical expression for the error PMF is that the error range, its statistics and percentiles can be easily deduced from the cumulative distribution function (CDF), and this method can be used in lieu of the time-intensive Monte Carlo simulations to evaluate performance of approximate circuits in terms of their output quality.

Our algorithm is schematically represented in Fig. 5.1 on an N-bit $\times$ N-bit unsigned multiplier. The statistics of the two N-bit operands A and B (i.e., the probability

that they take on values 1 and 0) are provided as an input to our approach. The multiplication process generates partial products (PP_1, $\cdots$, PP_{last}) as shown in the figure, and the computation proceeds by successively adding each partial product to the partial sum computed so far. Each such addition is performed by an array, Σ, of approximate and/or exact full adders, and is characterized by an error PMF for the adder array.

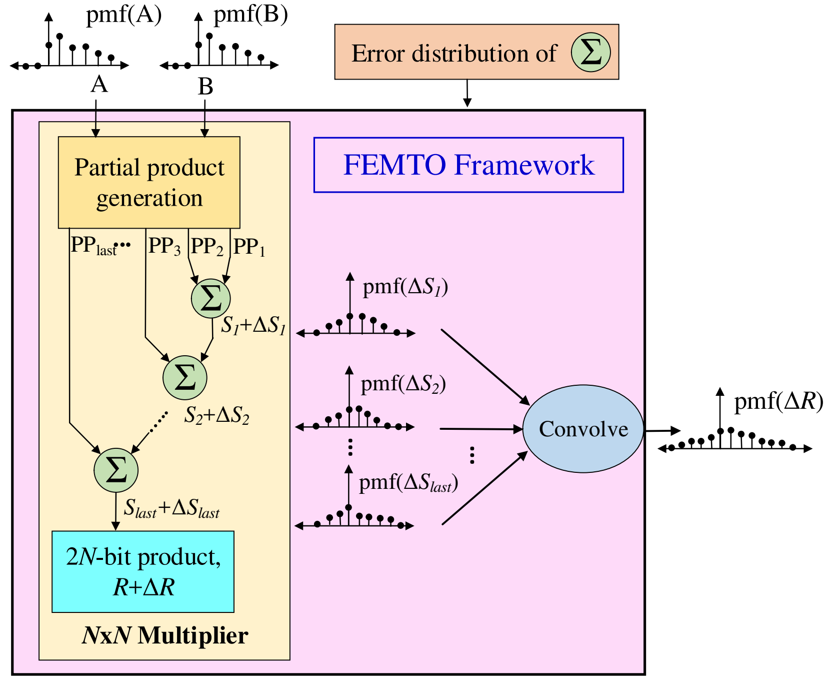


Figure 5.1: Schematic of the FEMTO algorithm on an unsigned multiplier.

Our approach proceeds as follows:

1. We obtain the PMF of the error of each full adder (Section 5.1).

2. We use these PMFs to compute the PMF of ΔS_i, the error introduced by the approximate adders at the ith level, over the statistical distribution of inputs A and B.

3. The total error introduced by the multiplication is the sum of the ΔS_i variables. We show that the PMF of the total error can be expressed as a convolution of a weighted set of error PMFs for individual full adders, and demonstrate how

this convolution is performed efficiently in the frequency domain in an intelligent manner, avoiding an explosion in the number of terms in the frequency-domain representation of the PMF (Section 5.2).

4. We enhance the speed of our algorithm by partitioning the N-bit operands into K-bit slices (Section 5.3).

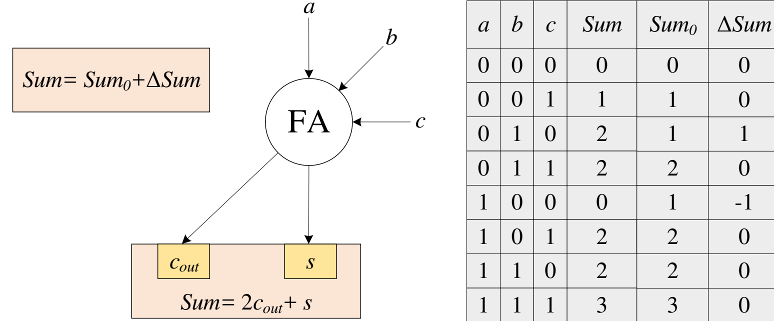
We experimentally validate our algorithm in Section 5.4 on a set of approximate multiplier schemes and demonstrate that it can achieve Monte Carlo-like accuracy, with a very good runtime performance.

5.1 Characterizing the PMF of Full Adders

In principle, the PMF of any combinational structure can be characterized through its truth table and the statistics of the inputs. However, the size of the truth table increases exponentially with the size of the input space, and such a direct characterization is impractical for a multiplier. Hence, we work with a fundamental unit that can reasonably be characterized – in this case, a full adder (FA) – and develop the error PMF for the multiplier hierarchically. Specifically, the error PMF of a single adder is used to obtain the error PMF of each row of FAs that sums the partial products, and finally the error distribution of the entire approximate multiplier. This section explains how we use the input distribution and Boolean function of an FA to obtain its output error distribution.

Let us explain our approach with the example of an approximate FA, *appx1* from [1], shown in Fig. 5.2. The truth table of its output, Sum, as compared against the exact output, Sum_0, is also shown in the figure. The inputs, a, b and c, are modeled as random variables with a known distribution, and the error injected by the multiplier is denoted as ΔSum. Since the inputs are binary, we represent their probability of being 1 as p_a, p_b and p_c, respectively. Similarly, $p'_x = 1 - p_x$, $(x = a, b, c)$, are the probabilities of a, b and c, respectively, to be 0. The PMF of the resultant sum (Sum) combining both the output bits, s and c_{out}, and the PMF of the error (ΔSum) in the resultant sum are defined by $f_{Sum}(n)$ and $f_{\Delta Sum}(n)$, respectively.

If the inputs are independent, and represented by an identical uniform distribution (p_a=p_b=p_c=0.5), then $f_{\Delta Sum}(n)$ and $f_{Sum}(n)$ can trivially be obtained from the truth



a	b	c	Sum	Sum_0	ΔSum
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	2	1	1
0	1	1	2	2	0
1	0	0	0	1	-1
1	0	1	2	2	0
1	1	0	2	2	0
1	1	1	3	3	0

Figure 5.2: Full adder (FA) with the associated truth table (*appx1* from [1]).

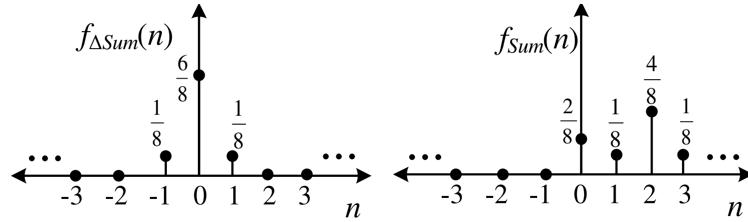


Figure 5.3: Output signal and error distribution for the *appx1* adder from [1].

table, and are depicted in the two plots in Fig. 5.3. For example, the PMF of ΔSum can be computed by observing that it takes the value 0 in six of eight entries in the truth table, and the values -1 and 1 in the remaining two entries. When the inputs are equiprobable, this leads to the PMF shown here. The PMFs in the figure can equivalently be represented as a weighted sum of discrete Kronecker delta functions as:

$$f_{\Delta Sum}(n) = \frac{6}{8}\delta(n) + \frac{1}{8}\delta(n-1) + \frac{1}{8}\delta(n+1) \tag{5.1}$$

$$f_{Sum}(n) = \frac{2}{8}\delta(n) + \frac{1}{8}\delta(n-1) + \frac{4}{8}\delta(n-2) + \frac{1}{8}\delta(n-3) \tag{5.2}$$

The coefficients of the delta functions in Eqs. (5.1) and (5.2) are the PMF values of the associated random variable (ΔSum or Sum) and are the length of the stems in Fig. 5.3. When the inputs are not equiprobable, the coefficient of $\delta(n-k)$ in the PMF of a random variable is the probability of the variable to be k; hence, assuming the inputs to be independent, the coefficients can also be expressed as functions of p_a, p_b and p_c as shown in Tables 5.1 and 5.2.

For a general approximate FA, ΔSum can range from -3 to 3, as the two output bits may have error of any of -1, 0 or 1 (although for the *appx1* adder shown in Fig. 5.2,

Table 5.1: PMF of the FA output.

n	$f_{Sum}(n)$
0	$p_0 = p'_a p'_b p'_c + p_a p'_b p'_c$
1	$p_1 = p'_a p'_b p_c$
2	$p_2 = p'_a p_b + p_a p'_b p_c + p_a p_b p'_c$
3	$p_3 = p_a p_b p_c$

Table 5.2: PMF of the errors in FA output.

n	$f_{\Delta Sum}(n)$
-1	$e_{-1} = p_a p'_b p'_c$
0	$e_0 = 1 - p_a p'_b p'_c - p'_a p_b p'_c$
1	$e_1 = p'_a p_b p'_c$

it only ranges from -1 to 1). Hence, the PMF of ΔSum and Sum can, in general, be expressed as a sum of Kronecker delta functions as:

$$f_{\Delta Sum}(n) = \textstyle\sum_{i=-3}^{3} e_i \delta(n-i) \tag{5.3}$$

$$f_{Sum}(n) = \textstyle\sum_{i=0}^{3} p_i \delta(n-i) \tag{5.4}$$

where the p_is and e_is are expressed as functions of p_a, p_b and p_c similar to Tables 5.1 and 5.2, respectively, and can be computed by substituting the values from the knowledge of the input distribution.

5.2 Overview of the FEMTO Algorithm

Consider a 4-bit×4-bit array multiplier with operands, A ($a_3a_2a_1a_0$) and B ($b_3b_2b_1b_0$), as shown in Fig. 5.4. The full adders in the array are each indexed as FA$_{ij}$, where i corresponds to the row number, starting from the top, and j to the position of the FA in the row, starting from the least significant bit (LSB), as shown in Fig. 5.4. The output of a single adder, FA$_{ij}$, is modeled as the random variable, $Sum_{ij} = Sum_{ij,0} + \Delta Sum_{ij}$, where $Sum_{ij,0}$ is the true sum (corresponding to an exact FA), and ΔSum_{ij} is the error due to the approximate addition, similar to the example of the FA in Fig. 5.2.

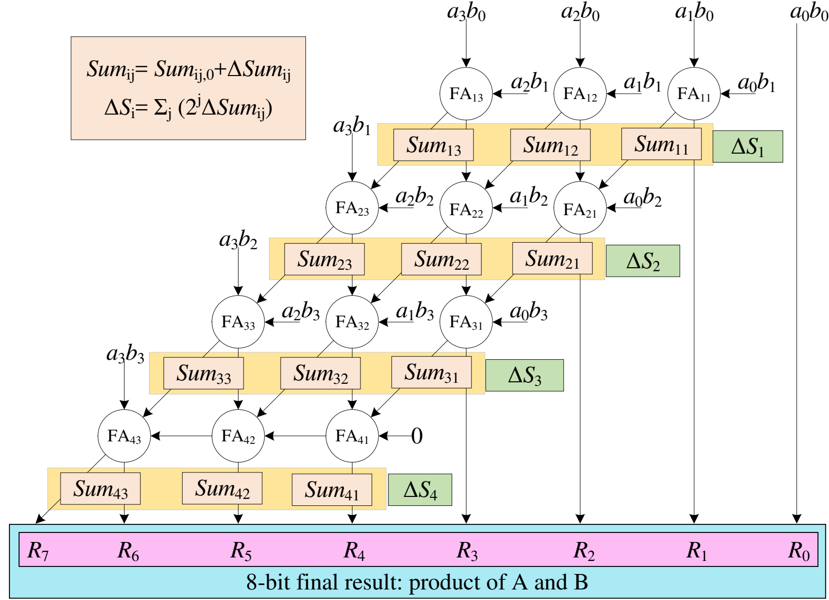


Figure 5.4: Structure of a 4-bit×4-bit array multiplier.

Before proceeding further, let us comment on the input data distribution, and our assumptions regarding the correlation between the various Sum_{ij} random variables. Although we assume the inputs, A and B, to be independent random variables, their distribution is a user-input (and can be any arbitrary distribution) from which the signal probability, p_{a_i} and p_{b_i}, of each input bit, a_i and b_i, respectively, can be inferred. In addition to the error PMF, FEMTO has the capability to produce the output PMF (signal probability of the output bits of the multiplier), which can be used as input signal probability in subsequent multipliers within a data flow graph. Thus FEMTO can propagate the probability distribution of the data from input to the output of the multiplier. Additionally, within the multiplier, we consider the correlation between the s and c_{out} bits of any adder by combining them into a two-bit output, Sum, and the corresponding error, ΔSum, both expressed in decimal, with their PMF characterized by similar methods as Tables 5.1 and 5.2, respectively. This technique captures the most important correlation which is the interdependence of the two output bits of any adder within the multiplier array. Although we ignore the correlations between the outputs of different adders by considering Sum_{ij}s to be independent random variables,

this assumption does not affect the quality of our results since correlations due to reconvergent fanout tend to be diluted as the logic depth of the reconvergent fanout paths increases. Finding the error PMF for the multiplier array involves three steps:

- **Step 1:** determining the input probabilities for all inputs of each individual FA_{ij}, and using the approach in Section 5.1 to compute the PMF of ΔSum_{ij},
- **Step 2:** finding the PMF of the error, ΔS_i, introduced by the ith row of the multiplier array, and
- **Step 3:** finding the PMF of the entire multiplier, i.e., the PMF of the sum of the ΔS_i variables over all rows, i.

Step 1: The first step simply involves probability propagation within a Boolean network, and we use established techniques for this purpose [69]. Based on this, we obtain the PMF of ΔSum_{ij}, denoted by $f_{\Delta Sum_{ij}}(n)$, as a sum of delta functions similar to Eq. (5.3).

Step 2: Next, we determine the error in the partial product accumulation, ΔS_i, in the ith row, which is the total error resulting from an array of $N-1$ approximate FAs, as depicted in Fig. 5.4 for $N=4$. For each row, $i \in \{1, \cdots, N\}$, a simple analysis yields:

$$\Delta S_i = \sum_{j=1}^{N-1} 2^{i+j-1} \Delta Sum_{ij} \tag{5.5}$$

Since we consider the ΔSum_{ij} random variables to be independent[1] , we can utilize the fact that the PMF of sum of independent random variables equals the convolution of the PMF of those random variables. hence, the PMF of ΔS_i can be expressed as:

$$f_{\Delta S_i}(n) = \bigotimes_{j=1}^{N-1} f_{2^{i+j-1}\Delta Sum_{ij}}(n) \tag{5.6}$$

where $\bigotimes$ is the convolution operator, applied here to convolve $N-1$ operands, and $f_{2^{i+j-1}\Delta Sum_{ij}}(n)$ is the PMF of the random variable, $2^{i+j-1}\Delta Sum_{ij}$.

If the absolute value of the largest output error of FA_{ij} is M (e.g., $M=1$ for the *appx1* FA in Section 5.1), then using Eq. (5.3),

$$f_{2^{i+j-1}\Delta Sum_{ij}}(n) = \sum_{k=-M}^{M} e_k^{(ij)} \delta(n - 2^{i+j-1}k) \tag{5.7}$$

[1] The assumptions are also borne out by results.

where $e_k^{(ij)}$ is the probability of ΔSum_{ij} to be k, and the superscript, (ij), indicates the position in the error-producing FA in the multiplier array.

Step 3: The error, ΔR, in the multiplier output is simply the sum of the errors, ΔS_i, over all N rows. Assuming the ΔS_i random variables to be independent, we obtain the error PMF of the multiplier result, $f_{\Delta R}(n)$, by convolving the $f_{\Delta S_i}(n)$ PMFs:

$$f_{\Delta R}(n) = \bigotimes_{i=1}^{N} f_{\Delta S_i}(n) = \bigotimes_{i=1}^{N} \bigotimes_{j=1}^{N-1} f_{2^{i+j-1}\Delta Sum_{ij}}(n) \tag{5.8}$$

We implement the following techniques to solve the above convolution problem to obtain the PMF of error in the final product:

1. Use the Z-transform [70] to convert the convolution into a friendlier multiplication in the frequency domain, yielding a polynomial in z. This polynomial can have exponential number of terms, and special techniques are required to manage the cost of working in the transform domain.

2. Use the inverse fast Fourier transform (IFFT) [70] to infer the PMF of ΔR from the polynomial obtained in the previous step.

In the next two subsections, we explain each of these techniques in detail.

5.2.1 Convolution using the Z-transformed Domain

According to the principles of transform calculus, the Z-transform of a convolution of multiple functions in the original domain is equivalent to the product of the Z-transforms of those individual functions in the transform domain. Hence, we can represent $F_{\Delta R}(z)$, the Z-transform of $f_{\Delta R}(n)$ in Eq. (5.8), as:

$$F_{\Delta R}(z) = \prod_{i=1}^{N} \prod_{j=1}^{N-1} F_{2^{i+j-1}\Delta Sum_{ij}}(z) \tag{5.9}$$

where $F_{2^{i+j-1}\Delta Sum_{ij}}(z)$, is the Z-transform of the PMF, $f_{2^{i+j-1}\Delta Sum_{ij}}(n)$. Applying the Z-transform to both sides of Eq. (5.7),

$$F_{2^{i+j-1}\Delta Sum_{ij}}(z) = \sum_{k=-M}^{M} e_k^{(ij)} z^{-2^{i+j-1}k} \tag{5.10}$$

Substituting Eq. (5.10) in Eq. (5.9), we can rewrite $F_{\Delta R}(z)$ as:

$$F_{\Delta R}(z) = \prod_{i=1}^{N} \prod_{j=1}^{N-1} \left(\sum_{k=-M}^{M} e_k^{(ij)} z^{-2^{i+j-1}k} \right) \tag{5.11}$$

$$= z^{-E} \sum_{i=0}^{2E} a_i z^i \tag{5.12}$$

$$= z^{-E} \phi(z) \tag{5.13}$$

where ΔR ranges from $-E$ to E, with $E = (2^{N-1} - 1)(2^N - 1)M$, and the a_is are the coefficients of the polynomial in z, denoted by $\phi(z)$ in Eq. (5.13). The derivation of E has been deferred to Appendix B for better readability. Performing the inverse Z-transform of Eq. (5.12), we obtain:

$$f_{\Delta R}(n) = \sum_{i=0}^{2E} a_i \delta(n + i - E) \tag{5.14}$$

Hence, a_i is the probability of the error, ΔR, to be $-(i - E)$. Thus finding the PMF of the error reduces to the problem of finding the coefficients, a_i, in $\phi(z) \left(= \sum_{i=0}^{2E} a^i z^i\right)$, which is a polynomial of degree $2E$ with non-negative coefficients.

While this scheme presents a clear picture of our computation scheme, the cost of a direct implementation of this idea is prohibitive. The most expensive step is the determination of the coefficients, a_i, by multiplying the terms in Eq. (5.11).

5.2.2 Inverse Fast Fourier Transform to infer the Error PMF of Multipliers

In this subsection, we present a method for efficiently computing the a_i coefficients in Eq. (5.14).

So far we have worked in the Z-transform domain to formulate the error PMF equation, $F_{\Delta R}(z)$. Let us now consider discrete Fourier domain to determine the coefficients, a_i, in $\phi(z)$ from Eq. (5.13), by using their Fourier-transformed values followed by performing inverse fast Fourier transform (IFFT). This interchange of domains is possible since, by definition, Z-transform is equivalent to the discrete time Fourier transform (DTFT) when the magnitude of $|z| = 1$ [70].

We begin by observing that the DTFT of the sequence, $\{a_0, a_1, \cdots, a_{2E}\}$, is given by the Fourier coefficients,

$$A_k = \sum_{i=0}^{2E} a_i \exp\left(-\mathbf{j}\frac{2\pi ik}{2E+1}\right) = \sum_{i=0}^{2E} a_i z_k^i \tag{5.15}$$

where $\mathbf{j} = \sqrt{-1}$ and $z_k = \exp\left(-\mathbf{j}\frac{2\pi k}{2E+1}\right)$. It is interesting to note that the values of z_k are the reciprocal of the $(2E+1)^{\text{th}}$ complex roots of unity.

Therefore, if we take $\phi(z)$ in Eq. (5.13) and substitute $z = z_k$, for the reciprocal of each of the $(2E+1)^{\text{th}}$ complex roots of unity, we obtain the Fourier coefficient, A_k. In other words,

$$A_k = \phi(z_k) = z_k^E \prod_{i=1}^{N} \prod_{j=1}^{N-1} \left(\sum_{k=-M}^{M} e_k^{(ij)} z_k^{-2^{i+j-1}k} \right) \tag{5.16}$$

This provides us with the discrete Fourier coefficients of the sequence of a_is, which are then obtained by performing inverse discrete time Fourier transform (IDFT) of the A_k values as:

$$a_i = \frac{1}{2E+1} \sum_{k=0}^{2E} A_k exp\left(\mathbf{j}\frac{2\pi ik}{2E+1}\right) \tag{5.17}$$

To compute the IDFT in Eq. (5.17) efficiently, we use the IFFT to obtain the values of the a_is. As explained in the previous subsection, obtaining the a_is directly provides the PMF of the error, ΔR, in the multiplier output.

5.3 Enhancing Efficiency of FEMTO

The error PMF obtained by the FEMTO algorithm provides probability of occurrence of errors with **unit-granularity**. In other words, we obtain $f_{\Delta R}(n)$ in (5.14), for each integer value of n within the error range, i.e., with unit spacing between successive values of n.

To enhance efficiency, we propose to process the multiplication by partitioning each of the N-bit operands, A and B, in an N-bit$\times N$-bit multiplier, into K-bit slices. The product of A and B is obtained using the results of K^2 N/K-bit$\times N/K$-bit multipliers as shown in Fig. 5.5. We call this the **partitioned-granularity** approach of obtaining the PMFs.

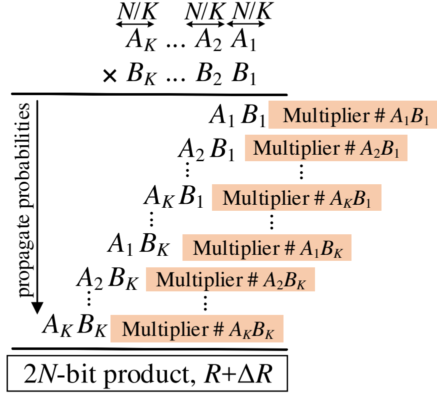


Figure 5.5: An N-bit$\times N$-bit multiplier from N/K-bit$\times N/K$-bit multipliers.

We then process the computation hierarchically. Instead of an FA, as in the previous section, we now use an N/K-bit$\times N/K$-bit multiplier as the fundamental unit. However, while the PMF of an FA can be exactly characterized through the truth table, this is not the case for the N/K-bit$\times N/K$-bit multiplier; instead, we use the approach in Section 5.2 to obtain this PMF. Characterizing the PMF of the N/K-bit$\times N/K$-bit multiplier requires two types of input data: the error distribution of the FAs, which is provided in Section 5.1, and the PMFs of the inputs to the multiplier, which is computed using the probability propagation algorithm in Step 1 of Section 5.2 (note that these probabilities are cheap to compute, and do not change whether we use this hierarchical scheme or the previous "flat" scheme).

Given the PMF of the N/K-bit$\times N/K$-bit multiplier as the fundamental block, the scheme in Section 5.2 can now be used to find the PMF of the multiplier error.

Practical runtime enhancement of FEMTO: The error range, $-E_K$ to E_K ($E_K \sim 2^{2N/K}$), of the N/K-bit$\times N/K$-bit multiplier, can be grouped or "binned" into P windows for further runtime enhancement of FEMTO, where P is chosen empirically. This idea is best explained with an example in Fig. 5.6. If the output error in the N/K-bit$\times N/K$-bit multiplier is represented by the random variable, ΔR, with $E_K = 15$, and the PMF of ΔR is $f_{\Delta R}(n)$, then we can group ΔR into $P = 5$ windows to obtain the binned version of the PMF, $f_{\Delta R,bin}(n)$ with fewer data points.

There is a trade-off between the computational effort, and the accuracy of the PMF

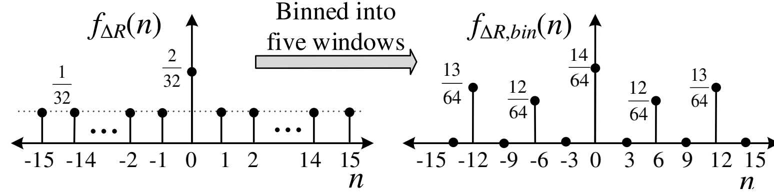


Figure 5.6: An error PMF (not to scale) with unit-granularity (left) and its binned version into five windows (right).

obtained by this approach. While the algorithm speeds up by $\left(\frac{2E_K}{P}\right)$X, inaccuracy is introduced due to the representation of $\frac{2E_K}{P}$ error values by a single value. Therefore, P should be chosen depending on the value of E_K to maintain an acceptable runtime-accuracy trade-off.

5.4 Experimental Setup and Results

We implement FEMTO in MATLAB R2010b in a 2.53 GHz Intel Core i3 CPU with 4Gb RAM and 64-bit Windows 7 OS, and present the results relating to approximate unsigned multipliers. The approximate adders that constitute the multipliers are the various transistor level approximations of the mirror adders [1], the Boolean expressions of which are mentioned in Table 5.3 for the two outputs, s and c_{out}, and inputs, a, b and c.

Table 5.3: Five versions of approximate adders from [1].

Approximate version	s	c_{out}
appx1	$\bar{a}\bar{b}c + abc$	$abc + ab\bar{c} + a\bar{b}c + \bar{a}bc + \bar{a}b\bar{c}$
appx2	$\overline{c_{out}}$	$ab + \bar{a}bc + a\bar{b}c$
appx3	$\overline{c_{out}}$	$abc + ab\bar{c} + a\bar{b}c + \bar{a}bc + \bar{a}b\bar{c}$
appx4	$\bar{a}\bar{b}c + \bar{a}bc + abc$	a
appx5	b	a

Since in practice, no more than ~50% of the resultant bits are usually approximated to maintain accuracy [1, 34], we use a similar strategy to approximate different number adders to implement the multipliers. We obtain the PMF of the errors normalized to the dynamic range of the output of the approximate multiplier. The normalization factor,

$\mathcal{R}$, is the total range (difference of maximum and minimum values) of the output when the multiplier is implemented using different combinations of approximate and accurate adders. The outputs are obtained by performing 2000 Monte Carlo logic simulations on the approximate multiplier. Such a normalization step is necessary since the same magnitude of error may have different levels of severity depending on the magnitude of the product.

First, we present the results for the combination of approximate and accurate adders to construct the multipliers, such that exactly 50% of the product bits from the LSB position are approximate. Referring to Fig. 5.4, this means that for the 4-bit×4-bit multiplier, the resultant bits, $R_1, \cdots, R_4$, are approximate and the rest are exact. The LSB, R_0, is exact since it is not produced by any adder and is simply the output of an AND gate. In general, for an $N \times N$ multiplier, to approximate 50% of the LSBs in the product, $N - i$ LSB adders in the ith row of the adder array in each partial product accumulation level should be approximate, with the rest being accurate, for $i = 1, \cdots, N$.

We implement FEMTO on 6-bit×6-bit, 8-bit×8-bit, and 16-bit×16-bit approximate multipliers. While the error PMFs for 6-bit×6-bit and 8-bit×8-bit are obtained by the unit-granularity approach, the error PMF for 16×16 multiplier is obtained by the partitioned-granularity approach using $K = 2$, i.e., using the results of the 8-bit×8-bit multiplier, and $P = 32$ windows to enhance the practical runtime, as explained in Section 5.3.

We compare the mean and standard deviation of the error PMFs obtained by FEMTO and Monte Carlo simulations using the absolute value of the normalized percentage error in mean and standard deviation, $\Delta\mu_{norm}$ and $\Delta\sigma_{norm}$, respectively, defined as:

$$\Delta\mu_{norm} = 100 \times \frac{|\mu_{FEMTO} - \mu_{MC}|}{\mathcal{R}} \tag{5.18}$$

$$\Delta\sigma_{norm} = 100 \times \frac{|\sigma_{FEMTO} - \sigma_{MC}|}{\mathcal{R}} \tag{5.19}$$

where μ_{FEMTO} (μ_{MC}) and σ_{FEMTO} (σ_{MC}) are the mean and standard deviation of the error PMF of each multiplier obtained by FEMTO (Monte Carlo simulation), respectively, and $\mathcal{R}$ is the normalizing factor defined earlier. We summarize the $\Delta\mu_{norm}$ and $\Delta\sigma_{norm}$ values, respectively, in Tables 5.4 and 5.5, corresponding to the error PMFs

of the 6×6, 8×8, and 16×16 approximate multipliers (with 50% of the product bits approximated).

Table 5.4: Normalized percentage error in estimated mean ($\Delta\mu_{norm}$) of PMF obtained by FEMTO compared against Monte Carlo simulation.

Multiplier → **Adder ↓**	6-bit×6-bit	8-bit×8-bit	16-bit×16-bit
appx1	0.23	0.12	0.12
appx2	0.09	0.06	0.41
appx3	0.14	0.13	0.67
appx4	0.01	0.01	0.02
appx5	0.02	0.02	0.26

Clearly, the error is less than 1% as observed by the $\Delta\mu_{norm}$ and $\Delta\sigma_{norm}$ values for all the approximate multipliers (with different versions of the approximate adders) considered here. This indicates that the error statistics obtained by FEMTO are very similar to those obtained by the Monte Carlo simulations.

Table 5.5: Normalized percentage error in estimated standard deviation ($\Delta\sigma_{norm}$) of PMF obtained by FEMTO compared against Monte Carlo simulation.

Multiplier → **Adder ↓**	6-bit×6-bit	8-bit×8-bit	16-bit×16-bit
appx1	0.10	0.06	0.26
appx2	0.00	0.01	0.26
appx3	0.01	0.04	0.03
appx4	0.38	0.11	0.03
appx5	0.44	0.18	0.31

Next, we consider cases when different percentages (45%, 50% and 55%) of LSBs are approximate in the product, and present the error CDFs of the 8-bit×8-bit multipliers as an example. Each column of subplots in Fig. 5.7 corresponds to a specific number of approximate bits in the 16-bit product of the 8-bit×8-bit multiplier. The numbers, 7, 8 and 9 bits, respectively, correspond to 45%, 50% and 55% of the 16-bits being approximated. Each row of subplots in Fig. 5.7 corresponds to the type of adders from Table 5.3 used to implement the approximate multiplier. The plot labeled **True** represents the CDF obtained by 2000 Monte Carlo simulations, and is assumed to be

the reference or golden CDF. The red plot labeled **Estimated** is the CDF obtained by FEMTO.

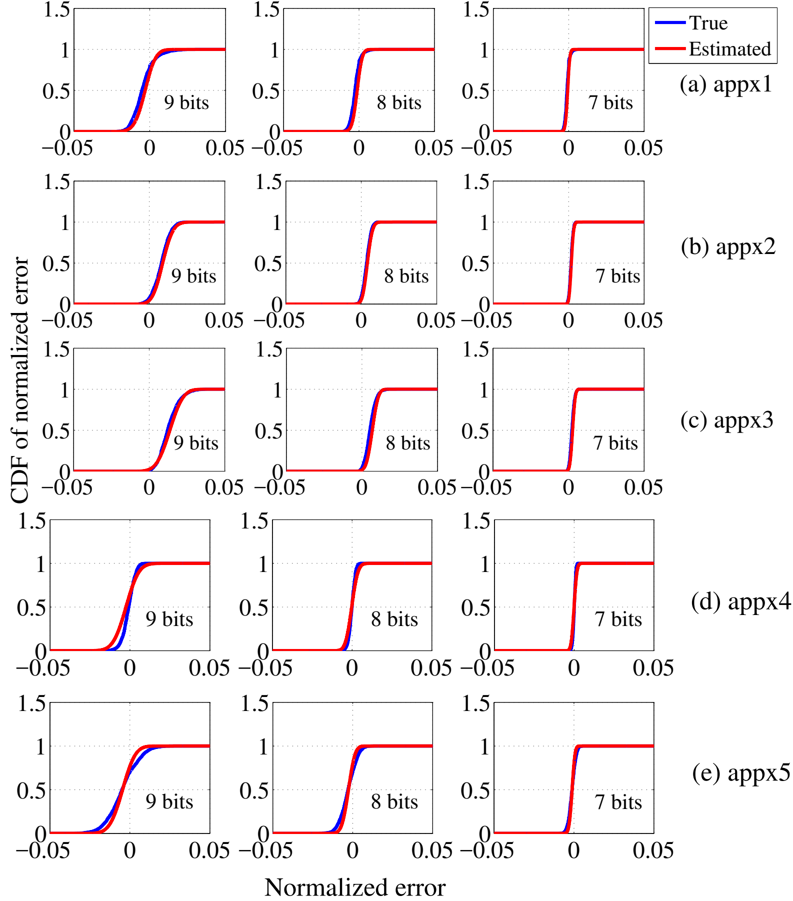


Figure 5.7: CDF of the normalized error for 8-bit×8-bit multipliers with different percentages (45%, 50% and 55%) of approximate LSBs in the product. The adders in (a)-(e) correspond to those from Table 5.3.

As expected, the errors are reduced with fewer approximate bits in the product. Additionally, we observe that the PMFs obtained by FEMTO are very close to those obtained by Monte Carlo simulation for different levels of approximation in the multiplier (as seen across each row of subplots in Fig. 5.7). Hence, for the various approximate multipliers considered here, FEMTO can predict the error distribution with accuracy comparable to Monte Carlo simulations.

To compare with one of the existing approaches of obtaining error PMF in approximate circuits, we also generate the error PMF of the multipliers using the modified interval arithmetic (MIA) based approach [63]. The authors of [63] had reported MIA to be very efficient in terms of runtime and storage complexity. Hence, we compare our approach with the MIA-based one. The multipliers are approximated such that 50% of the product LSBs are approximate, and the rest are accurate.

We compare the accuracy of the PMFs obtained by both FEMTO and MIA (implemented based on [63]), by observing the Hellinger distance [71] from the corresponding PMFs obtained from Monte Carlo simulations. Hellinger distance is a well-known metric to compare probability distributions, and is defined as:

$$\text{Hellinger distance} = \frac{1}{\sqrt{2}}\sqrt{\sum_{\text{all } n}\left(\sqrt{\hat{f}(n)} - \sqrt{f(n)}\right)^2} \tag{5.20}$$

where $\hat{f}(n)$ and $f(n)$ are the estimated and the true (Monte Carlo) PMFs, respectively. The factor, $\sqrt{2}$, ensures that this distance ranges from 0 to 1. We plot the Hellinger distances of the error PMFs obtained by FEMTO and MIA for the 6-bit×6-bit and 8-bit×8-bit multipliers in Fig. 5.8, from the PMFs obtained by Monte Carlo simulations. The smaller the distance, the closer is the estimate to the true PMF; thus clearly our approach yields more accurate PMFs compared to the MIA-based ones as seen by the lower values of the Hellinger distance for the approximate multipliers considered here.

The runtime to obtain the PMFs are summarized in Table 5.6. The 16-bit×16-bit multiplier did not finish computation within 3600 seconds which was set as the maximum observation time using the MIA-based approach, and hence, its runtime is labeled *timed out* in the table for all approximate versions of the adders.

Table 5.6: Runtime comparison to obtain the error PMFs.

Multiplier →	6-bit×6-bit		8-bit×8-bit		16-bit×16-bit	
Adder ↓	FEMTO	MIA	FEMTO	MIA	FEMTO	MIA
appx1	0.8s	6.6s	14.5s	174.2s	363.8s	*timed out*
appx2	0.7s	9.3s	14.0s	265.1s	364.6s	*timed out*
appx3	0.8s	9.7s	13.9s	257.5s	365.3s	*timed out*
appx4	0.8s	9.6s	14.4s	263.3s	369.9s	*timed out*
appx5	0.6s	10.1s	8.9s	293.5s	362.5s	*timed out*

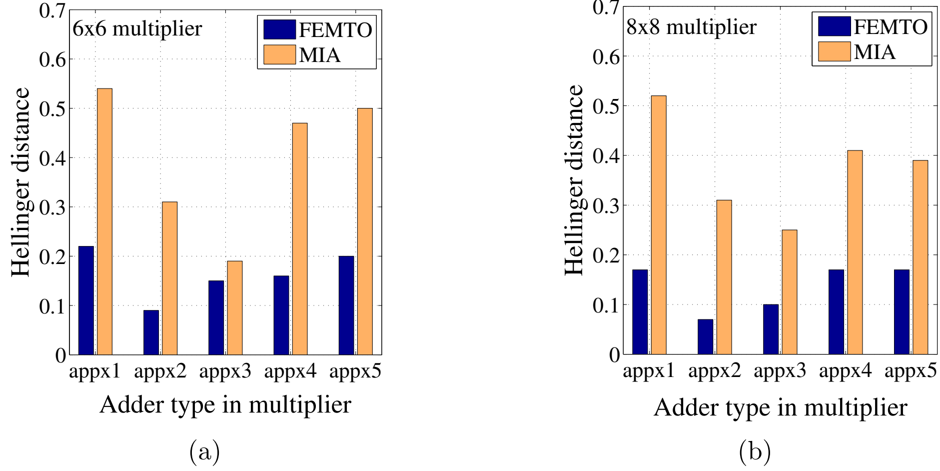


Figure 5.8: Hellinger distance between Monte Carlo-generated PMF, and PMFs obtained by FEMTO and our implementation of MIA for (a) 6-bit×6-bit, and (b) 8-bit×8-bit multipliers.

Clearly, not only is the accuracy of our method higher than MIA, but also the runtime shows excellent improvements. In conclusion, although MIA can quantify the error PMFs, due to the above reasons of inaccuracy and poor runtime, it has not been implemented yet in any multiplier design.

Our approach is thus superior to the existing approaches to quantify the error PMF, and can be incorporated into the multiplier design framework to speed up the performance evaluation process.

5.5 Conclusion

We have proposed a fast algorithm to analytically obtain the error PMF in an unsigned multiplier. The algorithm has been implemented to obtain error PMFs in different types and sizes of approximate multipliers comprising of transistor-level approximate adders. We have validated the results of our algorithm against Monte Carlo simulations, in terms of the various statistics and the error CDFs. The ease with which the PMF can be obtained will help approximate circuit designers to use FEMTO in an optimization framework to evaluate the circuit performance, and design low power approximate systems within a specified error tolerance.

Chapter 6

Generalized Error Analysis in Approximate Circuits

As explained in the previous chapter, a vital ingredient of any methodology based on approximate design is a fast and accurate procedure that can quantify the error injected into a computation by an approximation scheme. While earlier we had quantified the error PMFs in stand-alone unsigned approximate multipliers using the Fourier transform, here we extend the ideas to quantify error PMFs of approximate arithmetic circuits in general, implementing signed operations.

Conventionally, Monte Carlo simulation based approaches have been used to quantify error PMFs in approximate circuits. The look-up table and error composition based approach documented in [66] considers adders as the only arithmetic units in a circuit. While [72] proposes a technique for modeling operation-level error PMFs (in adders and multipliers) similar to our approach in the previous chapter, they do not explore the error propagation through a circuit, and only consider specific cases when the inaccuracy in output is due to the error generated at the operators. The ideas presented in this chapter, are capable of handling both adders (and subtractors) and multipliers, and computes the total error PMF in an approximate circuit considering both, the errors generated at each approximate operation, and propagated through other operations to the primary outputs of the circuit.

The input to our algorithm is the data flow graph of a circuit, represented as a

directed acyclic graph (DAG) whose nodes correspond to arithmetic units that can potentially be approximated, and whose edges indicate the connections between these units, as explained in Section 6.1. Since the most common circuits that are used to build hardware for error-resilient computations are adders [1,29–31,62] and multipliers [32–35], we consider these two operations within the approximate DAGs for our analysis in this chapter. The results can be extended to DAGs whose nodes contain subtractors and dividers as well since they can all be realized using adders and multipliers [36].

We consider signed operations using the two's complement representation, where approximation is introduced in each node by simplifying the logic function of the constituent full adders (FA) [1] of a fixed-width adder or multiplier array within the DAG under study. The Boolean function of the simplified FAs is also an input to our approach, along with the statistics of the operands to each node connected to primary inputs of the DAG. Our approach proceeds as follows:

1. For an FA, we obtain the PMF at the output, and the error in the the output as impulse functions from its truth table that can be computed using its Boolean function (Section 6.2).

2. Next, we use the above PMFs to compute the PMF of error generated at each approximate node of a circuit (Section 6.3). Specifically,

    - For adders, the operation proceeds through an array of approximate and/or exact FAs and the error generated in an approximate adder is the weighted sum of errors generated within the approximate FAs in the array.

    - For multipliers, the operation proceeds by generating partial products and by successively adding each partial product to the partial sum computed so far. Each such addition is performed by an array of approximate and/or exact FAs, and the error generated in the approximate multiplier is the weighted sum of errors within these adder arrays.

    We show that the PMF of the error generated at each node can be expressed as a convolution of a weighted set of error PMFs for individual FAs, and demonstrate how we perform this convolution efficiently in the frequency domain using the Fourier transform [70] on the PMFs.

3. Since the error at the output of a node within a DAG consists of both the generated error (explained in the previous step) and the error that is propagated from its inputs, we next obtain the PMF of such propagated errors, using suitable assumptions regarding the statistics of the inputs, and the errors in the inputs.

    - The error propagation through adders is implemented by adding the errors in its inputs, which translates to convolution of the input error PMFs in the frequency domain. We use the Fourier transform to efficiently solve this convolution problem.
    - The error propagation through multipliers proceeds by adding the products of error in one input with the true value of the other input. We propose a novel technique by using the Mellin transform [37], followed by the Fourier transform, to obtain the PMF of the propagated error in such cases.

    Finally, we combine the PMF of generated and propagated errors at the output of each node of a DAG, and through a topological traversal of the entire DAG, we compute the PMF of error at its primary outputs (Section 6.4).

We present the results of error PMF computation for representative DAGs [2] in Section 6.5 demonstrating the Monte Carlo-like accuracy of our method.

6.1 A DAG Model for an Approximate Circuit

An approximate circuit represented as a DAG can be illustrated in Fig. 6.1, and each node implements a fixed-width signed operation. Since a circuit may be used in a variety of user conditions (inputs vectors, voltage, temperature, etc.), to ensure its robustness, the inputs to an approximate circuit are typically considered as random variables [1,73] whose distribution depends on the type of application the circuit is used for. As a result, the output is also a random variable, and so is the error in the output introduced due to approximations in the circuit. A schematic of both the input and output PMFs are also depicted in Fig. 6.1.

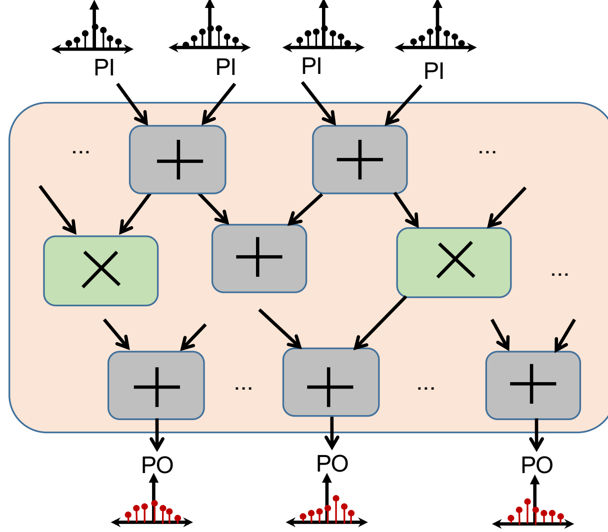


Figure 6.1: Representation of a DAG with approximate operations, producing erroneous outputs which can be characterized by PMFs.

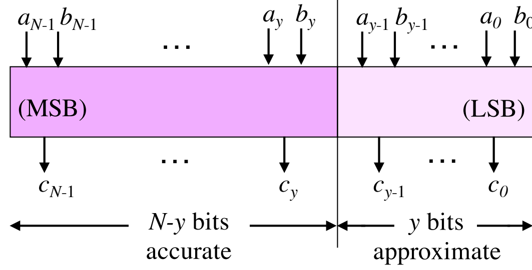


Figure 6.2: Representation of y approximate LSBs, where the subscript, i, refers to the values at the ith bit position.

Let us zoom into a node in Fig. 6.1, and consider a circuit representing the arithmetic operation associated with it, having two N-bit operands, $A(a_{N-1}, \cdots, a_0)$ and $B(b_{N-1}, \cdots, b_0)$, producing an output, $C(c_{N-1}, \cdots, c_0)$, as illustrated in Fig. 6.2. Since an approximate implementation of the hardware unit yields the benefit of using fewer resources [22] than its exact counterpart, typically some of the LSBs can be allowed to be erroneous, as this introduces a limited level of approximation. Hence, the hardware connected to y LSBs, as shown in the example in Fig. 6.2, is approximate, while that connected to the $(N-y)$ most significant bits (MSBs) is accurate. Clearly, the higher

the value of y, the greater are the power savings due to the imprecise hardware, although the error is also higher.

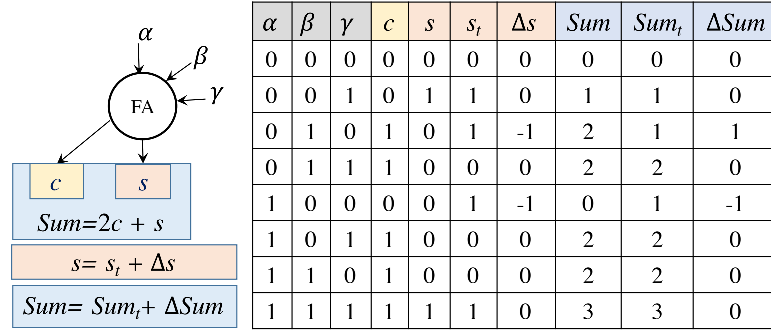
If the error due to the y approximate LSBs is e, then e can range from $-(2^y - 1)$ to $(2^y - 1)$, and its exact value depends on the inputs. Typically inputs are assumed to be random variables following a known distribution (e.g., uniform or normal) as represented schematically in Fig. 6.1. Hence, e, being a function of these inputs, is a random variable as well. Additionally, the output of this node is also a random variable, whether or not this node is approximated, since the inputs are modeled as random variables. Since the operands are all discrete in nature, we refer to their distributions as PMFs, as illustrated in Fig. 6.1. To obtain the error PMF at the primary outputs of the DAG, we also need the PMFs of the actual outputs at each internal node since they are inputs to successive nodes within the DAG. Hence, our proposed approach characterizes the PMF of the output, as well as the error in the output of a DAG whose nodes are candidates for approximation.

6.2 Output Distribution of Full Adders

In principle, the output distribution of any combinational structure can be characterized through its truth table and the statistics of the inputs. However, the size of the truth table increases exponentially with the size of the input space, and such a direct characterization is impractical for a multi-bit adder or a multiplier. Hence, we work with a fundamental unit that can reasonably be characterized – in this case, an FA – and develop the error PMF for a multi-bit adder and multiplier hierarchically. Specifically, for an adder, the error PMF of a single FA is used to obtain the error PMF of the output, and for a multiplier, the error PMF of a single FA is used to obtain the error PMF of each row of FAs that sums the partial products, and finally the error distribution of the entire approximate multiplier. This section explains how we use the input distribution and Boolean function of an FA to obtain its output and output error distribution.

Let us explain our approach with the example of an approximate FA, *appx1* from [1], shown in Fig. 6.3. The single-bit inputs are α, β and γ, and the outputs are c and s, which are combined as the two-bit output, Sum. The error injected by the adder is

denoted by ΔSum. Since we will be analyzing signed operations, for the FA in the MSB position, we also observe the error, Δs, injected in the sign bit, which corresponds to the sum bit of this FA. Hence, the truth table in Fig. 6.3 lists the outputs, s, c, and the combined Sum, along with the exact outputs, s_t and Sum_t. The inputs are modeled as random variables with a known distribution. Since the inputs are binary, we represent their probability of being 1 as p_α, p_β and p_γ, respectively, also known as the signal probability of those inputs. Similarly, $p'_x = 1 - p_x$, $(x = \alpha, \beta, \gamma)$, are the probabilities of α, β and γ, respectively, to be 0. The PMF of the adder output, Sum, combining both the output bits, s and c, and the PMF of the error, ΔSum, in the result, are defined by $f_{Sum}(n)$ and $f_{\Delta Sum}(n)$, respectively. Since we also need to compute the PMF of the error, Δs, in the sum bit, s, if the FA is at the MSB in an adder or a multiplier array, we denote its PMF by $f_{\Delta s}(n)$.



α	β	γ	c	s	s_t	Δs	Sum	Sum_t	ΔSum
0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	1	1	0
0	1	0	1	0	1	-1	2	1	1
0	1	1	1	0	0	0	2	2	0
1	0	0	0	0	1	-1	0	1	-1
1	0	1	1	0	0	0	2	2	0
1	1	0	1	0	0	0	2	2	0
1	1	1	1	1	1	0	3	3	0

Figure 6.3: Full adder (FA) with the associated truth table (*appx1* from [1]).

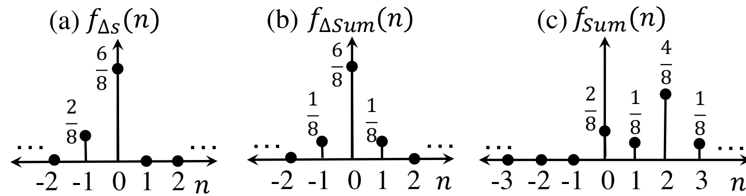


Figure 6.4: Output signal and error distribution for the *appx1* adder from [1].

If the inputs are independent, and represented by an identical uniform distribution (p_α=p_β=p_γ=0.5), then $f_{\Delta s}(n)$, $f_{\Delta Sum}(n)$, and $f_{Sum}(n)$ can trivially be obtained from the truth table, and are depicted in the three plots in Figs. 6.4(a)–(c). For example, the PMF of ΔSum can be computed by observing that it takes the value 0 in six of

eight entries in the truth table, and the values, -1 and 1, in the remaining two entries. When the inputs are equiprobable, this leads to the PMF shown in Fig. 6.4(b). The PMFs in the three figures can equivalently be represented as a weighted sum of discrete Kronecker delta functions as:

$$f_{\Delta s}(n) = \frac{6}{8}\delta(n) + \frac{2}{8}\delta(n+1) \tag{6.1}$$

$$f_{\Delta Sum}(n) = \frac{6}{8}\delta(n) + \frac{1}{8}\delta(n-1) + \frac{1}{8}\delta(n+1) \tag{6.2}$$

$$f_{Sum}(n) = \frac{2}{8}\delta(n) + \frac{1}{8}\delta(n-1) + \frac{4}{8}\delta(n-2) + \frac{1}{8}\delta(n-3) \tag{6.3}$$

The coefficients of the delta functions in Eqs. (6.1), (6.2), and (6.3) are the PMF values of the associated random variables (Δs, ΔSum, and Sum), and are the length of the stems in Fig. 6.4. For a general approximate FA, Δs can range from -1 to 1 since it is the error in the single bit variable, s. Similarly, ΔSum can range from -3 to 3, as the error within the two output bits may be any of -1, 0 or 1, while the value of Sum ranges from 0 to 3. It is to be noted that for the *appx1* adder shown in Fig. 6.3, Δs and ΔSum only range from -1 to 0 and -1 to 1, respectively.

Table 6.1: PMFs associated with of the FA output (Sum) and output errors (Δs and ΔSum).

n	$f_{\Delta s}(n) = x_n$	$f_{\Delta Sum}(n) = e_n$	$f_{Sum}(n) = p_n$
-1	$p'_\alpha p_\beta p'_\gamma + p_\alpha p'_\beta p'_\gamma$	$p_\alpha p'_\beta p'_\gamma$	$--$
0	$1 - p'_\alpha p_\beta p'_\gamma - p_\alpha p'_\beta p'_\gamma$	$1 - p_\alpha p'_\beta p'_\gamma - p'_\alpha p_\beta p'_\gamma$	$p'_\alpha p'_\beta p'_\gamma + p_\alpha p'_\beta p'_\gamma$
1	0	$p'_\alpha p_\beta p'_\gamma$	$p'_\alpha p'_\beta p_\gamma$
2	$--$	0	$p'_\alpha p_\beta + p_\alpha p'_\beta p_\gamma + p_\alpha p_\beta p'_\gamma$
3	$--$	0	$p_\alpha p_\beta p_\gamma$

When the inputs to the approximate FA are not equiprobable, the coefficient of $\delta(n-v)$ in the PMFs of Δs, ΔSum, and Sum, is the probability of the corresponding random variable to be v, where $v \in [-1, 1]$, $v \in [-3, 3]$, and $v \in [0, 3]$, for Δs, ΔSum, and Sum, respectively. Hence, assuming the inputs to be independent, the coefficients can, in fact, be expressed as functions of p_α, p_β and p_γ as shown in Table 6.1. Hence, the PMF of Δs, ΔSum, and Sum can, in general, be expressed as a sum of Kronecker

delta functions as:

$$f_{\Delta s}(n) = \sum_{v=-1}^{1} x_v \delta(n-v) \quad (6.4)$$

$$f_{\Delta Sum}(n) = \sum_{v=-3}^{3} e_v \delta(n-v) \quad (6.5)$$

$$f_{Sum}(n) = \sum_{v=0}^{3} p_v \delta(n-v) \quad (6.6)$$

where x_v, e_v, and p_v are, respectively, the probabilities of Δs, ΔSum, and Sum to be v, and expressed as functions of p_α, p_β, and p_γ similar to the terms in Table 6.1. The probabilities, p_α, p_β, and p_γ, can be computed from the knowledge of the input distribution, to obtain the PMFs in Eqs. (6.4) to (6.6).

6.3 Distribution of Generated Error in DAG Nodes

Let us consider an approximate node in a DAG, with two N-bit operands, $A(a_{N-1}a_{N-2}\cdots a_0)$ and $B(b_{N-1}b_{N-2}\cdots b_0)$, producing an output, $C(c_{N-1}c_{N-2}\cdots c_0)$ as shown in Fig. 6.5. We assume the inputs to be error-free (A_t and B_t being the true values of inputs without any error), and the output, C, to comprise of only the error generated at this approximate node denoted by ΔR. The true output, C_t, would have been the output if this node was exact. In this section we explain the methodology to obtain the PMF of errors generated at the individual nodes of a DAG, i.e., PMF of ΔR.

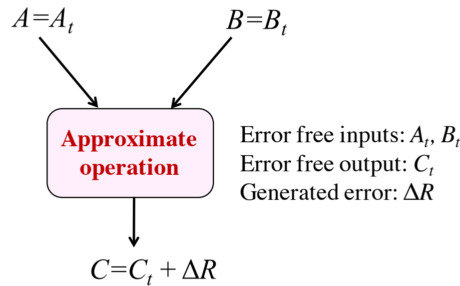


Figure 6.5: Approximate node of a DAG with exact inputs and erroneous output, where the error, ΔR, is generated at this node.

There are quite a few applications (e.g., image compression and classification, audio filtering, matrix multiplication etc.) that are amenable to approximation, which can be

implemented using add/subtract/multiply operations. Hence, in our work, we consider these three operators as candidates for approximate nodes within a DAG.

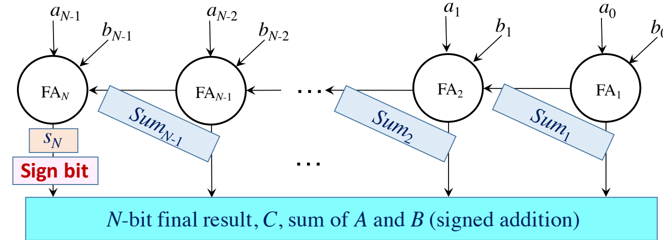


Figure 6.6: A signed adder where some of the LSB FAs can be approximated to introduce approximation in the circuit.

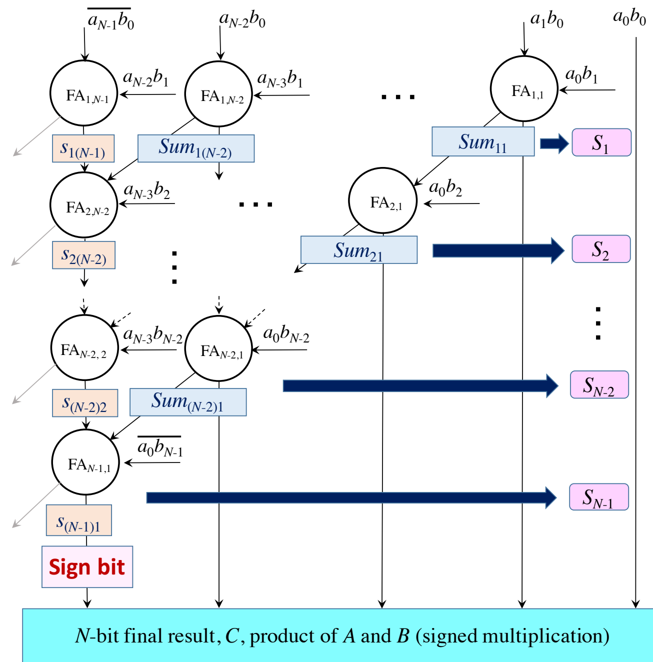


Figure 6.7: A signed array multiplier where some of the LSB FAs can be approximated to introduce approximation in the circuit.

We consider two's complement signed operations, and for the adder (or subtractor) we assume the sum bit output of the MSB FA to be the sign bit as shown in Fig. 6.6. For multipliers, we assume the lower triangle of the Modified Baugh-Wooley two's-complement multiplication array structure [36] as shown in Fig. 6.7, so that N LSBs

correspond to the actual result of the $N \times N$ multiplier, that goes into the subsequent stages. The sum bit output of the FA in the penultimate row and the MSB column of the array is the sign bit as highlighted in Fig. 6.7.

Before proceeding further, let us comment on the input data distribution, and our assumptions regarding the correlation between the random variables associated with the outputs and errors of each FA in the adder or the multiplier array. Although we assume the inputs, A and B, to be independent random variables, their distribution is a user-specified input (and can be any arbitrary distribution) from which the signal probability, p_{a_i} and p_{b_i}, of each input bit, a_i and b_i, respectively, can be inferred, $i \in 0, \cdots, N-1$. Additionally, we consider the correlation between the sum and carry bits of any FA and the error introduced in them, by combining them into a two-bit output, Sum, and the corresponding error, ΔSum, both expressed in decimal, with their PMFs characterized by similar methods as Table 6.1. This technique captures the most important correlation which is the interdependence of the two output bits of any adder within the multiplier or the adder array. Although we ignore the correlations between the outputs of different FAs within the adder and multiplier array by considering the output of the FAs in them to be independent random variables, this assumption does not affect the quality of our results since correlations due to reconvergent fanout tend to be diluted as the logic depth of the reconvergent fanout paths increases.

6.3.1 PMF of Generated Errors in Approximate Adders

For an adder, the FAs in the array are each indexed as FA$_i$, where i corresponds to the position of the FA in the adder, starting from the LSB side, so that i ranges from 1 to N for an N-bit signed adder as shown in Fig. 6.6. The two single bit outputs, s_i and c_i, from the ith adder, FA$_i$, are combined to a two-bit output, Sum_i, and modeled as the random variable, $Sum_i = Sum_{i,t} + \Delta Sum_i$, where $Sum_{i,t}$ is the true sum (corresponding to the output of an exact FA), and ΔSum_i is the error due to the approximate addition, similar to the example of the FA in Fig. 6.3. Although the structure shown in the figure corresponds to a Ripple Carry Adder, this concept can be extended to any adder architecture.

Finding the error PMF for the adder involves two steps:

- **Step 1**: determining the input probabilities for all inputs of each individual FA_i in the adder, and using the approach in Section 6.2 to compute the PMF of ΔSum_i, and

- **Step 2**: finding the error PMF of the entire adder, i.e., the PMF of the weighted sum of the ΔSum_i variables along with that of Δs_N which is the error in the sum bit of the MSB (N^{th}) FA and is also the sign bit in the output.

Let us now explain each step in further detail:

Step 1: The first step involves probability propagation within a Boolean network, and we use established techniques for this purpose [69]. For example, if the two inputs, α and β, to a two-input AND gate have signal probabilities of p_α and p_β, respectively, then the probability of the output of the AND gate to be 1 is $p_\alpha p_\beta$. Similarly, we can compute the probability of each signal within the adder array in Fig. 6.6 to be 1 or 0, using on the Boolean function of each FA. Based on this, we obtain the PMF of ΔSum_i, denoted by $f_{\Delta Sum_i}(n)$, as a sum of delta functions similar to Eq. (6.5). We also obtain the PMF of Δs_N, which is the error in the sum bit of the MSB FA, and denote it by $f_{\Delta s_N}(n)$ similar to Eq. (6.4).

Step 2: The generated error in the adder output, C (depicted in Fig. 6.6), is represented by ΔR, and is the weighted sum of the errors, ΔSum_i, over LSB $N-1$ FAs, and the error, Δs_N, in the sum bit of the MSB FA. Since we implement two's complement operation, Δs_N actually represents the error in the sign bit, and hence, has a negative weight associated with itself inside ΔR. A simple analysis yields:

$$\Delta R = \left(\sum_{i=1}^{N-1} 2^{i-1} \Delta Sum_i \right) - 2^{N-1} \Delta s_N \tag{6.7}$$

Since we consider the ΔSum_i, $i = 1, \cdots, N-1$ random variables, and Δs_N to be independent, we utilize the fact that PMF of sum of independent random variables equals the convolution of the PMF of those random variables. Hence, the PMF, $f_{\Delta R}(n)$, of the generated error, ΔR, at the adder node is obtained as:

$$f_{\Delta R}(n) = \left(\bigotimes_{i=1}^{N-1} f_{2^{i-1}\Delta Sum_i}(n) \right) \bigotimes f_{-2^{N-1}\Delta s_N}(n) \tag{6.8}$$

where $\bigotimes$ is the convolution operator. If the absolute value of the largest output error of FA_i is M, using Eqs. (6.4) and (6.5), we obtain the constituent PMFs in Eq. (6.8)

as:

$$f_{2^{i-1}\Delta Sum_i}(n) = \sum_{v=-M}^{M} e_v^{(i)} \delta(n - 2^{i-1}v) \tag{6.9}$$

$$f_{-2^{N-1}\Delta s_N}(n) = \sum_{v=-1}^{1} x_v^{(N)} \delta(n + 2^{N-1}v) \tag{6.10}$$

where $e_v^{(i)}$ and $x_v^{(N)}$ are the probabilities of ΔSum_i and Δs_N to be v, respectively, and the superscripts, (i) and (N), indicate the position in the error-producing FAs in the adder array. Since ΔSum_i (Δs_N) is shifted by 2^{i-1} (2^{N-1}) in Eq. (6.9) (Eq. (6.10)), the locations of the impulses in the error distributions are scaled by this factor as well, as shown inside the Kronecker delta functions.

We implement the following techniques to solve the convolution problem in Eq. (6.8) to obtain the PMF of ΔR:

1. Use the Z-transform [70] to convert the convolution into a friendlier multiplication in the frequency domain, yielding a polynomial in z. This polynomial can have an exponential number of terms, and special techniques are required to manage the cost of working in the transform domain.

2. Use the inverse fast Fourier transform (IFFT) [70] to infer the PMF of ΔR from the polynomial obtained in the previous step.

Next, we explain each of these techniques in detail.

Representing the convolution using the Z-transform

Based on the principles of transform calculus, the Z-transform of a convolution of multiple functions in the original domain is equivalent to the product of the Z-transforms of those individual functions in the transform domain. Hence, we can represent $F_{\Delta R}(z)$, the Z-transform of $f_{\Delta R}(n)$ in Eq. (6.8), as:

$$F_{\Delta R}(z) = \left(\prod_{i=1}^{N-1} F_{2^{i-1}\Delta Sum_i}(z) \right) F_{-2^{N-1}\Delta s_N}(z) \tag{6.11}$$

where $F_{2^{i-1}\Delta Sum_i}(z)$ and $F_{-2^{N-1}\Delta s_N}(z)$, are the Z-transform of the PMFs, $f_{2^{i-1}\Delta Sum_i}(n)$ and $f_{-2^{N-1}\Delta s}(n)$, as defined in Eqs. (6.9) and (6.10), respectively. Applying the Z-transform to both sides of these two equations, we obtain:

$$F_{2^{i-1}\Delta Sum_i}(z) = \sum_{v=-M}^{M} e_v^{(i)} z^{-2^{i-1}v} \tag{6.12}$$

$$F_{-2^{N-1}\Delta s_N}(z) = \sum_{v=-1}^{1} x_v^{(N)} z^{2^{N-1}v} \tag{6.13}$$

Substituting $F_{2^{i-1}\Delta Sum_i}(z)$ and $F_{-2^{N-1}\Delta s_N}(z)$ from Eqs. (6.12) and (6.13), respectively, in Eq. (6.11), we can rewrite $F_{\Delta R}(z)$ as:

$$F_{\Delta R}(z) = \prod_{i=1}^{N-1} \left(\sum_{v=-M}^{M} e_v^{(i)} z^{-2^{i-1}v} \right) \left(\sum_{v=-1}^{1} x_v^{(N)} z^{2^{N-1}v} \right) \tag{6.14}$$

$$= z^{-\mathcal{E}} \sum_{i=0}^{2\mathcal{E}} a_i z^i \tag{6.15}$$

$$= z^{-\mathcal{E}} \Phi(z) \tag{6.16}$$

where ΔR ranges from $-\mathcal{E}$ to $\mathcal{E}$, with $\mathcal{E} = 2^{N-1}(M+1) - M$, and the a_is are the coefficients of the polynomial in z, denoted by $\Phi(z)$ in Eq. (6.16). The derivation of $\mathcal{E}$ has been deferred to Appendix B for better readability. Performing the inverse Z-transform of Eq. (6.14), we obtain,

$$f_{\Delta R}(n) = \sum_{i=0}^{2\mathcal{E}} a_i \delta(n + i - \mathcal{E}) \tag{6.17}$$

Hence, a_i is the probability of the generated error, ΔR, in the adder output to be $-(i - \mathcal{E})$. Thus finding the PMF of the error reduces to the problem of finding the coefficients, a_i, in $\Phi(z) \left(= \sum_{i=0}^{2\mathcal{E}} a^i z^i \right)$, which is a polynomial of degree $2\mathcal{E}$ with non-negative coefficients.

While this scheme presents a clear picture of our computation scheme, the cost of a direct implementation of this idea is prohibitive. The most expensive step is the determination of the coefficients, a_i, by multiplying the terms in Eq. (6.14). Therefore, we develop an efficient scheme for finding the coefficients, a_i as explained next.

Using the IFFT to infer $f_{\Delta R}(n)$ from $\Phi(z)$ and $F_{\Delta R}(z)$

So far we have worked in the Z-transform domain to formulate the error PMF, $F_{\Delta R}(z)$. Let us now consider discrete Fourier domain to determine the coefficients, a_i, in $\Phi(z)$

from Eq. (6.16), by using their Fourier-transformed values followed by performing inverse fast Fourier transform (IFFT). This interchange of domains is possible since, by definition, the Z-transform is equivalent to the discrete time Fourier transform (DTFT) when the magnitude of $|z| = 1$ [70].

We begin by observing that the DTFT of the sequence, $\{a_0, a_1, \cdots, a_{2\mathcal{E}}\}$, is given by the Fourier coefficients,

$$\mathcal{A}_k = \sum_{i=0}^{2\mathcal{E}} a_i \exp\left(-\mathbf{j}\frac{2\pi ik}{2\mathcal{E}+1}\right) = \sum_{i=0}^{2\mathcal{E}} a_i z_k^i \quad (6.18)$$

where $z_k = \exp\left(-\mathbf{j}\frac{2\pi k}{2\mathcal{E}+1}\right)$ and $\mathbf{j} = \sqrt{-1}$. It is interesting to note that the values of z_k are the reciprocal of the $(2\mathcal{E}+1)^{\text{th}}$ complex roots of unity.

Therefore, if we take $\Phi(z)$ in Eq. (6.16) and substitute $z = z_k$, for the reciprocal of each of the $(2\mathcal{E}+1)^{\text{th}}$ complex roots of unity, we obtain the Fourier coefficient, $\mathcal{A}_k$. In other words,

$$\begin{aligned}\mathcal{A}_k &= \Phi(z_k) = z_k^{\mathcal{E}} F_{\Delta R}(z_k) \\ &= z_k^{\mathcal{E}} \prod_{i=1}^{N-1}\left(\sum_{v=-M}^{M} e_v^{(i)} z_k^{-2^{i-1}v}\right)\left(\sum_{v=-1}^{1} x_v^{(N)} z_k^{2^{N-1}v}\right)\end{aligned} \quad (6.19)$$

This provides us with the discrete Fourier coefficients of the sequence of a_is, which are then obtained by performing inverse discrete time Fourier transform (IDFT) of the $\mathcal{A}_k$ values as:

$$a_i = \frac{1}{2\mathcal{E}+1}\sum_{k=0}^{2\mathcal{E}} \mathcal{A}_k exp\left(\mathbf{j}\frac{2\pi ik}{2\mathcal{E}+1}\right) \quad (6.20)$$

To compute the IDFT in Eq. (6.20) efficiently, we use the IFFT to obtain the values of the a_is. As observed in Eq. (6.17), obtaining the a_is directly provides the PMF of the generated error, ΔR, in the adder output.

We summarize the steps to obtain the PMF of an approximate adder (or subtractor) node in Algorithm 1.

Algorithm 1 Pseudo-code to obtain the error PMF of an approximate adder (or subtractor) node within a DAG.

Input: Adder architecture; truth tables of the FAs.

Input: Signal probabilities of each input bit of the adder.

Output: Error PMF, $f_{\Delta R}(n)$, of the adder output.

1: Compute the internal signal probabilities, $e_{v_1}^{(i)}$, for $i \in [1, N-1]$, $v_1 \in [-M, M]$, and $x_{v_2}^{(N)}$, $v_2 \in [-1, 1]$, from the bitwise input signal probabilities, similar to Table 6.1
2: **for** each i from 1 to $N-1$ **do**
3: Compute $f_{2^{i-1}\Delta Sum_i}(n)$ from Eq. (6.9)
4: **end for**
5: Compute $f_{-2^{N-1}\Delta s_N}(n)$ from Eq. (6.10)
6: Formulate $f_{\Delta R}(n)$ as a convolution problem in Eq. (6.8)
7: To solve the convolution, perform Z-transform on $f_{\Delta R}(n)$ to formulate $F_{\Delta R}(z) = z^{-\mathcal{E}}\Phi(z)$ as Eq. (6.16), where $\mathcal{E}$ is derived in Appendix B
8: **for** each k from 0 to $2\mathcal{E}$ **do**
9: Compute $z_k = \exp\left(-\mathbf{j}\frac{2\pi k}{2\mathcal{E}+1}\right)$, $\mathbf{j} = \sqrt{-1}$
10: Evaluate $\mathcal{A}_k = \Phi(z_k)$ as shown in Eq. (6.19)
11: **end for**
12: IFFT: Using the $\mathcal{A}_k$s, obtain a_i, $i \in [0, 2\mathcal{E}]$ from Eq. (6.20)
13: Obtain $f_{\Delta R}(n) = \sum_{i=0}^{2\mathcal{E}} a_i \delta(n + i - \mathcal{E})$ as Eq. (6.17)

6.3.2 PMF of Generated Errors in Approximate Multipliers

For a multiplier, the FAs are each indexed as FA_{ij} in the array, where i corresponds to the row number, starting from the top ($i \in \{1, \cdots, N-1\}$), and j corresponds to the position of an FA in a particular row, starting from the LSB ($j \in \{1, \cdots, N-i\}$), as shown in Fig. 6.7. The output of FA_{ij} is modeled as the random variable, $Sum_{ij} = Sum_{ij,t} + \Delta Sum_{ij}$, where $Sum_{ij,t}$ is the true sum (corresponding to the output of an exact FA), and ΔSum_{ij} is the error due to the approximate addition, similar to the example in Fig. 6.3.

Similar to the error PMF of an adder, finding the error PMF for the multiplier array involves three steps, while the extra step is due to the two-dimensional structure of the multiplier as opposed to the one-dimensional one for the adder:

- **Step 1**: determining the input probabilities for all inputs of each individual FA_{ij}, and using the approach in Section 6.2 to compute the PMF of ΔSum_{ij},
- **Step 2**: finding the PMF of the error, ΔS_i, introduced by the ith row of the multiplier

array, and

- **Step 3**: finding the PMF of the entire multiplier, i.e., the PMF of the sum of the ΔS_i variables over all rows, i.

Let us now explain each step in further detail:

Step 1: The first step is similar to that of the generated error PMF computation of adder as explained in the previous subsection, to obtain the PMF of ΔSum_{ij} and $\Delta s_{i(N-i)}$ denoted by $f_{\Delta Sum_{ij}}(n)$ and $f_{\Delta s_{i(N-i)}}(n)$, respectively, where $\Delta s_{i(N-i)}$ is the error in the sign bit in the ith row.

Step 2: Next, we determine the error in the partial product accumulation, ΔS_i, in the ith row, which is the total error resulting from an array of $N-i$ FAs, as depicted in Fig. 6.7 for any general N, and $i \in \{1, \cdots, N-1\}$. For each row, $i \in \{1, \cdots, N-1\}$, a simple analysis yields:

$$\Delta S_i = \sum_{j=1}^{N-i-1} 2^{i+j-1} \Delta Sum_{ij} - 2^{N-1} \Delta s_{i(N-i)} \tag{6.21}$$

where the negative weight, -2^{N-1}, is associated with the sum bit, $s_{i(N-i)}$, in the ith row, which actually corresponds to the the sign bit in the multiplier result.

Since we consider the ΔSum_{ij} and $\Delta s_{i(N-i)}$ random variables to be independent, we can utilize the fact that the PMF of sum of independent random variables equals the convolution of the PMF of those random variables. Hence, the PMF of ΔS_i can be expressed as:

$$f_{\Delta S_i}(n) = \bigotimes_{j=1}^{N-i-1} f_{2^{i+j-1}\Delta Sum_{ij}}(n) \bigotimes f_{-2^{N-1}\Delta s_{i(N-i)}}(n) \tag{6.22}$$

where $f_{2^{i+j-1}\Delta Sum_{ij}}(n)$ and $f_{-2^{N-1}\Delta s_{i(N-i)}}(n)$ are the PMFs of the random variable, $2^{i+j-1}\Delta Sum_{ij}$ and $-2^{N-1}\Delta s_{i(N-i)}$, respectively.

If the absolute value of the largest output error of FA_{ij} is M, then using Eqs. (6.4) and (6.5), we obtain the constituent PMFs in Eq. (6.22) similar to Eqs. (6.9) and (6.10)

as:

$$f_{2^{i+j-1}\Delta Sum_{ij}}(n) = \sum_{v=-M}^{M} e_v^{(ij)} \delta(n - 2^{i+j-1}v) \tag{6.23}$$

$$f_{-2^{N-1}\Delta s_{i(N-i)}}(n) = \sum_{v=-1}^{1} x_v^{(i(N-i))} \delta(n + 2^{N-1}v) \tag{6.24}$$

where $e_v^{(ij)}$ and $x_v^{(i(N-i))}$ are the probabilities of ΔSum_{ij} and $\Delta s_{i(N-i)}$ to be v, respectively, and the superscripts, (ij) and $(i(N-i))$, indicate the position in the error-producing FAs in the multiplier array.

Step 3: The generated error, ΔR, in the multiplier output is simply the sum of the errors, ΔS_i, over all $N-1$ rows. Assuming the ΔS_i random variables to be independent, we obtain the error PMF, $f_{\Delta R}(n)$, by convolving the $f_{\Delta S_i}(n)$ PMFs from Eq. (6.22) as:

$$\begin{aligned} f_{\Delta R}(n) &= \bigotimes_{i=1}^{N-1} f_{\Delta S_i}(n) \\ &= \bigotimes_{i=1}^{N-1} \left(\bigotimes_{j=1}^{N-i-1} f_{2^{i+j-1}\Delta Sum_{ij}}(n) \right) \bigotimes f_{-2^{N-1}\Delta s_{i(N-i)}}(n) \end{aligned} \tag{6.25}$$

The techniques to solve the convolution problem in Eq. (6.25) are similar to those for Eq. (6.8) for adders as explained in Sections 6.3.1 and 6.3.1. Hence, we first represent $F_{\Delta R}(z)$, the Z-transform of $f_{\Delta R}(n)$ in Eq. (6.25), as:

$$F_{\Delta R}(z) = \prod_{i=1}^{N-1} \left(\prod_{j=1}^{N-i-1} F_{2^{i+j-1}\Delta Sum_{ij}}(z) \right) F_{-2^{N-1}\Delta s_{i(N-i)}}(z) \tag{6.26}$$

where $F_{2^{i+j-1}\Delta Sum_{ij}}(z)$ and $F_{-2^{N-1}\Delta s_{i(N-i)}}(z)$ are the Z-transforms of the PMFs, $f_{2^{i+j-1}\Delta Sum_{ij}}(n)$ and $f_{-2^{N-1}\Delta s_{i(N-i)}}(n)$, respectively, outlined in Eqs. (6.23) and (6.24). These two Z-transforms are similar to Eqs. (6.12) and (6.13), respectively, and can be formulated as:

$$F_{2^{i+j-1}\Delta Sum_{ij}}(z) = \sum_{v=-M}^{M} e_v^{(ij)} z^{-2^{i+j-1}v} \tag{6.27}$$

$$F_{-2^{N-1}\Delta s_{i(N-i)}}(z) = \sum_{v=-1}^{1} x_v^{(i(N-i))} z^{2^{N-1}v} \tag{6.28}$$

Substituting $F_{2^{i+j-1}\Delta Sum_{ij}}(z)$ and $F_{-2^{N-1}\Delta s_{i(N-i)}}(z)$ from Eqs. (6.27) and (6.28), respectively, in Eq. (6.26), we can rewrite $F_{\Delta R}(z)$ as:

$$F_{\Delta R}(z) = \prod_{i=1}^{N-1}\prod_{j=1}^{N-i-1}\left(\sum_{v=-M}^{M} e_v^{(ij)} z^{-2^{i+j-1}v}\right)\left(\sum_{v=-1}^{1} x_v^{(i(N-i))} z^{2^{N-1}v}\right) \tag{6.29}$$

$$= z^{-\mathcal{F}} \sum_{i=0}^{2\mathcal{F}} m_i z^i \tag{6.30}$$

$$= z^{-\mathcal{F}} \Theta(z) \tag{6.31}$$

where ΔR ranges from $-\mathcal{F}$ to $\mathcal{F}$, with $\mathcal{F} = 2^{N-1}(MN - 3M + N - 1) + 2M$, and the m_is are the coefficients of the polynomial in z, denoted by $\Theta(z)$ in Eq. (6.31). The derivation of $\mathcal{F}$ has been deferred to Appendix B for better readability. Performing the inverse Z-transform of Eq. (6.29),

$$f_{\Delta R}(n) = \sum_{i=0}^{2\mathcal{F}} m_i \delta(n + i - \mathcal{F}) \tag{6.32}$$

Hence, m_i is the probability of the generated error, ΔR, in the multiplier output to be $-(i - \mathcal{F})$, and can be computed using the ideas presented in Section 6.3.1.

We summarize the steps to obtain the PMF of an approximate multiplier node in Algorithm 2.

Algorithm 2 Pseudo-code to obtain the error PMF of an approximate multiplier node within a DAG.

Input: Multiplier architecture; truth tables of the FAs.

Input: Signal probabilities of each input bit of the multiplier.

Output: Error PMF, $f_{\Delta R}(n)$, of the multiplier output.

1: Compute the internal signal probabilities, $e_{v_1}^{(ij)}$ and $x_{v_2}^{(i(N-i))}$, where $i \in [1, N-1]$, $j \in [1, N-i-1]$, $v_1 \in [-M, M]$, and $v_2 \in [-1, 1]$

2: **for** each i from 1 to $N-1$ **do**

3:     **for** each j from 1 to $N-i-1$ **do**

4:         Compute $f_{2^{i+j-1}\Delta Sum_{ij}}(n)$ from Eq. (6.23)

5:     **end for**

6:     Compute $f_{-2^{N-1}\Delta s_{i(N-i)}}(n)$ from Eq. (6.24)

7: **end for**

8: Formulate $f_{\Delta R}(n)$ as the convolution problem in Eq. (6.25)

9: To solve the convolution, perform Z-transform on $f_{\Delta R}(n)$ to formulate $F_{\Delta R}(z) = z^{-\mathcal{F}}\Theta(z)$ as Eq. (6.31), where $\mathcal{F}$ is derived in Appendix B

10: **for** each k from 0 to $2\mathcal{F}$ **do**

11: Compute $z_k = \exp\left(-\mathbf{j}\frac{2\pi k}{2\mathcal{F}+1}\right)$, $\mathbf{j} = \sqrt{-1}$

12: Compute $\mathcal{M}_k = \Theta(z_k)$ similar to the $\mathcal{A}_k$s in Eq. (6.19)

13: **end for**

14: IFFT: Using the $\mathcal{M}_k$s, obtain m_i, $i\in[0, 2\mathcal{F}]$ similar calculating the a_is from $\mathcal{A}_k$s in Eq. (6.20)

15: Obtain $f_{\Delta R}(n) = \sum_{i=0}^{2\mathcal{F}} m_i\delta(n + i - \mathcal{F})$ as Eq. (6.32)

6.4 Distribution of Output Error in a DAG

The previous section summarized the computation of the error PMFs for individual adder, subtractor, and multiplier nodes of a DAG. In this section, we propose techniques to compute the error PMF for an entire DAG in general, comprising these nodes. In other words, in this section, we compose the block PMFs, as computed in Section 6.3, to determine the PMF for a DAG consisting of these blocks.

The error in the output, C, of any approximate node within a DAG comprises (a) the error generated at this node, and (b) the error propagated through the operands, A and B as illustrated in Fig. 6.8 for approximate adders and multipliers. We have already discussed the computation of the PMF of generated error in DAG nodes in Section 6.3. Here we present a methodology to obtain the PMF of the error, ΔC, at the output of a node combining the errors propagated through its inputs, A and B, and the errors, ΔR, generated at this node, as shown in Fig. 6.8.

To obtain the PMF of the total error, ΔC, propagated to the output of a node in a DAG, we need the PMF of the true value of the inputs, A_t and B_t, and the input errors, ΔA and ΔB. These are discrete random variables, and hence, their PMFs can

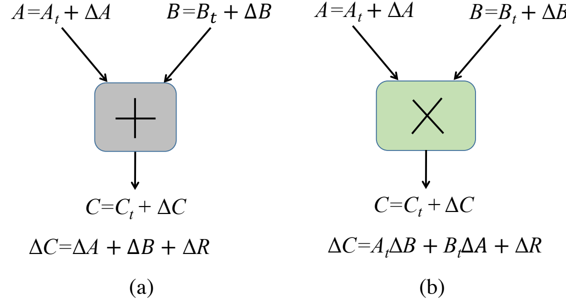


Figure 6.8: (a) Approximate adder, and (b) multiplier with inputs, A and B, both of which can be erroneous, producing approximate output, C.

be modeled as the sum of delta functions as:

$$f_{A_t}(n)=\sum_{-E}^{E} p_k^A \delta(n-k);\ f_{B_t}(n)=\sum_{-E}^{E} p_k^B \delta(n-k) \tag{6.33}$$

$$f_{\Delta A}(n)=\sum_{-F}^{F} p_k^{\Delta A} \delta(n-k);\ f_{\Delta B}(n)=\sum_{-F}^{F} p_k^{\Delta B} \delta(n-k) \tag{6.34}$$

where the inputs (input errors) range from $-E$ to E ($-F$ to F), and p_k^X represents the probability of the variable, X, to be k, where $X \in \{A_t, B_t, \Delta A, \Delta B\}$. It is to be noted that we drop the subscript, t, from $p_k^{A_t}$ and $p_k^{B_t}$ in Eq. (6.33) and simply use p_k^A and p_k^B, instead, for ease of notation. The expression for ΔC is shown in Figs. 6.8(a) and (b) for both adder and multiplier, and its PMF can be obtained using the PMFs listed in Eqs. (6.33) and (6.34), and the methods explained in the next two subsections.

6.4.1 Error Propagation through Adders

The total error, ΔC, at the output of an approximate adder is represented by $\Delta A + \Delta B + \Delta R$ as illustrated in Fig. 6.8(a). The generated error, ΔR, and both the input errors, ΔA and ΔB, can be assumed to be independent random variables. Hence, the PMF of ΔC can be formulated as:

$$f_{\Delta C}(n) = f_{\Delta A}(n) \bigotimes f_{\Delta B}(n) \bigotimes f_{\Delta R}(n) \tag{6.35}$$

where $f_{\Delta A}(n)$ and $f_{\Delta B}(n)$ are defined in Eq. (6.34), and $f_{\Delta R}(n)$ is obtained from Eq. (6.17) in Section 6.3.1. This convolution problem can be solved by using the concepts explained in Section 6.3.1, specifically, by first applying Z-transform on both

sides of Eq. (6.35), followed by evaluating the resulting polynomial at the reciprocal of $(2\mathcal{E}_C + 1)^{\text{th}}$ roots of unity, where $\mathcal{E}_C$ is the maximum value of ΔC from Eq. (6.35), and then performing an IFFT on those evaluations.

Similarly, PMF of C_t can be obtained by convolving the PMFs of A_t and B_t defined in Eq. (6.33). This is the PMF of the output of the adder node if it had implemented an exact operation, and may be used to characterize error PMFs in successive stages within the DAG.

6.4.2 Error Propagation through Multipliers

While ΔC for an adder is given by a simple sum of three random variables, ΔC for a multiplier involves the sum of product of random variables ($A_t\Delta B$, $B_t\Delta A$, and $\Delta A\Delta B$). We will employ the Mellin transform [37] to first compute the PMF of the product of random variables followed by established techniques using the Fourier transform to compute the PMF of their sum. Hence, let us first provide a few key concepts of the Mellin transform that we will use to compute the PMF of product or random variables.

Definition. *The Mellin transform is an integral transform that may be regarded as the multiplicative version of the Fourier transform [37]. It is defined only for functions defined on the positive real axis. For two positive random variables, X and Y, with PMFs, $f_X(n)$ and $f_Y(n)$, respectively, the following statements hold true regarding their Mellin transforms:*

(i) The Mellin transform of $f_X(n)$ and $f_Y(n)$, are represented by $F_X(s)$ and $F_Y(s)$, respectively, as:

$$F_X(s) = \mathcal{M}\left[f_X(n); s\right] = \int_0^\infty n^{s-1} f_X(n) dn$$
$$F_Y(s) = \mathcal{M}\left[f_Y(n); s\right] = \int_0^\infty n^{s-1} f_Y(n) dn$$

where s is a complex variable that corresponds to the transformed domain.

(ii) Product Rule: If X and Y are independent random variables in addition to being positive, then

$$\begin{aligned} F_{XY}(s) &= \mathcal{M}\left[f_{XY}(n); s\right] = \mathcal{M}\left[f_X(n); s\right] \mathcal{M}\left[f_Y(n); s\right] \\ &= F_X(s) F_Y(s) \end{aligned} \tag{6.36}$$

(iii) If $f_X(n)$ is represented by a sum of delta functions, $\sum_{k=1}^{N} x_k\delta(n-k)$, its Mellin transform is obtained as:

$$\mathcal{M}\left[\sum_{k=1}^{N} x_k\delta(n-k);\ s\right] = \sum_{k=1}^{N} x_k k^{s-1} \tag{6.37}$$

Now, in the total error, ΔC, propagated from the inputs of an approximate multiplier to its output (illustrated in Fig. 6.8(b)), we can ignore $\Delta A\Delta B$, since it is negligible compared to the other terms, so that ΔC can be reformulated as:

$$\Delta C \approx A_t\Delta B + B_t\Delta A + \Delta R \tag{6.38}$$

If A_t, ΔB, B_t, and ΔA are all independent random variables, the PMF of ΔC can be formulated as:

$$f_{\Delta C}(n) = f_{A_t\Delta B}(n) \bigotimes f_{B_t\Delta A}(n) \bigotimes f_{\Delta R}(n) \tag{6.39}$$

where $f_{\Delta R}(n)$ is obtained from Eq. (6.32) in Section 6.3.2. To solve the convolution in Eq. (6.39), we need the PMFs of $A_t\Delta B$ and $B_t\Delta A$. This can achieved by using the *Product Rule* of the Mellin transform of PMFs, as explained above. However, the Mellin transform is only valid for positive random variables, while both A_t and ΔB (or B_t and ΔA) can be either positive or non-positive (negative or zero) since we implement signed operations. Hence we use the positive and non-positive parts of X, $X \in \{A_t, B_t, \Delta A, \Delta B\}$, and apply the Mellin transform separately to handle the non-positive cases.

For example, to find the PMF of $C_1 = A_t\Delta B$, we compute four different PMFs for C_1, when (a) $A_t \geq 0$, $\Delta B \geq 0$, (b) $A_t \geq 0$, $\Delta B < 0$, (c) $A_t < 0$, $\Delta B \geq 0$, and (d) $A_t < 0$, ΔB<0, as sum of delta functions (impulse train), illustrated in Fig. 6.9(a)-(d). Clearly, the PMF of $C_1 = A_t\Delta B$ is the sum of all four impulse trains, and is depicted by $f_{C_1}(n)$ in Fig. 6.9(e). Similar logic is used to obtain the PMF of $B_t\Delta A$. Detailed derivation of these PMFs is deferred to Appendix B for better readability.

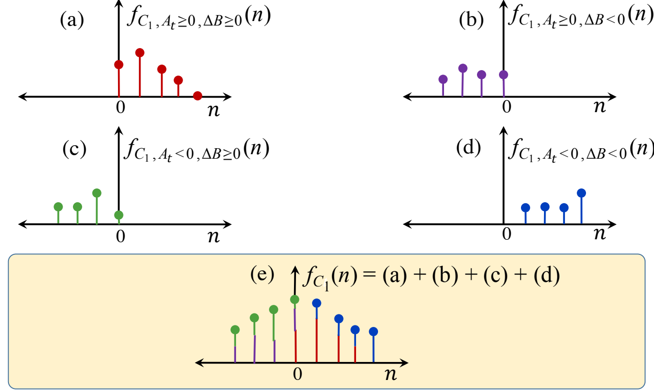


Figure 6.9: PMF of the product, $C_1 = A_t \Delta B$, as a sum of four PMFs depending on the sign of the two operands, A_t and ΔB.

Since the PMF, $f_{\Delta R}(n)$, of the generated error, ΔR, from Eq. (6.32) can also be represented by an impulse train, we can use the concepts explained in Section 6.3.1, to solve the convolution problem in Eq. (6.39). Specifically, we can apply Z-transform on both sides of Eq. (6.39), followed by evaluating the resulting polynomial at the reciprocal of $(2\mathcal{F}_C + 1)^{\text{th}}$ roots of unity, where $\mathcal{F}_C$ is the maximum value of ΔC from Eq. (6.39), and then performing an IFFT on those evaluations.

6.4.3 Error PMF at Output Nodes of a DAG

Each node of an approximate DAG can be traversed topologically while performing the unit operations of error generation and propagation through that node. The inputs connected to the primary input nodes of the DAG are error-free. The total error accumulates during the topological traversal, and appears as error at the primary output of the DAG.

6.5 Experimental Setup and Results

We implemented our ideas in C++, and the experiments were performed on a 64-bit Ubuntu server with a 3 GHz Intel® Core™2 Duo CPU E8400 processor. We use the Boost library to generate random numbers for performing Monte Carlo simulations against which we validate our proposed algorithm. We also use the FFTW library [74]

to perform the Fourier transform related calculations in C++, that is used in error PMF calculation of multiplier nodes, as well as to perform fast convolutions. Integrating Boost and FFTW libraries into our code renders it to be very fast as compared to MATLAB implementation, while providing the functional advantages of MATLAB.

The approximate adders that constitute the adder and multiplier nodes within an approximate DAG are the various transistor level approximations of the mirror adders [1], the Boolean expressions of which are mentioned in Table 5.3 in the previous chapter. Since in practice, at most ∼50% of the resultant bits are usually approximated to maintain accuracy [1, 34], we use a similar strategy while setting the number of approximate LSBs in each node of the DAG.

6.5.1 PMF of Generated Error in Approximate Nodes

We consider 10-bit Ripple Carry Adders and Modified Baugh-Wooley Multipliers as the approximate nodes of a DAG, and assume the magnitudes of the input to be represented by a Gaussian distribution with mean, 2^6 and variance, $2^6/6$, with either sign being equiprobable. Additionally, for the Monte Carlo simulations, we ensure that the selected input ranges do not result in overflow, since this reflects the typical use case of these computational blocks.

The generated error distributions of 10-bit adders and 10-bit×10-bit multipliers are computed assuming four LSBs to be approximate (40% approximation) by replacing the corresponding FAs by either of the five versions of the approximate FAs from Table 5.3. We assume the bitwise input signal probabilities to be 0.5, since for the distribution of the inputs is symmetric about zero. For internal nodes, although the inputs are not guaranteed to be symmetric, the propagated errors dominate the generated errors, and the signal probability assumption does not affect the results significantly.

We compare the cumulative distribution functions (CDFs) of the error PMFs obtained by our approach with those obtained by 5000 Monte Carlo simulations in Fig. 6.10 for adders (top row) and multipliers (bottom row). Each column of Fig. 6.10 indicated by (a)-(e), corresponds to the error CDF for an adder or a multiplier, when implemented using the approximate FAs from Table 5.3. Clearly from these figures, the estimated distributions show excellent match with those obtained from Monte Carlo simulations.

To compare the statistics, we normalize the output errors to the dynamic range

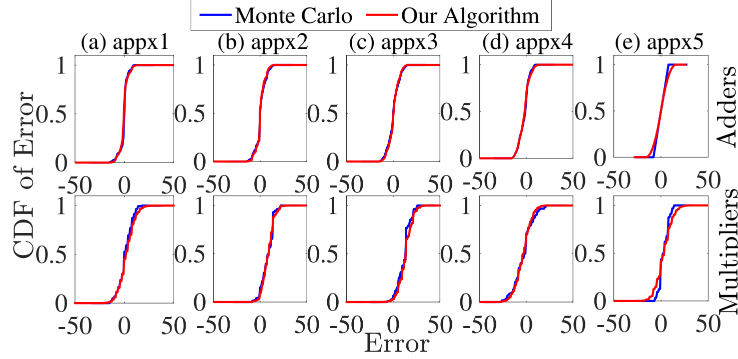


Figure 6.10: Distribution of error in an 10-bit signed adder (top row) and 10-bit×10-bit signed multiplier (botom row) implemented with the appx1-appx5 versions of the FAs from [1].

of the output of the approximate adder and multiplier since the same magnitude of error may have different levels of severity depending on the magnitude of the output. The normalization factor, $\mathcal{R}$, is the total range (difference of maximum and minimum values) of the output when the circuit (adder/multiplier) is implemented using different combinations of approximate and accurate FAs. We compare the standard deviation of the error PMFs obtained by our algorithm and Monte Carlo simulations, using the absolute value of the normalized percentage error in standard deviation, $\Delta\sigma_{norm}$, defined as:

$$\Delta\sigma_{norm} = 100 \times \frac{|\sigma_{est} - \sigma_{MC}|}{\mathcal{R}} \tag{6.40}$$

where σ_{est} and σ_{MC} are the standard deviations of the generated error PMF at adder/multiplier output, estimated by our algorithm, and through Monte Carlo simulation), respectively, and $\mathcal{R}$ is the normalizing factor defined earlier.

Table 6.2 lists the $\Delta\sigma_{norm}$ values of the 10-bit adders and multipliers in second and third columns, respectively, when approximation is introduced by replacing four LSB FAs by each approximate version of the FA from Table 5.3. Clearly, the error statistics obtained by our algorithm show an excellent match with those obtained by Monte Carlo simulations, as indicated by the negligible errors in Table 6.2.

Table 6.2: The values of $\Delta\sigma_{norm}$ for approximate adders and multipliers

DAG → **Adder Type↓**	Adders	Multipliers
appx1	0.01	0.11
appx2	0.16	0.02
appx3	0.15	0.03
appx4	0.09	0.15
appx5	0.50	0.33

6.5.2 Output Distribution in DAGs

We consider five representative DAGs for DSP applications, namely, MPEG Decoder (MPEG), Matrix Multiplier (MM), Horner-Bezier Filter (HB), Elliptic-Wave Filter (EWF), and Autoregression Filter (ARF), from the ExpressDFG Benchmark Suite [2], comprising of adder and multiplier nodes, and implementing a few generic math functions and signal processing related circuits, for demonstrating our algorithm.

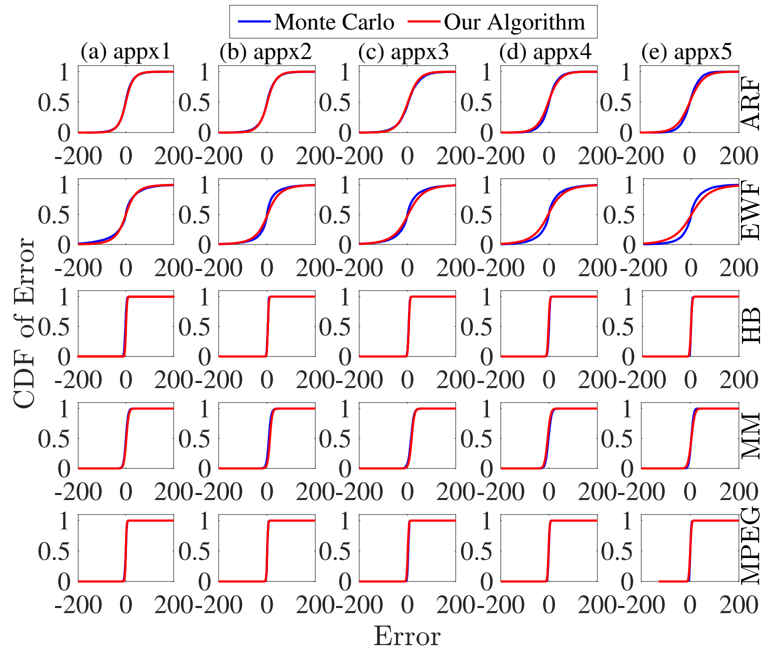


Figure 6.11: Distribution of error in the deepest output node of the DAGs consisting of 10-bit adders and multipliers implemented with the appx1-appx5 versions of the FAs from [1].

The number of approximate bits or the type of approximate FAs within each node of a DAG is a user-specified input. Here we assume all the nodes except those connected to the primary inputs to be approximated by each type of FAs from Table 5.3, having the same number of approximate LSBs (30%), similar to the approaches in [1]. We select the input statistics for each DAG differently, based on the number of stages in its deepest primary output, so that overflow-related errors can be avoided, while the inputs are large enough that the number of bits required to represent them is more than the number of bits approximated at each node. The error CDFs of the five DAGs are illustrated in Fig. 6.11. where each column of subplots (a)-(e), corresponds to the type of adders from Table 5.3 used to implement the approximate nodes within the respective DAG, which is denoted by each row of the subplots. The error distribution obtained by 5000 Monte Carlo simulations is assumed to be the reference. Clearly from Fig. 6.11, the CDFs obtained by our algorithm show a very good match with those obtained through Monte Carlo simulations.

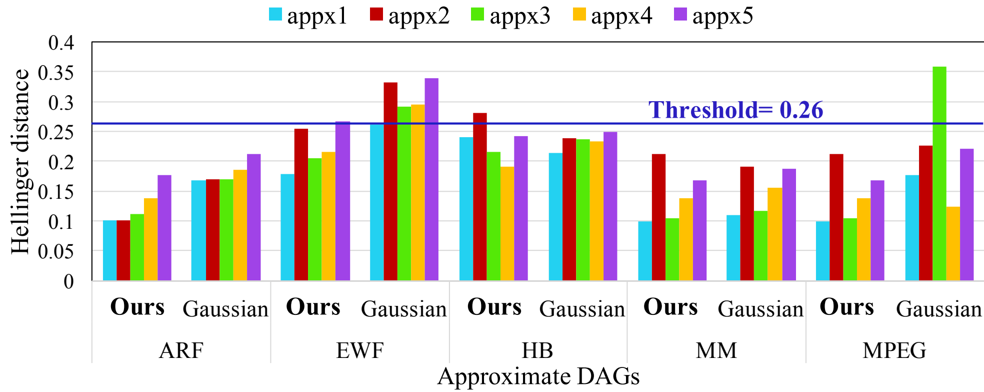


Figure 6.12: Hellinger distance between PMFs obtained by our approach and Gaussian approximation, as compared to the Monte Carlo PMFs. Our method is often well below the threshold, and our Hellinger distance is generally better than or comparable to the Gaussian approximation.

To quantify the Monte-Carlo like accuracy of our algorithm, we compute the Hellinger distance [75] between the estimated and Monte Carlo PMFs, and it is defined as:

$$\text{Hellinger distance} = \frac{1}{\sqrt{2}}\sqrt{\sum_{\text{all } n}\left(\sqrt{\hat{f}(n)} - \sqrt{f(n)}\right)^2} \tag{6.41}$$

where $\hat{f}(n)$ and $f(n)$ are the estimated and the Monte Carlo PMFs, respectively. The factor, $\sqrt{2}$, ensures that this distance ranges from 0 to 1. The threshold value to judge the closeness of the two PMFs so that below this value, the two PMFs can be practically assumed to be the same is chosen to be 0.26 [76]. We plot these distances between our estimated error PMFs and the corresponding PMFs from Monte Carlo simulation in Fig. 6.12, and for most of the cases, they are less than the threshold value. For comparison purposes in Fig. 6.12, we also construct Gaussian PMFs with the error mean and variances obtained by simply computing the statistics at the output of each node (instead of the entire PMF), and then plot the Hellinger distances between these PMFs and the Monte Carlo PMFs. This exercise gives an indication that although the error statistics may the same for the artificially generated PMFs, the error PMF is not completely Gaussian for many cases, as indicated by the threshold violation in Fig. 6.12 for DAGs where our method generates a more accurate error PMF.

Table 6.3: Normalized percentage errors in estimated standard deviations ($\Delta\sigma_{norm}$ defined in Eq. (6.40)) for five DAGs from [2].

DAG → **Adder Type↓**	ARF	EWF	HB	MM	MPEG
appx1	0.0	0.00	0.00	0.00	0.00
appx2	0.00	0.00	0.00	0.00	0.00
appx3	0.01	0.01	0.00	0.00	0.00
appx4	0.05	0.08	0.00	0.00	0.01
appx5	1.19	2.75	0.06	0.28	0.09

We summarize the $\Delta\sigma_{norm}$ values in Table 6.3, corresponding to the error PMFs of the five DAGs, introduced by approximating three LSBs in each node implementing 10-bit signed operations, using the FAs from Table 5.3. Clearly, the values of $\Delta\sigma_{norm}$ are less than 2% in most of the cases, indicating an excellent match. The EWF DAG has relatively higher discrepancy between our method and the Monte Carlo PMF for *appx5*. This arises because of the higher errors in the *appx5* version of the FA, which can be observed from its truth table using the Boolean functions in Table. 5.3. However, for all other cases where the FAs have a reasonable accuracy, our method shows an excellent match with those obtained from the Monte Carlo simulations.

Since the runtime for Monte Carlo simulations is proportional to the sample size,

Table 6.4: Runtime of our algorithm and 6000 Monte Carlo simulations.

DAG	ARF	EWF	HB	MM	MPEG
Monte Carlo	14.21s	34.59s	9.31s	33.71s	12.61s
Our Algorithm	0.60s	1.13s	0.53s	1.33s	1.23s
Speedup	23.68×	30.61×	17.56×	25.34×	10.25×

for fair comparison, we list the runtimes to obtain the error PMFs for the five DAGs (implemented by the *appx1* version of the FA from Table 5.3), by our algorithm, and 6000 Monte Carlo simulations in Table 6.4, based on the fact that the accuracy of our approach is also evaluated using 6000 such simulations. Our algorithm provides over 20× speedup on an average, with Monte Carlo like accuracy, thus indicating the feasibility of our approach to be used in a larger optimization/design framework.

6.6 Conclusion

We have proposed a novel technique in this chapter to analytically construct the PMF of errors generated due to approximating various adder and multiplier nodes in a DAG, using the concepts of transform calculus. We use the Fourier transform to propagate the errors through the adder nodes, while we implement a combination of the Mellin transform along with the Fourier transform, to propagate the errors through multiplier nodes within a DAG. The novelty of employing Mellin transform lies in its equivalence to the Fourier transform to compute the PMF of the product of random variables. Our technique generates the error PMF much faster than the traditional Monte Carlo simulations, and with very similar accuracy, as observed through the Hellinger distance metric as well as the difference in normalized standard deviations between our estimated and the Monte Carlo simulations.

Chapter 7

Approximate Circuit Design through SABER

The previous two chapters focussed on error analysis of approximate circuits for our study on intentional unreliability in circuits. In this chapter we focus on the design of these approximate circuits so that their power savings are maximized, while the errors injected into a computation by them, are bounded by user-specified budgets.

The design of approximate circuits has been explored at the system level [1, 77–79] as well as gate-level [80] for the past few years. At higher levels of design, [81] and [73] perform various high level synthesis transformations on abstract syntax trees and DAGs representing a circuit, respectively, with consideration of approximate components. However, all of these methods use coarse-grained decisions to choose among a few approximation options in conjunction with other high level decisions such as scheduling and binding.

We propose SABER (Selection of Approximate Bits for the Design of Error Tolerant Circuits), an optimization framework at the register transfer level that is solely focused on the tradeoff between approximation with power, but deployed at a much finer granularity level than [81] and [73]. More specifically, our optimization can continuously decide how many bits to be approximated for each arithmetic operation in a design.

The input to our optimization framework is the dataflow graph of the circuit, represented as a DAG whose nodes correspond to arithmetic units that can potentially

be approximated, and whose edges indicate the connections between these units. Our formulation maximizes the number of approximation bits in a circuit (which translates to power/area minimization) so that it uses minimal resources under a specified error budget. This work demonstrates results on fixed-point integer arithmetic operations. For convenience, in our exposition we assume operands to be integers since fractional operands can easily be scaled to integers and back again through simple shift operations To the best of our knowledge, this is the first work on design optimization considering approximation at bit-level granularity. We develop an analytical error model and a fast heuristic such that the computing cost is very low and highly scalable. This low cost provides the potential for our technique to be frequently called in the inner loop of a high level design-space exploration and optimization such as [81]. Our result serves as an initial solution for further refinement by gate-level synthesis methods [80].

For a fair comparison, instead of comparing against a no-approximation scheme (against which large improvements are easy to show), we compare our approach with methodologies where uniform approximation is used to approximate the circuit [1, 26], and demonstrate that our approach can outperform such methodologies by over 30% in power savings, for similar error specifications. The key contributions in this chapter are summarized as follows:

- **Precharacterization**: We perform gate level characterization of the error variance of multi-bit adders as a function of the number of approximated bits, starting from the LSB. This step is a one-time effort for a library of approximate gates.

- **Error Formulation**: We propose a computationally efficient, and accurate framework for expressing the error variance at the output of a DAG as a function of the number of approximate LSBs within each of its nodes, and model it as a nonlinear expression.

- **Design Optimization**: We formulate an optimization problem to maximize the total approximation in a circuit, constrained to an error budget. Since this optimization is an integer non-linear programming problem, we propose a heuristic to solve this NP-Hard problem. We generate an accurate starting point, followed by a fast approach to obtain the final solution in a simple, analytical form.

Through our optimization routine, we determine precisely if and how each node of a DAG should be approximated and optimize circuit performance under error specifications.

7.1 Error Characterization

The key ingredient of any methodology based on approximate design is an accurate quantification of the error injected into a computation by the approximation scheme. We use the variance of this error as the error metric to be constrained within a user-specified budget. Here we obtain an analytical expression of this error variance as a function of the total approximation in a circuit.

Let us consider a circuit representing an arithmetic operation with two N-bit operands, X and Y, producing an output, Z. An approximate implementation of the hardware unit yields the benefit of using fewer resources [22] than its exact counterpart. Typically some of the LSBs can be allowed to be erroneous, as this introduces a limited level of approximation. Hence, the hardware connected to y LSBs, for example, is approximate, while that connected to the $(N - y)$ MSBs is accurate. Clearly, the higher the value of y, the greater is the power saving due to the imprecise hardware, although the error is also higher. We use the parameter, y, referred to as the number of approximate LSBs, to quantify the amount of approximation.

We present an approach for characterizing the error variance of a DAG whose nodes are candidates for approximation. We begin by obtaining the error variance of an adder as a function of the number of approximate LSBs, y, in the adder. Using this function, we show how we can compute the error variance of any DAG whose nodes are approximate adders. The results for the adder DAG can be generalized to DAGs whose nodes contain adders, subtractors, multipliers, and dividers since the fundamental element of these operations is an adder [36], with shifters being implemented by appropriately routing the outputs of one DAG node to the inputs of others.

7.1.1 Error Precharacterization for an Adder

We consider transistor-level approximation where an N-bit approximate adder is implemented as an array of accurate and approximate full adders (FAs). If the error

due to y approximate LSBs is e, then e can range from $-(2^y - 1)$ to $(2^y - 1)$, and its exact value depends on the inputs. Typically inputs are assumed to be uniformly distributed random variables [1, 73]. Hence, e, being a function of these inputs, can be assumed to be a random variable as well. Let p_x be the probability of e to be x, where $x \in [-(2^y - 1),\ (2^y - 1)]$ is an integer owing to y being an integer. The error means are negligible compared to the variance [1]. Hence, we are concerned with the variance, $\sigma_e^2(y)$, given by:

$$\sigma_e^2(y) = \sum_{x=-(2^y-1)}^{(2^y-1)} x^2 p_x \tag{7.1}$$

Due to the x^2 term in Eq. (7.1), $\sigma_e^2(y)$ clearly depends on y. If x is uniformly distributed between $-(2^y - 1)$ and $(2^y - 1)$, $p_x = \frac{1}{2^{y+1}-1}, \forall x$, and $\sigma_e^2(y) = (2^{y+1} - 1)^2/12$. We also evaluate $\sigma_e^2(y)$ for normally distributed x in the later part of this section (Fig. 7.1). In fact, the exponential dependence on y holds for most practical error distribution functions (not just uniform or normal) for $y \leq N/2$, N being the word-length of the adder. Additionally, using the fact that $\sigma_e^2(y)$ should be zero for $y = 0$ (no approximation implies zero error), the variance of e is formulated empirically as:

$$\sigma_e^2(y) = a(2^{by} - 1) \tag{7.2}$$

where a and b are constants, obtained by fitting the error variance for different y, through Monte Carlo simulations.

Here we consider the specific transistor-level approximate FAs from [1] and the Lower-part-Or Adder (LOA) from [28] for our analysis. For a particular type of N-bit approximate adder with y approximate LSBs ($N = 10$ considered here), each simulation proceeds by uniformly sampling two inputs, X and Y, from $[0, 2^N - 1]$, to produce an approximate result, Z, and hence, the corresponding error, e, can be calculated. Since N is relatively small, we obtain the variance of e for a particular y by exhaustive simulations. This procedure is repeated for $y = 0, \cdots, N-1$, to obtain a and b in Eq. (7.2) through regression analysis.

The results are summarized in Table 7.1. The first column lists the type of adder studied in this work, followed by the respective values of a and b, defined in Eq. (7.2),

in the next two columns, respectively. The fourth and fifth columns list the adjusted R^2 values which refer to the goodness of the fit ($R^2 = 1$ indicates that the fitted model explains all variability) and the root mean square error (RMSE) values of the fitted curve, respectively. Both the quantities indicate that the model is a good fit for the actual data.

Table 7.1: Fitting parameters for adder error variance.

Adder type	a	b	adjusted R^2	RMSE
appx1	0.05	1.98	1.00	1.04
appx2	0.11	2.01	1.00	1.17
appx3	0.14	2.00	1.00	0.25
appx4	0.10	2.00	1.00	0.11
appx5	0.08	2.00	1.00	0.00
LOA	0.06	2.00	1.00	0.00

The simulations shown above assumed the two inputs, X and Y, of the adder node to be statistically independent. In a general scenario, the two inputs of an adder node within a DAG may have some correlation. Furthermore their distribution may not be uniform, as assumed in the above experiment. To observe the effect of a different input distribution that is correlated, we perform 5000 Monte Carlo simulations on 10-bit adders implemented using the FAs from Table 7.1, first with two independent 10-bit Gaussian inputs, and then with two correlated Gaussian inputs ($\rho = 0.5$), altering the number of approximate LSBs, y. We compare $\sigma_e^2(y)$, for both cases with that obtained through our model, for different values of y, as shown in Fig. 7.1.
In spite of the correlation among the inputs, which are also from a different distribution than what was used for precharacterization, variances of the error generated by the adder using a and b from Table 7.1, show an excellent match with those obtained from Monte Carlo simulations. Intuitively, this effect arises because the change in the distribution and correlation is more likely to affect the higher order bits, which are not approximated, and the distribution of the lower order bits is close to uniform regardless of correlation and for any reasonable distribution. Hence, we consider the generated adder errors as independent random variables to compute the total error variance of a DAG (in Eq. (7.3)).

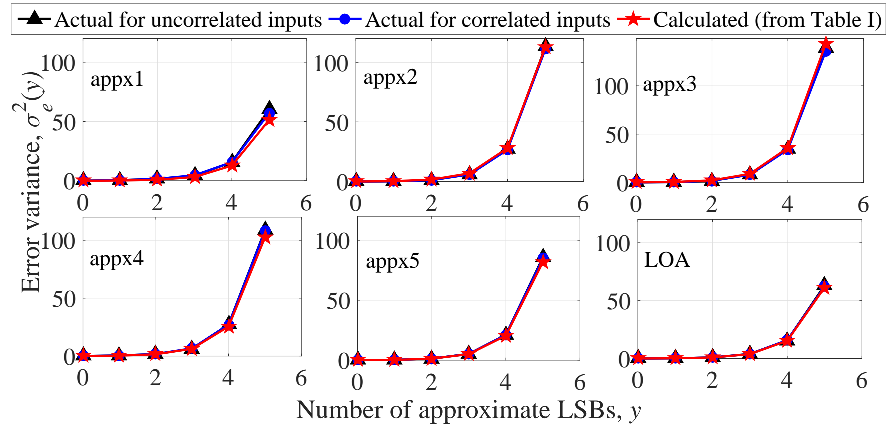


Figure 7.1: Error variance due to the uncorrelated and correlated adder inputs as a function of y.

7.1.2 Error Computation of a DAG

Let us consider a DAG consisting of adders and multipliers, and one primary output (PO) as shown in Fig. 7.2(a). A pair of adder and multiplier nodes from Fig. 7.2(a) has been highlighted in Fig. 7.2(b) to depict the implementation of the multiplier by add and shift operations. Overall, the DAG has T nodes, each representing an adder, and each edge is associated with a shift operation, denoted by $<<$.

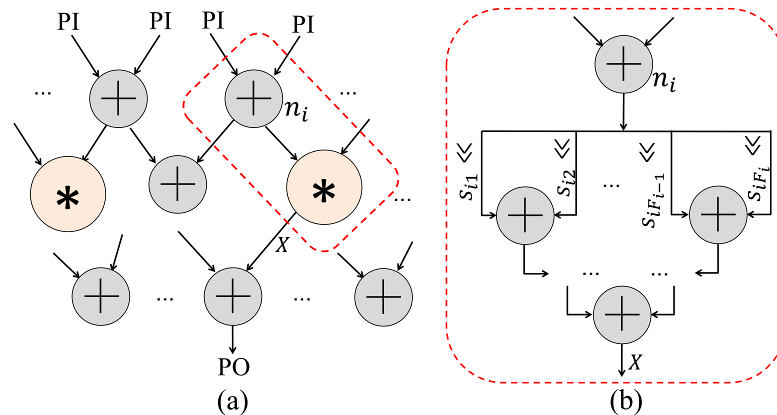


Figure 7.2: (a) A DAG consisting of adders and multipliers, (b) Representation of multipliers as shift and add operations.

Each node, n_i, is indexed by the subscript, $i \in [1, T]$, and the fanout of n_i is represented by F_i. Each of the F_i fanout edges of n_i is associated with a weight resulting from a shift operation of s_{ij} bits, such that j ranges from 1 to F_i. This nomenclature is depicted in Fig. 7.2(b).

Let the number of approximate LSBs in n_i be represented by y_i, hence the generated error variance in n_i is $\sigma_e^2(y_i)$, and is obtained by substituting $y = y_i$ in Eq. (7.2). The error generation among different approximate operations can be assumed to be independent for all practical purposes. However, error propagation exhibits structural correlation since the approximation in n_i not only affects its immediate fanouts, but also those in its fanout cone, through the edges (and the associated weights) connecting n_i to a PO via the transitive fanouts. We use the error sensitivity, β_i, of n_i, to a PO, to capture the structural correlations within the DAG. For a single PO, if an error, e, at n_i results in an error, E_i, at the PO, then $\beta_i = E_i/e$ is computed by a depth-first search of the DAG [73]. The total error variance, σ_t^2, which is a commonly used error metric [79] in the DAG, is obtained as:

$$\sigma_t^2 = \sum_{i=1}^{T} \sigma_e^2(y_i)\beta_i = \sum_{i=1}^{T} a(2^{by_i} - 1)\beta_i \tag{7.3}$$

where β_i is alternatively called the β value of the node, n_i.

When there are multiple POs in a DAG, to minimize error on all the POs, we simply add a dummy (but accurate) adder node with all the POs. This node is not a part of the design but conceptually indicates the summation of error variances of all nodes to compute the total error variance, σ_t^2 for a multioutput circuit. The addition of this dummy node is a simple device that enables the depth-first traversal of the DAG to compute the β values of the *real* nodes.

7.2 Optimization through SABER

In this section, we outline the SABER algorithm, which yields the number of approximate LSBs in each node of the DAG, maximizing the total power and area savings, while satisfying a specified error budget.

We first explain our proposed optimization problem, which is NP-Hard, and obtain a feasible solution by relaxing some of the constraints, to make it tractable. Next,

we propose a heuristic to solve the original problem, using the solution of the relaxed problem.

7.2.1 The Optimization Problem

Let us consider a DAG with T adder nodes. The power savings increase with increasing levels of approximation in the DAG, and all components of the power savings (dynamic and leakage) are proportional to the number of approximate bits, i.e., the number of approximate FAs. For adders, the proportionality of power savings to the number of approximate FAs has been empirically observed in previous work ([1] and [26]). Consider an approximate FA, as in [1], with a reduced transistor count. If S denotes the power savings per FA, then if k bits (i.e., k FAs) of a ripple-carry adder are approximated, the power savings equal kS. For more complex adder, such as a carry look-ahead adder, savings of kS continue to be seen in the FAs of the adder. If the carry logic is also approximated, the savings may be weakly super-linear, and currently, SABER does not model this, but an extension for this case is not difficult. The set of equations in (A.18) will be non-linear, but we can use well-established Newton-Raphson methods to solve the system. The heuristic for converting the real-valued solution to an integer solution as explained in Sec 7.2.2 can be used directly since the linearized version of the nonlinear equations is likely to be accurate over the small range of the number of approximate bits considered in this step. Even for a multiplier node, the linear proportionality holds true, because as discussed in Sec. 7.1.2, we decompose it into its constituent FAs, and hence the power savings are linear in the number of FAs in the decomposed graph. Therefore, the total number of approximate LSBs in the DAG, $\sum_{i=1}^{T} y_i$, is a good surrogate objective function that captures the essential trend of power savings, which we aim to maximize. The total error variance, σ_t^2, accumulated as a result of this approximation is given by Eq. (7.3). If the specified error variance budget is m, then σ_t^2 must be less than m. We thus formulate the optimization problem as:

$$\max \sum_{i=1}^{T} y_i, \quad \text{s.t} \sum_{i=1}^{T} a\left(2^{by_i} - 1\right)\beta_i \leq m, \quad y_i \in \mathbb{Z}^+ \tag{7.4}$$

where $\mathbb{Z}^+$ represents the set of non-negative integers. The constraint, $y_i \in \mathbb{Z}^+$, arises because the number of approximate LSBs in a node cannot be negative or fractional.

A feasible solution to the problem, which satisfies the error budget, always exists: it is the zero approximation solution. However, generating the optimal solution is NP-Hard since (7.4) is an integer non-linear problem. Hence, we *relax* the optimization problem, to make it tractable, and obtain a feasible solution. For this, we first remove the constraint, $y_i \in \mathbb{Z}^+$ in (7.4), and then convert the inequality constraint into an equality, since the optimal solution for the new maximization problem, will lie on the constraint surface. We obtain the solution through Theorem 7.1.

Theorem 7.1. *If the relaxed optimization problem from* (7.4) *is,*

$$\max \sum_{i=1}^{T} y_i, \quad \text{s.t} \sum_{i=1}^{T} a(2^{by_i} - 1)\beta_i = m \tag{7.5}$$

with β_i being the error sensitivity used in Eq. (7.3)*, then the solution, $\widetilde{y}_i$, is obtained as:*

$$\widetilde{y}_i = Y - \frac{1}{b}\log_2(\beta_i) \tag{7.6}$$

$$\textit{where } Y = \frac{1}{b}\log_2\left[\left(m + a\sum_{i=1}^{T}\beta_i\right)(aT)^{-1}\right] \tag{7.7}$$

The proof of Theorem 7.1 is deferred to Appendix A.

Next we impose the constraint on y_is to be integers. Since the $\widetilde{y}_i$s from Eq. (7.6), are not guaranteed to be integers, we use Lemma 1 to obtain a feasible solution.

Lemma 1. *A feasible solution of* (7.4) *is given by $\lfloor \widetilde{y}_i \rfloor$, where $\lfloor . \rfloor$ represents the* floor *function, and $\widetilde{y}_i$ is the optimal solution of the relaxed problem,* (7.5)*, and is defined in Eq.* (7.6)*.*

Proof: The left hand side (LHS) of the constraint in (7.4) is a monotonically increasing function of the state variables. Since, $\lfloor \widetilde{y}_i \rfloor \leq \widetilde{y}_i$, if $\widetilde{y}_i$ is a feasible solution (i.e., ensures the LHS to be less than m), then so is $\lfloor \widetilde{y}_i \rfloor$. □

Since, the solution from Lemma 1 may be suboptimal, or even negative, we propose heuristics to address these issues.

7.2.2 Heuristics to Solve the Original Problem

Let us observe the potential suboptimality associated with the *floor* function, through an example with two state variables, where the objective and constraint surfaces are depicted in Fig. 7.3. Both the optimal, $(\widetilde{x}_1, \widetilde{x}_2)$, and the corresponding $(\lfloor\widetilde{x}_1\rfloor, \lfloor\widetilde{x}_2\rfloor)$, are also plotted in the same figure. Clearly, $(\lfloor\widetilde{x}_1\rfloor, \lfloor\widetilde{x}_2\rfloor)$ is suboptimal. Since $(\widetilde{x}_1, \widetilde{x}_2)$ lies at the intersection of the constraint and the objective surface, $(\lfloor\widetilde{x}_1\rfloor, \lfloor\widetilde{x}_2\rfloor)$ must be pushed towards the constraint surface, pictorially depicted by the arrows in Fig. 7.3, to achieve a better solution.

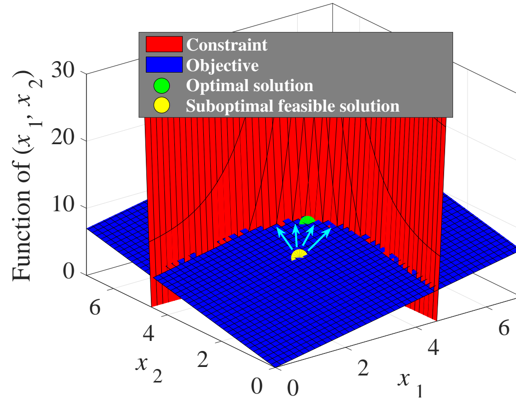


Figure 7.3: Example of an optimization problem similar to (7.5).

Similarly, we attempt to obtain the number of approximate LSBs, $\hat{y}_i$, in node, n_i, of the DAG in Fig. 7.2, by pushing the $\lfloor\widetilde{y}_i\rfloor$s towards the constraint surface while ensuring non-negativity.

For this, we first define $X = \lfloor Y \rfloor$ (with Y defined in Eq. (7.7)), so that each node now has $X - \frac{1}{b}\log_2 \beta_i$ approximate LSBs. This expression arises out of Eq. (7.6), where we apply the *floor* function only to a part of the solution, $\widetilde{y}_i$. Next we use Theorem 7.2 to obtain a parameter, K, denoting the number of nodes to which we add one more approximate LSB while satisfying the error constraints, thus further increasing the objective function in (7.4), while keeping the solution, feasible.

Theorem 7.2. *If the nodes of a DAG are indexed in the increasing order of their β values, such that each node, n_i, has $(X - \frac{1}{b}\log_2 \beta_i)$ approximate LSBs, where $X = \lfloor Y \rfloor$, and Y is defined in Eq. (7.7), then the number of first K nodes to which one more*

approximate LSB can be added to satisfy the error constraint, m, is given by:

$$K = \left\lfloor \left(m + a \sum_{i=1}^{T} \beta_i - a2^{bX} T \right) \Big/ \left(a2^{bX} (2^b - 1) \right) \right\rfloor \tag{7.8}$$

where T is the total number of adder nodes, and $K < T$.

The proof of Theorem 7.2 is deferred to Appendix A.

This analysis till now holds true even if one or more of the $\hat{y}_i$ values are negative. However, since a negative $\hat{y}_i$ does not have any physical meaning, we need to reassign these values to zero, which in turn may violate the error constraint in (7.4). To address this non-trivial issue we propose Algorithm 3 to obtain non-negative approximate LSBs while satisfying the error budget, m.

Algorithm 3 first resets the negative $\hat{y}_i$s to zero, and then, by decreasing the other $\hat{y}_i$s, computes the error to be compensated for. This compensation is performed by reducing $\hat{y}_i$ of the nodes corresponding to the highest β_i value as much as possible, before proceeding to the next with the second highest β value, and so on. Clearly, the complexity of our algorithm is dominated by the sorting of nodes in term of their β values, so that the overall complexity is $O(T \log T)$.

Algorithm 3 Algorithm to ensure non-negativity of $\hat{y}_i$.

Input: $\hat{y}_1, \cdots, \hat{y}_T$ values with decreasing order of β values.

Input: Error constraint, m, of the optimization problem in (7.4).

Output: Updated non-negative $\hat{y}_1, \cdots, \hat{y}_T$ values.

1: Initialize $C \leftarrow 0$ //Need to compensate error by C
2: **for** each j from 1 to T **do**
3:     **if** $\hat{y}_j < 0$ **then**
4:         $\hat{y}_j \leftarrow 0$ //Setting negative $\hat{y}_j$ to 0
5:     **end if**
6:     $C \leftarrow C + a(2^{b\hat{y}_j} - 1)\beta_j$ //Error from positive $\hat{y}_j$s
7: **end for**
8: **if** $C - m < 0$ **then**
9:     exit //Resetting $\hat{y}_j$ to 0 did not violate error
10: **end if**
11: Set $C \leftarrow C - m$ //Compensate by $C - m$, $\because C \geq m$

12: **for** each i from 1 to T with decreasing order of β_i **do**
13: **if** $C \leq a(2^{b\hat{y}_i} - 1)\beta_i$ **then**
14: Find Z where $(2^{b\hat{y}_i} - 1) - (2^{b(\hat{y}_i - Z)} - 1) = \frac{C}{a\beta_i}$
15: $\hat{y}_i \leftarrow \hat{y}_i - \lceil Z \rceil$
16: exit
17: **else**
18: $C \leftarrow C - a(2^{b\hat{y}_i} - 1)\beta_i$
19: $\hat{y}_i \leftarrow 0$
20: **end if**
21: **end for**

7.3 Experimental Setup and Results

We implement SABER in MATLAB R2015b on a 64-bit Ubuntu server with a 3GHz Intel® Core™2 Duo CPU E8400 processor. We consider the appx5 approximate adder which uses transistor-level approximation [1] for our analysis.

7.3.1 Optimization Results on an Example DAG

First we demonstrate the results of using SABER through an example structure, DAG10, with ten nodes, as shown in Fig. 7.4. Each node, n_i, can be represented by a 20-bit adder, where the approximation is introduced by replacing y_i LSB FAs with approximate FAs. A dummy node has been added as explained in Sec. 7.1.2, to obtain the error sensitivity (i.e., the β value) of the other nodes to the output. Each β_i is obtained by finding the number of paths from n_i through a depth-first search of the DAG considering the edge weights, and is depicted in Fig. 7.4.

We demonstrate our results for three error variance budgets, $m = 1\text{K}, 10\text{K}, 100\text{K}$, corresponding to the allowable error variance at the output of the dummy node. For comparison purposes, we consider the commonly-used uniform approximation case (e.g., in [1, 26]), when the number of approximate bits in each node is identical.

Using SABER, we compute the number of approximate LSBs in each of the ten nodes, whose distributions among the nodes are depicted in Fig. 7.5 for the three error variance budgets, m. Since the uniform approximation results in the same number of

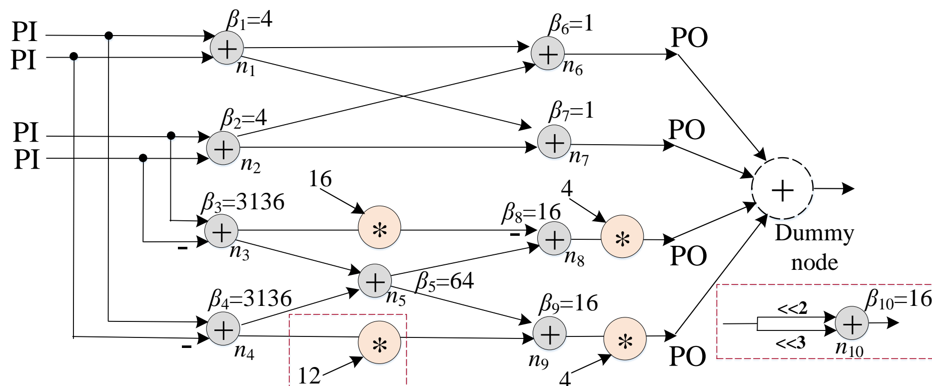


Figure 7.4: Structure of DAG10 with ten nodes.

approximate LSBs in each node, it is depicted by the stem plot, with a single stem of height ten, the rest being zero, in the same figure. Due to the dependence of the number of approximate LSBs in a node on its β value through Eq. (7.6), the distributions in Fig. 7.5 are also determined by the distribution of β values. As m increases, more approximation is possible, which is seen by the rightward horizontal shift in the bar charts.

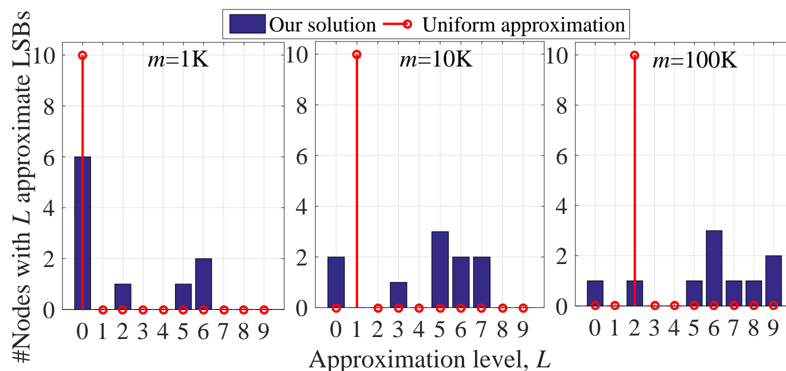


Figure 7.5: Distribution of the number of approximate LSBs over the ten nodes of DAG10.

The target, m, and the actual error variance, σ_t^2, arising out of the three approximate configurations, are listed in the first two columns of Table. 7.2. The difference between them arises due to the relaxation of the original problem in (7.4), and application of the proposed heuristics to make our solution feasible. However, this difference is less than 8% for all three cases, indicating the effectiveness of our methodology. The total number

Table 7.2: Total number of approximate FAs by SABER and the uniform approximation case for different error budgets.

Target, m ($\times 10^3$)	Achieved, σ_t^2 ($\times 10^3$)	Total approximate LSBs		Power savings (%)	
		SABER	Uniform	SABER	Uniform
1.00	0.98	19 LSBs	0 LSB	9.50	0.00
10.00	9.49	44 LSBs	10 LSBs	22.00	5.00
100.00	92.88	58 LSBs	20 LSBs	29.00	10.00

of approximate LSBs by SABER and the uniform approximation case, are listed in the next two columns of Table. 7.2, indicating that SABER clearly outperforms the uniform approximation for DAG10, from which power savings are calculated and listed in the last two columns. The proportionality constant between the number of approximate FAs and percentage power savings is 0.5%, as evident from the last four columns of the table, since each adder node is 20-bit, and 10 such adder nodes in the DAG lead to a total of 1/200×100=0.5% approximate FAs for each approximate LSB within the DAG. Since we use appx5 version of FA for approximation, the power savings for the entire DAG is also scaled by the same factor (0.5%) as appx5 has negligible power consumption compared to the exact FA [1]. Hence approximating k LSB FAs in the DAG leads to $0.5k\%$ power savings, which is accurate to a first order to indicate the advantage of using SABER to maximize power savings over the uniform approximation, without performing logic synthesis. The factor will be different for other versions of approximate FAs, and can be precomputed by the logic synthesis of the corresponding FA.

7.3.2 Optimization Results on FIR Filters

We evaluate our algorithm on a real-world example, by checking the sound quality of filtered signals from an approximate finite impulse response (FIR) filter. The results of filtering by approximate filters designed through SABER have been summarized within a compressed folder and uploaded to `http://conservancy.umn.edu/handle/11299/185544`. The signals under study comprise of 150K samples of eight different genres of audio clips [38] sampled at their prespecified frequency of 22.05KHz [82] and mixed with a high frequency noise. We constrain the signal to noise ratio (SNR) degradation between an exact filter and an approximate filter to be 50dB, to ensure comfortable loudness and clarity.

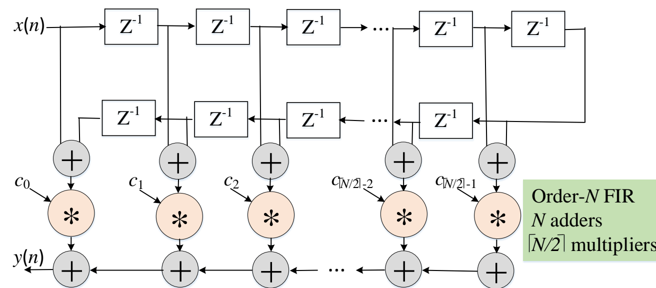


Figure 7.6: An FIR filter with symmetric coefficients [4].

The normalized pass band and stop band frequencies[1] of the FIR filter are 0.50 and 0.65, respectively, and the minimum order filter that MATLAB generated, had order=33.

The filter coefficients have been scaled by 1024 to facilitate integer arithmetic. All adders have word length of 20 bits, and the multiplications are implemented by array multipliers with add and shift operations. Since the coefficients are symmetric in an order-N FIR filter, we can reuse multipliers [4], resulting in only $\lceil N/2 \rceil$ multipliers and N adders, as shown in Fig. 7.6. In our order-33 FIR filter, the first 16 coefficients could be implemented simply by 30 adders (and shifters) based on their binary decomposition. Additionally the filter requires 33 adders. Hence, the resulting DAG for the optimization problem in (7.4) has $T = 63$ nodes.

To formulate the optimization problem, we need the error variance budget for each audio clip. Since the different genres of music are differently sensitive to approximation in the FIR filter, we select the respective error budgets from the tradeoff plot of SNR degradation versus error variance for each clip as depicted in Fig. 7.7. We obtain this plot by sweeping the error variance, m, to first obtain different configurations of approximate filters using SABER. We then filter the noisy signal using each such filter and compute the SNR degradation from the accurately filtered signal. Using this plot, we can select the target error variance, m, for various target SNR degradation values, an example of which is shown for 50dB by the line in Fig. 7.7. This plot can be generated very quickly since SABER takes less than a second to generate one configuration of the approximate filter.

[1] The normalized pass (stop) band frequency is defined as twice the ratio of the actual pass (stop) band frequency to the sampling frequency [70].

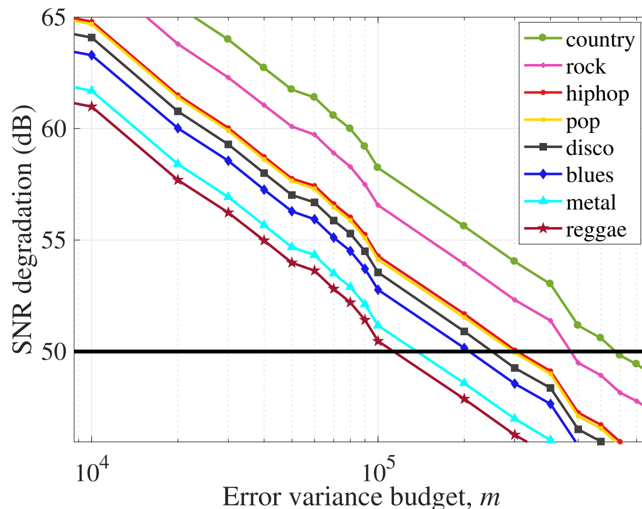


Figure 7.7: Tradeoff plot of SNR degradation with error variance in the FIR filter used to select the error budget.

For demonstration purposes, we select three values of m ($m = 100\text{K}, 200\text{K}, 400\text{K}$) from Fig. 7.7, to obtain three different approximate filters. The number of approximate LSBs in each node of the FIR filter for each m is then obtained by SABER, whose distribution among the 63 adder nodes is depicted in Fig. 7.8. The stem plots depicting the corresponding uniform approximation case are also provided in the same figures as reference.

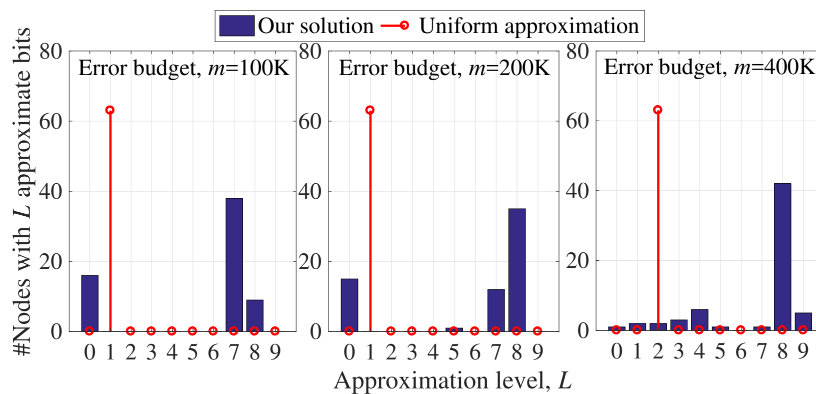


Figure 7.8: Distribution of the number of approximate LSBs over the 63 nodes of the FIR filter.

The power requirement of the approximate filters as a fraction of that of the accurate filter is listed in Table 7.3.

Table 7.3: Power dissipation of the approximate filters by SABER and uniform approximation, for three values of error budget, m, compared to the accurate filter.

	$m = 100\text{K}$	$m = 200\text{K}$	$m = 400\text{K}$
SABER	$0.73\times$	$0.70\times$	$0.66\times$
Uniform approximation	$0.95\times$	$0.95\times$	$0.90\times$
Improvement in power	30.14%	35.71%	36.36%

The second column denotes the values corresponding to the filter obtained by SABER with $m = 100\text{K}$, and by uniform approximation in the first two rows, respectively. The last row shows the percentage improvement of our method over the uniform approximation case for this m. Clearly, our algorithm not only achieves over 27% power savings over the exact implementation, but also outperforms the uniform approximation case by over 30%. These numbers increase as m is increased as seen from the last two columns of the table. Obtaining the solution through SABER can thus lead to significant power savings over the existing methodologies.

The resulting SNR degradation values for the eight audio clips while using the three filters are summarized in Table 7.4. The audio clips are listed in the first row, and the SNR degradation values for the error variance budgets, $m = 100\text{K}, 200\text{K}, 400\text{K}$, are listed in the next three rows, respectively. The degradation values are around 50dB

Table 7.4: SNR degradation (in dB) between the accurately filtered signal and those from the approximate filters constructed for different error budgets, m, using SABER.

Genre→	country	rock	hiphop	pop	disco	blues	metal	reggae
$m = 100\text{K}$	58.24	56.57	54.28	54.12	53.55	52.77	51.17	50.47
$m = 200\text{K}$	55.62	53.93	51.69	51.53	50.90	50.16	48.58	47.87
$m = 400\text{K}$	53.03	51.39	49.12	49.00	48.37	47.65	46.03	45.34

in all three rows indicating that the user experience is not compromised in spite of the approximations in the filter, and this fact can also be verified by directly playing the audio clips from `http://conservancy.umn.edu/handle/11299/185544`. For each clip, the site contains the noisy version, the exact filtered version, and filtered versions corresponding to the three error variance budgets, m, respectively.

7.4 Conclusion

We have proposed a bit-level optimization framework to design approximate circuits under specified error budgets, built upon an analytical expression for the number of approximate LSBs for each computational unit. The runtimes to obtain an approximate configuration of a DAG are shown to be very small due to the closed form solution, and outperforms the conventional approximation methods by over 30% in power savings.

Chapter 8

Thesis Conclusion

This thesis has developed novel techniques to manage circuit reliability by studying both the unintentional and intentional errors introduced in circuit outputs in course of its normal operation. Aging related issues are unavoidable in circuits, and hence, this thesis proposes algorithms to estimate delay degradation in a circuit under BTI and HCI aging, by using on-chip sensors, to tackle this unintentional reliability issue. An aging model has been proposed that provides a continuous and differentiable equation of the circuit delay, to facilitate the use of ROSC-based structures as surrogate aging sensors. The calibration factors computed using this model to translate sensor aging to that of the monitored circuit, can be stored in on-chip look-up tables, and easily read out while testing the circuit for aging, without interrupting its regular operation.

For the issue of intentional unreliability in circuits for error-tolerant applications, this thesis proposes algorithms for both error analysis of such circuits, and their design to reduce system power. For error analysis, the proposed algorithms compute error PMF in approximate circuits analytically, demonstrating how concepts from transform calculus can be used to enhance the speed of computation, while ensuring Monte Carlo-like accuracy. For approximate circuit design, this thesis proposes a novel algorithm for selecting the number of approximate bits in a node within a DAG by solving an integer non-linear optimization problem so that the total error variance of the approximate circuit is constrained by a user-specified budget.

The algorithms pertaining to both intentional and unintentional reliability issues, have been validated on standard benchmarks for the corresponding applications.

References

[1] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-Power Digital Signal Processing Using Approximate Adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2013. http://dx.doi.org/10.1109/TCAD.2012.2217962, Accessed March 24, 2017.

[2] EXPRESS Benchmarks. http://www.ece.ucsb.edu/EXPRESS/benchmark/, Accessed March 24, 2017.

[3] J. Keane, X. Wang, D. Persaud, and C. H. Kim. An All-In-One Silicon Odometer for Separately Monitoring HCI, BTI, and TDDB. *IEEE Journal of Solid-State Circuits*, 45(4):817–829, 2010. https://doi.org/10.1109/JSSC.2010.2040125, Accessed March 24, 2017.

[4] L. Aksoy, P. Flores, and J. Monteiro. A Tutorial on Multiplierless Design of FIR Filters: Algorithms and Architectures. *Circuits, Systems, and Signal Processing*, 33(6):1689–1719, 2014. https://doi.org/10.1007/s00034-013-9727-8, Accessed March 24, 2017.

[5] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance CMOS Variability in the 65-nm Regime and Beyond. *IBM Journal of Research and Development*, 50(4.5):433–449, 2006. http://dx.doi.org/10.1147/rd.504.0433, Accessed March 24, 2017.

[6] M. A. Alam and S. Mahapatra. A Comprehensive Model of PMOS NBTI Degradation. *Microelectronics Reliability*, 45(1):71–81, 2005. http://doi.org/10.1016/j.microrel.2006.10.012, Accessed March 24, 2017.

[7] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. An Analytical Model for Negative Bias Temperature Instability. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 493–496, 2006. https://doi.org/10.1145/1233501.1233601, Accessed March 24, 2017.

[8] A. Bravaix, C. Guerin, V. Huard, D. Roy, J.-M. Roux, and E. Vincent. Hot-Carrier Acceleration Factors for Low Power Management in DC-AC Stressed 40nm NMOS Node at High Temperature. In *Proceedings of the IEEE International Reliability Physics Symposium*, pages 531–548, 2009. https://doi.org/10.1109/IRPS.2009.5173308, Accessed March 24, 2017.

[9] T. Nigam. Impact of Transistor Level Degradation on Product Reliability. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 431–438, 2009. https://doi.org/10.1109/CICC.2009.5280815, Accessed March 24, 2017.

[10] J. Fang and S. S. Sapatnekar. Incorporating Hot-Carrier Injection Effects Into Timing Analysis for Large Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(12):2738–2751, 2014. https://doi.org/10.1109/TVLSI.2013.2296499, Accessed March 24, 2017.

[11] W. Wang, Z. Wei, S. Yang, and Y. Cao. An Efficient Method to Identify Critical Gates under Circuit Aging. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 735–740, 2007. https://doi.org/10.1109/ICCAD.2007.4397353, Accessed March 24, 2017.

[12] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. Adaptive Techniques for Overcoming Performance Degradation Due to Aging in CMOS Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(4):603–614, 2011. https://doi.org/10.1109/TVLSI.2009.2036628, Accessed March 24, 2017.

[13] E. Mintarno, J. Skaf, R. Zheng, J. B. Velamala, Y. Cao, S. Boyd, R. W. Dutton, and S. Mitra. Self-Tuning for Maximized Lifetime Energy-Efficiency in the Presence of Circuit Aging. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(5):760–773, 2011. https://doi.org/10.1109/TCAD.2010.2100531, Accessed March 24, 2017.

[14] L. Zhang and R. P. Dick. Scheduled Voltage Scaling for Increasing Lifetime in the Presence of NBTI. In *Proceedings of the IEEE Asia and South Pacific Design Automation Conference*, pages 492–497, 2009. https://doi.org/10.1109/ASPDAC.2009.4796528, Accessed March 24, 2017.

[15] T. H. Kim, R. Persaud, and C. H. Kim. Silicon Odometer: An On-Chip Reliability Monitor for Measuring Frequency Degradation of Digital Circuits. *IEEE Journal of Solid-State Circuits*, 43(4):874–880, 2008. https://doi.org/10.1109/JSSC.2008.917502, Accessed March 24, 2017.

[16] K. K. Kim, W. Wang, and K. Choi. On-Chip Aging Sensor Circuits for Reliable Nanometer MOSFET Digital Circuits. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 57(10):798–802, 2010. https://doi.org/10.1109/TCSII.2010.2067810, Accessed March 24, 2017.

[17] T. Iizuka, T. Nakura, and K. Asada. Buffer-Ring-Based All-Digital On-Chip Monitor for PMOS and NMOS Process Variability and Aging Effects. In *Proceedings of the IEEE Design and Diagnostics of Electronic Circuits and Systems*, pages 167–172, 2010. https://doi.org/10.1109/DDECS.2010.5491792, Accessed March 24, 2017.

[18] T. B. Chan, P. Gupta, A. B. Kahng, and L. Lai. DDRO: A Novel Performance Monitoring Methodology Based on Design-Dependent Ring Oscillators. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*, pages 633–640, 2012. https://doi.org/10.1109/ISQED.2012.6187559, Accessed March 24, 2017.

[19] Q. Liu and S. S. Sapatnekar. Synthesizing a Representative Critical Path for Post-Silicon Delay Prediction. In *Proceedings of the ACM International Symposium on Physical Design*, pages 183–190, 2009. https://doi.org/10.1145/1514932.1514973, Accessed March 24, 2017.

[20] S. Wang, J. Chen, and M. Tehranipoor. Representative Critical Reliability Paths for Low-Cost and Accurate On-Chip Aging Evaluation. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 736–741, 2012. https://doi.org/10.1145/2429384.2429543, Accessed March 24, 2017.

[21] Y. K. Chen, J. Chhugani, P. Dubey, C. J. Hughes, D. Kim, S. Kumar, V. W. Lee, A. D. Nguyen, and M. Smelyanskiy. Convergence of Recognition, Mining, and Synthesis Workloads and Its Implications. *Proceedings of the IEEE*, 96(5):790–807, 2008. `https://doi.org/10.1109/JPROC.2008.917729`, Accessed March 24, 2017.

[22] J. Han and M. Orshansky. Approximate Computing: An Emerging Paradigm For Energy-Efficient Design. In *Proceedings of the IEEE European Test Symposium*, pages 1–6, 2013. `https://doi.org/10.1109/ETS.2013.6569370`, Accessed March 24, 2017.

[23] A. B. Kahng and S. Kang. Accuracy-Configurable Adder for Approximate Arithmetic Designs. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 820–825, 2012. `https://doi.org/10.1145/2228360.2228509`, Accessed March 24, 2017.

[24] N. Zhu, W. L. Goh, and K. S. Yeo. An Enhanced Low-power High-speed Adder for Error-tolerant Application. In *Proceedings of the International Symposium on Integrated Circuits*, pages 69–72, 2009. `http://ieeexplore.ieee.org/document/5403865/`, Accessed March 24, 2017.

[25] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. A Low Latency Generic Accuracy Configurable Adder. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 86:1–86:6, 2015. `https://doi.org/10.1145/2744769.2744778`, Accessed March 24, 2017.

[26] L. B. Soares, S. Bampi, and E. Costa. Approximate Adder Synthesis for Area- and Energy-efficient FIR Filters in CMOS VLSI. In *Proceedings of the IEEE International New Circuits and Systems Conference*, pages 1–4, 2015. `https://doi.org/10.1109/NEWCAS.2015.7182095`, Accessed March 24, 2017.

[27] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. On Reconfiguration-oriented Approximate Adder Design and Its Application. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 48–54, 2013. `https://doi.org/10.1109/ICCAD.2013.6691096`, Accessed March 24, 2017.

[28] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas. Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-computing Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(4):850–862, 2010. https://doi.org/10.1109/TCSI.2009.2027626, Accessed March 24, 2017.

[29] N. Zhu, W. L. Goh, and K. S. Yeo. An Enhanced Low-Power High-Speed Adder For Error-Tolerant Application. In *Proceedings of the International Symposium on Integrated Circuits*, pages 69–72, 2009. http://ieeexplore.ieee.org/document/5403865/, Accessed March 24, 2017.

[30] Y. Emre and C. Chakrabarti. Low Energy Motion Estimation via Selective Approximations. In *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 176–183, 2011. https://doi.org/10.1109/ASAP.2011.6043266, Accessed March 24, 2017.

[31] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. MACACO: Modeling and Analysis of Circuits for Approximate Computing. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 667–673, 2011. https://doi.org/10.1109/ICCAD.2011.6105401, Accessed March 24, 2017.

[32] E. Swartzlander. Truncated Multiplication with Approximate Rounding. In *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, volume 2, pages 1480–1483, 1999. https://doi.org/10.1109/ACSSC.1999.831996, Accessed March 24, 2017.

[33] K. J. Cho, K. C. Lee, J. G. Chung, and K. K. Parhi. Design of Low-Error Fixed-Width Modified Booth Multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(5):522–531, 2004. https://doi.org/10.1109/TVLSI.2004.825853, Accessed March 24, 2017.

[34] B. Shao and P. Li. Array-Based Approximate Arithmetic Computing: A General Model and Applications to Multiplier and Squarer Design. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(4):1081–1090, 2015. https://doi.org/10.1109/TCSI.2015.2388839, Accessed March 24, 2017.

[35] C. Liu, J. Han, and F. Lombardi. A Low-Power, High-Performance Approximate Multiplier with Configurable Partial Error Recovery. In *Proceedings of the IEEE Design, Automation, and Test in Europe*, pages 1–4, 2014. https://doi.org/10.7873/DATE.2014.108, Accessed March 24, 2017.

[36] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. New York, NY: Oxford University Press, 2000.

[37] M. D. Springer. *The Algebra of Random Variables*. New York, NY: Wiley, 1979.

[38] MARSYAS Data Sets. http://marsyasweb.appspot.com/download/data_sets/, Accessed March 24, 2017.

[39] G. I. Wirth, R. D. Silva, and B. Kaczer. Statistical Model for MOSFET Bias Temperature Instability Component due to Charge Trapping. *IEEE Transactions on Electron Devices*, 58(8):2743–2751, 2011. https://doi.org/10.1109/TED.2011.2157828, Accessed March 24, 2017.

[40] J.-J. Kim, R. Rao, J. Schaub, A. Ghosh, A. Bansal, K. Zhao, B. Linder, and J. Stathis. PBTI/NBTI Monitoring Ring Oscillator Circuits with On-Chip Vt Characterization and High Frequency AC Stress Capability. In *Proceedings of the IEEE Symposium on VLSI Circuits*, pages 224–225, 2011. http://ieeexplore.ieee.org/document/5986047/, Accessed March 24, 2017.

[41] S. Chakravarthi, A. Krishnan, V. Reddy, C. Machala, and S. Krishnan. A Comprehensive Framework for Predictive Modeling of Negative Bias Temperature Instability. In *Proceedings of the IEEE International Reliability Physics Symposium*, pages 273–282, 2004. https://doi.org/10.1109/RELPHY.2004.1315337, Accessed March 24, 2017.

[42] C. Schlunder, S. Aresu, G. Georgakos, W. Kanert, H. Reisinger, K. Hofmann, and W. Gustin. HCI vs. BTI? - Neither One's Out. In *Proceedings of the IEEE International Reliability Physics Symposium*, pages 2F.4.1–2F.4.6, 2012. https://doi.org/10.1109/IRPS.2012.6241797, Accessed March 24, 2017.

[43] X. Feng, P. Ren, Z. Ji, R. Wang, K. Sutaria, Y. Cao, and R. Huang. Novel Voltage Step Stress (VSS) Technique for Fast Lifetime Prediction of

Hot Carrier Degradation. In *Proceedings of the IEEE International Conference on Solid -State and Integrated Circuit Technology*, pages 1–3, 2014. https://doi.org/10.1109/ICSICT.2014.7021226, Accessed March 24, 2017.

[44] Reliability PTM model. http://ptm.asu.edu/reliability/, Accessed March 24, 2017.

[45] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. NBTI-Aware Synthesis of Digital Circuits. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 370–375, 2007. https://doi.org/10.1145/1278480.1278574, Accessed March 24, 2017.

[46] M. Agarwal, V. Balakrishnan, A. Bhuyan, K. Kim, B. C. Paul, W. Wang, B. Yang, Y. Cao, and S. Mitra. Optimized Circuit Failure Prediction for Aging: Practicality and Promise. In *Proceedings of the IEEE International Test Conference*, pages 1–10, 2008. https://doi.org/10.1109/TEST.2008.4700619, Accessed March 24, 2017.

[47] X. Wang, M. Tehranipoor, and R. Datta. Path-RO: A Novel On-Chip Critical Path Delay Measurement Under Process Variations. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 640–646, 2008. https://doi.org/10.1109/ICCAD.2008.4681644, Accessed March 24, 2017.

[48] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra. Circuit Failure Prediction and Its Application to Transistor Aging. In *Proceedings of the IEEE VLSI Test Symposium*, pages 277–286, 2007. https://doi.org/10.1109/VTS.2007.22, Accessed March 24, 2017.

[49] Y. Li, S. Makar, and S. Mitra. CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns. In *Proceedings of the IEEE Design, Automation, and Test in Europe*, pages 885–890, 2008. https://doi.org/10.1109/DATE.2008.4484786, Accessed March 24, 2017.

[50] Y. Li, O. Mutlu, D. S. Gardner, and S. Mitra. Concurrent Autonomous Self-Test for Uncore Components in System-on-Chips. In *Proceedings of the IEEE VLSI Test*

Symposium, pages 232–237, 2010. https://doi.org/10.1109/VTS.2010.5469571, Accessed March 24, 2017.

[51] T. Liu, C.-C. Chen, and L. Milor. Accurate Standard Cell Characterization and Statistical Timing Analysis using Multivariate Adaptive Regression Splines. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*, pages 272–279, 2015. https://doi.org/10.1109/ISQED.2015.7085438, Accessed March 24, 2017.

[52] M. Orshansky, L. Milor, L. Nguyen, G. Hill, Y. Peng, and C. Hu. Intra-field Gate CD Variability and its Impact on Circuit Performance. In *Technical Digest of the IEEE International Electron Devices Meeting*, pages 479–482, 1999. http://dx.doi.org/10.1109/IEDM.1999.824197, Accessed March 24, 2017.

[53] Y. Zheng, A. Basak, and S. Bhunia. CACI: Dynamic Durrent Analysis towards Robust Recycled Chip Identification. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 1–6, 2014. https://doi.org/10.1145/2593069.2593102, Accessed March 24, 2017.

[54] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula. Predictive Modeling of the NBTI Effect for Reliable Design. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 189–192, 2006. https://doi.org/10.1109/CICC.2006.320885, Accessed March 24, 2017.

[55] ISCAS 1989 Benchmarks. http://www.pld.ttu.ee/~maksim/benchmarks/iscas89/verilog/, Accessed March 24, 2017.

[56] ITC 1999 Benchmarks. http://www.cad.polito.it/downloads/tools/itc99.html, Accessed March 24, 2017.

[57] IWLS 2005 Benchmarks. http://iwls.org/iwls2005/, Accessed March 24, 2017.

[58] NanGate 45nm Open Cell Library. http://www.si2.org/openeda.si2.org/projects/nangatelib, Accessed March 24, 2017.

[59] Synopsys, Inc. Design Compiler. http://beethoven.ee.ncku.edu.tw/testlab/course/VLSIdesign_course/course_96/Tool/Design_Compiler%20_User_Guide.pdf, Accessed March 24, 2017.

[60] D. Lee, D. Blaauw, and D. Sylvester. Runtime Leakage Minimization through Probability-Aware Dual-V_t or Dual-t_{ox} Assignment. In *Proceedings of the IEEE Asia and South Pacific Design Automation Conference*, pages 399–404, 2005. https://doi.org/10.1145/1120725.1120884, Accessed March 24, 2017.

[61] Wolfram Research, Inc. Mathematica Version 10.0, 2014. https://reference.wolfram.com/language/ref/Manual.html, Accessed March 24, 2017.

[62] J. Miao, K. He, A. Gerstlauer, and M. Orshansky. Modeling and Synthesis of Quality-Energy Optimal Approximate Adders. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 728–735, 2012. https://doi.org/10.1145/2429384.2429542, Accessed March 24, 2017.

[63] J. Huang, J. Lach, and G. Robins. Analytic Error Modeling for Imprecise Arithmetic Circuits. In *Proceedings of the Workshop on Silicon Errors in Logic - System Effects*, 2011. http://128.143.137.29/~robins/papers/SELSE_2011_camera_final.pdf, Accessed March 24, 2017.

[64] J. Stolfi and L. de Figueiredo. An Introduction to Affine Arithmetic. *Trends in Applied and Computational Mathematics*, 4(3):297–312, 2003. http://dx.doi.org/10.5540/tema.2003.04.03.0297, Accessed March 24, 2017.

[65] J. Huang, J. Lach, and G. Robins. A Methodology for Energy-Quality Tradeoff Using Imprecise Hardware. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 504–509, 2012. https://doi.org/10.1145/2228360.2228450, Accessed March 24, 2017.

[66] W. T. J. Chan, A. B. Kahng, S. Kang, R. Kumar, and J. Sartori. Statistical Analysis and Modeling for Error Composition in Approximate Computation Circuits. In *Proceedings of the IEEE International Conference on Computer Design*, pages 47–53, 2013. https://doi.org/10.1109/ICCD.2013.6657024, Accessed March 24, 2017.

[67] Y. C. Liao, H. C. Chang, and C. W. Liu. Carry Estimation for Two's Complement Fixed-Width Multipliers. In *Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation*, pages 345–350, 2006. https://doi.org/10.1109/SIPS.2006.352606, Accessed March 24, 2017.

[68] N. R. Shanbhag, R. A. Abdallah, R. Kumar, and D. L. Jones. Stochastic Computation. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 859–864, 2010. https://doi.org/10.1145/1837274.1837491, Accessed March 24, 2017.

[69] S. Devadas, K. Keutzer, and J. White. Estimation of Power Dissipation in CMOS Combinational Circuits Using Boolean Function Manipulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(3):373–383, 1992. https://doi.org/10.1109/43.124424, Accessed March 24, 2017.

[70] A. V. Oppenheim and A. S. Willsky. *Signals and Systems*. New Jersey, NJ: Prentice-Hall, 1997.

[71] M. S. Nikulin. Hellinger distance. *Encyclopedia of Mathematics*. http://www.encyclopediaofmath.org/index.php/Hellinger_distance, Accessed March 24, 2017.

[72] S. Lee, D. Lee, K. Han, E. Shriver, L. K. John, and A. Gerstlauer. Statistical Quality Modeling of Approximate Hardware. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*, pages 163–168, 2016. https://doi.org/10.1109/ISQED.2016.7479194, Accessed March 24, 2017.

[73] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu. Joint Precision Optimization and High Level Synthesis for Approximate Computing. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 104.1–104.6, 2015. https://doi.org/10.1145/2744769.2744863, Accessed March 24, 2017.

[74] M. Frigo and S. G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. https://doi.org/10.1109/JPROC.2004.840301, Accessed March 24, 2017.

[75] S.-H. Cha. Comprehensive Survey on Distance/Similarity Measures Between Probability Density Functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1(4):300–307, 2007. http://www.gly.fsu.edu/~parker/geostats/Cha.pdf, Accessed March 24, 2017.

[76] R. Beran. Minimum Hellinger Distance Estimates for Parametric Models. *The Annals of Statistics*, 5(3):445–463, 1977. http://www.jstor.org/stable/2958896, Accessed March 24, 2017.

[77] J. M. Jou, S. R. Kuang, and R. Der Chen. Design of Low-Error Fixed-Width Multipliers for DSP Applications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(6):836–842, 1999. https://doi.org/10.1109/82.769795, Accessed March 24, 2017.

[78] L. Chen, J. Han, W. Liu, and F. Lombardi. Design of Approximate Unsigned Integer Non-restoring Divider for Inexact Computing. In *Proceedings of the 25th edition on ACM Great Lakes Symposium on VLSI*, pages 51–56, 2015. https://doi.org/10.1145/2742060.2742063, Accessed March 24, 2017.

[79] F. S. Snigdha, D. Sengupta, J. Hu, and S. S. Sapatnekar. Optimal Design of JPEG Hardware Under the Approximate Computing Paradigm. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 106:1–106:6, 2016. https://doi.org/10.1145/2897937.2898057, Accessed March 24, 2017.

[80] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. SALSA: Systematic Logic Synthesis of Approximate Circuits. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 796–801, 2012. https://doi.org/10.1145/2228360.2228504, Accessed March 24, 2017.

[81] K. Nepal, Y. Li, R. Bahar, and S. Reda. ABACUS: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 1–6, 2014. https://doi.org/10.7873/DATE.2014.374, Accessed March 24, 2017.

[82] G. Tzanetakis and P. Cook. Musical Genre Classification of Audio Signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002. `http://hdl.handle.net/1828/1344`, Accessed March 24, 2017.

Appendix A

Proof of Theorems

The proofs of the theorems outlined in this thesis are detailed here for enhanced readability.

Proof of Theorem 3.1:

The functions, $x_i(t)$, and their envelope, $x_M(t) = \max_i(x_i(t))$, are depicted in Fig. A.1(a).

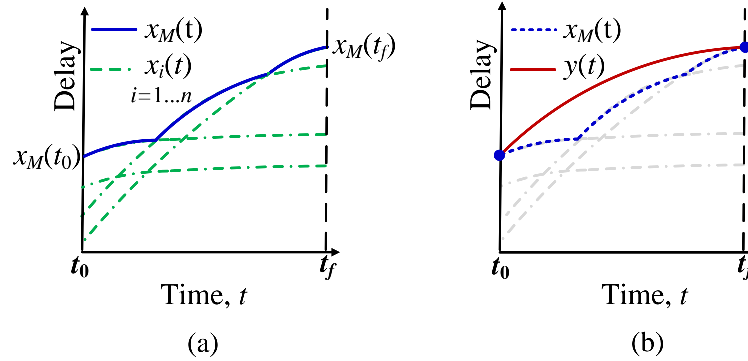


Figure A.1: (a) Maximum among the $x_i(t)$ functions, denoted by $x_M(t)$, (b) Smooth upper-bounded estimate of $x_M(t)$, denoted by $y(t)$.

Since for each $x_i(t)$, we have:

$$x_i(t_f) = x_i(t_0) + \theta_1^i \Delta f(t_f) + \theta_2^i \Delta g(t_f),$$

we obtain the relationship between each θ_1^i and θ_2^i as:

$$\theta_2^i = \frac{\Delta x_i(t_f) - \theta_1^i \Delta f(t_f)}{\Delta g(t_f)} \tag{A.1}$$

where $\Delta x_i(t) = x_i(t) - x_i(t_0)$.

To ensure that $y(t)$ as depicted in Fig. A.1(b) is a tight upper bound on $x_M(t)$, $y(t)$ should satisfy the following conditions:

$$y(t) = x_M(t), \quad t = t_0, t_f \tag{A.2}$$

$$y(t) \geq x_i(t), \quad \forall t \in [t_0, t_f] \tag{A.3}$$

Substituting $t = t_f$ in the definition of $y(t)$ in Eq. (3.1), and using Eq. (A.2), we obtain:

$$y(t_f) = x_M(t_f) = x_M(t_0) + \theta_1^M \Delta f(t_f) + \theta_2^M \Delta g(t_f) \tag{A.4}$$

Hence, θ_2^M is computed as:

$$\theta_2^M = \frac{\Delta x_M(t_f) - \theta_1^M \Delta f(t_f)}{\Delta g(t_f)} \tag{A.5}$$

where $\Delta x_M(t) = x_M(t) - x_M(t_0)$. Using Eqs. (A.5) and (A.1), $y(t)$ and each $x_i(t)$ can be rewritten as:

$$y(t) = x_M(t_0) + \theta_1^M \Delta f(t) + \left(\frac{\Delta x_M(t_f) - \theta_1^M \Delta f(t_f)}{\Delta g(t_f)} \right) \Delta g(t)$$

$$x_i(t) = x_i(t_0) + \theta_1^i \Delta f(t) + \left(\frac{\Delta x_i(t_f) - \theta_1^i \Delta f(t_f)}{\Delta g(t_f)} \right) \Delta g(t)$$

Hence, the error, $\delta_i(t) = y(t) - x_i(t)$, is obtained as:

$$\begin{aligned} \delta_i(t) = \Delta x_{M,i}(t_0) \left(1 - \frac{\Delta g(t)}{\Delta g(t_f)} \right) + \Delta x_{M,i}(t_f) \frac{\Delta g(t)}{\Delta g(t_f)} \\ + (\theta_1^M - \theta_1^i) \Delta f(t_f) \left(\frac{\Delta f(t)}{\Delta f(t_f)} - \frac{\Delta g(t)}{\Delta g(t_f)} \right) \end{aligned} \tag{A.6}$$

where $\Delta x_{M,i}(t) = x_M(t) - x_i(t)$. Analyzing each addend in Eq. (A.6) we make the following conclusions:

- Since $x_M(t) = \max_i(x_i(t))$, by definition, both $\Delta x_{M,i}(t_0)$ and $\Delta x_{M,i}(t_f)$ are nonnegative. Since $\Delta g(t) = t^{n_2} - t_0^{n_2}$ with $n_2 \in (0, 1)$, $0 \leq \frac{\Delta g(t)}{\Delta g(t_f)} \leq 1$, $\forall t \in [t_0, t_f]$. Hence, the first two addends in Eq. (A.6) are positive.

- For either form of $f(t) = t^{n_1}$ or $a + b\log t$, $\Delta f(t_f) \geq 0$, and by definition, $\theta_1^M \geq \theta_1^i$ in the third addend.

- Finally, to analyze the last parenthetical expression in Eq. (A.6) for both $f(t) = t^{n_1}$ and $a + b\log t$, we represent it by $e(t)$ for $t \in [t_0, t_f]$, i.e.,

$$e(t) = \left(\frac{\Delta f(t)}{\Delta f(t_f)} - \frac{\Delta g(t)}{\Delta g(t_f)} \right) \tag{A.7}$$

  Clearly, $e(t_0) = e(t_f) = 0$, and $e(t)$ is continuous, differentiable and real-valued in $[t_0, t_f]$. Hence, by Rolle's Theorem, there exists at least one point, $t = t_1 \in (t_0, t_f)$ for which $e'(t_1) = 0$, where $e'(t)$ is the derivative of $e(t)$ with respect to t. The values of t_1 are obtained as:

$$t_1 = \begin{cases} \left(\frac{n_1}{n_2} \left(\frac{t_f^{n_2} - t_0^{n_2}}{t_f^{n_1} - t_0^{n_1}} \right) \right)^{1/(n_2 - n_1)}, & \text{for } f(t) = t^{n_1} \\ \left(\frac{t_f^{n_2} - t_0^{n_2}}{n_2 \log(t_f/t_0)} \right)^{1/n_2}, & \text{for } f(t) = a + b\log t \end{cases}$$

  There is exactly one solution for $e'(t_1) = 0$ for a specific form of $f(t)$, when $n_1,\ n_2 \in (0, 1)$ and $n_2 > n_1$. To observe whether t_1 corresponds to a local maximum or minimum, we further obtain the second derivative of $e(t)$ with respect to t, at t_1, and simplify it as:

$$e''(t_1) = \begin{cases} \frac{-n_1(n_2 - n_1)t_1^{n_1 - 2}}{t_f^{n_1} - t_0^{n_1}}, & \text{for } f(t) = t^{n_1} \\ \frac{-n_2}{t_1^2 \log(t_f/t_0)}, & \text{for } f(t) = a + b\log t \end{cases}$$

  Clearly, $e''(t_1) < 0$ for both forms of $f(t)$, implying that t_1 is a maximum, or $e(t)$ is maximum at a single $t = t_1 \in (t_0, t_f)$. Hence, $e(t) \geq 0$ in the entire interval, $[t_0, t_f]$.

Being sum of all positive numbers, $\delta_i(t) \geq 0$ in Eq. (A.6), $\forall t \in [t_0, t_f]$. Hence, $y(t)$ as defined in Eq. (3.1), is indeed a smooth upper bound for the maximum of $x_1(t), \cdots$, $x_n(t)$. □

Proof of Theorem 3.2:

We first compute the K_B value of the UofM-based delay trajectory of C, or K_B^C, as the maximum among the K_B values of all near-critical paths. Let us denote this by $K_B^{p_m}$

for the path, $X = p_m$, of C in Eq. (3.12). Next, we obtain the K_B value of the ROSC delay trajectory, which is simply the K_B value of its single path, r, or K_B^r. Hence, using Eq. (3.12), K_B^C and K_B^R are obtained as:

$$K_B^C = K_B^{p_m} = \mathcal{K}_B^{p_m} F_B(V_{dd}, T)\Delta f(t) \tag{A.8}$$

$$K_B^R = K_B^r = \mathcal{K}_B^r F_B(V_{dd}, T)\Delta f(t) \tag{A.9}$$

where $\Delta f(t) = f(t) - f(t_0)$, and $F_B(.)$ is defined in Eq. (3.14). All terms related to V_{dd} and T in K_B^C and K_B^R in Eqs. (A.8) and (A.9) are identical. Hence, for $\xi_B^C = \frac{K_B^C}{K_B^R}$, these terms cancel out in the numerator and denominator, implying that the degradation ratio, ξ_B^C, is independent of V_{dd} and T.

For ξ_H^C, we first obtain the K_H value of the UofM-based delay trajectory of C, or K_H^C using Theorem 3.1 as:

$$\begin{aligned} K_H^C &= \frac{D^C(t_f) - D^C(t_0) - K_B^C \Delta f(t_f)}{\Delta g(t_f)} \\ &= \frac{D^{p_f}(t_f) - D^{p_0}(t_0) - \Delta D_B^{p_m}(t_f)}{\Delta g(t_f)} \end{aligned} \tag{A.10}$$

where $\Delta g(t) = g(t) - g(t_0)$, and $D^C(t_f)$ and $D^C(t_0)$ are, respectively, the CUT delays at times, t_f and t_0, when the paths, p_f and p_0, are critical in C. Hence, $D^C(t_f)$ and $D^C(t_0)$ have been replaced by $D^{p_f}(t_f)$ and $D^{p_0}(t_0)$, respectively, in the second line of Eq. (A.10). Similarly, since $K_B^C = K_B^{p_m}$ from Eq. (A.8) and $K_B^{p_m}\Delta f(t) = \Delta D_B^{p_m}(t)$ from Eq. (3.12), $K_B^C \Delta f(t_f)$ has been replaced by $\Delta D_B^{p_m}(t_f)$ in Eq. (A.10). Next, we rewrite $D^{p_f}(t_f)$ in Eq. (A.10) as:

$$D^{p_f}(t_f) = D^{p_f}(t_0) + \Delta D_B^{p_f}(t_f) + \Delta D_H^{p_f}(t_f) \tag{A.11}$$

where $\Delta D_B^X(t)$ and $\Delta D_H^X(t)$ are the delay shifts for any path, X, due to BTI and HCI, respectively, defined in Eqs. (3.12) and (3.13). Hence, by substituting this $D^{p_f}(t_f)$ in Eq. (A.10), we can simplify K_H^C as:

$$K_H^C = \frac{\Delta D^{p_f,p_0}(t_0) + (\Delta D_B^{p_f}(t_f) - \Delta D_B^{p_m}(t_f)) + \Delta D_H^{p_f}(t_f)}{\Delta g(t_f)} \tag{A.12}$$

where $\Delta D^{p_f,p_0}(t_0) = D^{p_f}(t_0) - D^{p_0}(t_0)$. Next, we use the following simplifications:

- $(\Delta D_B^{p_f}(t_f) - \Delta D_B^{p_m}(t_f)) = \Delta \mathcal{K}_B^{p_f,p_m} F_B(V_{dd},T)\Delta f(t_f)$ using Eq. (3.12), with $\Delta \mathcal{K}_B^{p_f,p_m} = (\mathcal{K}_B^{p_f} - \mathcal{K}_B^{p_m})$,
- $\Delta D_H^{p_f}(t_f) = \mathcal{K}_H^{p_f} F_H(V_{dd},T)\Delta g(t_f)$ from Eq. (3.13), with $F_H(.)$ defined in Eq. (3.15).

Now to compute ξ_H^C, we need the K_H value of the ROSC delay trajectory, which is simply equal to K_H^r from Eq. (3.13), where $K_H^r = \mathcal{K}_H^r F_H(V_{dd},T)$. Hence, using this K_H^r, and K_H^C from Eq. (A.12), we rewrite $\xi_H^C = \frac{K_H^C}{K_H^r}$ as:

$$\xi_H^C = \frac{\Delta D^{p_f,p_0}(t_0) + \Delta \mathcal{K}_B^{p_f,p_m} F_B(V_{dd},T)\Delta f(t_f)}{\mathcal{K}_H^r F_H(V_{dd},T)\Delta g(t_f)} + \frac{\mathcal{K}_H^{p_f}}{\mathcal{K}_H^r} \tag{A.13}$$

This is the expression of ξ_H^C as mentioned in Theorem 3.2. □

Proof of Theorem 4.1:

We present the proof by mathematical induction in terms of the number, N, of measurement instants.

Let $T_M = \{t_{m_0}, t_{m_1}, \cdots, t_{m_{N-1}}, t_{m_N}\}$ be the set of $N+1$ time instants in $[t_0, t_f]$, where the first N represent the measurement instants, with $t_{m_0} = t_0$ and $t_{m_N} = t_f$ denoting the beginning and end of lifetime of the CUT, C, respectively.

Base Case, $N = 1$:

Here, $T_M = \{t_{m_0}, t_{m_1} = t_f\}$, and the true delay measurement is available at a single instant $t = t_{m_0} = t_0$. Under this condition, $D_{re}^C(t) = D_{\mathrm{UofM}}^C(t)$, which has been proven in Sec. 3.1.1 to upper-bound the true circuit delay for $t \in [t_0, t_f]$.

Inductive Hypothesis, $N = r$:

When $T_M = \{t_{m_0}, t_{m_1}, \cdots, t_{m_{r-1}}, t_{m_r}\}$, we assume that $D_{re}^C(t)$, as defined in Eq. (4.1), together with (4.2) or (4.3), provides an upper bound on the true delay for all r measurement instants.

Inductive Step, $N = r+1$:

Here $T_M = \{t_{m_0}, t_{m_1}, \cdots, t_{m_{r-1}}, t_{m_r}, t_{m_{r+1}}\}$. Using the results of the inductive hypothesis, $D_{re}^C(t)$ forms an upper bound on true delay of the circuit for $t \in [t_0, t_{m_r}]$. At

$t = t_{m_r}$, $D^C_{re}(t) = D^C_a(t)$. Now for $t \in [t_{m_r}, t_{m_{r+1}}]$, we consider each of the bounds in Theorem 4.1.

(I) $D^{C,I}_{re}(t) = D^C_a(t_{m_r}) + K^{p_x}_B \Delta f_r(t) + K^{p_x}_H \Delta g_r(t)$

Hence, $D^{C,I}_{re}(t) = D^C_a(t_{m_r}) + (D^{p_x}_{pre}(t) - D^{p_x}_{pre}(t_{m_r}))$, where p_x is a near-critical path of the CUT with maximum delay degradation from t_{m_r} to $t_{m_{r+1}}$ under the worst-case workload characterized at the presilicon stage, among all near-critical paths. Since by definition of $D^{p_x}_{pre}(t)$, no other path under any real workload scenario undergoes as much degradation as p_x, the circuit delay change cannot exceed that of p_x over the interval $t \in [t_{m_r}, t_{m_{r+1}}]$. Hence, $D^C_{re}(t)$ forms an upper bound on delay trajectory under any real workload for $t \in [t_{m_r}, t_{m_{r+1}}]$.

(II) $D^{C,II}_{re}(t) = D^C_a(t_{m_r}) + K^{II}_B \Delta f_r(t) + K^{II}_H \Delta g_r(t)$

Now to ensure that $D^C_{re}(t)$ is an upper-bounding curve between t_{m_r} and $t_{m_{r+1}}$ over all paths $p_i \in S_{NC}$, we may simply apply Theorem 3.1, setting $t_0 = t_{m_r}$, $x_M(t_{m_j}) = D^C_a(t_{m_j})$, and $x_M(t_{m_{j+1}}) = D^C_{pre}(t_{m_{j+1}})$, and this yields the result. □

Proof of Theorem 7.1:

We rewrite the optimization problem from (7.5) as:

$$\max \ S = y_1 + \sum_{i=2}^{T} y_i \tag{A.14}$$

$$\text{s.t } a(2^{by_1} - 1)\beta_1 + \sum_{i=2}^{T} a(2^{by_i} - 1)\beta_i = m \tag{A.15}$$

Hence, using Eq. (A.15), we obtain y_1 as:

$$y_1 = \frac{1}{b}\log_2 \theta \tag{A.16}$$

$$\text{where } \theta = [m + a\beta_1 - a\sum_{i=2}^{T}(2^{by_i} - 1)\beta_i]/a\beta_1 \tag{A.17}$$

We rewrite, $S = \frac{1}{b}\log_2 \theta + \sum_{i=2}^{T} y_i$, by substituting y_1 from Eq. (A.16) in Eq. (A.14). Since S has to be maximized, we set $\frac{\partial S}{\partial y_i} = 0$, to obtain:

$$2^{by_i} = \frac{\theta\beta_1}{\beta_i} \tag{A.18}$$

Substituting 2^{by_i} in Eq. (A.17) and simplifying, we obtain:

$$\theta\beta_1 = \left(m + a\sum_{i=1}^{T}\beta_i\right)(aT)^{-1} \tag{A.19}$$

Finally, by substituting $\theta\beta_1$ in Eq. (A.18), we obtain the result. □

Proof of Theorem 7.2

Comparing the impact of adding one more approximate LSB to two nodes, n_1 and n_2, with β values, β_1 and β_2, respectively, where $\beta_1 < \beta_2$, we observe that n_1 introduces lower error in the DAG compared to n_2. In other words, for the same increase in approximation, the total error incorporated is lower if we start increasing the number of approximate LSBs in the nodes in the order of their increasing β values. Hence, we renumber the nodes in increasing order of the β values in Theorem 7.2, so that $\hat{y}_i = \lfloor X - \frac{1}{b}\log_2\beta_i + 1\rfloor$ for the first K nodes, while for the rest, $\hat{y}_i = \lfloor X - \frac{1}{b}\log_2\beta_i\rfloor$, $i \in [1, T]$. The new error variance with the increased number of approximate LSBs should satisfy the error constraint, m, in (7.5), such that,

$$\sum_{i=1}^{K}(2^{b(X-\frac{1}{b}\log_2\beta_i+1)} - 1)\beta_i + \sum_{i=K+1}^{T}(2^{b(X-\frac{1}{b}\log_2\beta_i)} - 1)\beta_i = \frac{m}{a}$$

Simplifying $\left(2^{b(X-\frac{1}{b}\log_2\beta_i)} - 1\right)\beta_i$ as $2^{bX} - \beta_i$, we obtain:

$$\frac{m}{a} = \sum_{i=1}^{K}(2^{b(X+1)} - \beta_i) + \sum_{i=K+1}^{T}(2^{bX} - \beta_i) \tag{A.20}$$

Expanding the right hand side of Eq. (A.20), we obtain:

$$m = a2^{bX}(2^b - 1)K - a\sum_{i=1}^{T}\beta_i + a2^{bX}T \tag{A.21}$$

Hence, K is obtained by simplifying the above equation. Additionally, $K < T$, since starting with $\lfloor\widetilde{y}_i\rfloor$, we can never increase all the $\lfloor\widetilde{y}_i\rfloor$s by one and remain in the feasible region, because such an increment will lead to $\lceil\widetilde{y}_i\rceil$ ($\lceil.\rceil$ being the *ceiling* function), and if it were in the feasible region, Theorem 7.1 would have chosen that solution over the fractional solution, $\widetilde{y}_i$. □

Appendix B

Miscellaneous Derivations

Derivations of various terms used in this thesis are detailed here for enhanced readability.

Derivation of E in Eq. (5.12):

In the expansion of Eq. (5.11), E is the magnitude of the exponent of z which corresponds to the term with the most negative exponent of z. Let this term be called T_E, where

$$T_E = \left(\prod_{i=1}^{N} \prod_{j=1}^{N-1} e_M^{(ij)} \right) z^{-E} \tag{B.1}$$

where

$$\begin{aligned} E = & M(2^1 + \cdots + 2^{N-1}) + M(2^2 + \cdots + 2^N) + \cdots \\ & + M(2^3 + \cdots + 2^{N+1}) + \cdots + M(2^N + \cdots + 2^{2N-2}) \end{aligned} \tag{B.2}$$

Simplifying, E is obtained as:

$$E = (2^{N-1} - 1)(2^N - 1)M \tag{B.3}$$

Derivation of $\mathcal{E}$ in Eq. (6.15):

In the expansion of Eq. (6.14), $\mathcal{E}$ is the magnitude of the exponent of z which corresponds to the term with the most negative exponent of z. Let this term be called $T_{\mathcal{E}}$, where

$$\begin{aligned} T_{\mathcal{E}} &= \left(\prod_{i=1}^{N-1} e_M^{(i)} z^{-2^{i-1}M} \right) x_{-1}^N z^{-2^{N-1}} \\ &= \left(x_{-1}^N \prod_{i=1}^{N-1} e_M^{(i)} \right) z^{-M(2^0+2^1+\cdots+2^{N-2})} z^{-2^{N-1}} \\ &= \left(x_{-1}^N \prod_{i=1}^{N-1} e_M^{(i)} \right) z^{-\mathcal{E}} \end{aligned} \tag{B.4}$$

Hence, $\mathcal{E}$ is obtained by adding the exponents of z in $T_{\mathcal{E}}$, as:

$$\mathcal{E} = M(2^{N-1} - 1) + 2^{N-1} \tag{B.5}$$

Simplifying, we obtain, $\mathcal{E} = 2^{N-1}(M+1) - M$.

Derivation of $\mathcal{F}$ in Eq. (6.30):

In the expansion of Eq. (6.29), $\mathcal{F}$ is the magnitude of the exponent of z which corresponds to the term with the most negative exponent of z. Let this term be called $T_{\mathcal{F}}$, where

$$\begin{aligned} T_{\mathcal{F}} &= \prod_{i=1}^{N-1} \left(\prod_{j=1}^{N-i-1} e_M^{(ij)} z^{-2^{i+j-1}M} \right) \left(x_{-1}^{(i(N-i))} z^{-2^{N-1}} \right) \\ &= \left(\prod_{i=1}^{N-1} \prod_{j=1}^{N-i-1} e_M^{(ij)} x_{-1}^{(i(N-i))} \right) z^{-\mathcal{F}} \end{aligned} \tag{B.6}$$

where

$$\begin{aligned} \mathcal{F} &= M(2^1 + \cdots + 2^{N-2}) + M(2^2 + \cdots + 2^{N-2}) + \cdots \\ &\quad + M(2^{N-3} + 2^{N-2}) + M(2^{N-2}) + (N-1)2^{N-1} \end{aligned} \tag{B.7}$$

Simplifying, $\mathcal{F}$ is obtained as:

$$\mathcal{F} = 2^{N-1}(MN - 3M + N - 1) + 2M \tag{B.8}$$

Derivation of PMF of $A_t\Delta B$ and $B_t\Delta A$ in Eq. (6.39):

As evident from Eqs. (6.33) and (6.34), the inputs (input errors) range from $-E$ to E ($-F$ to F). However, since by definition, the Mellin transform is only applicable on positive random variables, the domains of A_t, B_t, ΔA, and ΔB must be split into positive and non-positive parts, and treated separately. Hence, we rewrite the PMFs in Eqs. (6.33) and (6.34) as:

$$\begin{aligned} f_{A_t}(n) &= \sum_{-E}^{-1} p_k^A \delta(n-k) + \sum_{1}^{E} p_k^A \delta(n-k) + p_0^A \delta(n) \\ &= f_{A_{neg}}(n) + f_{A_{pos}}(n) + f_A(0) \end{aligned} \tag{B.9}$$

$$\begin{aligned} f_{B_t}(n) &= \sum_{-E}^{-1} p_k^B \delta(n-k) + \sum_{1}^{E} p_k^B \delta(n-k) + p_0^B \delta(n) \\ &= f_{B_{neg}}(n) + f_{B_{pos}}(n) + f_B(0) \end{aligned} \tag{B.10}$$

$$\begin{aligned} f_{\Delta A}(n) &= \sum_{-F}^{-1} p_k^{\Delta A} \delta(n-k) + \sum_{1}^{F} p_k^{\Delta A} \delta(n-k) + p_0^{\Delta A} \delta(n) \\ &= f_{\Delta A_{neg}}(n) + f_{\Delta A_{pos}}(n) + f_{\Delta A}(0) \end{aligned} \tag{B.11}$$

$$\begin{aligned} f_{\Delta B}(n) &= \sum_{-F}^{-1} p_k^{\Delta B} \delta(n-k) + \sum_{1}^{F} p_k^{\Delta B} \delta(n-k) + p_0^{\Delta B} \delta(n) \\ &= f_{\Delta B_{neg}}(n) + f_{\Delta B_{pos}}(n) + f_{\Delta B}(0) \end{aligned} \tag{B.12}$$

where p_k^X represents the probability of the variable, X, to be k, $X \in \{A_t, B_t, \Delta A, \Delta B\}$. The new variables, X_{neg} or X_{pos}, correspond to the negative and positive parts of X, and will be used to obtain the PMFs of $A_t\Delta B$ (and $B_t\Delta A$).

To simplify our notations, let us define a new variable, $C_1 = A_t\Delta B$, whose PMF, $f_{C_1}(n)$, comprises the following four functions, each representing the probability of C_1 to be n based on the positivity or negativity of A_t and ΔB.

- ➢ $f_{C_1,A_t\geq 0,\Delta B\geq 0}(n)$ is the probability of C_1 to be n when both A_t and ΔB are positive.
- ➢ $f_{C_1,A_t\geq 0,\Delta B<0}(n)$ is the probability of C_1 to be n when A_t (ΔB) is positive (negative).

- $f_{C_1,A_t<0,\Delta B\geq 0}(n)$ is the probability of C_1 to be n when A_t (ΔB) is negative (positive).
- $f_{C_1,A_t<0,\Delta B<0}(n)$ is the probability of C_1 to be n when both A_t and ΔB are negative.

If each of the above functions can be represented by the impulse train as shown in Figs. 6.9(a)–(d), for example, then $f_{C_1}(n)$ is simply the summation of them, as depicted in Fig. 6.9(e), and hence, given by:

$$f_{C_1}(n) = f_{C_1,A_t\geq 0,\Delta B\geq 0}(n) + f_{C_1,A_t\geq 0,\Delta B<0}(n) + f_{C_1,A_t<0,\Delta B\geq 0}(n) + f_{C_1,A_t<0,\Delta B<0}(n) \tag{B.13}$$

We had separated the negative and positive parts of A_t and ΔB in Eqs. (B.9) and (B.12), to define new random variables, X_{neg} and X_{pos} where $X = \{A_t, \Delta B\}$, and since the Mellin transform can only be applied when the random variables are positive, we inverse the domain of the PMFs when either of the multiplicands, A_t or ΔB, is negative. We also consider the cases differently when either of them is zero. Hence, we perform the following replacements to the terms in Eq. (B.13):

$$f_{C_1,A_t\geq 0,\Delta B\geq 0}(n) = f_{C_1,A_t=A_{pos},\Delta B=\Delta B_{pos}}(n) + f_{C_1,A_t=0,\Delta B=\Delta B_{pos}}(n) + f_{C_1,A_t=\Delta A_{pos},\Delta B=0}(n) \tag{B.14}$$

$$f_{C_1,A_t\geq 0,\Delta B<0}(n) = \mathcal{I}(f_{C_1,A_t=A_{pos},\Delta B=-\Delta B_{neg}}(n) + f_{C_1,A_t=0,\Delta B=-\Delta B_{neg}}(n)) \tag{B.15}$$

$$f_{C_1,A_t<0,\Delta B\geq 0}(n) = \mathcal{I}(f_{C_1,A_t=-A_{neg},\Delta B=\Delta B_{pos}}(n) + f_{C_1,A_t=-A_{neg},\Delta B=0}(n)) \tag{B.16}$$

$$f_{C_1,A_t<0,\Delta B<0}(n) = f_{C_1,A_t=-A_{neg},\Delta B=-\Delta B_{neg}}(n) \tag{B.17}$$

where $\mathcal{I}(.)$ represents the operation of taking the mirror image of the enclosed function about the $n = 0$ axis in Fig. 6.9. We can use the Mellin transform to rewrite Eqs. (B.14) to (B.17) as:

$$f_{C_1,A_t\geq 0,\Delta B\geq 0}(n) = \mathcal{M}^{-1}\big(\mathcal{M}[f_{A_{pos}}(n);s]\mathcal{M}[f_{\Delta B_{pos}}(n);s]\big) + \psi_1 \tag{B.18}$$

$$f_{C_1,A_t\geq 0,\Delta B<0}(n) = \mathcal{I}\left(\mathcal{M}^{-1}\big(\mathcal{M}[f_{A_{pos}}(n);s]\mathcal{M}[f_{-\Delta B_{neg}}(n);s]\big) + \psi_2\right) \tag{B.19}$$

$$f_{C_1,A_t<0,\Delta B\geq 0}(n) = \mathcal{I}\left(\mathcal{M}^{-1}\big(\mathcal{M}[f_{-A_{neg}}(n);s]\mathcal{M}[f_{\Delta B_{pos}}(n);s]\big) + \psi_3\right) \tag{B.20}$$

$$f_{C_1,A_t<0,\Delta B<0}(n) = \mathcal{M}^{-1}\big(\mathcal{M}[f_{-A_{neg}}(n);s]\mathcal{M}[f_{-\Delta B_{neg}}(n);s]\big) \tag{B.21}$$

where $\mathcal{M}^{-1}(.)$ denotes the inverse Mellin transform, and ψ_i, $i = 1 \cdots 3$, covers the cases when A_t and ΔB are zeros in Eqs. (B.14) to (B.16). Detailed solutions of Eqs. (B.18) to (B.21) using Eqs. (B.9) and (B.12), are provided below:

$\underline{f_{C_1,A_t\geq 0,\Delta B\geq 0}(n)}$: The two terms in Eq. (B.18) can be written as,

$$\begin{aligned}
&\mathcal{M}^{-1}\big(\mathcal{M}[f_{A_{pos}}(n);s]\mathcal{M}[f_{\Delta B_{pos}}(n);s]\big)\\
&= \mathcal{M}^{-1}\left((\sum_{k=1}^{E} p_k^A k^{s-1})(\sum_{k=1}^{F} p_k^{\Delta B} k^{s-1})\right)\\
&= \mathcal{M}^{-1}\left((\sum_{k=1}^{EF} c_k^{pp} k^{s-1})\right) = \sum_{k=1}^{EF} c_k^{pp}\delta(n-k) \qquad \text{(B.22)}
\end{aligned}$$

$$\text{and } \psi_1 = c_0^{pp}\delta(n) \qquad \text{(B.23)}$$

where c_k^{pp} can be computed as:

$$c_k^{pp} = \begin{cases} p_0^A \sum_{k=0}^{F} p_k^{\Delta B} + p_0^{\Delta B}\sum_{k=0}^{E} p_k^A - p_0^A p_0^{\Delta B}, & k = 0 \\ \sum_{i \text{ factor of } k} p_i^A p_{k/i}^{\Delta B}, & \text{otherwise} \end{cases}$$

Thus $f_{C_1,A_t\geq 0,\Delta B\geq 0}(n)$ from Eq. (B.14) is obtained as:

$$f_{C_1,A_t\geq 0,\Delta B\geq 0}(n) = \sum_{k=0}^{EF} c_k^{pp}\delta(n-k) \qquad \text{(B.24)}$$

$\underline{f_{C_1,A_t\geq 0,\Delta B<0}(n)}$: The two terms inside $\mathcal{I}(.)$ in Eq. (B.19) can be written as,

$$\begin{aligned}
&\mathcal{M}^{-1}\left(\mathcal{M}[f_{A_{pos}}(n);s]\mathcal{M}[f_{-\Delta B_{neg}}(n);s]\right)\\
&= \mathcal{M}^{-1}\left((\sum_{k=1}^{E} p_k^A k^{s-1})(\sum_{k=1}^{F} p_{-k}^{\Delta B} k^{s-1})\right)\\
&= \mathcal{M}^{-1}\left((\sum_{k=1}^{EF} c_k^{pn} k^{s-1})\right) = \sum_{k=1}^{EF} c_k^{pn}\delta(n-k) \qquad \text{(B.25)}
\end{aligned}$$

$$\text{and } \psi_2 = c_0^{pn}\delta(n) \qquad \text{(B.26)}$$

where c_k^{pn} can be computed as:

$$c_k^{pn} = \begin{cases} p_0^A \sum_{k=1}^{F} p_{-k}^{\Delta B}, & k = 0 \\ \sum_{i \text{ factor of } k} p_i^A p_{-k/i}^{\Delta B}, & \text{otherwise} \end{cases}$$

Thus $f_{C_1,A_t\geq 0,\Delta B<0}(n)$ from Eq. (B.15) is obtained as:

$$f_{C_1,A_t\geq 0,\Delta B<0}(n) = \mathcal{I}\left(\sum_{k=0}^{EF} c_k^{pn}\delta(n-k)\right) = \sum_{k=0}^{EF} c_k^{pn}\delta(n+k) \tag{B.27}$$

$\underline{f_{C_1,A_t<0,\Delta B\geq 0}(n)}$: The two terms inside $\mathcal{I}(.)$ in Eq. (B.20) can be written as,

$$\begin{aligned}
&\mathcal{M}^{-1}\left(\mathcal{M}[f_{-A_{neg}}(n);s]\mathcal{M}[f_{\Delta B_{pos}}(n);s]\right)\\
&= \mathcal{M}^{-1}\left((\sum_{k=1}^{E} p_{-k}^{A} k^{s-1})(\sum_{k=1}^{F} p_{k}^{\Delta B} k^{s-1}) \right)\\
&= \mathcal{M}^{-1}\left((\sum_{k=1}^{EF} c_k^{np} k^{s-1}) \right) = \sum_{k=1}^{EF} c_k^{np}\delta(n-k)
\end{aligned} \tag{B.28}$$

$$\text{and } \psi_3 = c_0^{np}\delta(n) \tag{B.29}$$

where c_k^{np} can be computed as:

$$c_k^{pn} = \begin{cases} p_0^{\Delta B}\sum_{k=1}^{E} p_{-k}^{A}, & k=0 \\ \sum_{i \text{ factor of } k} p_{-i}^{A} p_{k/i}^{\Delta B}, & \text{otherwise} \end{cases}$$

Thus $f_{C_1,A_t<0,\Delta B\geq 0}(n)$ from Eq. (B.16) is obtained as:

$$f_{C_1,A_t<0,\Delta B\geq 0}(n) = \mathcal{I}\left(\sum_{k=0}^{EF} c_k^{np}\delta(n-k)\right) = \sum_{k=0}^{EF} c_k^{np}\delta(n+k) \tag{B.30}$$

$\underline{f_{C_1,A_t<0,\Delta B<0}(n)}$: Eq. (B.21) can be written as,

$$\begin{aligned}
&\mathcal{M}^{-1}\left(\mathcal{M}[f_{-A_{neg}}(n);s]\mathcal{M}[f_{-\Delta B_{neg}}(n);s]\right)\\
&= \mathcal{M}^{-1}\left((\sum_{k=1}^{E} p_{-k}^{A} k^{s-1})(\sum_{k=1}^{F} p_{-k}^{\Delta B} k^{s-1}) \right)\\
&= \mathcal{M}^{-1}\left((\sum_{k=1}^{EF} c_k^{nn} k^{s-1}) \right) = \sum_{k=1}^{EF} c_k^{nn}\delta(n-k)
\end{aligned} \tag{B.31}$$

where c_k^{nn} can be computed as:

$$c_k^{nn} = \sum_{i \text{ factor of } k} p_{-i}^{A} p_{-k/i}^{\Delta B}$$

Since A_t and ΔB are strictly negative in this case, probability of C_1 to be zero is zero, and $f_{C_1,A_t<0,\Delta B<0}(n)$ from Eq. (B.16) is obtained as:

$$f_{C_1,A_t<0,\Delta B<0}(n) = \sum_{k=1}^{EF} c_k^{nn} \delta(n+k) \tag{B.32}$$

The impulse trains in Eqs. (B.24) to (B.32) are schematically depicted in Figs. 6.9(a)–(d), and can be simply added to obtain $f_{C_1}(n) = f_{A_t \Delta B}(n)$ as shown in Fig. 6.9(e).

Finally, by exchanging A_t and ΔB with B_t and ΔA, respectively, in the above discussion, and using the definitions of X_{neg} and X_{pos}, $X \in \{B_t, \Delta A\}$ from Eqs. (B.10) and (B.11), respectively, we can obtain the PMF, $f_{B_t \Delta A}(n)$, of $B_t \Delta A$ too.