A Statistical Approach to Stochastic Computing Design and Analysis

by

Timothy James Baker

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2023

Timothy James Baker

bakertim@umich.edu

ORCID iD: 0000-0001-9427-3840

DEDICATION

To my parents, Mark and Diane, for setting me down this path and to Kelly for keeping me on track.

ACKNOWLEDGMENTS

The research in this thesis would not have been possible without the support of many individuals. I feel truly lucky to know so many great people ranging from my mentors, supervisors, and professors to my family and friends. So many had a hand in this journey.

First, I'd like to thank my adviser and committee chair, Professor John P. Hayes. I have learned so much about research from John's guidance and mentorship. John was a patient adviser and gave me the space to take our work in the directions that interested me or that I thought were promising. At the same time, John was also always available for discussion and taught me a great deal about scientific writing. I am grateful to have had a mentor that gave me the freedom to develop as an independent researcher while offering frequent feedback and suggestions that helped guide the process.

I'd also like to thank the rest of my thesis committee, Professors Karem Sakallah, Zhengya Zhang, and Armin Alaghi for their service on the committee and helpful feedback. In addition to my committee, I am also deeply grateful for the many thought provoking conversations with my labmates and collaborators: Paishun Ting, Owen Hoffend, Florian Neugebauer, Ilia Polian, Stefan Holst, Yiqiu (Susan) Sun, and Jinjie Liu.

Next, I would like to thank some of the great professors at Michigan. Most notably, Professor Achilleas Anastasopoulos and his course on probability and random processes had a profound impact on my thesis research and interest in the topic. Meanwhile, Professors Satinder Singh, John Laird, and Thad Polk had a big influence on my future research interests through the various artificial intelligence courses that they taught. These courses aren't directly relevant to the work in this thesis, but rather have shaped my passion for research and my research interests going forward.

I would also like to thank my mentors from Rowan University where I earned my bachelor's degrees. In particular, Professor Karen Sauer has helped me in countless ways over many years and was the person who introduced me to academic research. She pushed me to apply to summer research programs after my freshman year and ultimately set me down this path. I am also grateful to Professor Hong Ling who was my research mentor during my senior year and taught me so much about research and physics. Finally, I would like to thank Professors Phil LaPorta, Trevor Smith, Jeffrey Hettinger, Eduardo Flores, Michael Lim (Go Blue!), Gabriela Hristescu, and Jack Myers as well as Dr. Patrick Chestnut for their mentorship, advice, and impact on my scholarship.

Challenging periods of time are made much easier through the support of friends and family. I would like to thank my parents, Mark and Diane who emphasized the value of education and loved me for many years. Without them, I would not be here today and I wish they could be here to see the fruits of their hard work. I would also like to thank my sister Jen and brother Mark as well as their partners Jeff and Briana for their love and support over the years.

I would also like to thank my friends from Rowan and New Jersey including Nick Keppler, Sammi Caramela, Anthony Barca, Jake Lightman, Paul Castellano, Mike Mordente, Garret Kennel, Kevin Momat, Lydia Bender, and so, so many others who made my undergraduate years a blessing. Wanting to make many of you proud was a big motivator for pushing through challenging times. Additionally, visiting New Jersey for the holidays always re-energized me when I returned to research in Michigan.

I am also grateful to my many great friends from Michigan including Rob Fonti, Scott Ensel, Sang won Choi, Jakin Zhang, Chris Tossas, Laura Saunders, Dan Matera, Ravi Raghani, Evan Gonzalez, Marco Magano, Pierre Luciat, Erik Bowall, Angelo Gagliardi, Yuxin Liu, Sam DePalma, Kennedy Gibson, Pieter van Bakel, Nic Tjahjadi, and many others. I deeply appreciate our times together. Ann Arbor is a great place to live and each of you made it even better. Many of you have enriched my life through inspiring me to push harder and to try new things. I want to especially thank Kelly Wang who has been my best friend and girlfriend for the past five years. She's been my companion for so long and I can't wait to venture into the next chapter of life with her.

Finally, I would like to thank the CSE staff members including Karen Liska, Ashley Andreae, Jasmin Stubblefield, Cindy Estell, Yolonda Coleman, Sarah Snay, Kimberley Clark, and Stephen Reger who were always quick to help and support me and other Ph.D. students. Lastly, thank you to the National Science Foundation for funding a large portion of this thesis under grant CCF-2006703.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

APC accumulative parallel counter

ASIC application-specific integrated circuit

BASE Bayesian analysis of stochastic errors

BMSE Bayes mean squared error

BNN binarized neural network

CAM correlation-adjusted multiplexer

CeMux correlation-enhanced multiplexer

CMP comparator

CNN convolutional neural network

ECG electrocardiogram

FIR finite impulse response

IVD input value distribution

LFSR linear feedback shift register

MMC multiplexer-majority chain

MSE mean squared error

MSSIM mean structural similarity index

PDF probability density function

PCC probability conversion circuit

PSA parallel sampling adder

RCED Roberts Cross edge detector

RNS random number source

SC stochastic computing

SB sequential binary

SBoNG S-box random number generator

SCC stochastic cross correlation

SN stochastic number

SNG stochastic number generator

SRB shift register based

UCB up/down counter based

WBG weighted binary generator

ABSTRACT

Stochastic computing (SC) is an unconventional computing style that uses probabilistic bitstreams to implement algorithms like those for machine learning, digital filtering, and image processing. SC's unusual encoding enables highly fault tolerant, low power, and small datapaths. For instance, a single AND gate performs multiplication in SC. SC's advantages make it attractive for use in small devices like wearables and biomedical implants, but widespread adoption of SC has been limited due to challenges including a steep accuracy-latency trade-off and expensive interface circuits. We address these challenges using a statistical approach. First, we develop a better framework for analyzing stochastic circuit errors and then use insights from statistical analysis to design and evaluate smaller and more accurate circuits.

Understanding SC's many error sources is key to developing more accurate designs. However, existing approaches to accuracy analysis fail to provide a comprehensive account of all error types. The first part of this thesis proposes a novel framework named Bayesian Analysis of Stochastic Errors (BASE). BASE is built on statistical estimation theory and identifies three key quantities that must be modeled or estimated to quantify a circuit's accuracy. BASE is used to better understand and address circuit errors and is compatible with prior methods for error analysis as well as new models that we propose in this thesis. Importantly, BASE reduces the reliance on black box simulation methods which provide limited insight and are prone to mistakes.

SC's low-cost multipliers make it attractive for neural network and digital filtering applications. However, accurate addition is difficult to implement in SC and overhead from SC's data conversion circuits reduces SC's advantage for these applications. We address these problems by using insights from statistical modeling to develop new and better SC designs. Three such designs are introduced: 1) CeMux, a multiplexer adder that is significantly more accurate and smaller than its predecessors, 2) parallel sampling adders which have a flexible area-accuracy trade-off that can be tuned for the application, and 3) multiplexer-majority chains which can reduce the overhead of SC's conversion units. We apply our new designs to tasks like ECG filtering, image edge detection, and image classification with binarized neural networks as well as higher-precision neural networks.

CHAPTER 1

Introduction

Smart sensors, brain-computer interfaces, biomedical implants, wearables, and other devices can benefit from the recent rapid progress in data-driven algorithms like neural networks. However, such small and battery powered devices often have stringent area and energy constraints which make on-chip implementations of their algorithms difficult. A potential solution to this problem is stochastic computing (SC), a neuro-inspired computing paradigm that encodes and processes data using probabilistic bitstreams called stochastic numbers (SNs). Features of SN datapaths include high fault tolerance, low power, and small size. For instance, a single AND gate can be used to multiply two SNs. These and other features have made SC an attractive paradigm for on-chip implementations of image processing, error correcting, neural networks, and other algorithms.

1.1 Motivation and Challenges

Due to memory, size, and energy limitations, many devices rely on cloud servers to make use of advanced machine learning algorithms like neural networks. In this approach, a device sends data to cloud where it is processed and the result is sent back to the device. For example, the echo dot, one of Amazon's smart speakers, sends user queries to a cloud server where it is addressed by machine learning models that implement the Alexa virtual assistant technology [47]. The result from Alexa is sent back to the echo dot where the answer is played over the speaker for the user.

Cloud-based solutions allow devices to overcome their local resource limitations and employ useful algorithms, but this approach also comes with many disadvantages [44, 69]. For instance, relying on data centers in the cloud requires a connection to the internet which is not always feasible. Additionally, data privacy also becomes a concern when sending sensitive user information to the cloud, especially in medical applications. The latency and energy consumption of data transfer can also be high [69]. Although challenging, implementing algorithms directly on a small device can help avoid these disadvantages.

One way to implement computationally expensive algorithms at the device level is to design

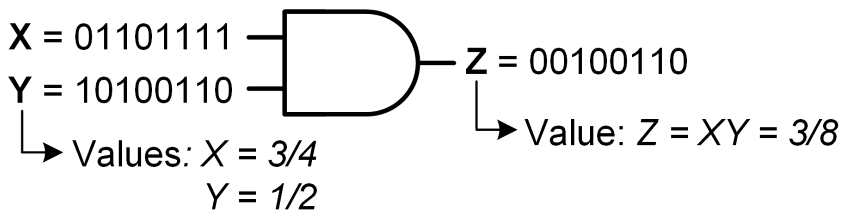


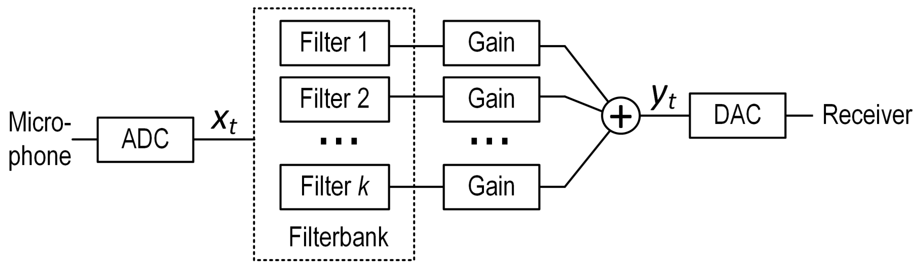
Figure 1.1: Example of multiplying two stochastic numbers with an AND gate.

new dedicated processors for certain workloads. For example, Apple's iPhones use a facial recognition algorithm called Face ID to unlock the device and to implement other authentication-related features. Face ID uses information from a variety of cameras and sensors on the phone to determine whether the user's face is present or not [52]. For best user experience, Face ID must execute quickly, minimize use of energy from the phone battery, and preserve user data privacy. To meet these constraints, Face ID uses the phone's 'neural engine', a dedicated hardware unit for deep neural networks separate from the iPhone's CPU and GPU [98]. In this case, using a cloud-based solution or the using phone's GPU may not meet application requirements for energy, latency and privacy. Thus, the inclusion of a dedicated processor for machine learning enables Face ID in iPhones [53].

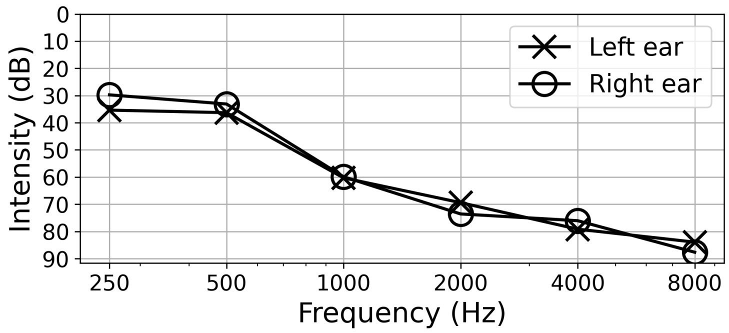
Designing application-specific integrated circuits (ASICs) that implement image processing, digital filtering, and machine learning algorithms on constrained devices is desirable, but can be difficult because these algorithms often require a large amount of memory or involve a large number of numeric operations. For example, the ResNet-50 neural network [43] requires more than 95MB of memory to store its network parameters and uses 3.8 billion multiplications to classify a single image [69]. Approximate or unconventional computing paradigms like stochastic computing (SC) can help address these challenges. SC's small and low power datapaths like the AND multiplier enable massively parallel implementations neural networks and other algorithms like finite impulse response (FIR) filters which rely heavily on multiplication [13, 66].

SC's advantages and challenges arise from its unconventional approach to encoding and processing data. SC uses stochastic numbers (SNs) to represent data. An SN is a stream of bits $\mathbf{X} = x_1x_2...x_L$ where $\mathbf{X}$'s value X is the probability that any of its bits take value 1. Two SNs can be multiplied by using an AND gate as illustrated in Figure 1.1 and fully explained in Chapter 2. Likewise, two SNs can be added using a two-way multiplexer. SC's low-cost multipliers and adders make it an attractive paradigm for designing ASICs for small, battery powered devices.

For example, our work has examined the use of SC in hearing aids [16]. Hearing aids provide a key solution to the problem of hearing loss which the World Health Organization estimates affects 430 million people worldwide [77]. A major component of a hearing aid is a filterbank; see



(a)



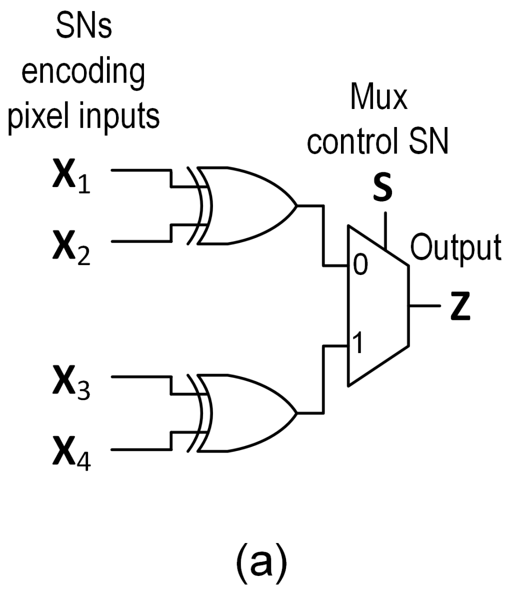
(b)

Figure 1.2: (a) Basic structure of a digital hearing aid; (b) example audiogram that indicates the least intense (faintest) sound that a patient can hear at the given frequency.

Figure 1.2a. The filterbank decomposes the input sound into frequency bands that are selectively amplified to match a specific pattern of hearing loss or audiogram. For example, the audiogram in Figure 1.2b indicates that the patient has severe high frequency hearing loss and requires more amplification at the upper end of the audio spectrum.

Hearing aid filterbanks typically require 100s to 1000s of multiplications. The area and power cost of filterbanks can make it difficult to meet the hearing aid's stringent constraints on physical size, response time, and power consumption [25]. In [16], we demonstrate that an SC-based filterbank can achieve 70% lower area and 4% to 65% lower power than a serial binary filterbank. Improvements to the SC filterbank would include increasing the stopband attenuation (a measure of filtering quality) and improving the power efficiency using techniques like our recently proposed SN adder and SN generator designs [14, 15].

In addition to low-cost datapaths, SC's probabilistic encoding also offers interesting avenues for design that are not present in conventional binary systems. For example, SC designs can exploit bit-level correlation to alter gate function. The general idea is that multiplication and other traditional SC designs are built on the assumption that all SN bits are independent or uncorrelated. Violating this assumption usually results in circuit inaccuracy [23], but recent work has shown



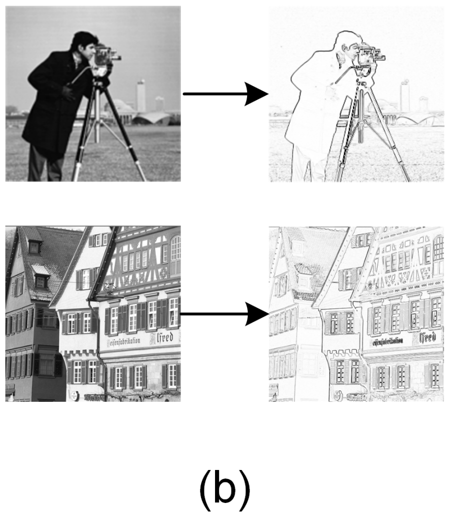


Figure 1.3: SC edge detection. (a) circuit design; (b) Example of images and their edge detected counterparts.

that correlation can be exploited in useful ways [3, 13, 19, 31]. For example, an AND gate with uncorrelated inputs implements multiplication, but the same AND gate will implement the $\min$ operation when its inputs are maximally correlated. Other examples of correlation-exploiting circuits include the OR and XOR gates which implement $\max$ and absolute difference, respectively. Non-linear operations like $\max$ are useful for implementing functions like max pooling and the ReLU activation functions found in neural network designs [31]. Other applications of correlation-based circuits include sorting networks [7] and the small edge detector design in Figure 1.3 which has been proposed for use in retina implants [5].

Another advantage of SC is the high fault tolerance of its SN encoding. For an SN, a bit-flip causes a change in value of $1/L$ where the L is the SN length. In contrast, a bit-flip on a highly significant bit in traditional binary format can result in a large value change. This feature of the SN encoding has made it a promising paradigm for use with emerging devices like memristors [7, 42, 56] and phase change memories [20]. Such devices have many advantages including high-density or speed, but suffer from variability and random noise in their operation. SC's fault tolerance can help overcome the noisy nature of these components and improve their reliability.

Despite its promise, SC has some challenges and limitations that have prevented its more widespread adoption.

Challenge 1. Poorly understood error theory. Existing methods fail to accurately quantify circuit errors which limits design tools and forces reliance on simulation that is computationally expensive and yields limited insight into how to improve accuracy.

Challenge 2. A steep accuracy-latency trade-off. The accuracy of estimating an SN's value scales, at best, linearly with bitstream length.

Challenge 3. Expensive data conversion circuits. The circuits used to generate SNs often require a large amount of circuit area which limits the benefits of cheap arithmetic circuits.

Our work addresses these challenges in the following ways. First, we address SC's poorly understood error theory by developing a new framework to error analysis and by introducing new techniques for modeling circuit behavior. Then, insights gained from accurately modeling circuit error led to new and more accurate circuit designs which address SC's accuracy-latency challenge. Specifically, large-scale adder design has been particularly difficult to implement efficiently in SC, but such designs serve a central role in implementing the many-input weighted additions found in operations like convolution. To address these challenges, we introduce CeMux, a new multiplexer adder that is more accurate and smaller than its predecessors [13]. We show CeMux greatly outperforms other SC designs for an electrocardiogram filtering task [13]. We then combine CeMux with another SC adder called the accumulative parallel counter to produce a new hybrid adder design named the parallel sampling adder (PSA) [14]. We demonstrate that PSAs have a flexible area-accuracy trade-off that can be tuned for the application, and we show that PSAs can be used to save about half the area of an SC neuron without sacrificing accuracy in an image classification application.

To address SC's expensive data conversion circuits and to improve understanding of correlation in modern SC designs, we also analyze SC's conversion circuits which are used to convert from traditional binary format into SC's SN format. We show how a well-known low-area conversion circuit named the weighted binary generator cannot reliably generate correlated SNs which results in high inaccuracy in correlation-reliant designs. To further analyze the correlation, area, and accuracy of SC's conversion circuits, we introduce multiplexer-majority chains (MMCs). MMCs can be used to better control correlation and area in SC conversion circuits and we demonstrate that combining MMCs with our CeMux design saves about 30% area in exchange for about 7% lower signal-to-noise ratio for a filtering application.

1.2 Outline of Thesis

The remainder of this thesis is organized as follows. We begin with a review of basic SC concepts in Chapter 2. More specific and technical background is introduced throughout the thesis as it becomes relevant. Then Chapter 3 details our proposed Bayesian Analysis of Stochastic Errors

(BASE) framework which addresses the challenge of SC's many error sources. BASE groups similar error types into one of two categories: systematic or random and then provides a formula for combining these error types into a single error metric, the mean squared error. BASE also highlights the application-dependence of stochastic circuit accuracy and provides a strong foundation for stochastic circuit accuracy analysis that complements existing simulation approaches.

BASE highlights three important quantities that must be statistically modeled or estimated to quantify a circuit's accuracy. Existing approaches to analyzing these quantities are either lacking key features or are inaccurate. In Chapter 4, we propose new probabilistic models that better represent circuit behavior and lead to accurate estimates for important accuracy influencing quantities like variance. We then show how accurate statistical models can be used to improve the accuracy of small circuits like mux adders and give a complete case study that applies BASE and our new models to SC neural network analysis.

Chapter 5 expands on the results of Chapter 4 by introducing two new large-scale adder designs, CeMux and PSAs. We demonstrate how both adders address the challenges of implementing many-input weighted addition in SC through example applications like electrocardiogram filtering and image classification with neural networks. Then in Chapter 6, we study how SN generator design impacts important circuit properties like input correlation. We show that low-area SN generator designs can't generate highly correlated bitstreams which greatly impacts the accuracy of correlation-reliant designs. Chapter 6 also details a new SN generator design style named multiplexer-majority chains. Chapter 7 summarizes our contributions and suggests direction for future work.

CHAPTER 2

Stochastic Computing

This chapter introduces relevant concepts from stochastic computing (SC). Some ideas are developed further in later chapters of the thesis. For example, the basics of stochastic number generation are covered here and then expanded on in Chapter 6 which focuses on stochastic number generator design. Helpful introductions to SC include a 2017 review paper by Alaghi, Qian and Hayes [6], a 2019 textbook written by established SC researchers and edited by Gross and Gaudet [40] and the 1969 article *Stochastic computing systems* by Gaines [34]. A history of stochastic computing is given in Chapter 1 of [40].

2.1 Stochastic Computing Basics

SC uses stochastic numbers (SNs) to represent data. An SN $\mathbf{X}$ is a sequence of random bits $\mathbf{X} = x_1x_2...x_L$ where each bit has the same probability of taking value 1, namely, $P_x = \mathbb{P}(x_t = 1)$. $\mathbf{X}$'s numeric value depends on the chosen SN format. When $\mathbf{X}$ is a unipolar SN, its value is simply $X = P_x$ while a bipolar SN $\mathbf{X}$ has value $X = 2P_x - 1$. Bipolar SNs are useful for representing negative values. For example, the 8-bit SN $\mathbf{X} = 00100110$ has an estimated unipolar value of $\hat{X} = 3/8$ and estimated bipolar value of $\hat{X} = -1/4$. Scaling can be used to represent numbers outside of the $[0, 1]$ or $[-1, 1]$ intervals.

A central advantage of SN encoding is that arithmetic is performed using simple logic elements. Consider the AND gate in Figure 2.1a with unipolar SN inputs $\mathbf{X}$ and $\mathbf{Y}$ and output $\mathbf{Z}$. The output value Z is $\mathbb{P}(z_t = 1) = P((x_t \wedge y_t) = 1)$. If the bits of $\mathbf{X}$ and $\mathbf{Y}$ are uncorrelated or independent, then $Z = \mathbb{P}(x_t = 1)\mathbb{P}(y_t = 1) = XY$. Thus, the output value is equal to the product of the input values implying the AND gate acts a single-gate SN multiplier. For bipolar SNs, multiplication is performed with an XNOR gate.

Like multiplication, addition is simple to implement in SC, but addition must be scaled or approximated due to the confinement of SN values to fixed intervals like $[0, 1]$. Scaled addition is usually performed with a multiplexer (mux) like the one in Figure 2.1b. Here, $\mathbf{S}$ is a control

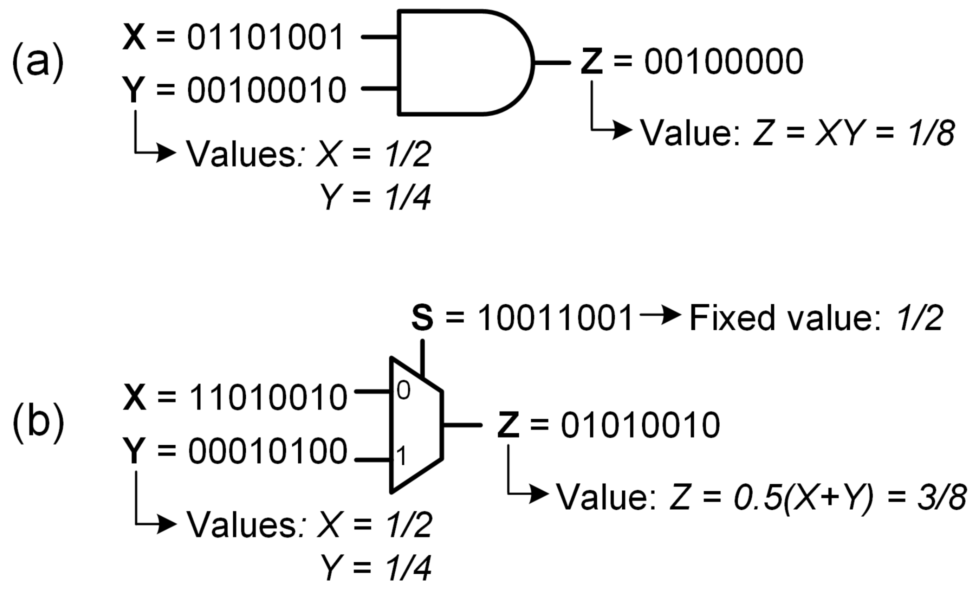


Figure 2.1: Stochastic logic elements: (a) unipolar multiplier (b) scaled mux adder.

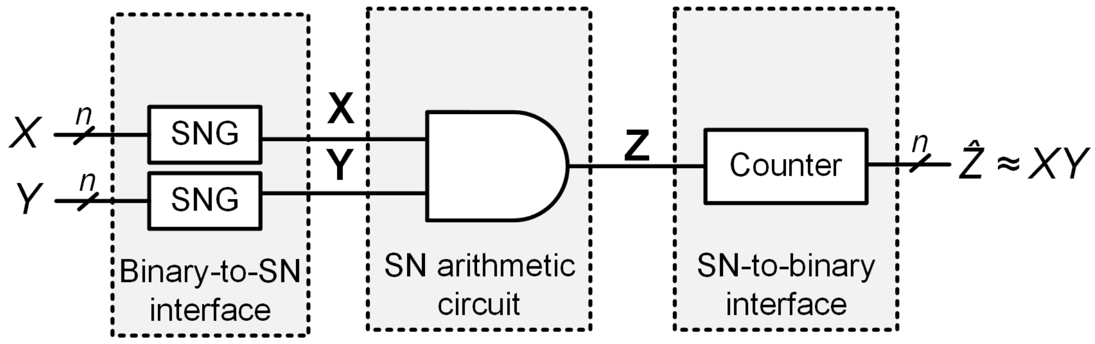


Figure 2.2: End-to-end SC system.

SN with a fixed probability of $P_s = 0.5$ and $\mathbf{X}$ and $\mathbf{Y}$ are data SNs that are added. With this configuration, the output $\mathbf{Z}$ has value $Z = 0.5(X + Y)$ when $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ are all unipolar SNs or when they are all bipolar SNs (the control $\mathbf{S}$ is always unipolar). Thus, the mux acts as a scaled unipolar or bipolar SN adder. Alternatively, an OR gate with inputs $\mathbf{X}$ and $\mathbf{Y}$ and output $\mathbf{Z}$ performs saturated addition on unipolar SNs: $Z = X + Y - XY$. In general, stochastic arithmetic is approximate, and accuracy depends on SN length, correlation levels, and other factors. For example, when viewed as an adder, the OR gate has an approximation error of $-XY$.

The general structure of an SC system that can be embedded into a traditional fixed-point system is shown Figure 2.2. Here, a small stochastic datapath, the AND multiplier, is surrounded by peripheral circuits that convert between binary fixed-point format and the SN format. The input side of the SC system consists a set of stochastic number generators (SNGs) used to create input

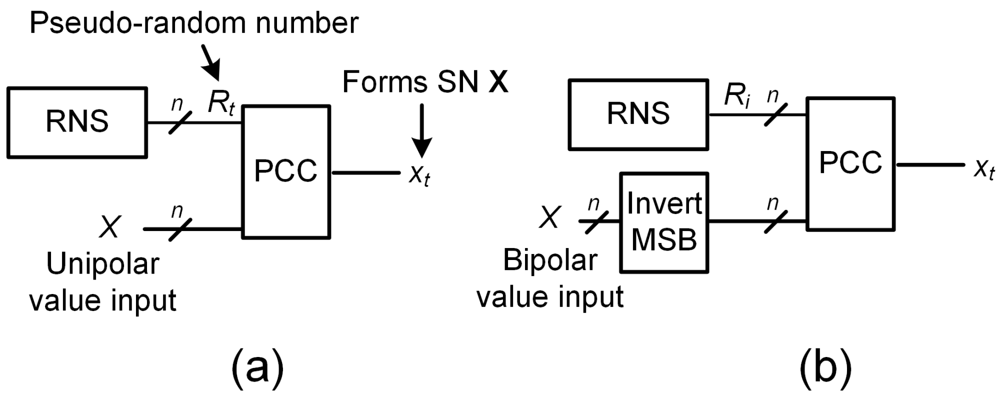


Figure 2.3: Stochastic number generators (SNGs): (a) unipolar SNG that encodes fixed-point X into SN $\mathbf{X}$; (b) bipolar SNG that encodes 2's complement X into $\mathbf{X}$.

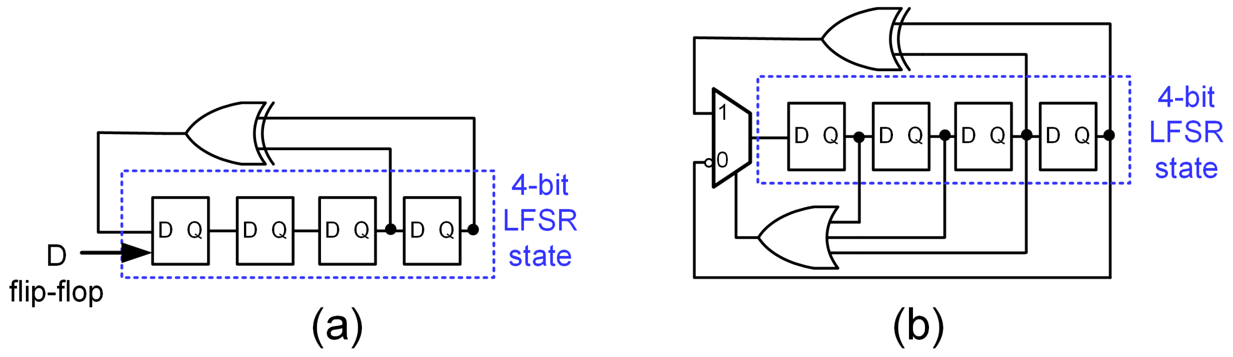


Figure 2.4: Feedback shift registers. (a) 4-bit maximal LFSR; (b) 4-bit maximal LFSR with all-0 state inserted to its state sequence.

SNs $\mathbf{X}$ and $\mathbf{Y}$. SNGs are typically constructed with a random number source (RNS) and a a probability conversion circuit (PCC) as illustrated in Figure 2.3a. The RNS produces a sequence of n-bit random numbers $R_1R_2...R_L$ which the PCC converts one-by-one to produce an SN $\mathbf{X} = x_1x_2...x_L$ with $\mathbb{P}(x_t = 1) = P_x$. For example, a common PCC is a digital comparator which outputs $x_t = R_t < X$. Since R_t is uniformly distributed, $\mathbb{P}(x_t = 1) = \mathbb{P}(R_t < X = 1) = X$ as desired. To generate bipolar SNs, the unipolar SNG of Figure 2.3a can be slightly modified as in Figure 2.3b.

An n-bit maximal linear feedback shift register (LFSR; Figure 2.4a) [37] is commonly employed as the RNS in an SNG because it is relatively low-cost and is sufficiently random for SC use [34, 37]. The LFSR's n-bit state S iterates through all values in $\{1/2^n, 2/2^n, \dots (2^n - 1)/2^n\}$ in a pseudo random order. Importantly, S never takes value 0 which can cause a small bias in SN generation and complicates later analysis. Fortunately, any maximal LFSR can be easily modified to include the all-zero state by adding a single $n-1$ input OR gate as shown in Figure 2.4b [8, 91].

Notation	Meaning
$\mathbf{X} = X_1 X_2 ... X_L$	An L-bit stochastic number (SN).
$X = \mathbb{P}(x_t = 1) = \mathbb{E}[\hat{X}]$	$\mathbf{X}$'s (expected) unipolar value.
$\hat{X} = \frac{1}{L}\sum_{t=1}^{L} x_t$	$\mathbf{X}$'s estimated unipolar value.
X^*	$\mathbf{X}$'s target value.
$\boldsymbol{\mathcal{X}}^* = [X_1^*, X_2^*, ..., X_M^*]$	Vector of M target SN values.
n	SN generator precision.
L	SN length. Usually set to 2^n.

Table 2.1: Major Notation and Definitions

All LFSRs used in this thesis are modified in this manner.

2.2 Stochastic Number Estimation

After generation, SNs are processed through an SN arithmetic circuit (e.g., an AND gate multiplier or a mux adder) and produce an output SN **Z** as illustrated in Figure 2.2. If **Z**'s length L is a power of two, a digital counter can be used to determine the frequency of 1s in **Z**, i.e.,

$$\hat{Z} = \frac{1}{L}\sum_{t=1}^{L} z_t \tag{2.1}$$

effectively converting from the SN format to traditional fixed-point format. Likewise a bipolar SN **Z** with value $Z = 2P_z - 1$ can have its value estimated as

$$\hat{Z} = 2\left(\frac{1}{L}\sum_{t=1}^{L} z_t\right) - 1 \tag{2.2}$$

An up-down counter that increments when $z_t = 1$ and decrements when $z_t = 0$ implements Equation (2.2) when the SN length L is a power of two. In both the unipolar and bipolar cases, the counter's output $\hat{Z}$ is an estimate of **Z**'s target value Z^*, which is the intended result of the computation. The difference between $\hat{Z}$ and Z^* is the circuit's error which fluctuates randomly due to the stochasticity of **Z**.

Before proceeding further, it is helpful to highlight the three distinct values associated with every SN such as $\mathbf{X}$. The first is $\mathbf{X}$'s expected value or just "value" X which is a function of the underlying probability P_x that its bits take value 1. In unipolar format $X = P_x$ and in bipolar format $X = 2P_x - 1$. The second is $\mathbf{X}$'s desired or target value X^* determined by the application and which may indicate a bias error if it differs from X. In many cases, $X = X^*$ and the SN is

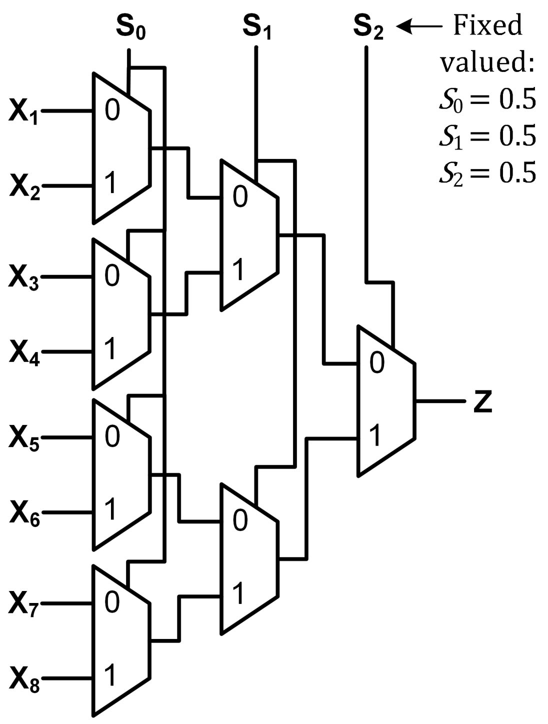


Figure 2.5: Basic mux tree adder.

said to be unbiased. The third value is **X**'s estimated value $\hat{X}$ found by counting the 1s in **X**, as in Equations (2.1) and (2.2). In traditional SC and this thesis, it is always the case that $X = \mathbb{E}[\hat{X}]$. Table 2.1 summarizes these values as well as some other important notation conventions used in this work.

In general, there will be some difference between an output SN's estimated value $\hat{Z}$ and its target value Z^*. Such errors come from a variety of sources like random fluctuations of SN bits and unwanted correlations between them. In general, the average error decreases as SN length increases. Thus, SC is an approximate computing paradigm with an inherent accuracy-latency trade-off. Quantifying this trade-off will be the central topic of Chapters 3 and 4. A final remark on SC accuracy is that SNs are very fault tolerant. Specifically, Equations (2.1) and (2.2) indicate that a bit-flip amounts to a small change in the estimated value of $1/L$ or $2/L$ while, in traditional fixed-point format, a bit-flip can cause a large error in value.

2.3 Many-input SN Adders

A tree of 2-way mux adders can be extended to add more SNs as shown in Figure 2.5. All mux adders sample their data inputs to implement scaled addition. For example, the mux tree in Figure 2.5 has a $1/8^{\text{th}}$ chance of sampling each input implying that **Z**'s expected value is

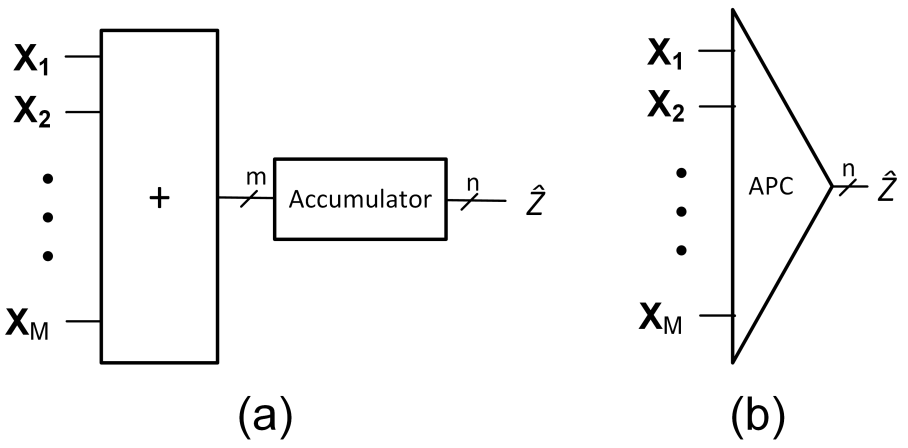


Figure 2.6: Accumulative parallel counter (APC) [78]. (a) APC structure consisting of an adder whose sum is accumulated over many clock cycles; (b) block symbol.

$Z = 1/8 \sum_{i=1}^{8} X_i$. Although mux tree adders are small, they are often very inaccurate especially when used to add 100s or 1000s of SNs [13, 71]. Mux inaccuracy is largely due its sampling behavior where only one input SN is sampled each clock cycle while the bits from all other inputs are ignored. Specifically, the chosen SN is randomly determined by the select bits and so the sampling process results in high noise and random fluctuation error. In Chapter 5, we analyze mux adder error using a new statistical model and then introduce CeMux, a new mux adder that is significantly more accurate and smaller than existing mux adder designs.

Along with mux adders, accumulative parallel counters (APCs) [78] are among the most popular SN adder designs. An M-input APC performs the popcount operation where it counts all 1s from each incident SN and accumulates the result each clock cycle as shown in Figure 2.6. The APC effectively sums its M input bitstreams while also converting them into a fixed-point estimate. In comparison to mux adders, APCs are very accurate due to their exhaustive counting approach, but also are costlier in terms of area and power. As mentioned earlier, conventional mux adders are usually inaccurate when summing many bitstreams and so APCs have become the preferred adder type for some applications like neural networks [66] which rely on large, many-input summations.

APCs are often used to perform weighted addition, a central operation in many important algorithms. Figure 2.7 illustrates a 4-input bipolar weighted APC. Here, the input values X_1 to X_4 and corresponding weights W_1 to W_4 are converted into bipolar SNs and are then multiplied using XNOR gates. The product SNs $\mathbf{Y}_1$ to $\mathbf{Y}_4$ with value $Y_i = W_i X_i$ are then summed using an APC. The APC output is $\hat{Z}$ with value $Z = \sum_{i=1}^{4} Y_i = \sum_{i=1}^{4} W_i X_i$. In general, weighted APCs implement $Z = \sum_{i=1}^{M} W_i X_i$ where M is the number of inputs.

In a weighted APC, using a separate SNG for each input SN and weight SN can be costly in terms of area and power. However, cost can be greatly reduced by sharing a single RNS across

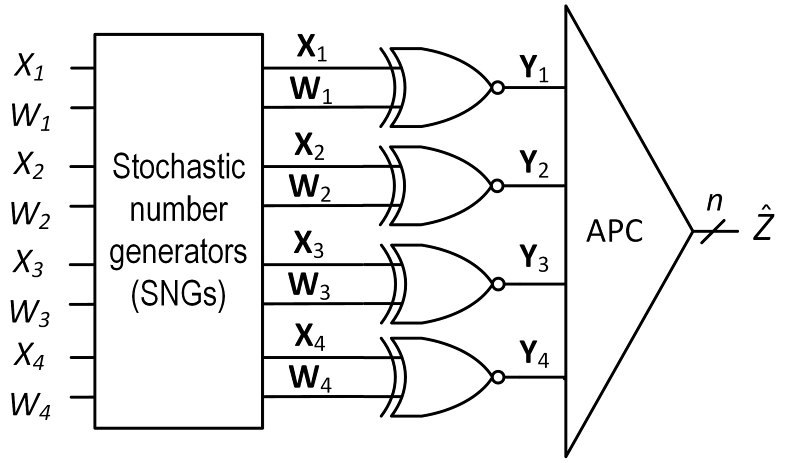


Figure 2.7: Bipolar weighted APC adder.

several SNGs. For weighted APCs, one RNS is generally shared amongst all input SNGs and a second RNS is shared amongst all weight SNGs [66]. This method employs as much RNS-sharing as possible while still allowing for accurate XNOR multiplication that requires inputs SNs and weight SNs to be uncorrelated. RNS-sharing is generally believed to not affect APC accuracy [66], but a case study in Section 5.2 proves that this is incorrect in some cases. Even with RNS-sharing, a weighted APC still has high area which we address in Section 5.3 with our parallel sampling adder design.

2.4 Correlation

As alluded to earlier, SN bits are usually assumed to be independent. However, correlations sometimes arise between SN bits with major implications on circuit function and accuracy. In SC, the stochastic cross correlation (SCC) metric [3] is typically used to quantify correlation between the bits of SNs $\mathbf{X}$ and $\mathbf{Y}$:

$$\mathrm{SCC}(\mathbf{X},\mathbf{Y}) = \begin{cases} \dfrac{P_{x\wedge y} - P_xP_y}{\min(P_x, P_y) - P_xP_y} & \text{if} P_{x\wedge y} \geq P_xP_y \\ \dfrac{P_{x\wedge y} - P_xP_y}{P_xP_y - \max(P_x + P_y - 1, 0)} & \text{otherwise} \end{cases} \tag{2.3}$$

where $P_{x\wedge y} = \mathbb{P}(x_t \wedge y_t = 1)$. SCC's numerator is the difference between the actual overlap of 1s in $\mathbf{X}$ and $\mathbf{Y}$ and expected overlap of 1s if $\mathbf{X}$ and $\mathbf{Y}$ were independent. SCC's denominator normalizes the value to range $[-1, 1]$.

If SNs $\mathbf{X}$ and $\mathbf{Y}$ are uncorrelated, then the 1s in these bitstreams are expected to overlap at a rate of P_xP_y. For example, when $P_x = 1/2$ and $P_y = 1/4$, the 1s in uncorrelated SNs $\mathbf{X}$ and $\mathbf{Y}$

Gate	**Function when** $\mathrm{SCC}(\mathbf{X},\mathbf{Y})=0$	**Function when** $\mathrm{SCC}(\mathbf{X},\mathbf{Y})=1$
AND	$Z = XY$	$Z = \min(X, Y)$
OR	$Z = X + Y - XY$	$Z = \max(X, Y)$
XOR	$Z = X + Y - 2XY$	$Z = \lvert X - Y \rvert$
MUX	$Z = 0.5(X + Y)$	$Z = 0.5(X + Y)$

Table 2.2: Unipolar Stochastic Operation with Independent or Correlated Inputs

overlap about $1/8^{\text{th}}$ of the time and $\mathrm{SCC}(\mathbf{X},\mathbf{Y})$ is 0. When the 1s in $\mathbf{X}$ and $\mathbf{Y}$ overlap more often than expected, $\mathrm{SCC} > 0$ and $\mathbf{X}$ and $\mathbf{Y}$ are said to be correlated. Likewise, when their 1s overlap less often than expected, $\mathrm{SCC} < 0$ and $\mathbf{X}$ and $\mathbf{Y}$ are said to be anti-correlated. SCC is maximized with a value of ± 1 when the 1s in $\mathbf{X}$ and $\mathbf{Y}$ overlap or do not overlap as much as possible based on P_x and P_y. For example, $\mathbf{A} = 110101$ and $\mathbf{B} = 100101$ have an estimated SCC($\mathbf{A}$,$\mathbf{B}$) of $+1$ while $\mathbf{A}$ and $\mathbf{C} = 001010$ have an estimated SCC($\mathbf{A}$,$\mathbf{C}$) of -1.

In SC design, correlation can be exploited to change circuit function [3, 19]. For example, an AND gate normally multiplies uncorrelated inputs $\mathbf{X}$ and $\mathbf{Y}$ when $\mathrm{SCC}(\mathbf{X},\mathbf{Y}) = 0$, but an AND gate instead outputs $\mathbf{Z}$ with $Z = \min(X, Y)$ when $\mathbf{X}$ and $\mathbf{Y}$ are maximally correlated with $\mathrm{SCC} = 1$. Table 2.2 gives other examples of how correlation changes gate function in useful ways. Using correlated SNs to implement functions like min, max and absolute difference has practical applications in image processing [1, 19, 61], sorting networks [7] and neural networks [2, 31]. For example, neural network max pooling and ReLU activation functions rely on the max operation which can be implemented with just an OR gate if the input SNs are maximally correlated.

In addition to modifying circuit function, correlated SNs can improve circuit accuracy compared to using independent SNs. For example, in Chapter 4 we introduce a large mux-based SN adder named CeMux [13] that is significantly more accurate than traditional mux-based adders that don't leverage correlation. Other circuits like sorting networks or median filters can also achieve very low or even zero error under certain conditions [19]. If designers are not careful, however, correlation may increase output error as in the case of a mux adder with negatively correlated inputs [12].

2.5 Sequential Stochastic Circuits

Sequential stochastic circuits are another type of design that uses memory to implement useful arithmetic functions. For example, the circuit in Figure 2.8 implements $Z \approx X^2$. The D flip-flop in the circuit creates a delayed copy of input $\mathbf{X}$ where $d_t = x_{t-1}$. $\mathbf{D}$'s value $D = \mathbb{P}(d_t = 1) \approx \mathbb{P}(x_t = 1) = X$ therefore approximates $\mathbf{X}$'s value and the AND gate multiplies $\mathbf{X}$ and $\mathbf{D}$ to yield

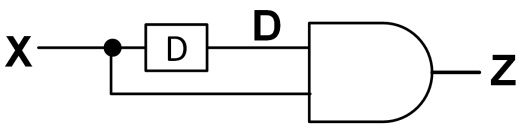


Figure 2.8: Stochastic squarer circuit.

$\mathbf{Z}$ with $Z = XD \approx X^2$. Other sequential stochastic circuits can be used to implement useful functions like $\tanh$ and $\exp$ [18, 64].

Sequential stochastic circuits can be analyzed using Markov chains [18, 64] as described in detail in Section 4.3. Such circuits usually assume that there is no correlation within an SN or, in other words, that the input SNs are have no autocorrelation. For example, the SN $\mathbf{X} = 01010101$ is highly autocorrelated because a 1 always follows a 0 and vice versa. $\mathbf{X} = 11110000$ is another example of a highly autocorrelated SN which will produce an inaccurate result if used in a sequential circuit like the squarer. It is understood that autocorrelation causes high error, but simulation was the only method quantifying that error for a given design. In Section 4.3, we formalize autocorrelation error and propose an analytic approach for quantifying this error type.

2.6 SC Applications

SC has been applied to many domains including image processing, digital filtering, error correction codes, and neural networks. SC is well-suited to these areas given its low-cost computational elements, its fault tolerance, and its potential for massive parallelism. SC synergizes best with applications that can tolerate or benefit from the random noise inherent to SC's processing. For example, a neural network's class prediction is usually found by taking the argmax of the neuron values in the network's final layer. These neuron values can contain random noise without changing the overall class prediction as long as the largest neuron value is sufficiently higher than the rest. Moreover, random noise during neural network training can improve classification performance [73, 84] and random noise during neural network inference can help defend against adversarial attacks [89]. Thus, SC is well-suited for designing low-cost implementations of neural networks.

Table 2.3 lists some of the many SC applications with references, and indicates whether this thesis includes an example of the application. Our work largely focuses on SC's application to neural networks, image processing, and digital filtering. For example, our CeMux adder design is applied to filtering noisy electrocardiogram (ECG) signals for heart monitoring [13, 14, 15] and applied to filtering audio signals for hearing aids [16]. Meanwhile our parallel sampling adder (PSA) design improves SC-based neural network performance [14] and our multiplexer-majority chain (MMC) design [15] can be used to better understand and improve SNG overhead in any design.

Application	Reference(s)	Covered in this work
Image classification with neural networks	[18, 29, 31, 45]	Sections 4.5, 5.3, 6.4
Digital filtering	[13, 21, 51, 91]	Sections 5.2, 6.4
Median filtering	[19]	Section 6.4
Edge detection	[5, 19]	Section 6.4
Error correction with LDPC codes	[39, 49]	-
Sorting networks	[7]	-
Encoding for noisy emergent devices	[20, 45, 56]	-
Near-sensor processing	[30, 60]	-
Bayesian inference	[28, 42]	-
Brain-inspired computing	[76, 86]	-
Drone stabilization	[26]	-

Table 2.3: Examples of Stochastic Computing Applications

CHAPTER 3

Bayesian Analysis of Stochastic Errors

Understanding accuracy and its trade-offs is key to evaluating the growing list of SC circuit designs. Prior works have identified many sources of error in SC systems which can be difficult to combine. Prior approaches to error analysis have focused on one particular error type [10, 23, 67, 74] or have focused on deriving conservative upper bounds on error [79]. However, a lack of a comprehensive and general framework for errors impedes the design process and makes comparing designs difficult. This chapter summarizes our progress towards a more general theory of SC errors by detailing our Bayesian analysis of stochastic errors (BASE) framework. The work in this chapter is mostly published in [11], but the name BASE is introduced here for the first time.

3.1 Error Types

In general, error is the difference between a circuit's target output value and its estimated output value found by counting the output SN's bits. For example, a stochastic circuit used to multiply $1/4$ by $3/4$ has target output $3/16$, but may output $\mathbf{Z} = 0000100011000100$ which has four 1s and sixteen total bits. In this case, the output's estimated value is $4/16$ which constitutes an error of $1/16$. Stochastic circuit inaccuracy arises from a variety of sources including approximation, quantization, correlation and random fluctuation [74, 79].

Approximation error is the difference between the target function and the function implemented by the circuit. Approximating a target function can help reduce hardware cost if the approximating function is simpler to implement than the target function. For example, an OR adder implements $Z = X + Y - XY$ which approximates standard addition $Z^* = X + Y$ with an approximation error of $-XY$. Approximation error also arises when using reconfigurable architectures where target functions must be approximated by a specific functional form, such as a Bernstein polynomial [79].

Quantization error is due to the limited precision of the SNGs. For example, a 4-bit SNG can only generate SNs with value $\{0, 1/16, 2/16, \ldots, 15/16\}$. An input SN $\mathbf{X}$ with $X^* = 0.72$ has its target value rounded by truncation to $X = 11/16$ producing a quantization error of $11/16 - 0.72 \approx$

-0.03. Other works [48, 74] may define quantization error in terms of the smallest value a SN can represent based on its length, but our concept of quantization is based on the SN's underlying probability and is separate from SN length. For example, in our definition, $\mathbf{X} = x_1$ with unipolar value $X = 0.5$ and length $L = 1$ has zero quantization error if $\mathbf{X}$'s lone bit takes value 1 with probability 0.5.

Correlation error is another, more subtle error source due to unwanted similarities between bit-streams. Most traditional SC circuits are designed assuming that all input SNs are independent or uncorrelated. In practice though, SNs may be correlated thus changing the circuit's expected output and introducing error [23]. For example, consider an AND gate with no quantization error and with inputs $\mathbf{X}$ and $\mathbf{Y}$. If $\mathbf{X}$ and $\mathbf{Y}$ are highly correlated, the output $\mathbf{Z}$ will have $Z = \min(X, Y)$ rather than $Z^* = XY$. The correlation error in this case is $\min(X, Y) - XY$. Interestingly, the $\min$ function, and hence deliberately introduced correlation, is useful in some applications [3]. In general, correlation error arises whenever SNs do not have the desired amount of correlation.

Random fluctuation error is usually the largest source of error in SC. In general, estimated values like $\hat{Z}$ fluctuate around their expected value $Z = \mathbb{E}[\hat{Z}]$ because the bits of $\mathbf{Z}$ are stochastic. For example, an AND gate may have expected output value $Z = 0.5$, but may actually output $\mathbf{Z} = 01100001$. In this case, $\mathbf{Z}$'s value would be estimated as $\hat{Z} = 3/8$ which constitutes a random fluctuation error of $-1/8$. In general, the expected magnitude of random fluctuation errors tends to decrease proportionally to $1/\sqrt{L}$ or $1/L$ where L is SN length [4, 74].

Since there are many error types, it can be difficult to quantify a circuit's overall error. In [79], an upper bound for each error type is identified and then combined to produce an upper bound for overall error. However, such an approach ignores cancellations between errors. For example, a positive approximation error can partly cancel with negative quantization error. Ignoring such cancellations causes the analysis to overestimate the overall error's true upper bound. Other error analysis approaches focus mostly on random fluctuation error because it is usually the most prominent error source [67, 74]. However, random fluctuation error is not always the dominant error source as correlation error can sometimes be very high [23].

Overall, it is desirable to have a general and comprehensive error analysis framework that simultaneously accounts for all error sources and can produce accurate estimates for the circuit's expected error which is often the desired metric of interest. Our Bayesian analysis of stochastic errors (BASE) framework makes significant progress on this goal. The framework consists of two key concepts which lead to a better understanding of stochastic circuits. The first concept is that all error types fall into one of two fundamental categories: systematic or random. The second concept is that a circuit's expected error is application dependent. Overall, the BASE framework simultaneously accounts for all error types, provides a useful decomposition of expected error, and leads to a methodology for comparing the accuracy of different stochastic or deterministic circuit

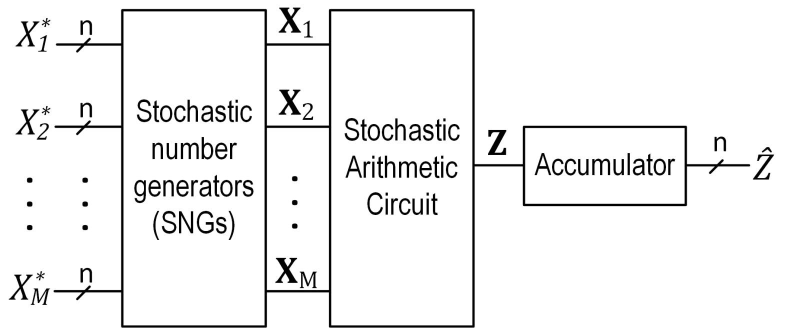


Figure 3.1: Generic end-to-end SC system consisting of stochastic number generators, a stochastic datapath and an output accumulator.

designs.

3.2 Systematic and Random Errors

Existing methods in other areas of the statistical sciences can be leveraged to build an error analysis framework for SC. Specifically, stochastic circuits can be viewed as estimators of a target value. Through this useful lens, results and insights from statistical estimation theory [54] can be utilized to develop a framework for stochastic circuit error analysis. We detail this formulation for unipolar SC, but it readily extends to the bipolar format by replacing all unipolar SN values with bipolar ones.

Consider the generic SC system in Figure 3.1. The M-input stochastic circuit has target input values $\boldsymbol{\mathcal{X}}^* = [X_1^*, X_2^*, ..., X_M^*]$ and a target output function $Z^* = f(\boldsymbol{\mathcal{X}}^*)$. The circuit's output is an SN $\mathbf{Z} = z_1 z_2 ... z_L$ whose value $Z = \mathbb{P}(z_t = 1)$ equals or approximates the target Z^*. However, Z is not available directly and must be estimated by using an accumulator to count the 1s in **Z**:

$$\hat{Z} = \frac{1}{L}\sum_{t=1}^{L} z_t. \tag{3.1}$$

The circuit's error ϵ_Z is the difference between the estimate $\hat{Z}$ and the target value Z^*.

$$\epsilon_Z = \hat{Z} - Z^* \tag{3.2}$$

A cost or loss function can be used to determine the severity of this estimation error. No standard cost function exists for SC, but $L(\epsilon) = \epsilon^2$ (quadratic error) and $L(\epsilon) = |\epsilon|$ (absolute error) are commonly used. We focus on quadratic cost due to its useful bias-variance decomposition, but the following formulation, especially Equations (3.3) and (3.9) can readily be framed in terms of other cost functions.

The circuit's mean squared error (MSE) given target input values $\boldsymbol{\mathcal{X}}^*$ is defined as

$$\mathrm{MSE}(Z^*, \hat{Z}|\boldsymbol{\mathcal{X}}^*) = \mathbb{E}[(\hat{Z} - Z^*)^2|\boldsymbol{\mathcal{X}}^*] \tag{3.3}$$

In Equation (3.3), it is explicit that error is dependent on the target input values $\boldsymbol{\mathcal{X}}^*$ which is normally the case in SC. MSE also usually depends on other parameters like SN length and SNG precision whose influence is implicit in (3.3).

In standard statistical analysis [54], the MSE of an estimator such as $\hat{Z}$ can be decomposed into a combination of bias and variance, the so-called bias-variance decomposition of MSE:

$$\mathrm{MSE}(Z^*, \hat{Z}|\boldsymbol{\mathcal{X}}^*) = \mathrm{Bias}^2(Z^*, \hat{Z}|\boldsymbol{\mathcal{X}}^*) + \mathrm{Var}(\hat{Z}|\boldsymbol{\mathcal{X}}^*) \tag{3.4}$$

where

$$\mathrm{Bias}(Z^*, \hat{Z}|\boldsymbol{\mathcal{X}}^*) = \mathbb{E}[\hat{Z}|\boldsymbol{\mathcal{X}}^*] - \mathbb{E}[Z^*|\boldsymbol{\mathcal{X}}^*] \tag{3.5}$$

$$\mathrm{Var}(\hat{Z}|\boldsymbol{\mathcal{X}}^*) = \mathbb{E}[(\hat{Z} - \mathbb{E}[\hat{Z}])^2|\boldsymbol{\mathcal{X}}^*] \tag{3.6}$$

Bias is the difference between the target value and the circuit's expected estimated value. Bias quantifies error as if the SNs had infinite length and all randomness was averaged out. In other words, bias captures the systematic error of the circuit. Bias is the combined error of consistent and usually independent error sources like approximation, quantization and correlation errors. For example, an approximation error of 0.02, quantization error of -0.05 and correlation error of 0.04, combine to give an overall bias of 0.01. A circuit's bias can be reduced by addressing these error sources by, for example, redesigning the circuit to improve approximation error, increasing SNG bitwidth to reduce quantization error, or by using decorrelation techniques to reduce correlation error [87].

Variance, the counterpart to bias, captures the random fluctuation error of the circuit by quantifying the expected difference between the estimated output value $\hat{Z}$ and its expected value $Z = \mathbb{E}[\hat{Z}]$. Variance decreases as SN length increases because random fluctuations tend to average out over many clock cycles. Variance depends on the SNG design (e.g., what RNS type is used) and on the datapath design, but does not depend on the target value Z^*. Thus, variance is a property of the circuit rather than the target application. In many cases, variance is much larger than bias and bitstream length has a dominating influence on accuracy.

Although bias-variance decomposition is well-known in statistical estimation theory, to our knowledge, our work [11] was the first to explicitly apply it to the analysis of SC circuits. One work on correlation error analysis [23] derived some equations that resemble bias-variance decomposition, but did not develop the analysis to include other error types like approximation and

Algorithm 3.1 Stochastic Circuit Error Analysis

```
Input: $\boldsymbol{\mathcal{X}}^* = [X_1^*, X_2^*, ..., X_M^*]$                ▷ Target input values
Output: $\mathrm{MSE}(Z^*, \hat{Z}|\boldsymbol{\mathcal{X}}^*)$        ▷ Mean squared error given target input values $\boldsymbol{\mathcal{X}}^*$
1: $Z^* \leftarrow f(\boldsymbol{\mathcal{X}}^*)$                  ▷ Compute target output based on application
2: $\boldsymbol{\mathcal{X}} \leftarrow \lfloor 2^n \boldsymbol{\mathcal{X}}^* \rfloor / 2^n$      ▷ Use truncation to quantize target inputs to SNG precision $n$
3: $Z \leftarrow g(\boldsymbol{\mathcal{X}})$           ▷ Compute expected output value based on stochastic circuit logic
4: $B \leftarrow Z - Z^*$                                  ▷ Compute bias using (3.5)
5: $V \leftarrow h(\boldsymbol{\mathcal{X}})$                              ▷ Compute variance using (3.6)
6: $E \leftarrow B^2 + V$                                  ▷ Compute MSE using (3.4)
7: return $E$
```

quantization. Other works acknowledge SC's various error sources, but focus mostly on analyzing random error (variance) [67, 74].

The notions of bias and variance are a key component of BASE. They each quantify a fundamental error type: bias quantifies systematic error and variance quantifies random error. A high MSE may indicate a high bias, a high variance, or both. Understanding which is the case is paramount to addressing inaccuracies. For example, increasing SN length will greatly reduce MSE if variance is the dominant error source, but will not mitigate a high MSE if bias is the dominant error source since SN length does not affect bias. Algorithm 3.1 is based on MSE's bias-variance decomposition and describes a general process for determining a circuit's MSE given its target input values. The following two examples illustrate Algorithm 3.1 in action.

Example 3.1: Consider using the SC system in Figure 3.2a to multiply $X^* = 15/32$ and $Y^* = 13/32$ with $n = 4$-bit SNGs and $L = 16$-bit SN length. Following Algorithm 3.1 to analyze circuit accuracy, the target output value is first noted as $Z^* = X^*Y^* \approx 0.19$ (Step 1). Then, X^* and Y^* are truncated to 4-bit precision to yield $X = 7/16$ and $Y = 6/16$; these truncated values are input to the SNGs in Figure 3.2a (Step 2). Based on the circuit's logic and the independence of $\mathbf{X}$ and $\mathbf{Y}$, the expected output value is then found to be $Z = XY \approx 0.164$ (Step 3) which yields a squared bias of 6.95×10^{-4} (Step 4). Next, simulation or analysis can be used to find the variance of $\hat{Z}$ (Step 5). In Chapter 4, the variance of Figure 3.2a is derived as

$$\mathrm{Var}(\hat{Z}|\boldsymbol{\mathcal{X}}^*) = \frac{XY(1-X)(1-Y)}{L-1} \tag{3.7}$$

which yields $\mathrm{Var}(\hat{Z}) = 3.85 \times 10^{-3}$ with this example's X, Y and L. Finally, squared bias and variance are combined to give an overall MSE of about 4.54×10^{-3} (Step 6). Here, MSE has been derived analytically and matches the results of using simulation to estimate MSE.

Example 3.1 demonstrates a few important aspects of errors in stochastic circuits. First, bias can arise due to quantization of input values, just as in traditional computing systems. This type of quantization error can be reduced by increasing SNG bitwidth and is unaffected by SN length.

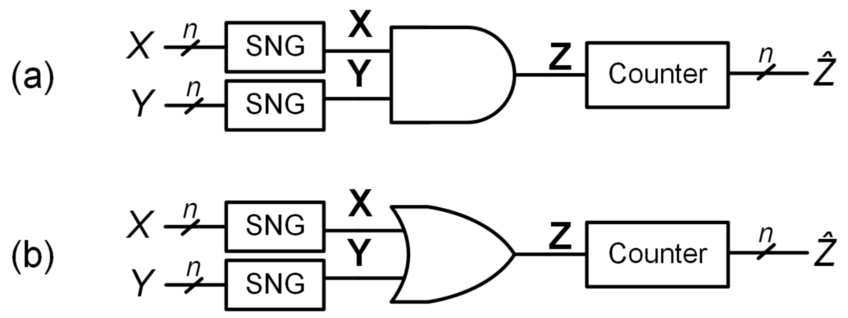


Figure 3.2: Examples of basic SC circuits: (a) unipolar multiplier; (b) unipolar biased adder.

Second, in this example, variance is about 5.6x higher than bias squared and thus dominates MSE. Consequently, the small bias from input quantization is sometimes ignored in analysis [74, 79].

Example 3.2: Next, other features of errors in stochastic circuits are demonstrated by applying Algorithm 3.1 to the circuit in Figure 3.2b which is intended to implement addition $Z^* = X^* + Y^*$ with SN length L and target input values X^* and Y^*. For this example, the SNG bitwidth is sufficiently large that there is no quantization error implying that $X = X^*$ and $Y = Y^*$. Based on the logic of an OR gate and independence of $\mathbf{X}$ and $\mathbf{Y}$, the circuit's expected output value is $Z = X + Y - XY$ which yields squared bias of X^2Y^2. The variance of OR adder can be derived as Equation (3.7), i.e., the variance is the same as in Example 3.1 [67]. Thus, the overall MSE of Figure 3.2b is

$$\text{MSE}(Z^*, \hat{Z}|\boldsymbol{\mathcal{X}}^*) = X^2Y^2 + \frac{XY(1-X)(1-Y)}{L-1} \tag{3.8}$$

Figure 3.3 illustrates the bias-variance decomposition of Equation (3.8) when $Y = 0.20$, $L = 32$ and X is varied. In Figure 3.3, the (non-squared) bias is visualized as the difference between the black target value curve and the red circuit expected value curve. Due to random fluctuations, the estimated value $\hat{Z}$ will fluctuate around its expected value as shown by the red dots in Figure 3.3. Each dot represents the estimated value $\hat{Z}$ of one simulation run. Sometimes $\hat{Z}$ fluctuates away from its mean and towards target value Z^* constituting lower overall error while other times $\hat{Z}$ fluctuates away from Z^* constituting higher overall error. The expected variation of $\hat{Z}$ is visualized in Figure 3.3 as the red shaded region whose width is $\hat{Z}$'s standard deviation (square root of variance). The average effect of fluctuation on MSE is captured as variance in MSE's bias-variance decomposition in Equation (3.4). Although $\hat{Z}$ can sometimes fluctuate closer to its target value, variance is always non-negative indicating that random fluctuations are always expected to increase overall error in terms of MSE.

Example 3.2 demonstrates a few further aspects of errors in stochastic circuits. First, rather than plain addition, an OR gate implements $X + Y - XY$ which leads to approximation bias that is much higher than last example's bias from quantization. Second, Figure 3.3 illustrates that both

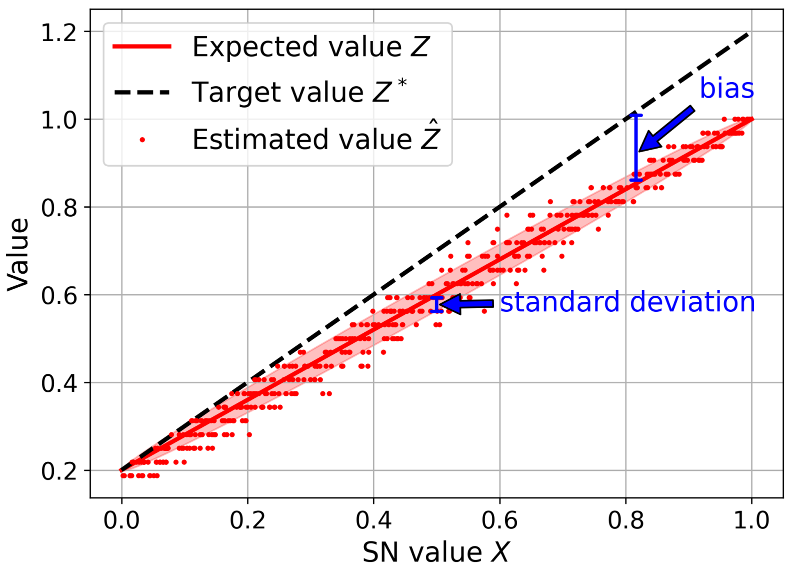


Figure 3.3: Visualization of bias-variance decomposition for an OR-gate adder with one input value fixed to 0.2.

bias and variance vary with the input value X. For example, bias is very low for small X which reflects the conventional wisdom that the OR gate adder is more accurate for small X and Y. Third, variance (the second term in Equation 3.8), decreases with L, but bias (the first term in Equation 3.8) does not. As L approaches infinity, the MSE converges to the bias and, importantly, does not approach zero. Thus, it is paramount to understand whether a stochastic circuit's error is due mainly to bias or due to variance because in the former case, error will not improve significantly by simply increasing L.

In summary, Algorithm 3.1 built around MSE's bias-variance decomposition encapsulates the first aspect of our BASE framework. Algorithm 3.1 has the following useful features:

1. Separability: Bias-variance decomposition enables systematic errors and random errors to be analyzed separately and then easily combined into a single overall error metric, MSE.

2. Flexibility: Both bias and variance can be derived from statistical analysis as in Examples 3.1 and 3.2 or bias and variance can be estimated via simulation. Thus, even pure simulation approaches can benefit from Algorithm 3.1's approach to error analysis.

3. Actionability: The relative magnitude of bias and variance can provide actionable insight into design improvement as we will demonstrate throughout this thesis. High variance can be combated by increasing SN length, by improving RNS design, or by improving circuit design as shown later. High bias is unaffected by SN length, but can instead be mitigated through

means like increasing SNG bitwidth, improving correlation control and using datapaths with lower approximation error.

3.3 Application-Specific Performance

In the foregoing error analysis framework of Algorithm 3.1, error is expressed as a function of the target inputs values $\boldsymbol{\mathcal{X}}^*$ and SN length L. While such formulas provide useful insights, it would also be helpful to have a single number which encapsulates a circuit's expected accuracy and which reproduces simulation error estimates that are often given as a single numeric value. The next part of our framework details such a metric, namely, the Bayes mean squared error (BMSE) defined [54] as

$$\mathrm{BMSE}(Z^*, \hat{Z}) = \mathbb{E}[\mathrm{MSE}(Z^*, \hat{Z}|\boldsymbol{\mathcal{X}}^*)] \tag{3.9}$$

where the expectation is taken with respect to $\boldsymbol{\mathcal{X}}^*$'s distribution. Like plain MSE, BMSE also has a bias-variance decomposition.

$$\mathrm{BMSE}(Z^*, \hat{Z}) = \mathbb{E}[\mathrm{Bias}^2(Z^*, \hat{Z}|\boldsymbol{\mathcal{X}}^*)] + \mathbb{E}[\mathrm{Var}(\hat{Z}|\boldsymbol{\mathcal{X}}^*)] \tag{3.10}$$

where the expectations are again taken with respect to $\boldsymbol{\mathcal{X}}^*$'s distribution.

One advantage of BMSE over plain MSE is that, given SN length L, BMSE is a single numeric value which can be used to rank stochastic circuits against other stochastic circuits and their non-stochastic counterparts. BMSE can be intuitively understood by expanding the expectation operator in Equation (3.9):

$$\mathrm{BMSE}(Z^*, \hat{Z}) = \underbrace{\int_{x^* \in \mathbb{U}^M}}_{\substack{\text{integrate} \\ \text{over all } x^*}} \underbrace{f_{\boldsymbol{\mathcal{X}}^*}(x^*)}_{\substack{\text{probability} \\ \text{density for } x^*}} \underbrace{\mathrm{MSE}(Z^*, \hat{Z}|\boldsymbol{\mathcal{X}}^* = x^*)}_{\substack{\text{MSE given} \\ \text{input values } x^*}} \, dx^* \tag{3.11}$$

where $\mathbb{U} = [0, 1]$ is the unit interval. In Equation (3.11), the MSE for each particular set of input values $\mathrm{MSE}(Z^*, \hat{Z}|\boldsymbol{\mathcal{X}}^* = x^*)$ is weighted by its corresponding probability density $f_{\boldsymbol{\mathcal{X}}^*}(x)$ and the MSE contributions from all valid sets of input values are integrated to compute a single error metric, BMSE. Note that the limits of integration in Equation (3.11) may change depending on the application, but usually unipolar SC circuits have target input values confined to the unit interval.

Bayes MSE and our BASE framework are "Bayesian" in the sense that prior knowledge is being leveraged to reason about something random in the future. In this case, knowledge about the circuit's input value distribution is being leveraged to provide a better numerical estimate of the circuit's expected error. The leveraging of prior information is a central idea in Bayesian

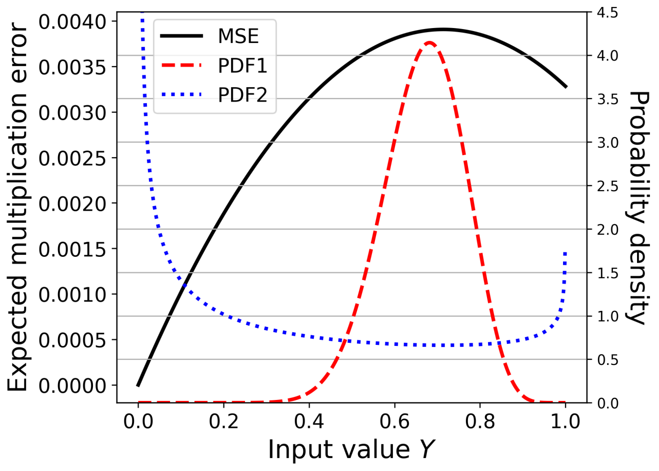


Figure 3.4: MSE curve and example PDFs for an AND-gate multiplier with one input value fixed to 0.7.

probability theory as exemplified by Bayes' theorem [54]

$$\mathbb{P}(H|D) = \frac{\mathbb{P}(D|H)\mathbb{P}(H)}{\mathbb{P}(D)} \tag{3.12}$$

where, for instance, H might be some hypothesis and D might be some data influencing belief in H. Equation (3.12) indicates how the prior probability of the hypothesis H, $\mathbb{P}(H)$, affects the posterior belief in the hypothesis given the data, $\mathbb{P}(H|D)$. In the case of our BASE framework, the prior belief about the input value distribution informs our posterior belief about the circuit's expected error (i.e., its BMSE).

BMSE is well-known in statistical estimation theory [54] and is a natural choice of error metric for SC. In fact, although not recognized by name, BMSE and other forms of Bayesian cost is what many simulations studies in SC implicitly measure [51, 79, 91] when reporting a numeric error metric from simulation. Usually, such works will refer to a simulation's numeric error estimate as 'mean absolute error' or 'mean squared error', but in our BASE framework, these values would be called 'Bayes mean squared error' or 'Bayes mean absolute error' while terms like 'mean squared error' are used to refer to expressions for error in terms of input values. Importantly, BMSE highlights the fact that stochastic circuit performance can be highly application-dependent because the expectation in Equation (3.9) is taken with respect to the input value distribution (IVD) $f_{\mathcal{X}^*}$ which is determined by the application. The following example shows the application-dependence of stochastic circuit accuracy in action.

Example 3.3 Consider the AND gate multiplier of Figure 3.2a with two uncorrelated 64-bit input SNs $\mathbf{X}$ and $\mathbf{Y}$ and output $\mathbf{Z}$. To illustrate the influence of an application's IVD on the multiplier's MSE, the circuit is simulated with $X = 0.7$ held fixed and Y drawn from one of two possible probability density functions (PDFs). Figure 3.4 illustrates the PDFs: PDF1 (red dashed line) is a peaked distribution centered at 0.70 while PDF2 (blue dotted line) is a U-shaped distribution with most of its density concentrated near 0.0 and 1.0. For both PDFs, $R = 50,000$ simulation runs are used to estimate BMSE as

$$\mathrm{BMSE}(Z^*, \hat{Z}) \approx \frac{1}{R}\sum_{r=1}^{R}(Z_r^* - \hat{Z}_r)^2 \tag{3.13}$$

where and $\hat{Z}_r$ and Z_r^* are **Z**'s estimated and target values during simulation run r. The BMSE was found to be 3.82×10^{-3} when using PDF1 to choose values for Y, but the BMSE is 2.28×10^{-3} (40% lower) when using PDF2 to choose values for Y.

This large difference in MSE is due to PDF2's more favorable IVD for Y. Specifically, PDF2 has a higher density for Y values that result in very low multiplication error while PDF1 has higher density for Y values that result in high error. Thus, the circuit is more accurate on average when Y is drawn from PDF2. Evidently, Y's PDF exerts a large influence on the average output error as captured by BMSE. This leads to an important conclusion: the accuracy of a stochastic circuit can be highly dependent on an application's IVD. Having knowledge of the IVD enables a better understanding of average accuracy which can lead to a more informed choice about SN length and thus circuit latency.

Case studies presented later in this thesis will give other concrete examples of the IVD's importance. For example, the neural network case study of Section 4.5 shows that the MNIST image classification benchmark [58] has a very favorable IVD that results in about five times lower BMSE than other benchmarks like CIFAR10 [57]. The analysis concludes that SC-based neural networks may perform worse on CIFAR10 not only because it is a harder classification benchmark than MNIST, but also because the CIFAR10's IVD results in noisier stochastic processing as indicated by its five times higher BMSE.

3.4 Comparing Circuit Accuracy

BMSE can be used to compare the accuracy of two circuits, e.g., by computing the difference or ratio of BMSE. In some cases, however, one circuit may have a lower BMSE than another circuit for some IVDs, but not for others. Such a scenario points to a potential problem with SC simulation studies that only use one method of choosing input values. For example, it may seem intuitive to compare different adder designs with uniformly random input data to determine how the adders compare. In this case, however, only one BMSE (or similar metric) is calculated and compared, but the comparison may not generalize well to other IVDs. For general design studies, it is important to appreciate that accuracy is application-specific and test against several benchmark applications or IVDs to get a full picture of accuracy behavior. Section 3.5 gives an example where various different IVDs yield different estimates for circuit error and conclusions about required SN length.

Comparing circuits using multiple IVDs is generally a good idea, but it is sometimes the case that circuit A is never less accurate than another circuit B for any IVD. In that case, circuit A is said to dominate B. Formally, a circuit A with output $\hat{A}$ dominates another circuit B with output $\hat{B}$ with respect to a target value Z^* when [54]:

1. $\mathrm{MSE}(Z^*, \hat{A}|\boldsymbol{\mathcal{X}}^* = \boldsymbol{x}^*) < \mathrm{MSE}(Z^*, \hat{B}|\boldsymbol{\mathcal{X}}^* = \boldsymbol{x}^*)$ for at least one input value set $\boldsymbol{x}^*$.

2. $\mathrm{MSE}(Z^*, \hat{A}|\boldsymbol{\mathcal{X}}^* = \boldsymbol{x}^*) \leq \mathrm{MSE}(Z^*, \hat{B}|\boldsymbol{\mathcal{X}}^* = \boldsymbol{x}^*)$ for all other values of $\boldsymbol{x}^*$.

In words, circuit A dominates B when A is more accurate for at least one input combination $\boldsymbol{x}^*$ and no less accurate than B for all other input combinations. Here, MSE is the chosen error metric, but dominance can also be phrased in terms of other cost functions like mean absolute error or mean percent error. The notion of estimator dominance leads to the following theorem.

Theorem 3.1: Consider two circuits A and B with the same input $\boldsymbol{\mathcal{X}}^*$ and the same target function Z^*. If A dominates B, then A never has a greater BMSE than B for any distribution of $\boldsymbol{\mathcal{X}}^*$.

Theorem 3.1 follows from the definition of dominance and the fact that BMSE is computed as a weighted integral (i.e., infinite sum) of MSEs where all weights are positive. When a circuit dominates another circuit, then the former is always at least as accurate as the latter for any IVD implying that proving dominance can be very useful. For example, our novel CeMux adder introduced in Chapter 5, dominates all other mux adders while also having lower area implying that it is the best mux-based adder design for any application.

3.5 Application of Error Analysis

One key application of error analysis is determining the critical SN length, i.e., the minimum SN length required to achieve a user-specified level of accuracy. Since error varies with the circuit's input values, latency can be determined in terms of the worst-case error [74], in terms of average-case error or in terms of expected error (BMSE) for a given IVD. As shown next, each approach can yield significantly different critical SN lengths.

Example 3.4: Consider again the SN multiplier of Figure 3.2a. Assuming inputs $\mathbf{X}$ and $\mathbf{Y}$ have length $L = 2^n$ and are generated with n-bit SNGs. The BMSE is derived for four different IVDs: (1) PDF1 of Figure 3.4, (2) PDF2 of Figure 3.4, (3) the worst-case IVD that maximizes BMSE

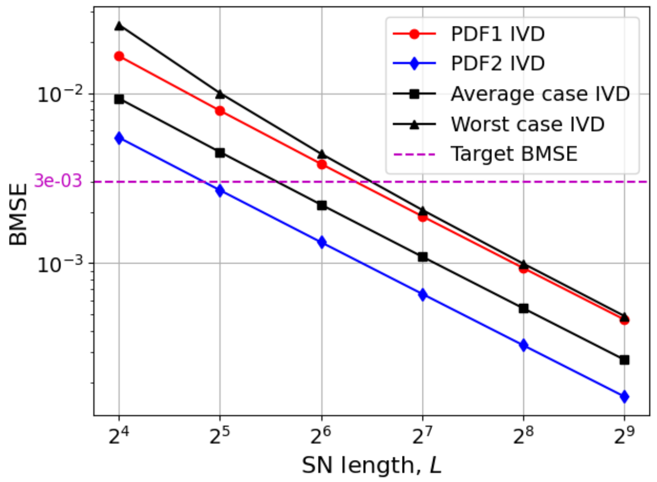


Figure 3.5: BMSE for an AND multiplier with target input values drawn from various IVDs.

and (4) the average-case IVD defined as when X^* and Y^* are chosen uniformly randomly from the unit interval $\mathbb{U} = [0, 1]$.

The BMSE results are plotted in Figure 3.5. The latency-error curve is lowest for the PDF2 case because input SN values that result in low error occur with high probability. The opposite situation occurs for PDF1 where the latency-error curve is high because input SN values that result in high error occur with high probability. Meanwhile, the average case latency-error curve falls between PDF1's and PDF2's error curve, which highlights the fact that average case analysis is not representative of these (and many other) IVDs. The BMSE curve for the worst-case IVD is the highest by definition.

The relationship between error and SN length in Figure 3.5 can be used to derive the critical SN length for a given accuracy. For instance, if the target BMSE is 3×10^{-3}, then Figure 3.5 shows that 32-bit SNs are needed when IVD is PDF2, 64-bit SNs are needed for the average case IVD and 128-bit SNs are needed for the worst case IVD or when the IVD is PDF1. The factor of four difference in required SN length demonstrates the influence that the IVD can have on the error. Note that SN length is restricted to be a power-of-two so that the output SN's value can be estimated using only a counter.

In summary, average case or worst case analysis can be useful, but more reliable results are achievable through analyzing the application's IVD. For example, stochastic circuits are designed to operate for a specific application such as image processing. The IVD of natural image pixels contrasts starkly with random values as in average case analysis and with adversarial examples as in worst case analysis. Later, it is shown that our CeMux adder design performs significantly better on real-world data, supporting the argument that application-specific performance should be considered when assessing stochastic circuit accuracy. Specifically, the case study in Section 5.2

finds that CeMux's error is over 100 times lower when applied to electrocardiogram denoising than when applied to uniformly random input data.

3.6 Summary

Our Bayesian analysis of stochastic errors (BASE) framework is composed of two important components. The first is MSE's bias-variance decomposition which allows systematic errors and random errors to be analyzed separately and then combined to yield the overall expected error. Specifically, systematic errors like approximation, quantization and correlation combine to yield the circuit's bias while random errors like SN fluctuation and transient faults are quantified as the circuit's variance. Bias squared and variance sum to give the circuit's overall error in terms of MSE.

The main advantage of using MSE and its bias-variance decomposition is that SC's various error sources can be accounted for in a straightforward manner because the analysis of systematic errors and random errors is separated. Furthermore, knowing whether bias or variance dominates the MSE can guide attempts to improve accuracy. For example, increasing SN length can help address a high circuit variance, but will not affect circuit bias.

The second component of our BASE framework is the Bayes MSE (BMSE) metric which gives a single numeric estimate for circuit accuracy that can be used to compare different SC designs. BMSE and other Bayesian cost functions [54] are what SC simulation studies implicitly measure when they report error estimates. Thus, BMSE reproduces simulation-derived MSE which allows for the validation of simulation results and enables a better understanding of simulation results as will be further illustrated in the upcoming NN case study of Section 4.5. In addition, BMSE also stresses the influence of the input value distribution on a circuit's expected accuracy. This fact underpins the importance of assessing a circuit's accuracy performance using a variety of expected applications in addition to the usual practice of testing circuits with uniformly random input data.

Our overall BASE framework is encapsulated in Algorithm 3.1 and illustrated in Figure 3.6. There are three key parameters to determine when evaluating a stochastic circuit's accuracy as measured by BMSE: the circuit's bias, the circuit's variance, and the input value distribution. In the next chapter, we show how these quantities can modeled for different important circuits like multipliers and adders and for different applications including digital filtering and neural networks. These examples will also help clarify and reinforce this chapter's key ideas.

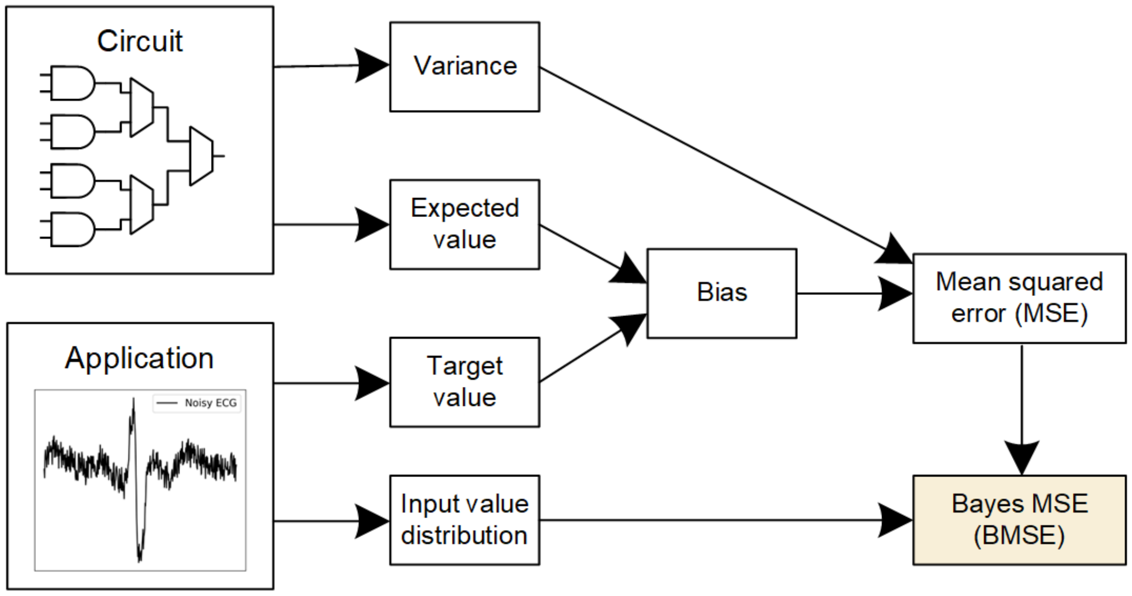


Figure 3.6: Bayesian analysis of stochastic errors (BASE) framework for error analysis. An arrow indicates influence between components of the framework.

CHAPTER 4

Statistical Models for Error Analysis

The Bayesian analysis of stochastic errors (BASE) framework proposed in Chapter 3 serves as a mathematical basis for analyzing circuit accuracy. BASE stresses that overall circuit accuracy depends on the circuit variance, circuit bias, and input value distribution (IVD). These components can be estimated using simulation or be analyzed statistically. Employing both methods and achieving matching results can increase confidence in the results and statistical models can further lead to better design understanding and innovations. Unfortunately, prior work on statistical analysis in SC is limited and, where it does exist, often produces incorrect results due to assumptions that do not map on to real circuit designs. In this chapter, we give examples of where existing analysis techniques produce inaccurate error estimates and propose several new statistical models for bias, variance, and the IVD. The work in this chapter is mostly published in [10, 11, 12].

4.1 Interplay Between Simulation and Analysis

Simulation models and statistical models can both be used to analyze stochastic circuit accuracy, but with differing advantages and benefits. Simulation provides an unequivocal account of error given that the simulation uses enough samples for statistical significance and that the circuit's logic is faithfully emulated. Thus, circuits should always be simulated for error analysis. Simulation may be slow in some cases, but when given the circuit's logic, it is straightforward to build a simulation model whereas it may not be obvious how to build a statistical model for a given design.

On the other hand, statistical models can provide a detailed account of how error varies with parameters like SN length, input values and correlation. For example, the error of an AND multiplier can sometimes be expressed as $\mathrm{MSE} = XY(1-X)(1-Y)/(L-1)$ [12]. Such expressions provide a fine-grain account of how error varies with the input values X, Y and SN length L. Statistical models can also inspire new designs and design improvements. For example, modeling mux adders as sampling units leads to various design improvements explained later. Statistical models should not strictly replace simulation models because analytic results should always be tested

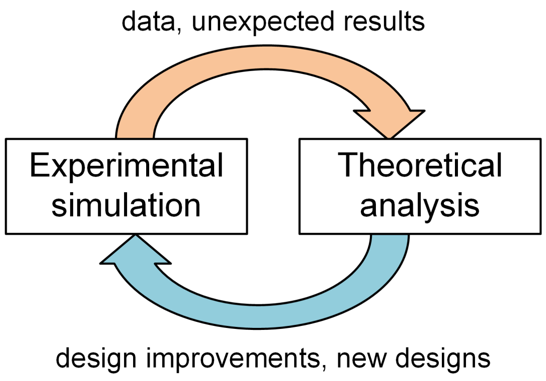


Figure 4.1: The synergistic interplay between simulation and analysis.

with simulation to validate their correctness. In short, simulation and analysis are complementary methods, not competing approaches.

Iterating between simulation and analysis can be an effective process as illustrated in Figure 4.1. Simulation provides data to validate analysis and to test new designs. Unexpected simulation results also provide new directions for statistical modeling. Meanwhile, analyzing unexpected or known phenomena can lead to new designs for simulation to test. Based on these benefits and others, a soundly based statistical approach to complement simulation error analysis is desirable. For example, our adder designs in Chapter 5 and SNG designs in Chapter 6 were developed as a result of many iterations of analysis and simulation that aimed to understand the errors of various stochastic circuits.

However, due to shortcomings in current SC error theory, simulation is often the only method used to estimate a circuit's accuracy which breaks the synergistic process in Figure 4.1. Some prior works have proposed analytic techniques that provide useful insights, but also fall short in some respects. For example, the SN model in [74] leads to overestimation of the variance in common SC designs while the variance analysis approach in [68] cannot be easily extended to include correlated SNs. In this chapter, we present new statistical models that overcome some of these shortcomings and accurately predict the error of important circuits like adders. Like all models, the ones we propose in this chapter are not perfect, but they are useful in that they have explained unexpected simulation results and have led to new and improved circuit designs.

4.2 Variance and SN Models

Variance quantifies a stochastic circuit's random error. The output SN's variance can be expressed as

$$\mathrm{Var}(\hat{Z}) = \mathrm{Var}\left(\frac{1}{L}\sum_{t=1}^{L} z_t\right) \tag{4.1}$$

which expands to

$$\mathrm{Var}(\hat{Z}) = \frac{1}{L^2}\sum_{t=1}^{L}\mathrm{Var}(z_t) + \frac{1}{L^2}\sum_{t=1}\sum_{\substack{k=1\\k\neq t}}\mathrm{Cov}(z_t, z_k) \tag{4.2}$$

where

$$\mathrm{Var}(z_t) = \mathbb{E}[z_t^2] - \mathbb{E}[z_t]^2 \tag{4.3}$$

is the variance of bit z_t and

$$\mathrm{Cov}(z_t, z_k) = \mathbb{E}[z_t z_k] - \mathbb{E}[z_t]\mathbb{E}[z_k] \tag{4.4}$$

is the covariance between bits z_t and z_k. Since variance quantifies squared error, taking its square root can be useful:

$$\sigma_Z = \sqrt{\mathrm{Var}(\hat{Z})} \tag{4.5}$$

where σ_Z is known as the standard deviation. Variance and standard deviation are used interchangeably in this thesis except where the distinction matters.

Equation (4.2) makes no assumptions about the SN bits and thus can be applied to any SN generated by any means. It applies to unipolar SNs and also to bipolar SNs whose bits are redefined to take values $\{-1, 1\}$ rather than $\{0, 1\}$ (see also Appendix A). Importantly, Equation (4.2) reveals that the variance of an SN's estimated value depends on the distribution of its bits $z_1 z_2 ... z_L$. Specifically, $\mathrm{Var}(\hat{Z})$ depends on the variance of $\mathbf{Z}$'s bits and the correlations between bits of $\mathbf{Z}$. Thus, modeling variance can be accomplished by modeling $\mathbf{Z}$'s bits. In the following section, the traditional SN model is covered. Weaknesses of the traditional approach is then discussed and our novel hypergeometric SN model [12] is proposed.

4.2.1 Traditional Binomial SN Model

In the traditional model of stochastic computing, all bits within a newly generated an SN are assumed to be independent [34, 74]. More formally, the traditional model presumes every bit of an L-bit SN $\mathbf{X}$ is an independent Bernoulli trial with P_x probability of success. Consequently, $\mathbf{X}$'s estimated unipolar value $\hat{X} = \frac{1}{L}\sum_{t=1}^{L} x_t$ is a scaled Binomial random variable with L trials and

P_x probability of success per trial. Formally, $\hat{X} = \frac{1}{L}A$ where $A \sim \text{Binomial}(P_x, L)$ [38]. Given the connection to the binomial distribution, we refer to such SNs as binomial SNs.[1] When the inputs to a combinational stochastic circuit are binomial SNs, the output **Z** is also a binomial SN [74]. Thus, **Z**'s estimated value is a scaled binomial random variable with

$$\text{Var}(\hat{Z}) = \frac{Z(1-Z)}{L} \tag{4.6}$$

where L is SN length and $Z = \mathbb{E}[\hat{Z}]$ by definition. Equation (4.6) can be derived from Equation (4.2) by noting that binomial SN **Z** has $\text{Var}(z_t) = Z(1-Z)$ and $\text{Cov}(z_t, z_k) = 0$. This section focuses on unipolar SNs, but it is worth noting that a version of Equation (4.6) exists for bipolar SNs as well. The bipolar variance expression is

$$\text{Var}(\hat{Z}) = \frac{1-Z^2}{L} \tag{4.7}$$

which can be derived from Equation (4.2) by noting that $\text{Var}(z_t) = 1 - Z^2$ and $\text{Cov}(z_t, z_k) = 0$ for bipolar binomial SN **Z** [74].

Two things are immediately apparent from Equations (4.6) and (4.7). First, $\text{Var}(\hat{Z})$ is straightforward to determine because Z is usually known or simple to compute. For example, an AND multiplier with $Z = XY$ would be found to have $\text{Var}(\hat{Z}) = XY(1-XY)/L$ where L is SN length. Second, variance scales as $1/L$ meaning that standard deviation decreases as $1/\sqrt{L}$. This relatively poor accuracy-latency trade-off is an oft-cited criticism of SC and motivates our work on developing more accurate circuits. The discussion of modeling variance of combinational circuits could end here, however, as shown next, Equation (4.6) often fails to predict the accuracy of basic stochastic circuits designed using traditional approaches.

Example 4.1: Consider an AND multiplier with inputs $\mathbf{X}$, $\mathbf{Y}$, and output $\mathbf{Z}$. The SN length is $L = 256$ bits. According to Equation (4.6), the circuit's standard deviation is predicted to be

$$\sigma_Z = \sqrt{\frac{XY(1-XY)}{256}} \tag{4.8}$$

which is plotted as the yellow surface in Figure 4.2a. This deviation can also be estimated using simulation. In this example, the SNG bit-width is large enough that there is no quantization error,

[1]In previous work, such SNs have occasionally been referred to as "Bernoulli SNs." Here, we opt to use the term binomial to better contrast with the SN model introduced later.

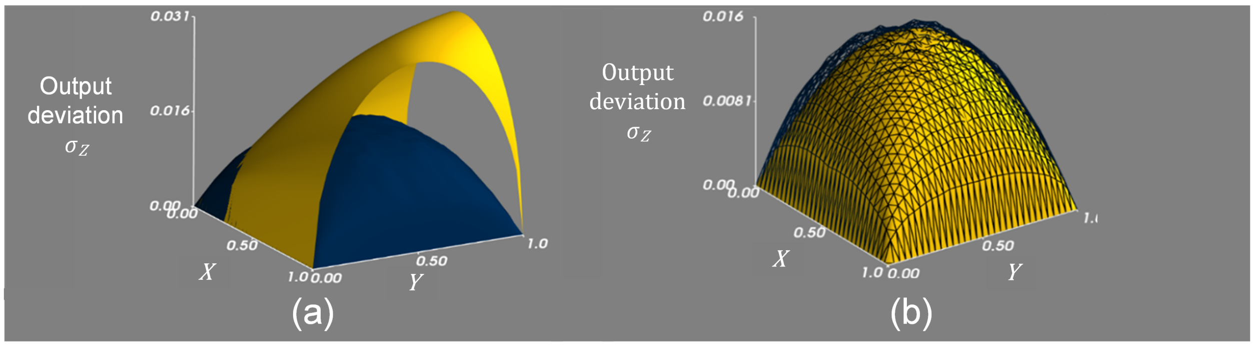


Figure 4.2: Analysis-derived deviation (yellow surface) and simulation-derived deviation (blue surface or mesh) for an AND-gate multiplier. (a) Binomial SN analysis; (b) hypergeometric SN analysis.

so $\hat{Z}$'s bias is zero. Thus, σ_Z can be estimated accurately as

$$\sigma_Z \approx \sqrt{\frac{1}{R}\sum_{r=1}^{R}(\hat{Z}_r - Z_r)^2} \tag{4.9}$$

where R is the number of simulation runs, $\hat{Z}_r$ is $\mathbf{Z}$'s estimated value during simulation run r, and Z_r is $\mathbf{Z}$'s expected (target) value during run r.

The multiplier is simulated with traditional SNGs built around linear feedback shift registers (LFSRs) and $R = 10,000$ simulation runs are used for each pair of values X, Y to achieve statistically significant estimates for σ_Z. The results are plotted as the blue surface in Figure 4.2a. Despite the simplicity of the AND multiplier, Figure 4.2a shows that the theory greatly overestimates the error and also poorly predicts how error varies with X, Y as indicated by difference in shape between the yellow and blue surfaces. This disagreement between theory and practice stems from assumptions made about LFSR SNs, i.e., SNs generated by LFSR-based SNGs. In short, LFSR SNs are not well modeled as binomial SNs.

4.2.2 Hypergeometric SN Model

The large discrepancy between the surfaces in Figure 4.2a suggests the need for a new model of LFSR SNs and we proposed such a model in [12]. The model is motivated by how LFSRs produce random numbers. Figure 4.3a shows a LFSR-based SNG for **X** while Figure 4.3b shows a 4-bit modified maximal LFSR state sequence.[2] Each clock cycle, the LFSR state acts as the pseudo-random number that determines **X**'s generated bit, but each LFSR state is only visited once during its state sequence. Put another way, LFSR states are visited in a pseudo-random order without

[2]Recall all LFSRs in this work are modified to include the all-0 state.

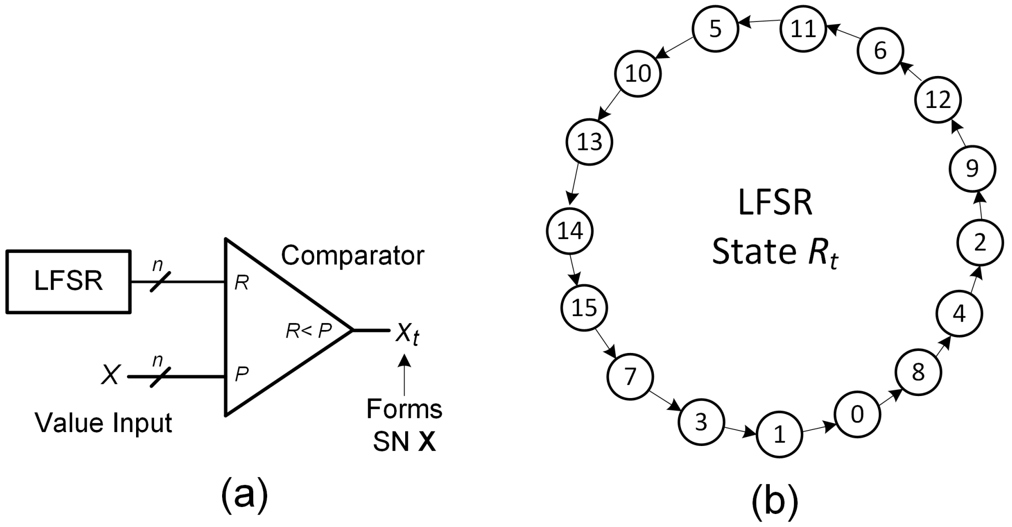


Figure 4.3: LFSR-based SN generation. (a) generic SNG with LFSR RNS; (b) 4-bit LFSR state diagram with added state 0.

replacement. Therefore, bits within $\mathbf{X}$ are not independent which violates the traditional binomial model's assumptions. For example, if a 1 is generated for $\mathbf{X}$ in the first clock cycle, then the probability of generating a 1 in the next clock cycle decreases since the LFSR state that led to the 1 being generated will not be repeated. This pattern of generating bits emulates trials of a hypergeometric random variable [38] which serves as the basis of our hypergeometric SN model [12].

As suggested by its name, the hypergeometric SN model is based upon the hypergeometric distribution [38]. A hypergeometric random variable $A \sim \text{Hypergeometric}(M, K, N)$ is defined by three parameters: the population size M, the total number of success states in the population $K \leq M$, and the number of trials $N \leq M$. For each trial, a state is drawn from the population without replacement and A's value is the number of success states drawn during the N trials. Note that if states are instead drawn with replacement, A would be a binomial random variable with probability of success K/M and N trials (i.e., $A \sim \text{Binomial}(K/M, N)$).

To contrast hypergeometric and binomial random variables, let $p = \frac{K}{M}$ and consider two random variables: $A \sim \text{Hypergeometric}(M, K, N)$ and $B \sim \text{Binomial}(p, N)$. A's expectation, and variance are

$$\mathbb{E}[A] = pN \tag{4.10}$$

$$\text{Var}(A) = p(1-p)N\frac{M-N}{M-1} \tag{4.11}$$

while, for B, these formulas are similar

$$\mathbb{E}[B] = pN \tag{4.12}$$

$$\mathrm{Var}(B) = p(1-p)N \tag{4.13}$$

The only difference between these statistics is that A's variance is lower than B's variance by $\frac{M-N}{M-1}$, a factor which decreases as the number of trails N increases. Intuitively, A has lower variance than B because hypergeometric random variables model situations involving sampling without replacement whereas binomial random variables model sampling with replacement behavior. The former has less opportunity for random fluctuations and thus lower variance.

Returning to SN models, suppose that $\mathbf{X}$ is an L-bit SN generated by an n-bit LFSR SNG with value input X. Since LFSRs sample random states without replacement, the bits of $\mathbf{X}$ can be modeled as trials of a hypergeometric random variable A with a 2^n population size, 2^nX success states and L trials: $A \sim \mathrm{Hypergeometric}(2^n, 2^nX, L)$. Consequently, $\mathbf{X}$'s estimated value is a scaled hypergeometric random variable: $\hat{X} = \frac{1}{L}A$. Equations (4.10) and (4.11) then imply that $\mathbb{E}[\hat{X}] = X$ and $\mathrm{Var}(\hat{X}) = 0$ in the common case that the entire LFSR sequence is used ($L = 2^n$). In short, compared to a similar binomial SN, a hypergeometric SN has the same expected value, but less variance and often with zero variance. Other work has also noted that LFSRs can generate input SNs that have no variance in their estimated value [51, 67]. Our work, however, goes further, by proposing an explicit model for such SNs and then deriving important consequences of the model including new, more accurate circuit designs.

Example 4.1, Part 2: The simulation of Example 4.1 used 8-bit LFSRs to generate 256-bit SNs and Figure 4.2a showed that the binomial SN model poorly predicted the circuit's accuracy. Since $\mathbf{X}$ and $\mathbf{Y}$ are LFSR SNs, they can be better modeled as hypergeometric SNs. First, it is noted that $\mathrm{Var}(\hat{X}) = \mathrm{Var}(\hat{Y}) = 0$ since the entire LFSR sequence is used. Next, the methodology presented in [67] is used to derive the output variance. Overall, the multiplier output's deviation is found to be

$$\sigma_Z = \sqrt{\frac{XY(1-X)(1-Y)}{L-1}} \tag{4.14}$$

which is plotted as the yellow surface in Figure 4.2b. The blue mesh in Figure 4.2b again shows the result of using simulation to approximate σ_Z accurately. The two surfaces closely match implying that modeling the LFSR input SNs $\mathbf{X}$ and $\mathbf{Y}$ as hypergeometric SNs leads to a correct prediction of output error.

Note that Equation (4.14) applies when $\mathbf{X}$ and $\mathbf{Y}$ are generated using the LFSR's entire sequence. In other words, $\mathbf{X}$ and $\mathbf{Y}$ have length 2^n where n is the LFSR bitwidth. Unless otherwise noted, all remaining analysis in this chapter will assume that any LFSR SN is also generated using

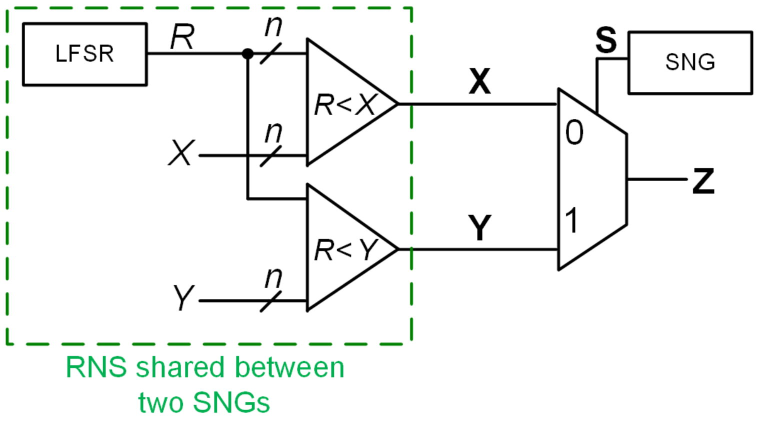


Figure 4.4: Mux adder with LFSR SNGs that share an RNS.

the entire LFSR sequence which is almost always the case in recent SC work. Next, other common circuits are analyzed using the hypergeometric SN model.

Example 4.2: Consider a standard 2-input mux adder where n-bit LFSR SNGs are used to generate input SNs $\mathbf{X}$, $\mathbf{Y}$, and control SN $\mathbf{S}$. The SN length is $L = 2^n$. Mux adders often fix $\mathbf{S}$'s value to $S = P_s = 0.5$, but here arbitrary S is considered such that the mux performs weighted addition of the form

$$Z = \bar{S}X + SY \tag{4.15}$$

where $\bar{S} = 1 - S$. Assuming no correlation is present, the hypergeometric model used with the variance propagation methodology in [67] yields the following prediction for the standard deviation of $\hat{Z}$.

$$\sigma_Z = \sqrt{\frac{S\bar{S}\big(X(1-X) + Y(1-Y)\big)}{L-1}} \tag{4.16}$$

For comparison, σ_Z can also be found using the binomial SN model and Equation (4.6).

$$\sigma_Z = \sqrt{\frac{\bar{S}X(1-\bar{S}X) + SY(1-SY) - 2S\bar{S}XY}{L}} \tag{4.17}$$

The deviation in the hypergeometric case is always less than or equal to the deviation in the binomial case for all X, Y when $S = 0.5$ and $L \geq 2$. This fact implies that hypergeometric input SNs generally lead to more accurate mux output than binomial input SNs.

An important result derived from the binomial SN model is that a single LFSR can be shared between $\mathbf{X}$'s and $\mathbf{Y}$'s SNGs without affecting accuracy [3, 51]. Figure 4.4 shows an example of such sharing which can greatly reduce circuit area, but also correlates $\mathbf{X}$ and $\mathbf{Y}$. Correlation amongst input SNs often causes a degradation in accuracy because SNs are usually required to

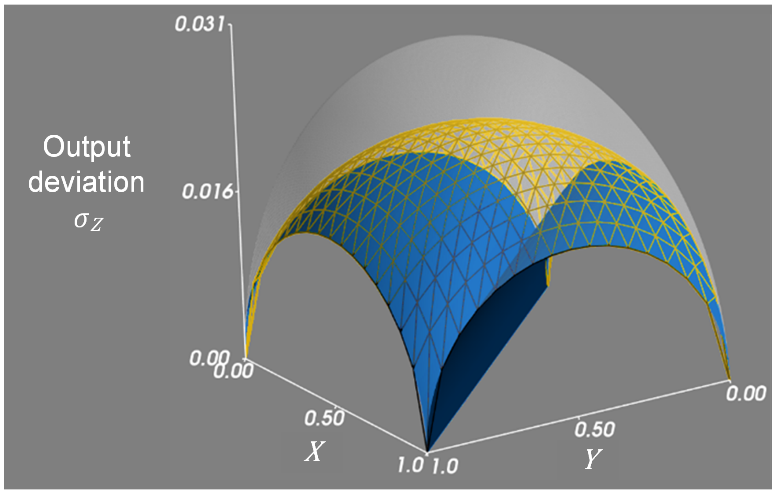


Figure 4.5: Mux adder output deviation with maximally correlated inputs (blue surface), with uncorrelated inputs (yellow mesh), or with anti-correlated input (translucent white surface).

be uncorrelated. However, the binomial model predicts that correlation amongst $\mathbf{X}$ and $\mathbf{Y}$ causes no change in accuracy in the case of mux addition. When tested, simulations involving LFSR-generated SNs shows that this prediction is incorrect. Since the mux circuit employs LFSR input SNs, the hypergeometric SN model can be used to develop new insight.

According to our hypergeometric SN model, correlation amongst hypergeometric input SNs significantly affects mux variance. Specifically, positive correlation ($\mathrm{SCC} > 0$) decreases mux variance while negative correlation ($\mathrm{SCC} < 0$) increases variance. In [12], the variance of a mux adder with hypergeometric inputs was derived for when $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = 1$ and for when $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = -1$. These two SCC levels are especially important because they occur often as the result of sharing an RNS between $\mathbf{X}$'s and $\mathbf{Y}$'s SNGs. These variance expressions were derived using the hypergeometric SN model and the general SN variance formula Equation (4.2). When $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = 1$, the output variance is

$$\sigma_Z = \sqrt{\frac{S\bar{S}(Y - X)(1 - (Y - X))}{L - 1}} \tag{4.18}$$

where L is the SN length and $X \leq Y$ is assumed without loss of generality. When $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = -1$, the output variance is:

$$\sigma_Z = \sqrt{\frac{S\bar{S}\Big(X(1 - X) + Y(1 - Y) + 2\min\big(XY, (1 - X)(1 - Y)\big)\Big)}{L - 1}} \tag{4.19}$$

Figure 4.5 visualizes the hypergeometric mux variance equations, (4.16), (4.18) and (4.19) by

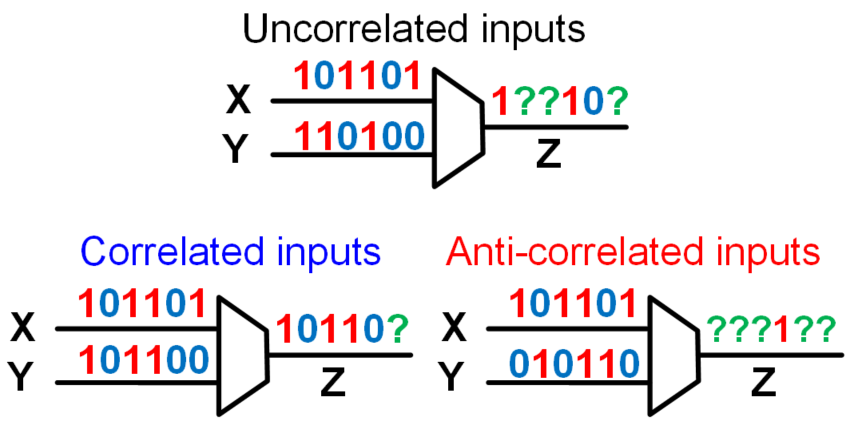


Figure 4.6: Comparison of mux behavior with varying levels of input correlation. An output bit denoted as ‘?’ is uncertain and depends on the select input.

plotting the output deviation when $S = 0.5$, $L = 256$, and X and Y are varied. These plots match those produced by estimating σ_Z via simulation of the mux with LFSR input SNs. Most noteworthy is that the standard deviation is lowest when $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = 1$ and highest when $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = -1$. In fact, the error is 0 when $X = Y$ in the $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = 1$ case. Thus, there is motivation to correlate mux inputs to save circuit area (through RNS sharing), and also to improve accuracy. If analysis relied only on the traditional binomial SN model, then these important conclusions would be missed which emphasizes the usefulness and importance of correct statistical models.

Statistical analysis and simulation both conclude that correlation substantially impacts mux adder accuracy for hypergeometric inputs, but not for binomial inputs. Figure 4.6 illustrates the intuition behind this result. When $\mathbf{X}$ and $\mathbf{Y}$ are correlated, their 1s overlap more often and the output becomes more certain compared to the uncorrelated case. In other words, the mux processing appears less random when $\mathbf{X}$ and $\mathbf{Y}$ are correlated which suggests the variance may be lower. When $\mathbf{X}$ and $\mathbf{Y}$ are binomial SNs, however, the increased determinism of the mux processing does not decrease variance because the variance of inputs $\mathbf{X}$ and $\mathbf{Y}$ propagates to the output $\mathbf{Z}$: noisy inputs plus less noisy processing equals noisy output. In contrast, when $\mathbf{X}$ and $\mathbf{Y}$ are hypergeometric or LFSR SNs, their variance is zero, and the less random mux processing induced by correlation leads to lower variance: non-noisy inputs plus less noisy processing equals less random output.

To summarize the foregoing analysis, Table 4.1 reports the average mux adder deviation for various input SN types and correlation levels. The calculation of average σ_Z assumes the mux input values X and Y are randomly chosen from $[0, 1]$, the mux select value is $S = 0.5$ and the SN length is $L = 256$. The hypergeometric model's predictions aligns closely with results from simulations of LFSR SNs which further validates our model. Overall, Table 4.1 yields three important conclusions about applying SN models to mux adders with LFSR SNs: 1) the binomial model greatly

SN model	$\mathrm{SCC}(\mathbf{X}, \mathbf{Y})$	**Average standard deviation** σ_Z	**Ratio of** σ_Z **to uncorrelated hypergeometric** σ_Z
Binomial	Any	0.0282	1.59
Hypergeometric	0	0.0178	1
Hypergeometric	$+1$	0.0123	0.69
Hypergeometric	-1	0.0214	1.21

Table 4.1: Average Mux Output Deviation

overestimates σ_Z by 59% on average in the uncorrelated input case, 2) the binomial model incorrectly concludes that correlation does not affect accuracy and 3) the hypergeometric model shows that maximum positive correlation decreases deviation by 31% on average while anti-correlation increases deviation by 21% on average compared to the uncorrelated case. These conclusions may have been missed in the past without using insights gleaned from the hypergeometric SN model.

4.2.3 Low-Discrepancy SNs

LFSRs are a common RNS-type used in SC because of their relative low area and sufficient randomness for accurate SN computation. When LFSRs are used to generate SNs, the stochastic circuit's root BMSE usually decreases proportionally to $1\sqrt{L}$ where L is SN length. This accuracy-latency trade-off can sometimes be substantially improved by using Sobol sequence generators instead of LFSRs [29, 65, 72].

Sobol sequences are an example of low-discrepancy sequences [4, 65]. Low discrepancy sequences are fully deterministic but share certain properties with random numbers. Figure 4.7 illustrates how Sobol sequences differ from random sequences and LFSR sequences. In each plot,

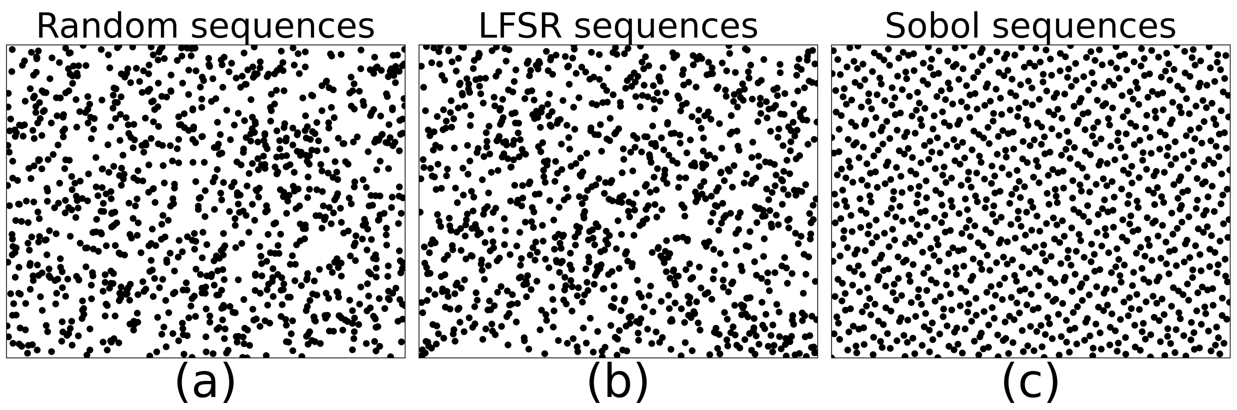


Figure 4.7: Comparison of random, LFSR, and Sobol sequences.

Target value	Sobol SN	Target value	Sobol SN
0/16	$\mathbf{X} = 0000\,0000\,0000\,0000$	8/16	$\mathbf{X} = 1010\,1010\,1010\,1010$
1/16	$\mathbf{X} = 1000\,0000\,0000\,0000$	9/16	$\mathbf{X} = 1110\,1010\,1010\,1010$
2/16	$\mathbf{X} = 1000\,0000\,1000\,0000$	10/16	$\mathbf{X} = 1110\,1010\,1110\,1010$
3/16	$\mathbf{X} = 1000\,1000\,1000\,0000$	11/16	$\mathbf{X} = 1110\,1110\,1110\,1010$
4/16	$\mathbf{X} = 1000\,1000\,1000\,1000$	12/16	$\mathbf{X} = 1110\,1110\,1110\,1110$
5/16	$\mathbf{X} = 1010\,1000\,1000\,1000$	13/16	$\mathbf{X} = 1111\,1110\,1110\,1110$
6/16	$\mathbf{X} = 1010\,1000\,1010\,1000$	14/16	$\mathbf{X} = 1111\,1110\,1111\,1110$
7/16	$\mathbf{X} = 1010\,1010\,1010\,1000$	15/16	$\mathbf{X} = 1111\,1111\,1111\,1110$

Table 4.2: Examples of 16-bit Sobol SNs

Counter state	Sobol state	Sobol value	Counter state	Sobol state	Sobol value
0000	0000	0	1000	0001	1
0001	1000	8	1001	1001	9
0010	0100	4	1010	0101	5
0011	1100	12	1011	1101	13
0100	0010	2	1100	0011	3
0101	1010	10	1101	1011	11
0110	0110	6	1110	0111	7
0111	1110	14	1111	1111	15

Table 4.3: 4-bit Sobol RNS Implementation

a random (x,y) coordinate is repeatedly chosen using either random sequences, LFSR sequences or Sobol sequences. The points in all three plots appear random to some extent, but the points in the Sobol plot fill the 2D space more evenly. This is a key feature of low-discrepancy sequences: they tend to appear random while also sampling spaces more evenly than truly random sequences, as shown in the 2D space of Figure 4.7c. Thus, low-discrepancy sequences are sometimes referred to as quasi-random sequences because of their similarity, yet distinct differences with random sequences.

When an SN is generated with a Sobol sequence, the resulting Sobol SN has its 1s roughly equally distributed throughout its length, whereas binomial SNs and LFSR SNs have their 1s randomly distributed. Table 4.2 shows examples of Sobol SNs. For instance, $\mathbf{X}$ with $X = 6/16$ is generated by comparing the Sobol sequence $R = 0, 8, 4, 12, 6, 10, 2, 14, 1, 9, 5, 13, 7, 11, 3, 15$ to X interpreted as the integer 6. On the first clock cycle, 0 is compared to X yielding a 1 for $\mathbf{X}$. Next X is compared with 8 and then with 4 to yield a 0 and 1, respectively. The process continues until all 16 bits of $\mathbf{X}$ are generated.

Filter name	**No RNS sharing average absolute error** ($\times 10^{-2}$)	**Basic RNS sharing average absolute error** ($\times 10^{-2}$)
f4-1	2.09	1.58
fs4-2	1.81	1.78
fs4-3	2.27	2.09
fs5-1	2.57	2.23
fs5-2	2.75	2.27
...	...	...
Average	4.22	4.03

Table 4.4: Excerpt of Experimental Results from Table 4 of [51].

Prior work has shown that using Sobol SNs improves SCs accuracy-latency trade-off [29, 65, 72]. For instance, when multiplying two L-bit Sobol SNs, the average error decreases as $1/L$, a notable improvement over the $1\sqrt{L}$ scaling when using LFSR-generated SNs. In terms of area cost, the simplest Sobol RNS can be implemented as the reverse state of an ordinary base-2 counter as in Table 4.3, but additional Sobol RNSs have much higher area cost than LFSRs. Despite their higher area, using Sobol RNSs tends to be more energy efficient because their higher accuracy allows the use of shorter SNs [65]. The benefits of Sobol SNs are most pronounced when a circuit contains few RNSs.

Sobol SNs share some features with hypergeometric SNs. If the SN length is $L = 2^n$, then any Sobol SN will have a precise number of 1s just like hypergeometric SNs. However, a Sobol SN will have its 1s roughly uniformly distributed throughout its length as shown in Table 4.2 whereas a hypergeometric SN will have its 1s randomly distributed throughout its length. Thus, the hypergeometric SN model is not perfectly applicable to Sobol SNs. For example, when modeling the error of an AND multiplier with Sobol SN inputs, our hypergeometric model overestimates the error. Although a new SN model is needed to fully capture the behavior of Sobol SNs. our hypergeometric model can yield some insight. For example, mux adders with Sobol input SNs also benefit from correlation just like mux adders with hypergeometric SNs.

4.2.4 Applications of Accurate SN Models

One useful application of the hypergeometric SN model is that it can help explain unexpected results found in simulations performed in prior studies. In [51], mux adders with hundreds of inputs are used to implement finite impulse response (FIR) filters with SC. The filter designs aggressively share LFSR RNSs among many SNGs and this sharing is expected to introduce some additional error in return for area savings. Data from Table 4 in [51] is presented here as Table

4.4. It gives error estimates from simulation and, surprisingly, shows that in most cases and on average, basic LFSR sharing produces smaller errors than the the no RNS sharing case. This unexpected accuracy improvement is mentioned but not explained in [51] presumably because the paper's focus is on other aspects of SNG design. However, our analysis of mux adders with LFSR SNs clearly explains the result: LFSR sharing introduces correlation amongst the input SNs which leads to higher mux adder accuracy compared to not sharing LFSRs. This example helps highlights that accurate analysis can help explain anomalous simulation results which is one key motivation for this line of work.

Another useful application of our statistical analysis is that it can prove circuit dominance amongst different designs by comparing the derived variance equations. Such equations include the AND multiplier standard deviation equations (4.8), (4.14) and the mux adder deviation equations (4.16) to (4.19). Recall that a circuit dominates another circuit when the former is no less accurate than the latter for all possible input value combinations and is more accurate for at least one input combination. Dominance is a strong and useful statement about circuit accuracy. The following are a few of the dominance relations and their consequences.

1. For practical SN lengths $L \geq 4$, AND multipliers and mux adders with uncorrelated hypergeometric inputs dominate equivalent circuits with binomial inputs. Therefore, these simple circuits should use LFSR RNS instead of other RNS like S-box random number generator (SBoNG) because the former generates hypergeometric SNs while the latter generates more binomial-like SNs [75].

2. Mux adders with correlated hypergeometric inputs dominate those with uncorrelated hypergeometric inputs. Therefore, basic mux circuits should share their LFSR RNS to save both area and accuracy.

3. Mux adders with uncorrelated hypergeometric input SNs dominate those with anti-correlated hypergeometric inputs. Therefore, anti-correlation should be avoided whenever possible in favor of independent or correlated SNs.

A third application of our statistical analysis is actionable design improvement. Consider the subtractor circuit in Figure 4.8a that operates on bipolar SNs $\mathbf{X}$ and $\mathbf{Y}$ and computes $Z = 0.5(X - Y)$ [63]. Since the subtractor is built around a mux, the conventional binomial SN model (incorrectly) dictates that an LFSR can be shared amongst $\mathbf{X}$ and $\mathbf{Y}$'s SNGs to save area and not affect accuracy. This reasoning results in the design of Figure 4.8a which is sub-optimal due to correlation. Specifically, $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = 1$ after generation, but then $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}') = -1$ after $\mathbf{Y}$ passes through the inverter. Thus, the inputs to the subtractor's mux are anti-correlated and, because an LFSR RNS is used, the hypergeometric model correctly predicts that the mux's error will be relatively high.

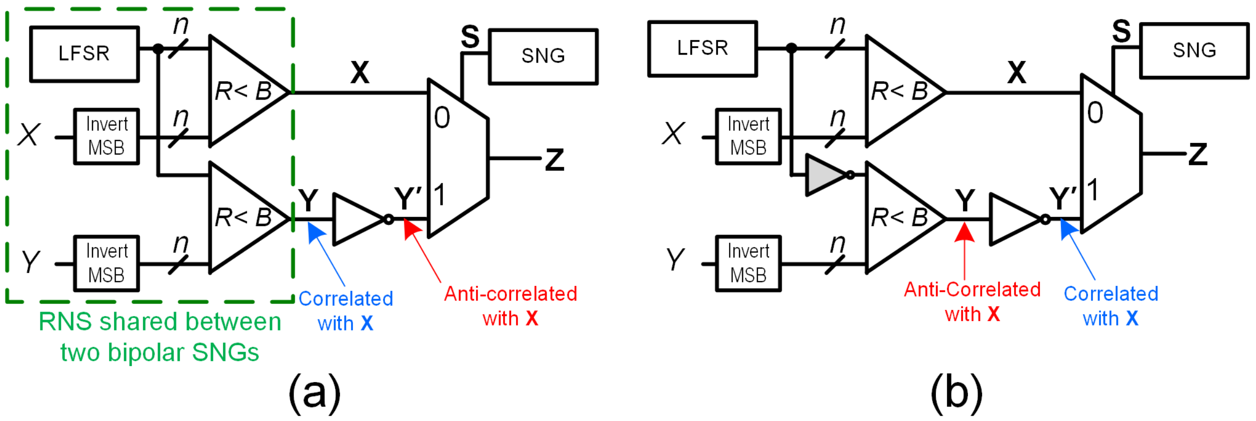


Figure 4.8: SN subtractor designs: (a) traditional subtractor circuit; (b) CAM subtractor.

Fortunately, a relatively small modification can be used to improve the subtractor design: the LFSR state can be inverted before it enters $\mathbf{Y}$'s comparator as illustrated in Figure 4.8b. This change causes $\mathbf{Y}$ to initially be anti-correlated with $\mathbf{X}$ with $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = -1$. Then, after passing through the inverter and before entering the mux, $\mathbf{Y}'$ becomes correlated with $\mathbf{X}$ with $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}') = 1$. Our resulting design is named the correlation-adjusted multiplexer (CAM) subtractor and is significantly more accurate than the traditional subtractor design. Assuming all values of $X, Y \in [-1, 1]$ are equally likely to occur, the CAM subtractor reduces standard deviation by 39% on average compared to the original subtractor circuit. The CAM subtractor is an exemplar of how insight derived from improved analysis can be leveraged to improve circuit design.

4.2.5 SN Model Summary

Guided by the foregoing analysis, we propose hypergeometric SNs, i.e., SNs whose bits are treated as hypergeometric trials. In general, the hypergeometric model is suitable for SNs produced from SNGs built around any uniform, non-repeating RNS such as an LFSR.[3] Once an input SN has been identified as hypergeometric, its variance can be computed using Equation (4.11) and then used with the variance propagation formulas of [67] to predict the output variance of a stochastic circuit when the inputs are uncorrelated. When correlation is present, output variance can be derived using the definition of variance (Equation (4.2)) and the circuit's logic.

Overall, the comparison of binomial and hypergeometric SNs demonstrates the value of having accurate analytic models of stochastic computing. The benefits include:

[3]Another notable, but rarely used, RNS of this type is the rule 90-150 hybrid cellular automata that has been shown to be a competitive alternative to the LFSR for SN generation [18, 46]

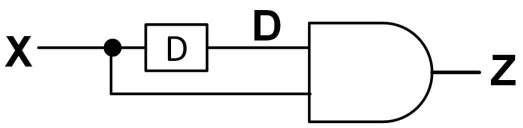


Figure 4.9: Stochastic squarer circuit with $Z \approx X^2$.

1. Explanations of anomalous simulation results. The simulation analysis of [51] noted that accuracy was noticeably higher when mux inputs were correlated whereas the prediction in [51] was that correlation would not affect accuracy. Our analysis of mux adders and correlation explains this data and validates the unexpected simulation results in [51].

2. Definitive proof of circuit dominance. Hypergeometric input SNs nearly always lead to lower output variance than binomial input SNs for important circuits like multipliers and adders. Consequently, LFSRs should be favored over other RNS like SBoNG for these designs. Without analytic models, simulation alone can offer evidence, but not definite proof for circuit dominance.

3. Insights for new designs. Our analysis on mux adders and correlation led to the design of the novel CAM subtractor which requires similar logic as a basic SN subtractor, but is 39% more accurate on average. In Chapter 5 these insights developed further and culminate in the CeMux design, a many-input mux adder that is significantly more accurate than other mux adders while also occupying less area.

4.3 Bias and Autocorrelation Error

As mentioned earlier, bias can arise from many sources including approximation, quantization and correlation. One type of correlation-related bias is autocorrrelation error which occurs when bits *within* a single SN do not meet required correlation levels. Autocorrelation causes bias for sequential stochastic circuits like the squaring circuit mentioned in the introduction and shown again in Figure 4.9.

Although recognized, autocorrelation has been largely neglected in the SC context while correlation between SNs (also known as cross-correlation) has been the topic of many studies [3, 12, 13, 23, 51]. In particular, mathematical tools are not available for statistically modeling autocorrelation in SC or measuring its impact on system accuracy. Here, an approach for managing autocorrelation in the SC context is proposed. It includes a metric for quantifying SN autocorrelation and a method for determining autocorrelation bias via analysis. The content presented in this section is published in [10]. Before introducing our autocorrelation model, we provide a motivating example of autocorrelation error.

Example 4.3: Consider the stochastic squaring circuit in Figure 4.9. The circuit has a single input $\mathbf{X} = x_1 x_2 ... x_L$ and a delay unit implemented with a D flip-flop. The flip-flop output $\mathbf{D} = d_1 d_2 ... d_L$ is a delayed copy of $\mathbf{X}$ with $d_t = x_{t-1}$ and $d_0 = 0$. If L is large, then $\mathbf{D}$ will have approximately the same value as $\mathbf{X}$:

$$D = \mathbb{P}(d_t = 1) \approx \mathbb{P}(x_t = 1) = X \tag{4.20}$$

If $\mathbf{X}$ has no autocorrelation, then $\mathbf{X}$ and $\mathbf{D}$ will be uncorrelated, and the AND multiplier will output $Z = XD \approx X^2$. However, suppose $X^* = 0.5$ and $\mathbf{X}$ is highly autocorrelated in that a 1 always follows a 0 and vice versa: $\mathbf{X} = 101010\ldots$. In this case, the squarer output would be all 0s, i.e., $\mathbf{Z} = 000000...$ with $\hat{Z} = 0$ regardless of SN length. Noting that the target output is $Z^* = 0.5^2 = 0.25$, the large bias due to autocorrelation is $\hat{Z} - Z^* = -0.25$.

As Example 4.3 shows, autocorrelation can cause very large bias in sequential circuits which motivates the development of a methodology that quantifies autocorrelation error for a given circuit and input autocorrelation. First, a metric for quantifying autocorrelation is needed. In general, autocorrelation describes the similarity between values in a sequence of random variables, such as the bits of an SN. The k-th order *autocovariance function* [10, 17] of SN $\mathbf{X} = x_1 x_2 ... x_L$ is

$$r_{\mathbf{X}}(k) = \mathbb{E}[x_t x_{t+k}] - \mathbb{E}[x_t]^2 \tag{4.21}$$

where autocovariance measures the dependence between a bit of $\mathbf{X}$ at some time t and a bit of $\mathbf{X}$ k cycles later. Note that $\mathbb{E}[x_t] = P_x$ since $x_t \in \{0, 1\}$ which leads to a close connection between $r_{\mathbf{X}}(k)$ and conditional probability given by

$$r_{\mathbf{X}}(k) = P_{x_{t+k}|x_t} - P_x^2 \tag{4.22}$$

where $P_{x_{t+k}|x_t}$ is shorthand for $\mathbb{P}(x_{t+k} = 1 | x_t = 1)$.

The autocovariance $r_{\mathbf{X}}$ can be derived through analysis or be estimated for $\mathbf{X} = x_1 x_2 ... x_L$ as

$$r_{\mathbf{X}}(k) \approx \frac{1}{L} \sum_{t=1}^{L-k} x_t x_{t+k} - \Big(\frac{1}{L} \sum_{t=1}^{L} x_t \Big)^2 \tag{4.23}$$

In the SC context, autocovariance has a drawback in that its range depends on the SN's value, so comparing the severity of autocorrelation for SNs can be difficult. For example, if $X = 0.5$, then $r_{\mathbf{X}}$ generally varies between $[-0.25, 0.25]$ but if $X = 0.25$, then $r_{\mathbf{X}}$ generally varies between $[-0.0625, 0.1875]$. To normalize autocorrelation values across SN values, the k-th order *autocor-*

relation function can be used. It is defined as

$$\rho_{\mathbf{X}}(k) = \frac{r_{\mathbf{X}}(k)}{P_x(1-P_x)} \tag{4.24}$$

for an SN $\mathbf{X}$ where k is the lag time of interest [17]. For example, $\rho_{\mathbf{X}}(2)$ measures the self-similarity between $\mathbf{X}$ and $\mathbf{X}$ delayed by two clock cycles. Put another way, $\rho_{\mathbf{X}}(2)$ measures the correlation between bits x_t and x_{t-2}.

The autocorrelation function $\rho_{\mathbf{X}}$ normalizes autocovariance based on the $\mathbf{X}$'s value and is useful for comparing autocorrelation of SNs with different values. In general, $\rho_{\mathbf{X}}$ is bound to the range of $[-\delta, 1]$ where a value of 1 corresponds to a high amount of autocorrelation in $\mathbf{X}$. Likewise, $-\delta = -\min(\frac{P_x}{1-P_x}, \frac{1-P_x}{P_x})$ corresponds to a high amount of anti-autocorrelation in $\mathbf{X}$ and 0 corresponds to no autocorrelation $\mathbf{X}$. In this work, $r_{\mathbf{X}}(k)$ is used for analysis while $\rho_{\mathbf{X}}(k)$ is used when comparing autocorrelation error for SNs of different values as in an error plot.

For our autocorrelation error analysis, conditional probabilities derived from the auto-covariance function $r_{\mathbf{X}}(k)$ are used extensively. For instance, four quantities determine x_{t+1}'s dependence on x_t. Two of the quantities are $\mathbb{P}(x_{t+1} = 1 | x_t = 1)$ which is abbreviated as $P_{x|x}$ and $\mathbb{P}(x_{t+1} = 0 | x_t = 0)$ abbreviated as $P_{\bar{x}|\bar{x}}$. The other two quantities are $P_{x|\bar{x}}$ and $P_{\bar{x}|x}$. These four quantities can be derived from autocovariance $r_{\mathbf{X}}(1)$ in the following manner:

$$P_{x|x} = r_{\mathbf{X}}(1) + P_x^2 \tag{4.25}$$

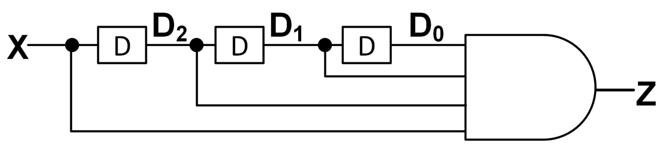
$$P_{\bar{x}|x} = 1 - P_{x|x} \tag{4.26}$$

$$P_{\bar{x}|\bar{x}} = \frac{P_{x|x}P_x}{1-P_x} \tag{4.27}$$

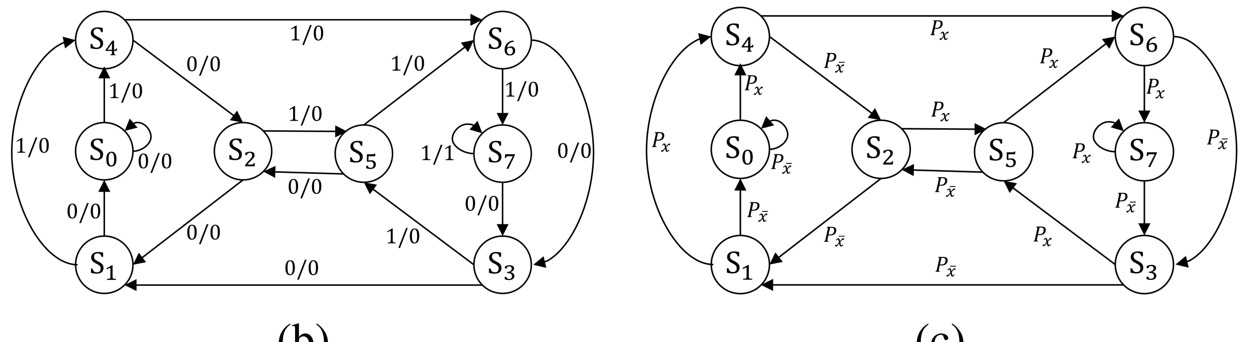
$$P_{\bar{x}|\bar{x}} = 1 - P_{x|\bar{x}} \tag{4.28}$$

where Equation (4.27) is derived using Bayes' theorem. These four quantities completely describe x_{t+1}'s dependence on x_t. Next, the methodology for applying these conditional probabilities is developed.

Sequential stochastic circuits are usually modeled using Markov chains [18, 64, 88]. Consider the quartic circuit in Figure 4.10a where each of the three D flip-flops outputs a delayed copy of $\mathbf{X}$. The three delayed copies are then multiplied together with $\mathbf{X}$ to yield $Z \approx X^4$. The operation of the quartic circuit can also be understood in a more systematic way by examining its state diagram shown in Figure 4.10b. The transitions in the state diagram are probabilistic since they are determined by $\mathbf{X}$'s stochastic bits. Thus, the state diagram can be redrawn as a Markov chain as in Figure 4.10c. The steady state of this Markov chain can then be analyzed to determine $\mathbf{Z}$'s expected value as explained next.



(a)



(b) (c)

Figure 4.10: Stochastic quartic circuit where $Z = X^4$: (a) circuit diagram; (b) Mealy-style state diagram; (c) Markov chain.

The Markov chain in Figure 4.10c has a matrix representation $\boldsymbol{M} = [m_{ij}]$, where each element m_{ij} is the probability of transitioning from state S_i to state S_j.

$$M = \begin{bmatrix} 1-P_x & 0 & 0 & 0 & P_x & 0 & 0 & 0 \\ 1-P_x & 0 & 0 & 0 & P_x & 0 & 0 & 0 \\ 0 & 1-P_x & 0 & 0 & 0 & P_x & 0 & 0 \\ 0 & 1-P_x & 0 & 0 & 0 & P_x & 0 & 0 \\ 0 & 0 & 1-P_x & 0 & 0 & 0 & P_x & 0 \\ 0 & 0 & 1-P_x & 0 & 0 & 0 & P_x & 0 \\ 0 & 0 & 0 & 1-P_x & 0 & 0 & 0 & P_x \\ 0 & 0 & 0 & 1-P_x & 0 & 0 & 0 & P_x \end{bmatrix} \tag{4.29}$$

The circuit's state during clock cycle t is then represented by an 8-element vector $\boldsymbol{\pi}_t = [\pi_{t,0}, \ldots, \pi_{t,7}]$ where $\pi_{t,i}$ is the probability that the state is S_i during cycle t. The next state $\boldsymbol{\pi}_{t+1}$ can be determined from present state with

$$\boldsymbol{\pi}_{t+1} = \boldsymbol{\pi}_t \boldsymbol{M} \tag{4.30}$$

In the limit of $t \to \infty$, $\boldsymbol{\pi}_t$ will converge into a steady state $\boldsymbol{\pi}^*$ where $\boldsymbol{\pi}^* = \boldsymbol{\pi}^* \boldsymbol{M}$. This steady state

$\boldsymbol{\pi}^* = [\pi_1^*, \pi_2^*, ..., \pi_7^*]$ can be used to find $\mathbf{Z}$'s expected value as

$$Z \approx \sum_{i=0}^{i=7} \pi_i^* P_{z|S_i} \tag{4.31}$$

where $P_{z|S_i}$ is the probability that $\mathbf{Z}$ is 1 given the circuit is in state S_i. Equation (4.31) approximates Z because steady state analysis does not account for the circuit's start state. However, if the SN length is long, then the influence of the start state is relatively small and the approximation will be accurate.

According to its state diagram, the quartic circuit has $P_{z|S_i} = 0$ for all $S_i \neq 7$ and $P_{z|S_7} = P_x$. Meanwhile, the steady state $\boldsymbol{\pi}^*$ can be found with Equations (4.29) and (4.30). Putting everything together in Equation (4.31) gives $Z \approx \pi_7^* P_x = P_x^4 = X^4$ proving that the quartic circuit performs as reasoned previously. The advantage of the Markov chain approach is that it can be systematically applied to more complicated circuits where Z cannot be intuitively derived. The Markov chain approach also serves as the basis for our autocorrelation error analysis.

The foregoing Markov chain analysis of the quartic circuit offers no opportunity to include autocorrelation information since the analysis only used P_x, $P_{z|S_i}$ and standard techniques for analyzing Markov chains. Thus, the analysis would yield the same result, $Z \approx X^4$, for uncorrelated $\mathbf{X}$ or highly autocorrelated $\mathbf{X}$. However, autocorrelation analysis is possible by modifying the transitions in the Markov chain. Observe that the D flip-flops in the quartic circuit form a shift register which acts as a finite memory of 3 bits. For example, when the circuit is in state S_3, the D flip-flop bits are $d_{2,t}d_{1,t}d_{0,t} = 011$. To arrive at such a state, the previous three bits of $\mathbf{X}$ must have been $x_{t-1}x_{t-2}x_{t-3} = 011$. Thus, S_3's outgoing P_x and $P_{\bar{x}}$ transitions in the Markov chain can be relabeled as $P_{x|\bar{x}xx} = \mathbb{P}(x_t = 1|\bar{x}_{t-1}, x_{t-2}, x_{t-3})$ and $P_{\bar{x}|\bar{x}xx} = \mathbb{P}(x_t = 0|\bar{x}_{t-1}x_{t-2}x_{t-3})$, respectively.

The reasoning applied to S_3 can be applied to any state S_i because $d_{2,t} = x_{t-1}$, $d_{1,t} = x_{t-2}$ and $d_{0,t} = x_{t-3}$ is true for all states. The result is the Markov chain in Figure 4.11a which contains conditional probabilities that carry autocorrelation information unlike the original Markov chain in Figure 4.10c. Unfortunately, deriving or estimating the 16 different conditional probabilities (e.g., $P_{x|xxx}$ and $P_{\bar{x}|\bar{x}\bar{x}x}$) in Figure 4.11a can be laborious or difficult. Instead, each conditional probability can be simplified to contain only $\mathbf{X}$'s most recent bit x_{t-1}. For instance, $P_{x|x\bar{x}\bar{x}}$ becomes $P_{x|x}$ and $P_{\bar{x}|xx\bar{x}}$ becomes $P_{\bar{x}|x}$. Simplifying all conditional probabilities in this manner results in the Markov chain of Figure 4.11b which contains just four distinct conditional probabilities that can be easily derived from autocovariance using Equations (4.25) to (4.28).

The Markov chain in Figure 4.11b containing simplified conditional probabilities can be analyzed in same the manner as before to yield a different steady state $\boldsymbol{\pi}^*$ which is in terms of conditional probabilities. The steady state $\boldsymbol{\pi}^*$ can be difficult to determine by hand, but can be de-

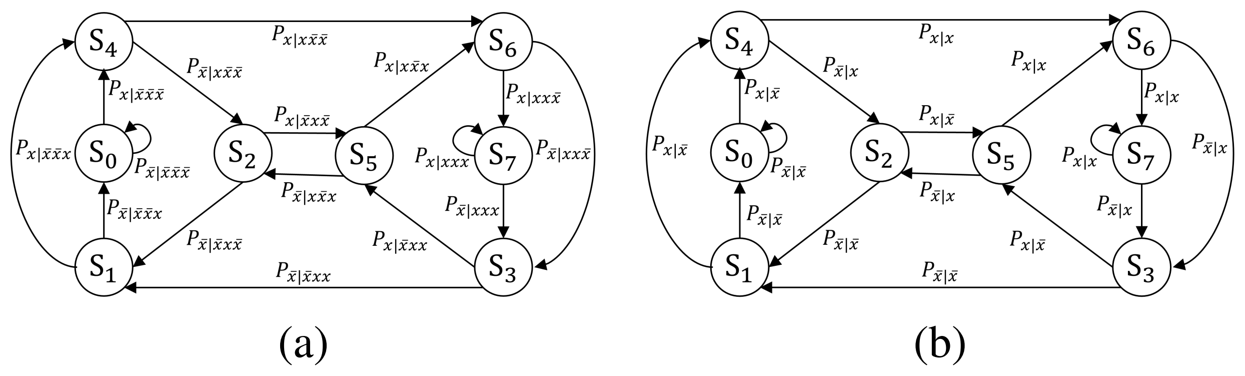


Figure 4.11: Modified quartic circuit Markov chains. Markov chain with: (a) with conditional probabilities denoting state transition probabilities; (b) simplified conditional probabilities

rived numerically. The expected output value can then be estimated using Equation (4.31) which yields $Z = \pi_7^* P_{x|x}$. And finally, the circuit's bias can be estimated as

$$\text{Bias}(Z^*, Z) \approx \pi_7 P_{x|x} - P_x^4 \tag{4.32}$$

If there is no quantization error, then the bias is completely due to autocorrelation and Equation (4.32) is approximately the autocorrelation error. Figure 4.12a shows how the autocorrelation error varies with input value $X = P_x$ and with first-order autocorrelation $\rho_{\mathbf{X}}(1)$. When autocorrelation is large (close to +1), the autocorrelation error is very high and, importantly, it can not be decreased

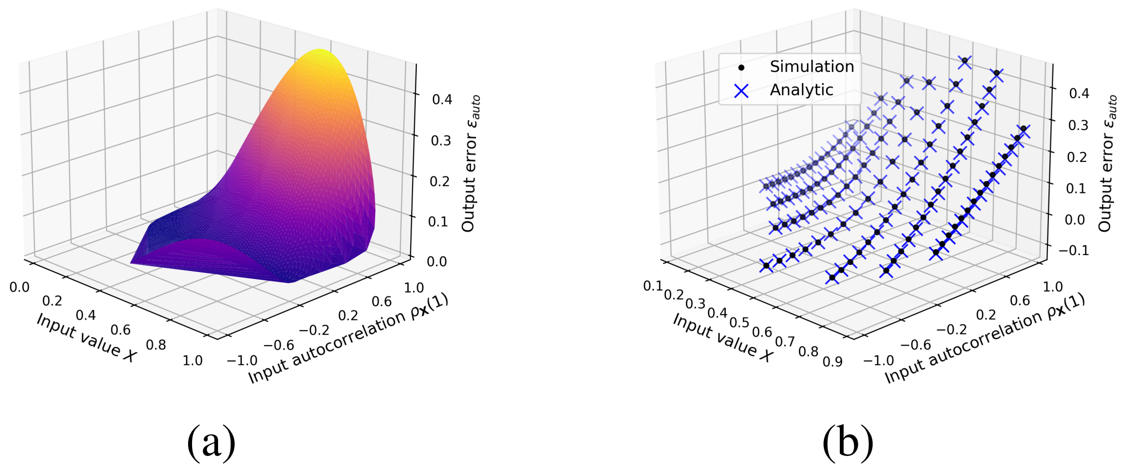


Figure 4.12: Quartic circuit autocorrelation error: (a) analysis results; (b) comparison of analysis and simulation results.

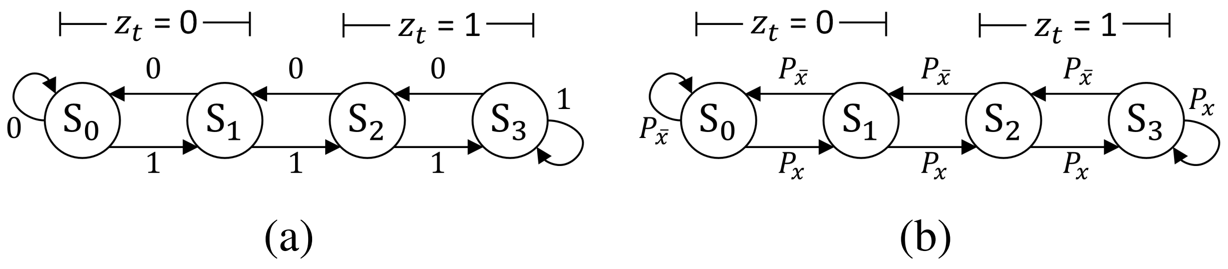


Figure 4.13: Diagrams for a stochastic tanh circuit with input $\mathbf{X}$, $G = 4$ states and output $\mathbf{Z}$: (a) Moore-style state diagram; (b) corresponding Markov chain.

by increasing SN length unless increasing the SN length also decreases $\mathbf{X}$'s autocorrelation. This analysis was confirmed by simulation in [10] and the results are shown in Figure 4.12b where the analysis predictions closely match the results of bitstream simulation.

While our autocorrelation analysis was successful, it must be further generalized to apply to other sequential stochastic circuits. In [88], two major classes of sequential stochastic designs are identified: shift register based (SRB) and up/down counter based (UCB). The quartic and squarer circuits are examples of SRB designs. Such designs are characterized by their implicit use of shift registers as multi-bit delays which is a useful method of generating copies of SNs [87]. Since the circuit's state is a shift register, $\mathbf{X}$'s previous bits can always be determined by the current state. For example, the quartic circuit had input $\mathbf{X}$ and state bits $d_{2,t}d_{1,t}d_{0,t}$ with $d_{2,t} = x_{t-1}$, $d_{1,t} = x_{t-2}$ and $d_{0,t} = x_{t-3}$. This property of SRB circuits allows conditional probabilities to be easily added to the circuit's Markov chain diagram, but most sequential circuits, including UCB designs, do not have this useful property.

UCB designs are used to implement functions like stochastic $\tanh$ (Stanh) and stochastic $\exp$. [18, 64]. They are characterized by a "linear" state diagram with a variable number of states, G. For example, Figure 4.13a is the state diagram for an Stanh element with input $\mathbf{X}$ and $G = 4$. As before, the behavior of this sequential circuit can be modeled by translating its state diagram into a Markov chain (Figure 4.13b) and analyzing the steady state. Assuming no autocorrelation in $\mathbf{X}$, the Stanh circuit outputs $\mathbf{Z}$ with $Z \approx \tanh\left(\frac{G}{2}X\right) = \tanh(2X)$. Designs based on up-down counters have long been the focus of research into stochastic sequential circuits. It has often been noted that autocorrelation in the inputs to these designs lowers accuracy [18, 65, 88], but no analytic method for quantifying this error existed before the following approach that we introduced in [10].

Quantifying autocorrelation error in the Stanh circuit is difficult because conditional probabilities cannot be added to its Markov chain in a straightforward manner. For example, state S_2 in Figure 4.13b has an incoming transition from S_1 where $x_{t-1} = 1$ and another incoming transition from S_3 where $x_{t-1} = 0$. Therefore, given that the present state is S_2, one cannot identify if x_{t-1}

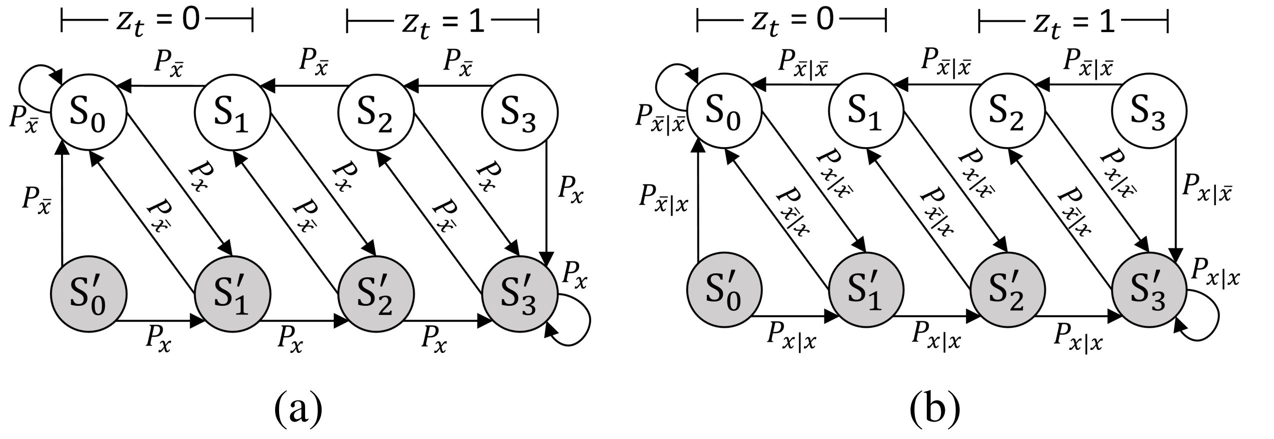
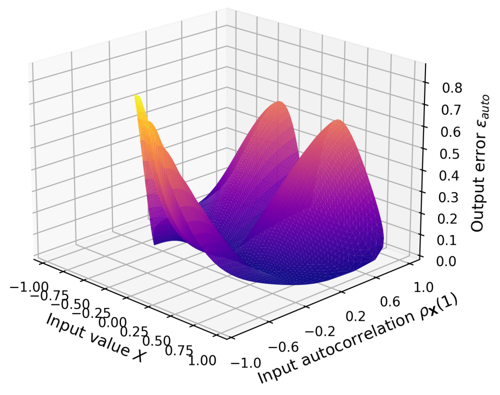


Figure 4.14: Modified Stanh Markov chains: (a) extended Markov chain; (b) extended Markov chain with with conditional probabilities.

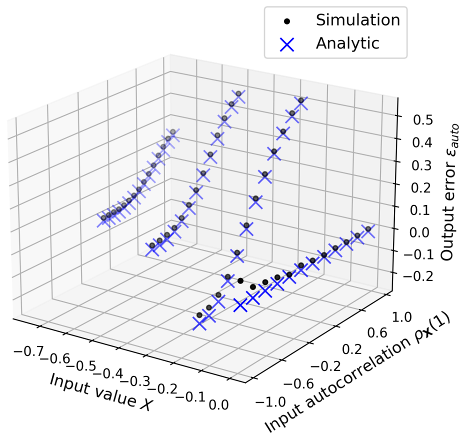
was 0 or was 1 implying that S_2's outgoing transitions cannot be readily replaced by conditional probabilities. A solution to this problem is to extend the Markov chain state space to allow states to encode information about previous input bits. An extended Markov chain of this type is shown in Figure 4.14a where each state S_i from the original Markov chain is split into two copies: an unshaded version S_i that can only be reached when $x_{t-1} = 0$, and a shaded version S'_i that can only be reached when $x_{t-1} = 1$. The copied states output the same bits as the original state, so that the modified design preserves the overall circuit function.

The purpose of the extended Markov chain in Figure 4.14a is to enable states to encode information about previous input bits. Once this is done, conditional probabilities can be used for the state transitions as before when analyzing the quartic circuit. For example, all shaded states S'_i can only be reached when $x_{t-1} = 1$. Therefore the outgoing transition probabilities of these states (P_x and $P_{\bar{x}}$) can be relabeled as $P_{x|x}$ and $P_{\bar{x}|x}$, respectively. In short, Figure 4.14b is a Markov chain for the Stanh circuit that includes conditional probabilities necessary for autocorrelation analysis and this Markov chain can be analyzed using the same steady state analysis that was used to quantify the quartic circuit's autocorrelation error.

The quartic circuit experiments are repeated for an Stanh circuit with bipolar input $\mathbf{X}$, bipolar output $\mathbf{Z}$ and $G = 8$ states. The extended Markov chain for this circuit is derived in a similar manner as Figure 4.14b. In this case, $Z^* = tanh(4X)$ and $Z \approx \pi^*_4 + \pi^*_5 + \pi^*_6 + \pi^*_7$ where π^*_i is the steady-state probability of being in S_i or S'_i. Figure 4.15a summarizes the analysis by plotting the autocorrelation error as a function of input value X and input autocorrelation $\rho_{\mathbf{X}}(1)$. The autocorrelation error is high when $\rho_{\mathbf{X}}$ is very negative or positive which matches observations in other works [18, 65]. Our autocorrelation analysis is compared against simulation and the results



(a)



(b)

Figure 4.15: Stanh autocorrelation error: (a) analysis results; (b) comparison of analysis and simulation results.

appear in Figure 4.15b. As in the quartic circuit experiment, there is strong agreement between the analytic model's predicted error, and the errors measured by simulation. However, some small deviations between analysis and simulation occur due to the influence of the circuit's initial state which is not accounted for in steady state analysis. Only negative values for $\mathbf{X}$ are shown for clarity in Figure 4.15b, but their positive counterparts behave similarly.

Understanding the sensitivity of SC elements to autocorrelated inputs can help inform design choices. For example, consider using the Stanh element in an SC neural network. If the inputs to the Stanh element typically have a certain level of autocorrelation, then a decision can be made whether to include some form of decorrelation. Using the preceding analysis, a designer could determine the amount by which shuffling would improve output accuracy and weigh that against the area and power cost of including the shuffler circuit. Overall, the accuracy, and thus the potential application range of SC, can be significantly increased if autocorrelation-induced errors can be identified, measured and controlled.

4.4 Input Value Distribution

The impact of the input value distribution (IVD) on circuit error is rarely explicitly considered in the SC literature, but it has significant implications for design and for interpreting simulation results as shown in later examples. Modeling the IVD $f_{\boldsymbol{\mathcal{X}}^*}$ requires some knowledge about the target input values $\boldsymbol{\mathcal{X}}^*$ in the form of data or in the form of prior belief. It is important to note that the current objective is to model the distributions of SN target values values like X^*. This

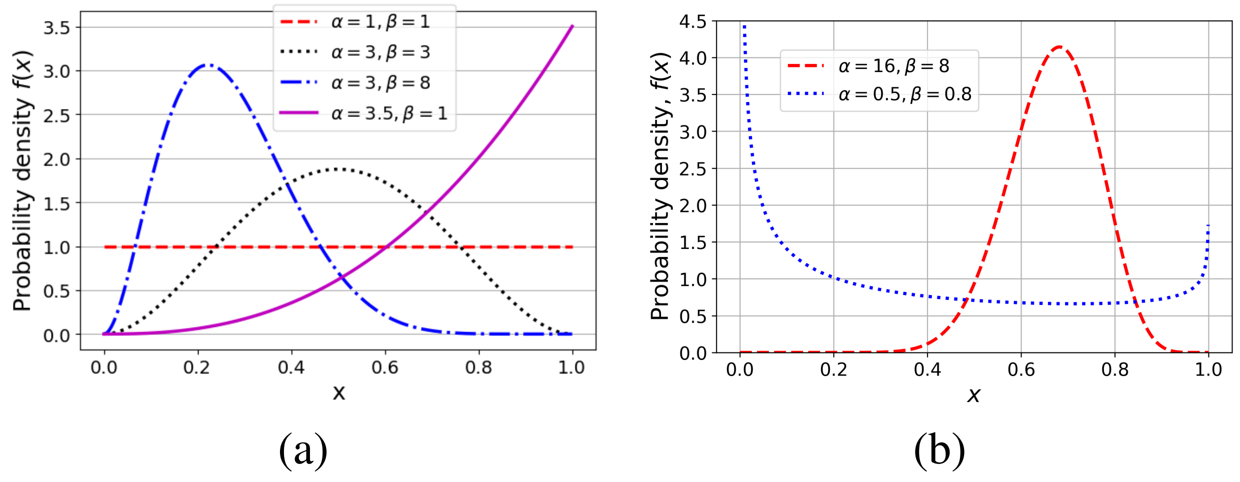


Figure 4.16: (a) Example beta PDFs; (b) beta PDFs from Example 3.3.

distribution is completely determined by the application. The present analysis is not currently concerned with SN estimated values like $\hat{X}$ which is another random quantity distinct from X^* and which was the focus of the earlier discussion of bias and variance.

4.4.1 Beta Distribution

In this section, the beta distribution is proposed a flexible tool for modeling the IVD. The beta distribution is a continuous probability distribution with two shape parameters, $\alpha > 0$ and $\beta > 0$. The PDF of a random variable $A \sim \text{Beta}(\alpha, \beta)$ is

$$f_{\text{beta}}(a|\alpha, \beta) = \frac{a^{\alpha-1}(1-a)^{\beta-1}}{\mathrm{B}(\alpha, \beta)} \tag{4.33}$$

where B is the beta function used to normalize the distribution [38].

$$\mathrm{B}(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}\, dt \tag{4.34}$$

The beta distribution is defined on the $(0, 1)$ interval if $\alpha, \beta < 1$, or on the $[0, 1]$ interval otherwise. It is thus a suitable distribution for modeling probabilities, proportions or, in this case, SN values [11].

Figure 4.16a illustrates the versatility of the beta distribution achieved by varying α and β. When $A \sim \text{Beta}(\alpha, \beta)$ and $\alpha = \beta = 1$, A has uniform density across $[0, 1]$. Thus the uniform distribution is a special case of the beta distribution. When $\alpha, \beta > 1$, A has a unimodal distribution reminiscent of a Gaussian distribution, but where A is bounded to the unit interval. The sharpness

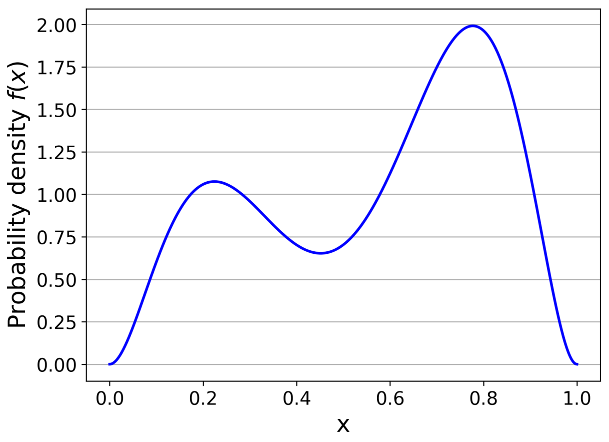


Figure 4.17: Example of a beta mixture model with $\pi_1 = 0.35, \alpha_1 = 3, \beta_2 = 8, \pi_2 = 0.65, \alpha_2 = 8$ and $\beta_2 = 3$.

of the distribution's peak depends on the magnitude of α, β while the location of the peak's center depends on the ratio of α to β. When $\alpha, \beta < 1$, A has bimodal distribution with peaks at 0 and 1, respectively. The PDFs from Example 3.3 are also beta distributions and are shown again in Figure 4.16b. Overall, the beta distribution is bound to the $[0, 1]$ interval and can flexibly represent a variety of shapes which makes it a strong candidate for modeling SN values.

Once it is decided that an SN target value will be modeled as beta random variable, the parameters α and β must be estimated from available data or be based on prior belief. Data driven methods include the method of moments, maximum likelihood estimation and maximum a posteriori estimation [54]. The method moments is less computationally costly than the latter two methods whereas the latter two methods may yield better estimates for α and β. A key distinction between maximum likelihood and maximum a posteriori estimates is that maximum a posteriori estimates can incorporate prior beliefs about α and β into its estimate.

Although the beta distribution is fairly flexible, it cannot represent some PDF types such as a bimodal distribution with peaks at 0.3 and 0.8. Nevertheless, the representational power of the beta distribution can be increased by using a so-called mixture model [82]. For example, an SN target value X^* can be distributed as a mixture of two beta distributions with parameters α_1, β_1 and α_2, β_2. In that case, the PDF of X^* is

$$f_{X^*}(x^*|\alpha_1, \beta_1, \alpha_2, \beta_2, \pi) = (1 - \pi) f_{\text{beta}}(x^*|\alpha_1, \beta_1) + \pi f_{\text{beta}}(x^*|\alpha_2, \beta_2) \tag{4.35}$$

where $0 \leq \pi \leq 1$ determines the relative weighting of the two beta distributions in the mixture model. By tuning the parameters $\alpha_1, \beta_1, \alpha_2, \beta_2$ and π, the PDF of X^* can represent a variety of distribution shapes including bimodal ones; Figure 4.17 shows an example. In general, mixture models can contain an arbitrary number of component distributions. For a k component mixture model, the PDF is

$$f_{X^*}(x^*|\boldsymbol{\theta}) = \sum_{i=1}^{k} \pi_i f_{\text{beta}}(x^*|\alpha_i, \beta_i) \tag{4.36}$$

where π_i is the weight of the i-th component distribution, $\sum_{i=1}^{k} \pi_i = 1$ and $\boldsymbol{\theta}$ is the set containing all the distributions parameters (i.e., the α_i's, β_i's and π_i's). Mixture models with more components have more expressive power, but such models require more computation for parameter estimation and can be prone to over-fitting. Parameter estimation for beta mixture models can be done using the modified expectation maximization algorithm presented in [82].

Thus far, we have proposed modeling a single SN target value X^* as a beta random variable or beta mixture model. When modeling a set of input values $\boldsymbol{\mathcal{X}}^* = [X_1^*, X_2^*, ..., X_M^*]$ it could be assumed that all elements of $\boldsymbol{\mathcal{X}}^*$ are independent implying that the joint distribution is simply the product of the marginal distributions:

$$f_{\boldsymbol{\mathcal{X}}^*}(\boldsymbol{x}^*) = f_{X_1^*}(x_1^*) f_{X_2^*}(x_2^*) ... f_{X_M^*}(x_m^*) \tag{4.37}$$

where $\boldsymbol{x}^* = [x_1^*, x_2^*, ..., x_M^*]$ is a realization of $\boldsymbol{\mathcal{X}}^*$. In this case, each $f_{X_i^*}$ can be modeled separately using the aforementioned (or other) techniques for modeling a single SN value's distribution. In some settings, however, the circuit's input values may be correlated[4] and Equation (4.37) cannot be used. For example, values of adjacent pixels in an image tend to be highly correlated. In situations like these, modeling the input distribution becomes a domain-specific task.

The beta distribution is just one useful way to model SN values. An advantage of this modeling approach is that beta distributions and beta mixture models can flexibly represent a variety of distribution shapes and there are relatively few parameters that must be estimated from data. Other approaches such as using a piece-wise polynomial could also be used. We demonstrate the usefulness of the beta distribution in an neural network application in the following section.

4.5 Case Study: Neural Networks

The foregoing error framework and modeling methodology is now applied to an SC-based convolutional neural network (CNN) trained for image classification. Artificial neural networks have long been seen as a promising application for SC because of the huge number of multiplications they require to implement inner-product operations of the form $\sum_{i=1}^{M} W_i X_i$, where the W_i

[4]It is important to note that correlated SN values and correlated SN bits are distinct concepts. Two SNs $\mathbf{X}$ and $\mathbf{Y}$ can have correlated values X, Y but have independent bits x_t, y_t. Likewise, $\mathbf{X}$ and $\mathbf{Y}$ can have correlated bits but independent values. Although the concepts of correlated SN values and correlated SN bits sound superficially related, they are essentially separate ideas.

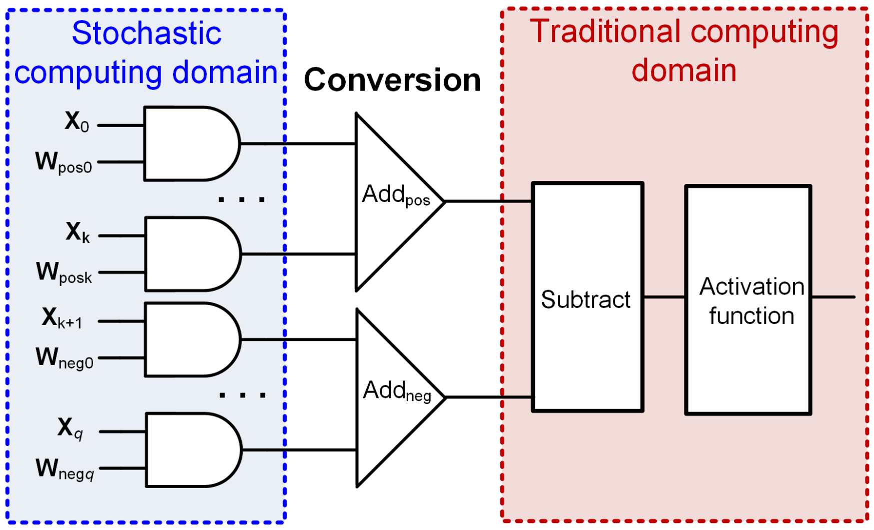


Figure 4.18: Convolution layer of a CNN based on the design in [29].

are the constant trained network weights and the X_i are variable network inputs. This case study investigates stochastic circuit accuracy when used to implement inference on a trained CNN.

A basic part of a high-performing SC-based CNN design is shown in Figure 4.18 [29]. The convolution layer is largely implemented in the SC domain where LFSRs (not shown in Figure 4.18) are used to generate the SNs. In this SC CNN design style, positive and negative weights are separated into two groups and multiplication is performed using the unipolar format.[5] As illustrated in Figure 4.18, the products with positive weights and the products with negative weights are summed separately using APCs. Then the difference is taken between the positive weight group's sum and negative weight group's sum to compute the overall inner product between weights and inputs. Since multiplication is done in the SC domain, the goal of this analysis is to characterize the multiplication accuracy of the CNN using simulation and our BASE framework.

For this case study, two CNNs with structures similar to the classic LeNet-5 [58] are trained. One network is trained for handwritten digit recognition using the grayscale MNIST handwritten digit dataset [58] while the other is trained to classify animals or objects using the colored CIFAR-10 natural image dataset [57]. Both MNIST and CIFAR-10 are popular 10-class benchmarks used in machine learning, and Figure 4.19 gives example images from both datasets.

The two objectives of this case study are 1) investigate how the input value distribution (IVD) affects SC accuracy and 2) accurately model SC error using our BASE framework. In both CNNs, the first convolutional layer's multiplications are implemented with SC and the accuracy of this operation will be the focus of analysis. Since one objective is for our analysis to reproduce the results of simulation, the first step is to establish the simulation-based results. The two trained

[5]The absolute value of negative weights are used for SN generation purposes

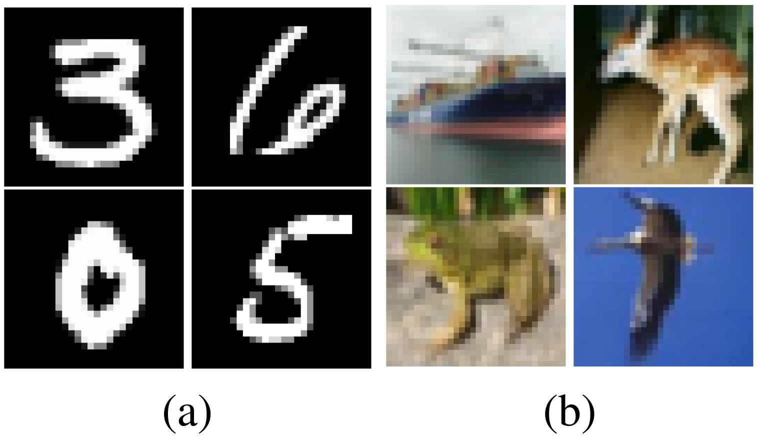


Figure 4.19: Example images from (a) MNIST dataset [58]; (b) CIFAR-10 dataset [57].

networks are simulated with the corresponding MNIST or CIFAR-10 test dataset as input. Since 8-bit precision is commonly used for image processing, both networks use 8-bit LFSRs for SN generation and the SN length is set to 256-bits.

The multiplication BMSE derived from simulation is 1.18×10^{-5} for the MNIST CNN and 5.61×10^{-5} for the CIFAR-10 network. Despite both CNNs being very similar, the BMSE is nearly five times higher for multiplications in the CIFAR-10 CNN. The reason for this large error difference can be speculated on these simulation results alone, but a principled analysis using our BASE framework provides a structured approach that arrives at a clear answer.

Circuit bias, variance, and input value distribution must be modeled to predict BMSE analytically using BASE. In this experiment, there is very little quantization error because the SNG precision is 8-bits and the image pixels from these benchmarks are also represented in 8-bit precision. There is also no correlation error since separate LFSRs are used to generate the input SNs and weight SNs and there is no approximation error since AND gate implements multiplication. Thus, the overall bias is relatively small compared to variance which will be the focus of analysis.

$$\text{Bias}(Z^*, \hat{Z}|X^*, W^*) \approx 0 \tag{4.38}$$

Since bias is approximated as zero, the input target values X^* and W^* will simply be written as X and W, i.e., without the usual star notation. Except where stated, notation will return to its standard after this section concludes.

Next, variance is modeled. The variance of an AND multiplier with L-bit LFSR SN inputs $\mathbf{X}$ and $\mathbf{W}$ and output $\mathbf{Z}$ was given in Example 4.1 Part 2.

$$\text{Var}(\hat{Z}|X, W) = \frac{X(1-X)W(1-W)}{L-1} \tag{4.39}$$

The BMSE can now be expressed in terms of bias and variance.

$$\mathrm{BMSE}(Z^*, \hat{Z}) = \mathbb{E}[\mathrm{Bias}^2(Z^*, \hat{Z}|X, W)] + \mathbb{E}[\mathrm{Var}(\hat{Z}|X, W)] \tag{4.40}$$

$$\mathrm{BMSE}(Z^*, \hat{Z}) = 0^2 + \int_0^1 \int_0^1 f_{X,W}(x, w)\,\mathrm{Var}(\hat{Z}|X = x, W = w)\,\mathrm{d}x\,\mathrm{d}w \tag{4.41}$$

The weights and pixel values are modeled as independent meaning that $f_{X,W}(x, w) = f_X(x)f_W(w)$. This simplification along with Equation (4.39) can be used to re-express Equation (4.41) as

$$\mathrm{BMSE}(Z^*, \hat{Z}) = \frac{1}{L-1} \int_0^1 f_X(x)x(1-x)\,\mathrm{d}x \int_0^1 f_W(w)w(1-w)\,\mathrm{d}w \tag{4.42}$$

In this case, BMSE is the product of two integrals of the same form. We now turn our attention to the x integral since the w integral will be handled in the same manner.

The distributions of pixel values f_X and network weights f_W are modeled as mixtures of K_X and K_W beta distributions, respectively. The x integral can be re-expressed using the beta PDF (Equation (4.33)) along with the beta mixture model PDF (Equation (4.36)

$$\int_0^1 f_X(x)x(1-x)\,\mathrm{d}x = \sum_{k=1}^{K_X} \pi_k^{[X]} \int_0^1 \frac{x^{\alpha_k^{[X]}}(1-x)^{\beta_k^{[X]}}}{\mathrm{B}\left(\alpha_k^{[X]}, \beta_k^{[X]}\right)} \tag{4.43}$$

where $\pi_k^{[X]}, \alpha_k^{[X]}, \beta_k^{[X]}$ are the coefficient and parameters for the k-th component of X's beta mixture model. Equation (4.43) can be further simplified by evaluating the definite integral.

$$\int_0^1 f_X(x)x(1-x)\,\mathrm{d}x = \sum_{k=1}^{K_X} \pi_k^{[X]} \frac{\alpha_k^{[X]}\beta_k^{[X]}}{\alpha_k^{[X]} + \beta_k^{[X]} + 1} \tag{4.44}$$

Equation (4.44) and its counterpart that uses w in place of x can then be substituted into the BMSE equation (4.42) to yield

$$\mathrm{BMSE}(Z^*, \hat{Z}) = \frac{1}{L-1} \sum_{j=1}^{K_X} \sum_{k=1}^{K_W} \pi_j^{[X]}\pi_k^{[W]} \frac{\alpha_j^{[X]}\beta_j^{[X]}\alpha_k^{[W]}\beta_k^{[W]}}{(\alpha_j^{[X]} + \beta_j^{[X]} + 1)(\alpha_k^{[W]} + \beta_k^{[W]} + 1)} \tag{4.45}$$

where $\pi_k^{[W]}, \alpha_k^{[W]}, \beta_k^{[W]}$ are the coefficient and parameters for the k-th component of W's beta mixture model.

To summarize, the objective of this analysis is to derive the BMSE of the neural network's SC multipliers. The first step of analysis found that multiplier bias was zero and the variance of the multiplier was Equation (4.39). Next, it was decided that the IVD would be modeled using

Distribution	π_1	α_1	β_1	π_2	α_2	β_2
MNIST pixels	1	0.0362	0.1817	-	-	-
CIFAR-10 pixels	0.6653	2.2919	3.2944	0.3347	0.9236	0.8315
MNIST weights	1	1.2800	7.5486	-	-	-
CIFAR-10 weights	1	1.6704	17.782	-	-	-

Table 4.5: Estimated Beta Distribution Parameters

two independent beta mixture models – one mixture model for the input pixels and one mixture model for the network weights. The BMSE was then expressed as Equation (4.42) and eventually simplified to Equation (4.45). The remaining steps of the analysis is to estimate the beta mixture model parameters, i.e., the $\alpha_k^{[X]}$'s, $\beta_k^{[W]}$'s, $\pi_k^{[X]}$'s, etc. and compute a numeric value for BMSE.

The beta mixture model parameters will be estimated using available data. There are four sets of beta mixture model parameters that must be estimated in this way – one parameter set for the MNIST pixel values, one for the CIFAR-10 pixel values, one for the MNIST CNN weights, and one for the CIFAR-10 CNN weights. Although our objective is to compute the BMSE of the MNIST and CIFAR-10 test datasets, the parameters for the pixel value distributions will be estimated only using the MNIST and CIFAR-10 training data. This decision makes our analysis more representative of a real-world setting and will demonstrate that this analysis generalizes to unseen data. The parameters for the CNN weight distributions will be estimated with the weights from the respective trained networks since these are known during inference.

The parameters for each beta mixture model are estimated using the aforementioned data and the algorithm in [82]. Beta mixture models of up to three components were considered and the best-fitting model was chosen. Overall, the CIFAR-10 pixel distribution was fitted with a two-component mixture model whereas all other distributions were fitted with a single component mixture model (i.e., as a plain beta distribution). The estimated parameters are reported in Table 4.5, while the distributions and best fits are shown in Figure 4.20. The beta mixture model fits are representative of the target distribution in all cases.

The final step of the BMSE analysis can now be completed. The BMSE for both the MNIST and CIFAR-10 test datasets are computed using the fitted model parameters found in Table 4.5 with Equation (4.45). Both the simulation-derived BMSE mentioned earlier and analysis-derived BMSE are reported in Table 4.6. There is close agreement between both methods of estimating BMSE suggesting that the analysis and simulation are correct. Results obtained during the statistical analysis can now be leveraged to better understand circuit performance.

Many insights not apparent from simulation alone can be gained from the BMSE analysis. For example, the multiplier variance expression (4.39) implies that error is highest with $X = W = 0.5$

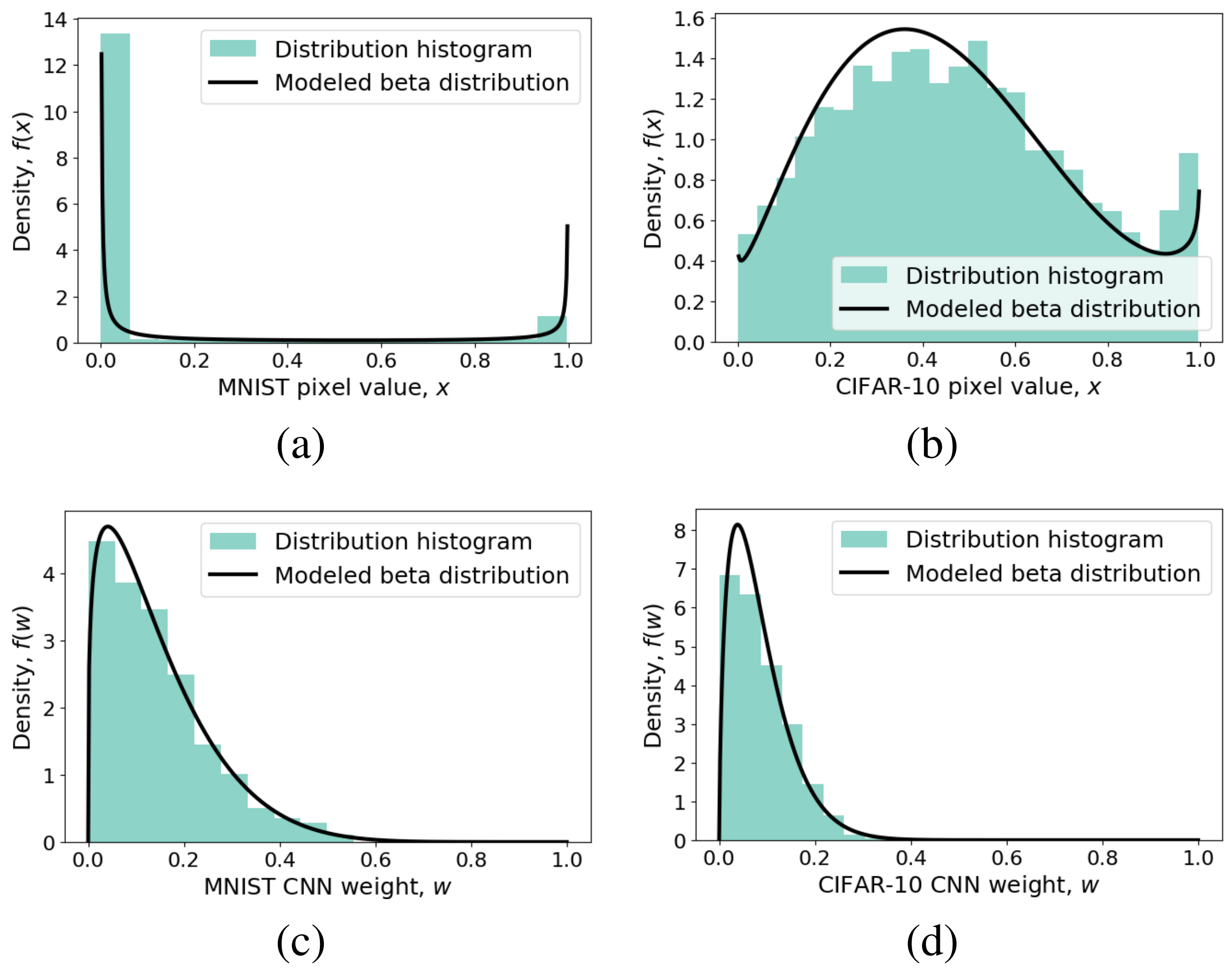


Figure 4.20: Representative IVDs and their beta distribution models: (a) MNIST pixel values; (b) CIFAR-10 pixel values; (c) MNIST CNN weights (absolute values); (d) CIFAR-10 CNN weights (absolute values).

and lowest when $X = 0, 1$ or $W = 0, 1$. As Figure 4.20 shows, pixel values in the MNIST dataset almost always take value 0 or 1 while the MNIST networks weights are concentrated near 0. The high probability density for X and W values that correspond to low variance implies implying the multiplication will be very accurate in the MNIST case. In the CIFAR-10 case, however, pixel values vary across the whole interval [0,1] while the weights are concentrated towards 0, implying that multiplication will be fairly accurate, but not as accurate as the MNIST case. Put simply, the favorable IVDs for pixels and weights in the MNIST case explains why multiplication BMSE is significantly lower for the MNIST case study than for the CIFAR-10 case study.

In general, CIFAR-10 poses a more difficult classification task than MNIST, but in the SC setting there is an added wrinkle of higher multiplication error in the CIFAR-10 case. In other words, the extremely favorable IVD for the MNIST task implies that SC CNN results on MNIST

Network	Simulation-derived BMSE	Analysis-derived BMSE
MNIST	1.18×10^{-5}	1.083×10^{-5}
CIFAR-10	5.61×10^{-5}	5.55×10^{-5}

Table 4.6: BMSE for Multiplications in the MNIST and CIFAR-10 CNNs

may poorly generalize to other classification tasks like CIFAR-10. For instance, in [29], accurate MNIST classification was achieved with an impressive SN length of just 8 bits. The favorable MNIST IVD could be one reason that high performance was possible with such short bitstreams as SC CNNs sometimes require SN lengths of hundreds or even thousands of bits. If a designer wishes to extend these results to other datasets with less favorable IVDs (like CIFAR-10), then knowing multiplication error will be higher informs the design process and may suggest that longer bitstreams will be needed.

In sum, the IVDs should be carefully considered when assessing the accuracy of a circuit design. These IVDs should be representative of data in the application domain of the circuit, or, in the case of general designs, should represent a variety of different distributions. The beta distribution can be useful for constructing or modeling input distributions with a variety of shapes. Alternatively, multiple datasets can used to assess a circuit's accuracy where each dataset represents an IVD as in this case study. When comparing circuits, if circuit dominance cannot be proved, then it is not sufficient to simply simulate the designs with uniformly random input values and declare design as superior to another. One should test multiple IVDs or sample from the target applications IVD to gain a comprehensive comparison of circuit accuracy.

4.6 Summary

Modeling a circuit's bias, variance, and input value distribution (IVD) leads to better understanding of its accuracy. In this chapter, we proposed the hypergeometric SN model which can better predict the variance of circuits that employ non-repeating RNSs like LFSRs. Insights from the hypergeometric SN model led to the CAM subtractor design which is much more accurate on average than a standard subtractor. In the following chapter, these same insights will form the basis of our CeMux adder which is more accurate and smaller than standard mux adders.

In addition to the hypergeometric SN model, we also proposed a method for modeling autocorrelation error in sequential stochastic circuits. This method improves the understanding of sequential stochastic circuits and can inform trade-off decisions involving decorrelation methods. We also proposed using the beta distribution to flexibly model the IVD. The chapter culminated in a neural network case studied which combined many of our results and demonstrated the usefulness of these statistical models and the BASE framework. Overall, the BASE framework combined with our statistical models gives insights into the circuit design process. In the next chapter, we further develop these insights and present new advances in stochastic computing adder design.

CHAPTER 5

Large-Scale Stochastic Adder Design

Stochastic computing's most effective applications include image processing and digital filtering because they are error tolerant to some degree and have high computational demand. In these applications, the bulk of the computing cost often comes from the vast number of many-input inner products that underlie important algorithms like convolution and artificial neural networks. Multiplication is relatively accurate in SC, but highly accurate addition is much more challenging, particularly when there are hundreds or thousands of summands. For instance, when used to add many SNs, basic mux adders have low area but poor accuracy while another common SN adder design, the accumulative parallel counter (APC), has good accuracy but high area. This chapter addresses the challenge of large-scale SN adder design by improving existing adders and introducing new adder types. The work in this chapter is mostly published in [13, 14].

5.1 Mux Adders

Mux adders are the traditional SN adder design type. Figure 5.1 shows a 2-input mux adder with data inputs $\mathbf{A}$, $\mathbf{B}$, select input $\mathbf{S}$, and output $\mathbf{Z}$. The select SN has $P_s = 0.5$ and the output value is

$$Z = \frac{1}{2}(A + B) \tag{5.1}$$

One way to understand Equation (5.1) is to envision the mux as a sampling unit, where each clock cycle the control input $\mathbf{S}$ determines whether $\mathbf{A}$ or $\mathbf{B}$ is sampled and has its bit propagated to the output. Since $P_s = 0.5$, $\mathbf{A}$ and $\mathbf{B}$ have an equal chance of being sampled each clock cycle implying that, on average, half of $\mathbf{Z}$'s bits will be from $\mathbf{A}$ and half from $\mathbf{B}$. Thus, $\mathbf{Z}$'s value is given by Equation (5.1). The viewpoint of mux adders as sampling units is illustrated in Figure 5.1 and will serve an important role in understanding how to improve the accuracy of mux adders.

The two-input mux that performs two-input scaled addition can be generalized to an M-input

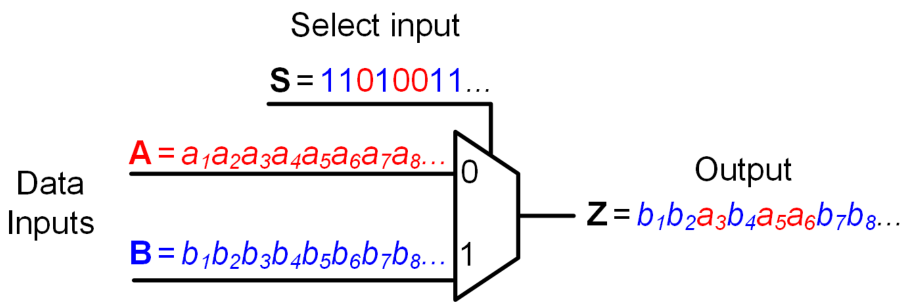


Figure 5.1: Multiplexer (mux) sampling adder.

mux adder that performs weighted addition of the form

$$Z = \frac{1}{\sum_{i=1}^{M} |W_i|} \sum_{i=1}^{M} W_i X_i \tag{5.2}$$

where $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_M$ are bipolar input SNs with weights $W_1, W_2, ..., W_M$, respectively. Conventional bipolar mux adders that compute Equation (5.2) do not use bipolar SNs to encode weight values. Instead, they use a two stage implementation: an XOR gate array followed by a mux adder tree as shown in Figure 5.2 [51, 91, 94, 97]. The XORs multiply each bipolar input $\mathbf{X}_i$ with the sign of its corresponding weight W_i. Then, each XNOR output $\mathbf{Y}_i$ with $Y_i = \text{sign}(W_i)X_i$ is routed into a mux tree that computes $Z = \frac{1}{\sum_i^M |W_i|} \sum_{i=1}^{M} |W_i|Y_i$ which is equivalent to Equation (5.2). In short, the XOR array accounts for the sign of the adder weights while the mux tree accounts for

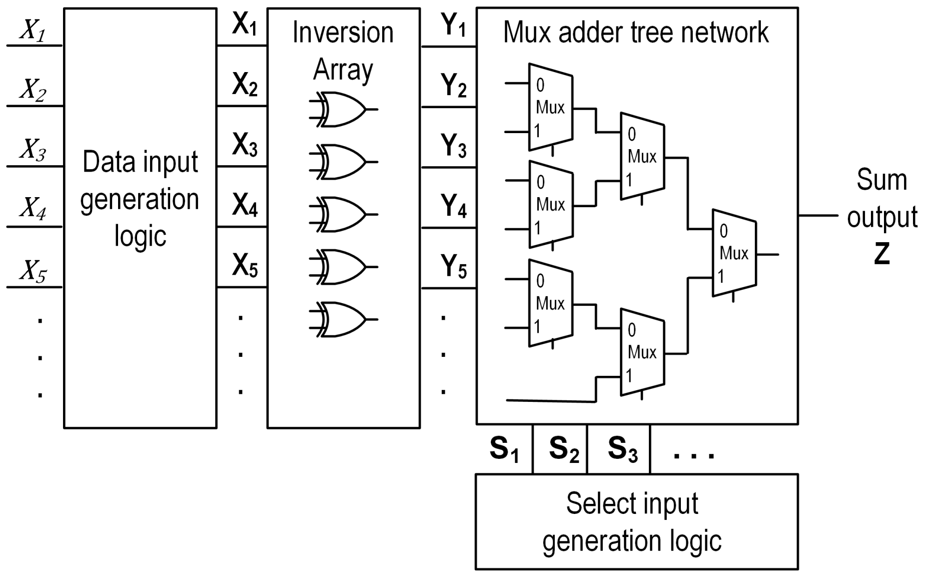


Figure 5.2: Generic M-way mux weighted adder design.

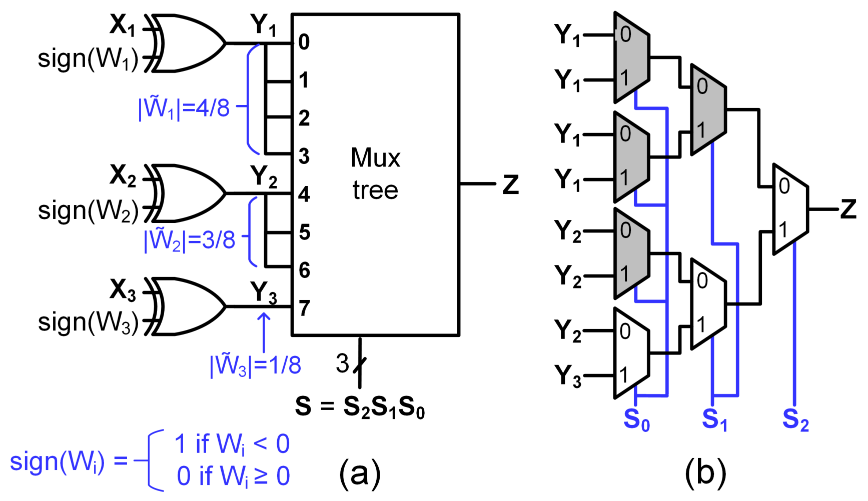


Figure 5.3: Hardwired mux tree example: (a) full design; (b) internal mux tree structure where the shaded two-way muxes are redundant and can be removed.

the magnitude of the adder weights.

There are various mux tree designs that, together with an XOR array, implement Equation (5.2). The trees mainly differ in how they use the mux select inputs to encode the normalized magnitude of each weight $|\widetilde{W_i}| = \frac{|W_i|}{\sum_{i=1}^{M}|W_i|}$. One basic design is the *hardwired mux tree* [91] which is best explained with an example. The hardwired mux tree in Figure 5.3 computes

$$Z = \frac{1}{2}Y_1 + \frac{3}{8}Y_2 + \frac{1}{8}Y_3 \tag{5.3}$$

The hardwired mux's select inputs $\mathbf{S}_2$, $\mathbf{S}_1$, and $\mathbf{S}_0$ all have value 0.5 and are shared amongst muxes on the same level of a full mux tree as shown in Figure 5.3. With this configuration, all 8 mux tree inputs have probability $1/8$ of being sampled each clock cycle and the normalized weight magnitudes can be implemented by hardwiring each $\mathbf{Y}_i$ to one or more input mux slots. For example, $\mathbf{Y}_2$ is hardwired to three of the eight mux tree inputs because $\widetilde{W_2} = 3/8$. Likewise, $\mathbf{Y}_1$ is hardwired to half the mux tree inputs since $\widetilde{W_1} = 1/2$ and $\mathbf{Y}_3$ is hardwired to just one input because $\widetilde{W_3} = 1/8$. In general, the number of levels or height of the mux tree determines the values to which the normalized weights must be quantized, and Algorithm 5.1 describes the quantization procedure. In Figure 5.3, the height is 3 and all $|\widetilde{W_i}|$ are quantized to 3-bit precision. The hardwired mux tree is most useful in resource-limited applications where the weights are not expected to be updated, such as in hearing aid filters [16] or electrocardiogram filtering [9].

In cases where weights are expected to be updated, the *biased selector mux tree* introduced in [21] can be used to implement weighted addition. In this case, the weights are not hardwired, but rather are encoded into the select input SNs' values which are no longer all set to 0.5. When

Algorithm 5.1 Hardwired Mux Tree Weight Normalization and Quantization

```
Input: $\mathbf{w} = \left[W_1^*, W_2^*, ..., W_M^*\right]$          ▷ Target adder weights
Input: $h$          ▷ Desired height of hardwired mux tree
Output: $\left[|\widetilde{W}_1|, |\widetilde{W}_2|, ..., |\widetilde{W}_M|\right]$          ▷ Absolute values of the quantized, normalized weights.
 1: $\mathbf{a} \leftarrow \text{abs}(\mathbf{w})$          ▷ Take the element-wise absolute value of $\mathbf{w}$.
 2: $\mathbf{t} \leftarrow 2^h\mathbf{a}/\,\text{sum}(\mathbf{a})$          ▷ Compute the numerators of the normalized weights.
 3: $\mathbf{q} \leftarrow \text{round}(\mathbf{t})$          ▷ Quantize $\mathbf{t}$ to nearest integer.
 4:
 5: ▷ Sometimes after rounding, the sum of the quantized normalized weights is not equal to 1.
 6: ▷ and slight adjustments are needed:
 7: while $\text{sum}(\mathbf{q}) > 2^h$ do
 8:     $i \leftarrow \text{argmax}(\mathbf{q} - \mathbf{t})$
 9:     $q_i \leftarrow q_i - 1$          ▷ Decrement the numerator that results in smallest bias.
10: end while
11:
12: while $\text{sum}(\mathbf{q}) < 2^h$ do
13:     $i \leftarrow \text{argmax}(\mathbf{t} - \mathbf{q})$
14:     $q_i \leftarrow q_i + 1$          ▷ Increment the numerator that results in smallest bias.
15: end while
16:
17: return $\mathbf{q}/2^h$
```

a change in summand weights is needed, the select input values can be updated. The weight flexibility comes at a high area cost, however, since many additional SNGs are needed for the select input SNs. Recent work aims at reducing this SNG overhead [51, 97]. Section 3 in [51] gives a detailed explanation of the biased selector mux tree design methodology. Note that the terminology "biased selector tree" is not used in [51], but was introduced in our CeMux work [13] to help differentiate mux tree designs.

Mux adders are often criticized for their high error when used to add many SNs [66, 71]. Inaccuracy must be addressed by using very long bitstreams or by using a costlier adder design like an APC. Mux inaccuracy stems from its sampling behavior. Every clock cycle, an M-input mux samples one input SN and ignores the bits from the other $M - 1$ inputs. When M is large, a significant amount of information is lost which leads to inaccuracy. This explanation for mux accuracy inspires two techniques that substantially improve mux accuracy: precise sampling and full correlation. These techniques reduce the amount of information lost during sampling by employing correlation in useful ways. A new formula for mux variance that we introduced in [13] and repeated below explains how these techniques affect mux accuracy.

$$\text{Var}(Z) = \epsilon_{\text{noise}} + \epsilon_{\text{samp}} + \epsilon_{\text{corr}} \tag{5.4}$$

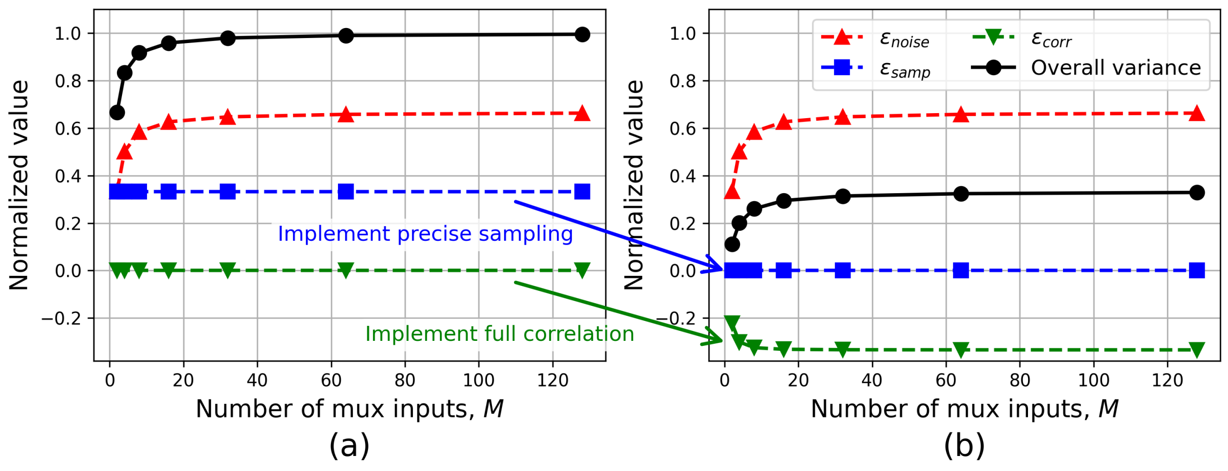


Figure 5.4: Bipolar mux adder variance: (a) conventional mux adder; (b) mux adder with precise sampling and full correlation. Values are normalized by multiplying by 512, the bitstream length.

Formal definitions and derivations for ϵ_{noise}, ϵ_{samp} and ϵ_{corr} are given in Appendix A. Here, the focus is a high-level explanation of Equation (5.4). Note that Equation (5.4) flexibly applies to both Bernoulli and hypergeometric SN models and applies to both unipolar and bipolar SC.

Like most variance expressions, ϵ_{noise}, ϵ_{samp} and ϵ_{corr} all depend on the SN length L, the input values, and the summation weights. Decomposing mux variance into these three components highlights how mux adder accuracy can be improved. For instance, Figure 5.4 illustrates how these three variance components and overall mux variance vary with the number of bipolar mux adder inputs. Figure 5.4a corresponds to a conventional mux adder and shows that the overall variance quickly saturates to a high value of $1/L$ as the number of inputs increases [71]. Figure 5.4b corresponds to a mux adder with our proposed precise sampling and full correlation methods. Comparing Figures 5.4a and 5.4b reveals that precise sampling reduces ϵ_{samp} from about $1/(3L)$ to zero while full correlation pushes ϵ_{corr} from zero to about $-1/(3L)$. These reductions are additive and lead to a significant overall variance reduction of 67%. This accuracy improvement is further amplified in our CeMux design where a low-discrepancy RNS type is used in place of an LFSR RNS.

5.1.1 Precise Sampling and Full Correlation

A mux adder's inaccuracy largely stems from its sampling behavior. Every clock cycle, an M-input mux adder samples a bit from one of its input SNs and ignores the bits from the other $M-1$ inputs. Due to the amount of unused information, the mux adder output has high variability as quantified by variance. Two new techniques, precise sampling and full correlation, can significantly decrease this variability and thus greatly improve mux accuracy. Both techniques address one component of mux adder variance: precise sampling reduces ϵ_{samp} to zero while full correlation reduces ϵ_{corr} from zero to a negative value.

Mux addition relies on sampling the input SNs and ϵ_{samp} quantifies the variation in the number of times each input is sampled. For instance, consider the previously discussed hardwired mux tree of Figure 5.3 with data inputs $\mathbf{Y}_1$, $\mathbf{Y}_2$, $\mathbf{Y}_3$, adder weights $\widetilde{W_1} = 1/2$, $\widetilde{W_2} = 3/8$, $\widetilde{W_3} = 1/8$, and SN length $L = 16$. Since $|\widetilde{W_1}| = 1/2$, $\mathbf{Y}_1$ is expected to be sampled eight times ($|\widetilde{W_1}|L = 8$), Likewise, $\mathbf{Y}_2$ is expected to be sample six times since $|\widetilde{W_2}|L = 6$ and $\mathbf{Y}_3$ is expected to be sampled two times since $|\widetilde{W_3}|L = 2$. However, $\mathbf{Y}_1$, $\mathbf{Y}_2$ and $\mathbf{Y}_3$ may each be sampled anywhere from zero to sixteen times due to random fluctuations in the mux select inputs $\mathbf{S}_2$, $\mathbf{S}_1$ and $\mathbf{S}_0$. When inputs are not sampled their expected number of times, the mux output will be biased thus causing error that is characterized by ϵ_{samp}.

Let C_i be a random variable denoting the number of times bipolar mux input $\mathbf{Y}_i$ is sampled. Then

$$\epsilon_{\text{samp}} = \frac{1}{L^2} \sum_{i=1}^{M} \sum_{j=1}^{M} Y_i Y_j \operatorname{Cov}(C_i, C_j) \tag{5.5}$$

where L is the SN length, M is the number of mux inputs, $\operatorname{Cov}(C_i, C_j)$ is the covariance between C_i and C_j, and $\operatorname{Cov}(C_i, C_i) = \operatorname{Var}(C_i)$ by the definition. For the foregoing example of Figure 5.3, $L = 16$, $M = 3$ and the covariance terms depend on how the select SNs are generated. The covariance terms in Equation (5.5) highlight the fact that ϵ_{samp} (and thus part of the mux's variance) is dependent on the variation in the number of times each input is chosen. Reducing this variation motivates the concept of precise sampling.

Let L-bit SNs $\mathbf{Y}_1, \mathbf{Y}_2, ..., \mathbf{Y}_M$ with normalized weights $|\widetilde{W_i}|$ be input to a mux tree and let C_i be a random variable representing the number of times $\mathbf{Y}_i$ is sampled by the tree. The expected number of times $\mathbf{Y}_i$ is sampled is $\mathbb{E}[C_i] = |\widetilde{W_i}|L$. A mux tree performs precise sampling when $\mathbb{P}(|C_i - \mathbb{E}[C_i]| < 1) = 1$ for all $1 \leq i \leq M$. In simpler terms, precise sampling is when each input $\mathbf{Y}_i$ is always (i.e., with probability 1) sampled its expected number of times, up to a rounding error (i.e., $|C_i - \mathbb{E}[C_i]| < 1$). In the case where $\mathbb{E}[C_i]$ is an integer for all i, implementing precise sampling guarantees that $C_i = \mathbb{E}[C_i]$ for all i implying that $\operatorname{Var}(C_i) = \operatorname{Cov}(C_i, C_j) = 0$ and, as a result, $\epsilon_{\text{samp}} = 0$. A mux tree that does not perform precise sampling is said to implement *noisy sampling*.

Conventional hardwired mux trees such as HWA [91] perform noisy sampling because separate and independent n-bit RNSs drive the mux select lines. Fluctuations between these RNSs cause sampling variation as seen in the noisy sampling behavior of Figure 5.5a. Instead, we propose to generate the select inputs of a height h hardwired mux tree with a single RNS as shown in Figure 5.5b. This new construction saves considerable area compared to the former construction because

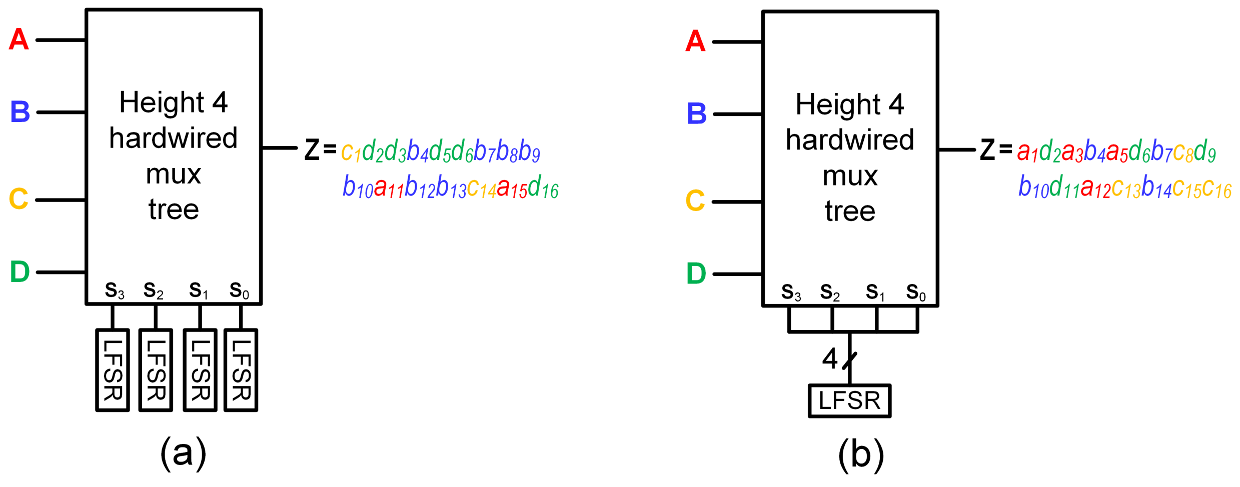

A
B
C
D
Height 4 hardwired mux tree
S_3 S_2 S_1 S_0
LFSR
LFSR
LFSR
LFSR
$Z = c_1 d_2 d_3 b_4 d_5 d_6 b_7 b_8 b_9 b_{10} a_{11} b_{12} b_{13} c_{14} a_{15} d_{16}$
(a)
A
B
C
D
Height 4 hardwired mux tree
S_3 S_2 S_1 S_0
4
LFSR
$Z = a_1 d_2 a_3 b_4 a_5 d_6 b_7 c_8 d_9 b_{10} d_{11} a_{12} c_{13} b_{14} c_{15} c_{16}$
(b)

Figure 5.5: Sampling inputs from a hardwired mux tree: (a) noisy sampling; (b) precise sampling.

a single RNS is used in place of h RNSs. Moreover, it allows a straightforward implementation of precise sampling.

An n-bit RNS is used to implement precise sampling of SNs with length 2^n. The RNS is constrained to sample numbers from $[0, 2^n - 1]$ without replacement. For example, an LFSR could be used. Then, for a height h mux tree, h out of the n RNS bits are used to drive the mux select lines. This restriction ensures that the mux adder performs precise sampling because each mux input slot is sampled exactly 2^{n-h} times as in Figure 5.5b where $n = h = 4$. Suitable choices for the RNS include an n-bit LFSR as in Figure 5.5b or a counter as in our CeMux adder design.

Full correlation is another technique that can reduce mux variance. Input correlation arises when the input SNs share an RNS. The sharing of a single RNS amongst the M data inputs of a mux adder is common practice in SC because it saves considerable area and because correlation among the data inputs of mux adders was believed to have no effect on output error [3, 51]. In Section 4.2, our hypergeometric SN model [12] showed that this belief is incorrect by demonstrating that correlation has a substantial effect on the variance of a 2-input mux adder. Full correlation generalizes these ideas and analysis to M-input mux adders.

To illustrate correlation's effect on a larger mux adder, consider the mux tree in Figure 5.6a with four uncorrelated inputs $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ each with unipolar value 0.5, length 8 and weight 0.25. During each clock cycle i, some of the input bits are 0 and others 1. By happenstance, it is possible for the mux to propagate a 0 every single clock cycle resulting in $\mathbf{Z} = 00000000$. In this somewhat unlikely, yet possible, case, $\mathbf{Z}$'s estimated value $\hat{Z}$ is 0 which poorly represents Z's expected value $Z = \mathbb{E}[\hat{Z}] = 0.5$ and target value $Z^* = 0.5$.

In contrast, Figure 5.6b shows the same mux adder configuration, but when $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ are maximally correlated by sharing an RNS. Since $\mathbf{A}$ to $\mathbf{D}$ have the same value of 0.5, they contain

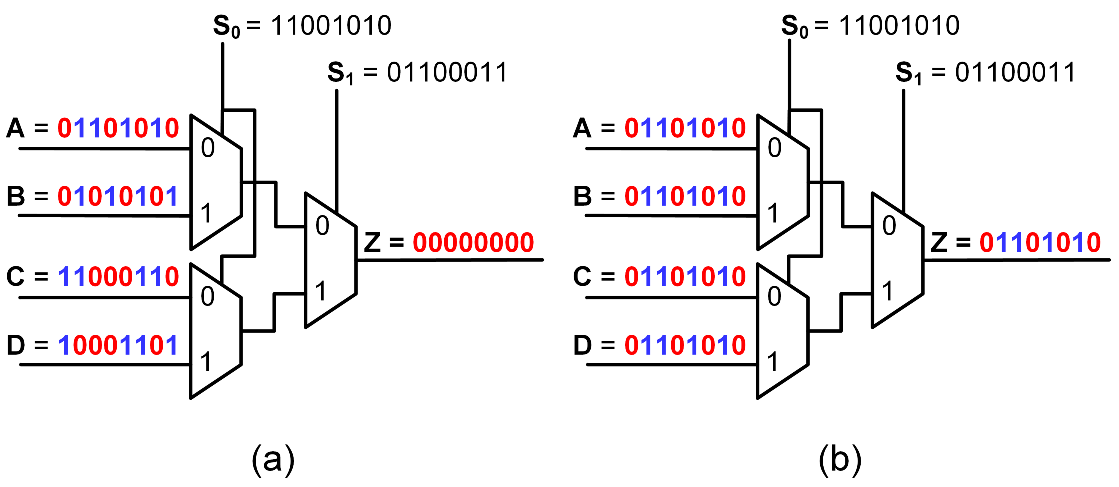


Figure 5.6: Effect of input correlation on mux behavior. Possible behavior of mux tree with: (a) uncorrelated inputs; (b) maximally correlated inputs.

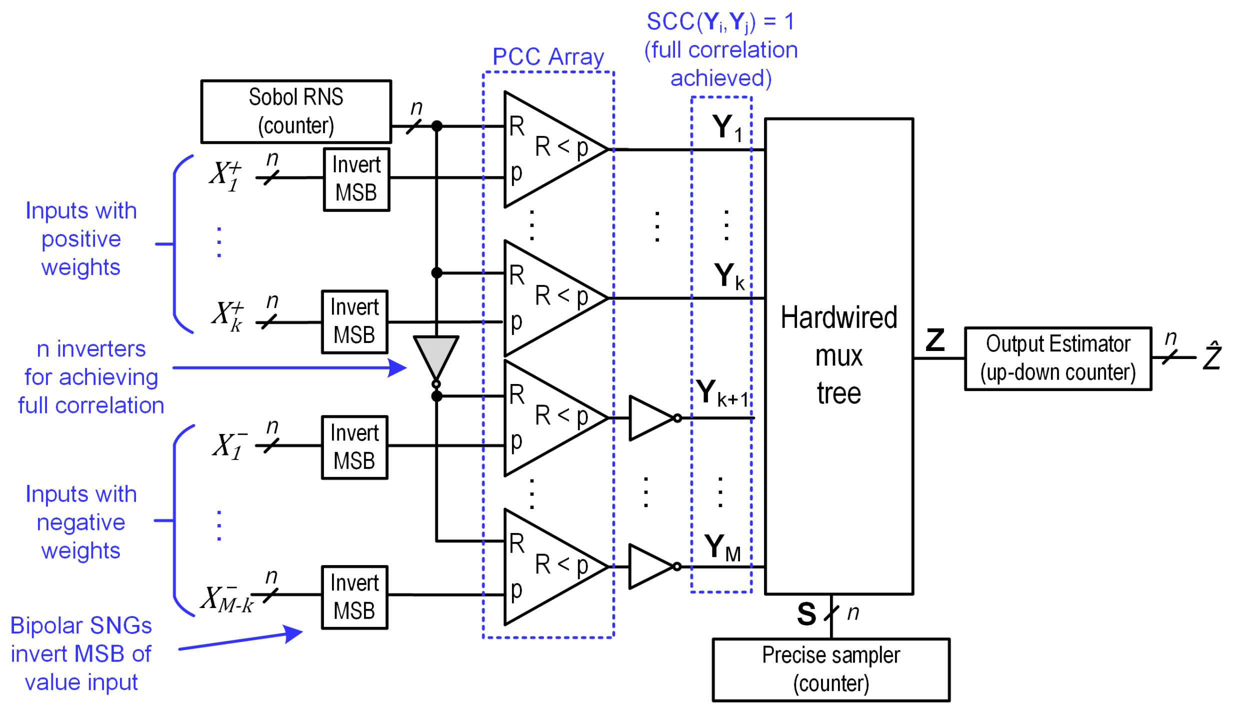
the same number of 1s and since they share an RNS, those 1s occur during the same clock cycle. Thus, $\mathbf{A}$ to $\mathbf{D}$ are identical bitstreams implying that the output bitstream $\mathbf{Z}$ will be the same for any realization of select inputs S_0 and S_1. Essentially, it does not matter which SN is sampled each clock since each SN is identical due to correlation. In this case, $\mathbf{Z}$ will have a correct estimated value of $\hat{Z} = Z = 0.5$ with zero variance when inputs $\mathbf{A}$ to $\mathbf{D}$ are hypergeometric SNs. Figure 5.6b is an extreme example since mux input SNs rarely all have the same value. In general, when mux input values differ, the input SNs will not be identical because each SN will contain a different number of 1s. In that case, the output variance is nonzero, but lower when input correlation is higher.

The component ϵ_{corr} quantifies how mux variance is impacted by bit-level correlation amongst mux data inputs:

$$\epsilon_{\text{corr}} = \frac{1}{L^2} \sum_{i=1}^{M} \sum_{\substack{j=1 \\ j \neq i}}^{M} \mathbb{E}[C_i]\mathbb{E}[C_j] \operatorname{Cov}(x_{i,k}, x_{j,l}) \tag{5.6}$$

where L is the SN length, M is the number of mux inputs, C_i is the number of times $\mathbf{X}_i$ is sampled, $x_{i,k}$ is a bit of $\mathbf{X}_i$ during arbitrary clock cycle k, and $x_{j,l}$ is a bit of a different SN $\mathbf{X}_j$ during a different, but otherwise arbitrary, clock cycle l ($k \neq l$).

ϵ_{corr} is 0 when no correlation is present or when the inputs are binomial SNs. ϵ_{corr} is positive when the inputs are anti-correlated hypergeometric SNs while ϵ_{corr} is negative when the inputs are correlated hypergeometric SNs. Most importantly, ϵ_{corr} is lowest when all mux data inputs are maximally correlated with a pairwise SCC of $+1$. When correlation is maximized in this manner, the mux is said to achieve full correlation. Implementing full correlation is not as straightforward



SCC(Y_i,Y_j) = 1
(full correlation achieved)
PCC Array
Sobol RNS (counter)
n
Inputs with positive weights
Invert MSB
R
p
R < p
Y_1
Y_k
Y_k+1
Y_M
Hardwired mux tree
Z
Output Estimator (up-down counter)
Ẑ
n inverters for achieving full correlation
Inputs with negative weights
Bipolar SNGs invert MSB of value input
S
Precise sampler (counter)

Figure 5.7: CeMux bipolar weighted adder.

as simply sharing an RNS amongst mux input SNs because of the XOR array found between the SNGs and mux tree (see Figure 5.2). A method for implementing full correlation is given in the following section which covers our CeMux design.

5.1.2 CeMux

Correlation-enhanced multiplexer (CeMux) refers to a bipolar weighted mux adder that combines our two correlation-inspired techniques, precise sampling and full correlation, with other recent advances in SC to form a particularly efficient design method for large adders. Figure 5.7 shows CeMux which implements weighted addition (Equation (5.2)). CeMux uses an XOR array and hardwired mux tree as in the general mux adder structure of Figure 5.2. Since the weights are fixed and known ahead of time, Figure 5.7 simplifies the XOR array by explicitly showing that inputs with negative weights are inverted by the XOR array while inputs with positive weights are unmodified by the XOR array. The remainder of this section explains CeMux component by component.

Like other compact mux adders [51, 91, 97], CeMux uses a single RNS to generate its data input SNs. While LFSRs are a popular and relatively low-cost RNS-type, a stochastic circuit's accuracy can sometimes be improved by using Sobol sequence generators as the RNS for SN generation [29, 65]. Sobol sequences and the properties of SNs generated by such sequences are covered

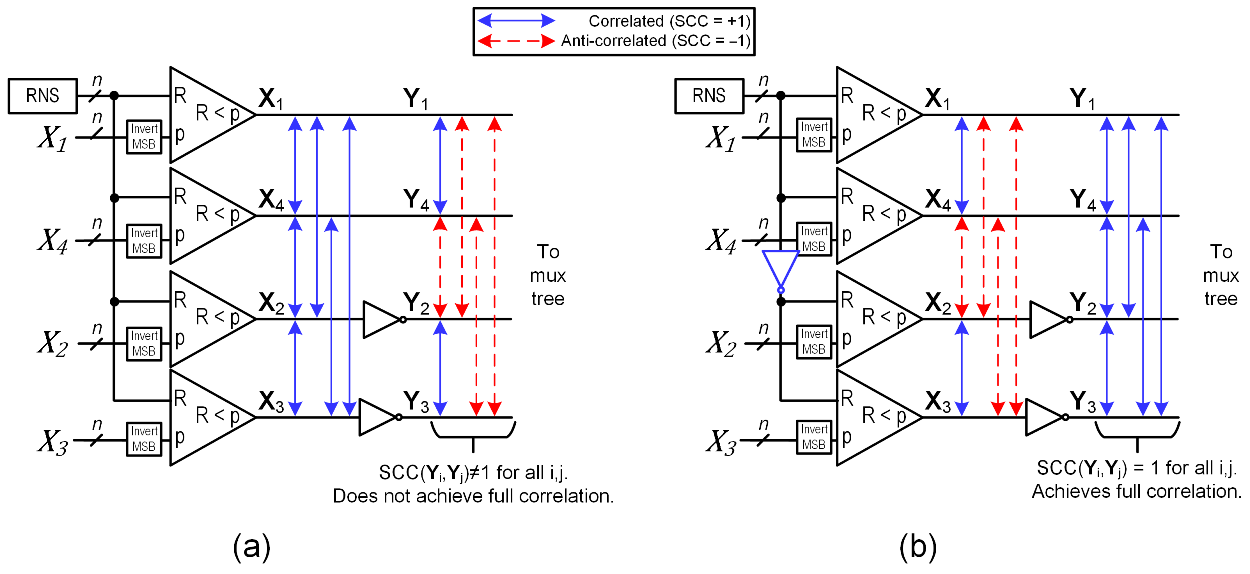

Correlated (SCC = +1)
Anti-correlated (SCC = –1)
RNS
n
X1
X4
X2
X3
Invert MSB
R
p
R < p
X1
X4
X2
X3
Y1
Y4
Y2
Y3
To mux tree
SCC(Yi,Yj)≠1 for all i,j.
Does not achieve full correlation.
SCC(Yi,Yj) = 1 for all i,j.
Achieves full correlation.
(a)
(b)

Figure 5.8: Visualizing the correlation of mux tree inputs: (a) conventional SN generation with RNS-sharing; (b) full correlation generation method.

in Section 4.2.3. CeMux requires a single RNS and uses the simplest Sobol RNS which can be implemented with the reverse state of a standard binary counter [72]. Our later experiments show that using a Sobol RNS in place of an LFSR improves CeMux's accuracy.

CeMux's mux tree achieves full correlation when $\mathrm{SCC}(\mathbf{Y}_i, \mathbf{Y}_j) = +1$ for all i, j. Ensuring maximum correlation is not as straightforward as simply sharing the RNS amongst all data input SNGs. Figure 5.8a illustrates this point where a mux containing four inputs $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, $\mathbf{X}_4$ with corresponding weights $W_1, W_4 > 0$ and $W_2, W_3 < 0$ is shown. In this case, the pairwise SCC amongst $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, $\mathbf{X}_4$ is 1. However, since $W_2, W_3 < 0$, $\mathbf{X}_2$ and $\mathbf{X}_3$ are inverted by the inversion array which results in $\mathrm{SCC}(\mathbf{Y}_1, \mathbf{Y}_4) = \mathrm{SCC}(\mathbf{Y}_2, \mathbf{Y}_3) = 1$ and $\mathrm{SCC} = -1$ for the other four pairwise combinations of $\mathbf{Y}_1$, $\mathbf{Y}_2$, $\mathbf{Y}_3$, $\mathbf{Y}_4$. Thus, full correlation is not achieved for the mux tree in Figure 5.8a.

In contrast, CeMux's SNG configuration for these SNs is shown in Figure 5.8b. A single RNS is shared by the SNGs, but the RNS output is inverted for $\mathbf{X}_2$'s and $\mathbf{X}_3$'s SNGs. The result is $\mathrm{SCC}(\mathbf{X}_1, \mathbf{X}_4) = \mathrm{SCC}(\mathbf{X}_2, \mathbf{X}_3) = 1$, and $\mathrm{SCC} = -1$ for all other SN pairings. Following the inversion of $\mathbf{X}_2$ and $\mathbf{X}_3$, the SCC between all pairings of $\mathbf{Y}_i$ and $\mathbf{Y}_j$ is $+1$ and full correlation is achieved by the mux tree. More generally, to achieve full correlation in CeMux, all inputs share an RNS, but inputs with negative weights use the inverted RNS output for SN generation while inputs with positive weights use the unaltered RNS output for SN generation.

CeMux's hardwired mux tree height is set to n where 2^n is the SN length. This is the largest tree height that enables precise sampling to function fully and yields the lowest quantization error. A

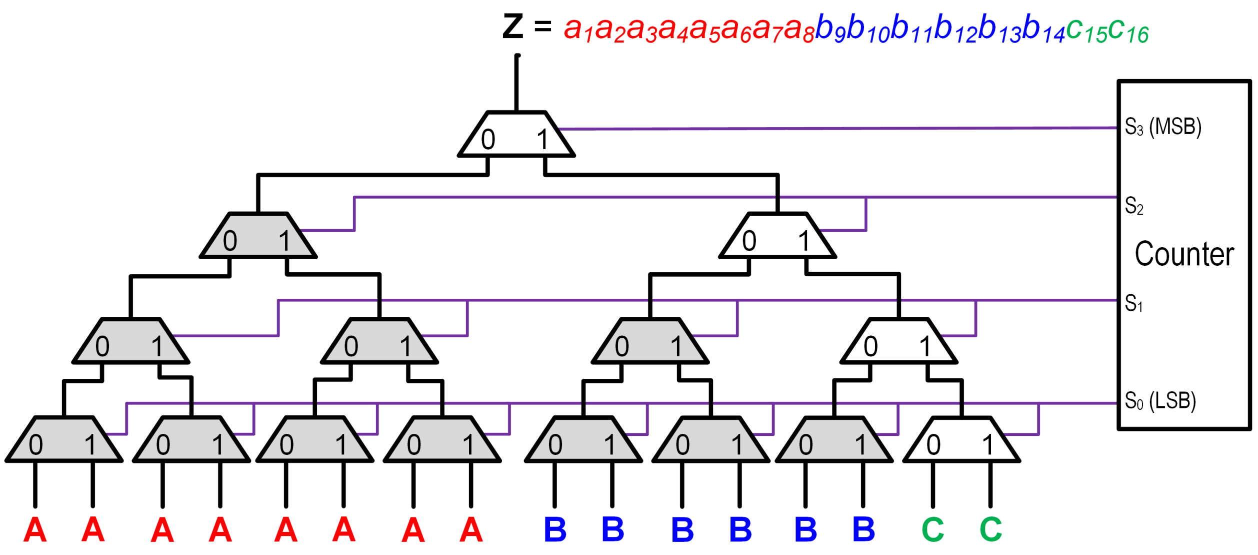


Figure 5.9: CeMux precise sampling implementation. The shaded muxes are redundant and can be removed.

high mux height n might seem to imply a high area cost due to the exponential number of 2-input muxes needed to construct the tree. That is not the case, however, because many of the 2-way muxes in a hardwired mux tree have identical data inputs and thus can be eliminated. For instance, all the shaded muxes in Figure 5.9 can be removed, reducing the mux count from fifteen to four. In general, a loose upper bound on the number of 2-input muxes in a height-n hardwired mux tree with M unique input SNs is $\min(Mn-1, 2^n-1)$ [13]. Thus, for a given input size M, the number of mux gates grows linearly with tree height n rather than exponentially as 2^n. CeMux implements precise sampling by using an n-bit counter's state as the mux select input lines. The counter's i-th MSB is connected to muxes on the i-th level of CeMux's hardwired mux tree as in Figure 5.9.

Overall, CeMux is built around a hardwired mux tree adder design similar to HWA [91], but with three new key optimizations: full correlation, precise sampling, and a Sobol RNS. The following experiments measure the root Bayes mean squared error (BMSE) of a hardwired mux tree adder with and without CeMux's these three optimizations. For each configuration, the adder is simulated with random bipolar input values X_i and weights W_i. The circuit precision is fixed at $n = 10$, and the number of inputs M is varied from 8 to 256. The SN length is $L = 2^{10}$ and $R = 5,000$ simulation runs are used. Root BMSE is estimated as

$$\text{Root BMSE} \approx \sqrt{\frac{1}{R}\sum_{r=1}^{R}(\hat{Z}_r - Z_r^*)^2} \tag{5.7}$$

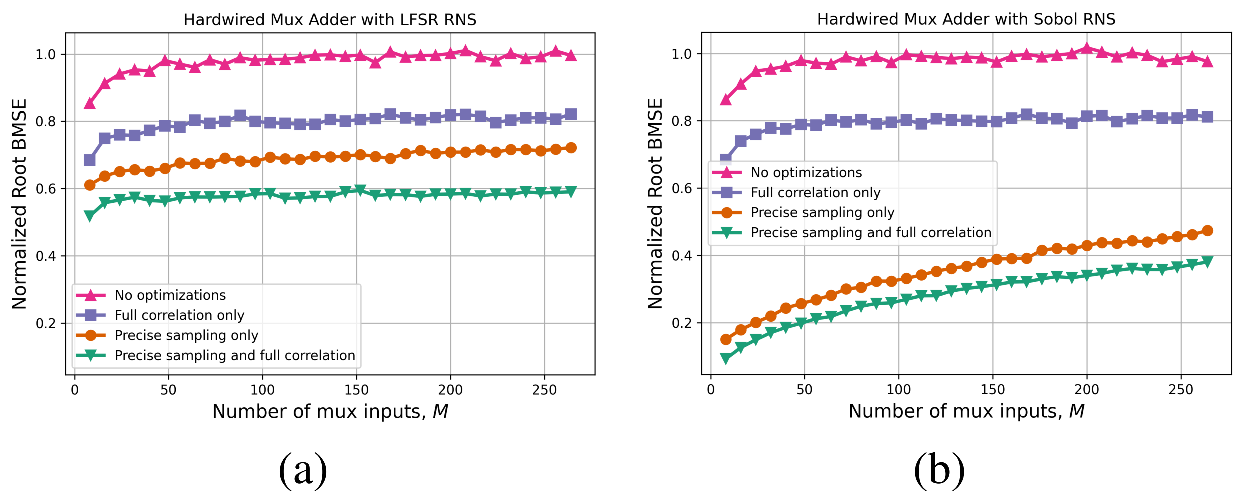


Figure 5.10: Error of a mux adder with various optimizations and when using: (a) an LFSR RNS; (b) a Sobol RNS.

where $\hat{Z}_r$ and Z_r^* are, respectively, $\mathbf{Z}$'s estimated and target values during simulation run r. Figure 5.10a shows the accuracy results when the adder uses a LFSR RNS while Figure 5.10b shows the accuracy results when using a Sobol RNS. The curves within both Figures 5.10a and 5.10b correspond to different configurations of using or not using full correlation and precise sampling. The "Precise sampling and full correlation" curve in Figure 5.10b corresponds to our full CeMux design while the "No optimizations" curve in Figure 5.10a corresponds to a traditional mux adder like HWA [91]. These error results are normalized by multiplying by $\sqrt{L}$ where $L = 1024$ is the SN length used in the experiments.

As the number of inputs increases, the root BMSE for every configuration increases and then saturates which is consistent with other mux adder studies [51, 71, 91, 97]. When using an LFSR RNS, the mux accuracy is exactly as our earlier analysis predicted. Full correlation and precise sampling decrease variance by about 33% each which translates to a reduction of about 40% in terms of root BMSE.[1] Comparing Figures 5.10a and 5.10b illustrates that using the Sobol RNS greatly improves accuracy when precise sampling is also used. Depending on the input size, using a Sobol RNS with precise sampling and full correlation is 1.5 times to 5.7 times more accurate than using a LFSR with both correlation techniques. This improvement arises because the method of precise sampling with a counter works particularly well with the Sobol RNS as explained earlier.

In all, the full CeMux design is 2.6 times to 9.3 times more accurate than a conventional hard-wired mux tree design like HWA [91]. In terms of area, CeMux is also smaller than existing mux adder designs because the precise sampling uses much less hardware than prior approaches that

[1]In this case, bias is relatively low so root BMSE is approximately equal to the square root of variance.

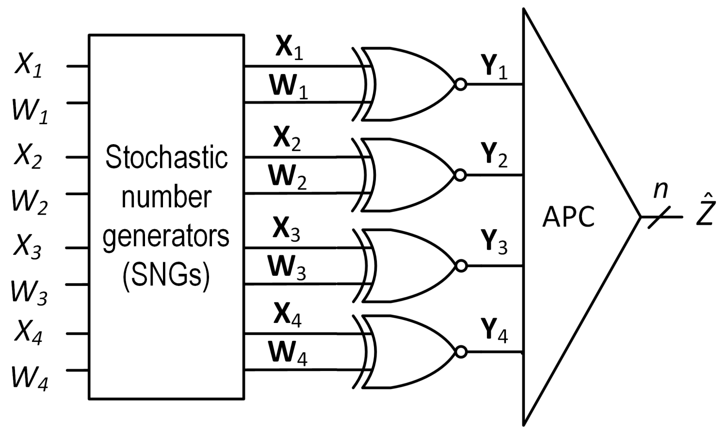


Figure 5.11: APC-based bipolar weighted adder with $M = 4$ input size.

have noisy sampling. For example, the later case study in Section 5.2 finds that CeMux's area is 30% to 75% less than traditional mux adder areas.

5.2 ECG Filtering Case Study

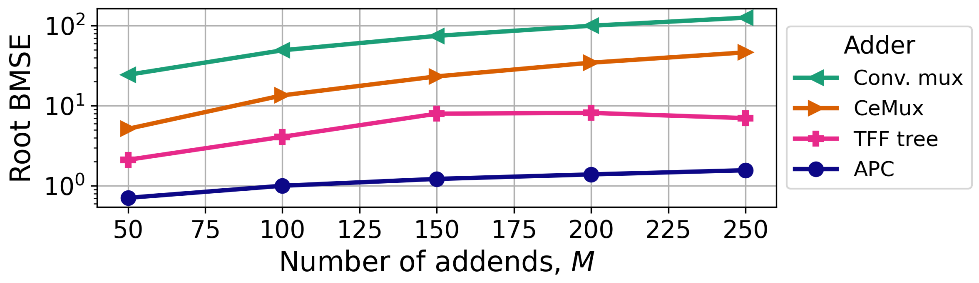
The World Health Organization estimates that 17.9 million people died of cardiovascular disease in 2019, representing about 32% of global deaths [93]. Low-power continuous heart rate monitoring can help to better address the risk of cardiovascular disease. A key preprocessing step in heart monitors is the denoising of the electrocardiogram signal with digital filters [83]. Large-scale SC weighted adders have been applied to the design of linear digital finite impulse response (FIR) filters [13, 50, 91] that are often used to denoise signals and have desirable properties like linear phase response. An M-tap filter implements

$$Z_t = \sum_{k=0}^{M-1} h_k X_{t-k} \tag{5.8}$$

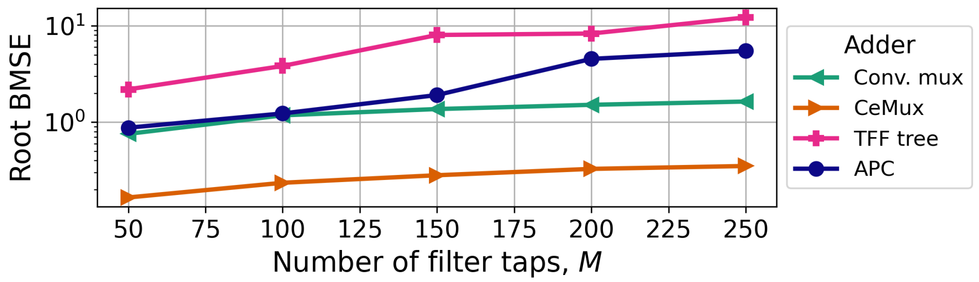
where X_t is the noisy input signal, h_k are the M filter coefficients and Z_t is the filtered output signal. Generally, FIR filters with more taps (higher M) perform better filtering at the cost of more computational resources. Here, various large-scale bipolar SC adders are used to implement an FIR filter that denoises ECG signals.

The purpose of this case study is to determine how various large-scale SN adders perform when used implement FIR filtering applied to ECG denoising. The tested designs include a conventional hardwired mux tree design [91], a conventional weighted APC design like that of Figure 5.11, a design built around a tree of T flip-flops (TFF tree) [60] and our CeMux design [13].

Before testing the ECG filtering application, each design is evaluated using a baseline test where



(a)



(b)

Figure 5.12: Error of bipolar weighted SC adders with: (a) uniformly random input and weight values; (b) ECG filtering input and weight values.

the adders are configured to add bipolar SNs with random values $X_i \in [-1, 1]$ and weights $W_i \in [-1, 1]$. The circuit precision is $n = 10$ bits and SN length is $L = 2^{10}$ bits. The input size M is varied as each adder's root BMSE is estimated using $R = 10,000$ runs per value of M.

This baseline experiment with random inputs and weights helps validate the correctness of software implementations and establishes expectations for how the designs will perform on other tests. Figure 5.12a shows the results after error values were normalized to account for difference in adder scale factor.[2] The APC has the lowest error because it takes an exhaustive counting approach to summation. The next most accurate design is the TFF adder which was introduced as a more accurate, but costlier alternative to the mux adder [60]. Finally, CeMux has the next best accuracy followed by the conventional mux adder.

The ECG filtering case study uses a similar setup to the baseline test: the circuit precision is $n = 10$ bits and the SN length is $L = 2^{10}$ bits. The number of filter taps M is varied as each adder's root BMSE is estimated using $R = 10,000$ simulations runs per value of M. The input values are derived from Physiobank's MIT arrhythmia data-base [36] where random noise is added to the

[2]Specifically, mux adders have scale factor $\alpha = \sum_{k=0}^{M-1} |h_k|$, the APC has no scale factor $\alpha = 1$ and the TFF tree has a scale factor that depends on the tree height h: $\alpha = 2^h$. The TFF tree height is determined by the number of adder inputs $h = \lceil \log_2(M) \rceil$. All scale factors are multiplied out so that error comparisons are fair.

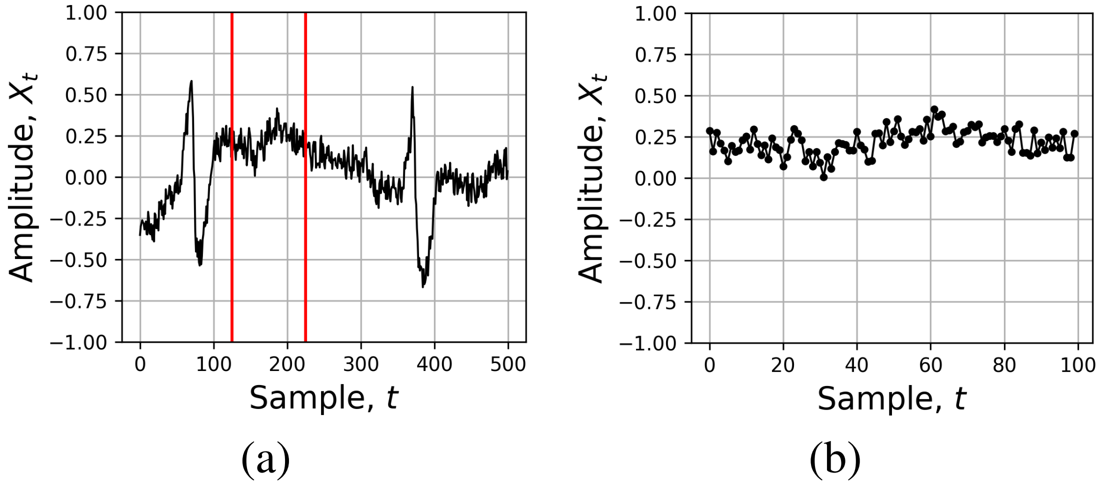


Figure 5.13: Noisy ECG signals: (a) 500 samples of a noisy ECG signal; (b) 100 samples of a noisy ECG signal taken from the region between the red vertical lines in (a).

benchmark signals as in [32] to simulate three major noise types: device noise, electro-surgical noise, and noise from muscle contractions. Filter coefficients are derived using MATLAB. The APC and TFF adders are expected for perform the best on the ECG task based on the random input baseline results. Intuitively, these designs are also expected to be more accurate because they are less random, albeit costlier, alternatives to the mux adder.

Figure 5.12b plots the root BMSE of each SC adder against the number of filter taps M. Compared to the baseline test, the APC's error is 1.3 times to 3.5 times higher depending on M. In contrast, the conventional mux and CeMux error is 31 times to 132 times lower compared to the baseline test. This is a significant decrease in error completely caused by the change in input value distribution. Overall, Figure 5.12 shows that the magnitude of each adder's error as well as the relative ranking of adder accuracy differ significantly between the baseline test and ECG case study. The two major contributing reasons for the change in adder performance are that 1) the ECG input value distribution is especially favorable for mux adders and 2) the weighted APC's accuracy is degraded by correlation-related error.

As explained in Section 4.4, an important, but sometimes overlooked influence on circuit accuracy is the application's input value distribution (IVD) [11]. In the case of ECG filtering, the weight values are derived using MATLAB and the input SN values are derived from a noisy ECG signal like that of Figure 5.13. These are highly non-uniformly random IVDs that lead to a different BMSE than was measured in the baseline test. Moreover, consecutive samples from an ECG signal are highly similar in value. For example, a 100-tap FIR filter may have the highly similar ECG samples shown in Figure 5.13b as input. Similarity of input values especially benefits CeMux's

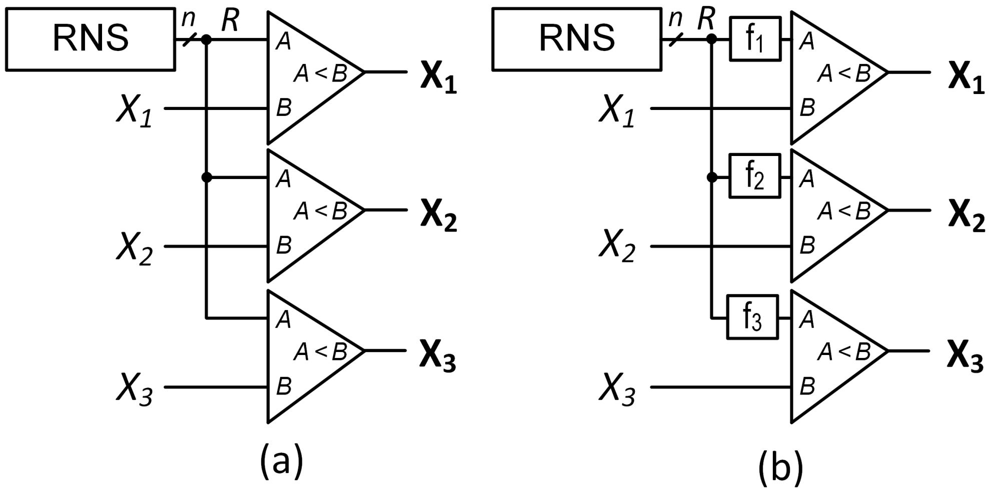


Figure 5.14: Sharing an RNS amongst several SNGs: (a) direct sharing; (b) sharing with inversion networks f_i.

accuracy [12, 13] as explained fully in Appendix A.2. Our data reflects these claims as CeMux has up to 132 times lower error when applied to ECG filtering than when applied to the random input baseline. This result underpins the importance of accounting for the IVD.

In contrast to CeMux, the APC's error was up to 3.5 times higher on the ECG filtering case study than on the random baseline test. This accuracy degradation is due to correlation from RNS-sharing which is normally thought to not affect APC accuracy [66]. Consider a 4-input weighted APC like that of Figure 5.11. Since the APC counts all 1s of the product SNs $\mathbf{Y}_1$ to $\mathbf{Y}_4$, the APC's error $\epsilon_Z = \hat{Z} - Z^*$ can be expressed as the sum of multiplication errors $\epsilon_{Y_i} = \hat{Y_i} - Y_i^*$ where $\hat{Y_i}$ is $\mathbf{Y}_i$'s estimated value and $Y_i^* = W_i^* X_i^*$ is $\mathbf{Y}_i$'s target value.

$$\epsilon_Z = \sum_{i=1}^{M} \epsilon_{Y_i} \tag{5.9}$$

When no RNS sharing is used, the $\mathbf{Y}_i$'s are independent and, on any given simulation run, about half the multiplication errors ϵ_{Y_i} are positive and about half are negative. The positive and negative errors partially cancel out when the $\mathbf{Y}_i$ are summed by the APC which results in a low overall APC error ϵ_Z. For example, if the multiplication errors of a 4-input weighted APC are 0.02, -0.1, 0.05 and -0.03, then the overall APC error is -0.06.

In contrast, when RNS-sharing is employed, the product SNs $\mathbf{Y}_i$ and the multiplication errors ϵ_{Y_i} are correlated. In this case, on any given simulation run the magnitude of the multiplication errors are the same as the no RNS-sharing case, but about 60% of the multiplication errors have the same sign. Consequently, the overall weighted summation error ϵ_Z is higher as multiplication

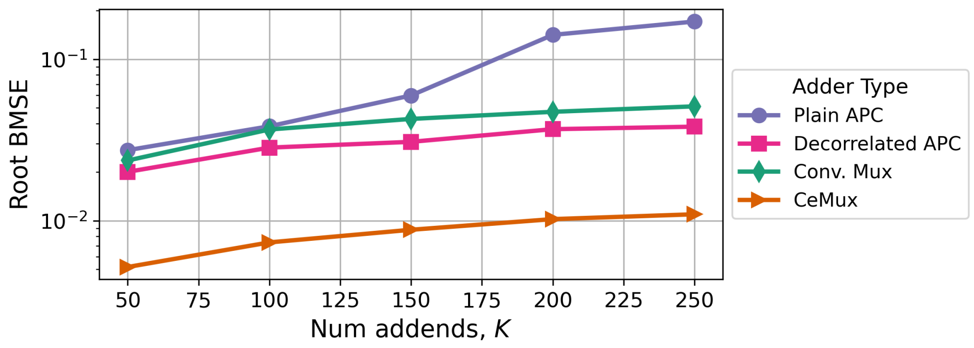


Figure 5.15: ECG filtering error for various bipolar SC adders.

errors accumulate much more often and cancel out less often. For example, if the multiplication errors are 0.02, 0.1, 0.05, and 0.03, then the overall APC error is 0.2 which is much higher than the previous example because all the multiplication errors have matching signs.

In [14], we proposed a low-cost technique for decorrelating the weighted APC design. It involves applying a randomly chosen network of inverters to the shared RNS output before it enters each SN's PCC as illustrated in Figure 5.14b. Using the decorrelation scheme lowers the APC's root BMSE for ECG filtering by 1.4 times to 4.5 times depending on the input size. Although the decorrelation scheme significantly improves the accuracy of the APC in the ECG filtering case, it does not affect APC accuracy in the baseline or in any of the other case studies in this work. In those cases, the multiplication errors ϵ_{Y_i} are uncorrelated even without decorrelation. The sensitivity of APCs to input correlation therefore depends on the application's input value distribution. In all of our experiments, however, the proposed decorrelation technique either decreased error or left it unchanged compared to direct sharing. Given that this decorrelation technique is low-cost, it appears useful in general.

Even with decorrelation, the APC's error is still up to 3.8 times higher than CeMux's error for ECG filtering as shown in Figure 5.15. This result is extremely surprising since APCs are usually more accurate than mux adders and since, intuitively, APCs should be more accurate due to their exhaustive counting approach. However, the ECG case study's IVD is favorable for CeMux whose accuracy depends on differences in input values. One advantage of the APC design over CeMux, however, is that the weights are programmable whereas CeMux implements weights via hardwiring. Therefore, CeMux is best used with field programmable gate arrays or for applications where the weights are not expected to change as in hearing aid filtering [16, 25] or ECG filtering [9].

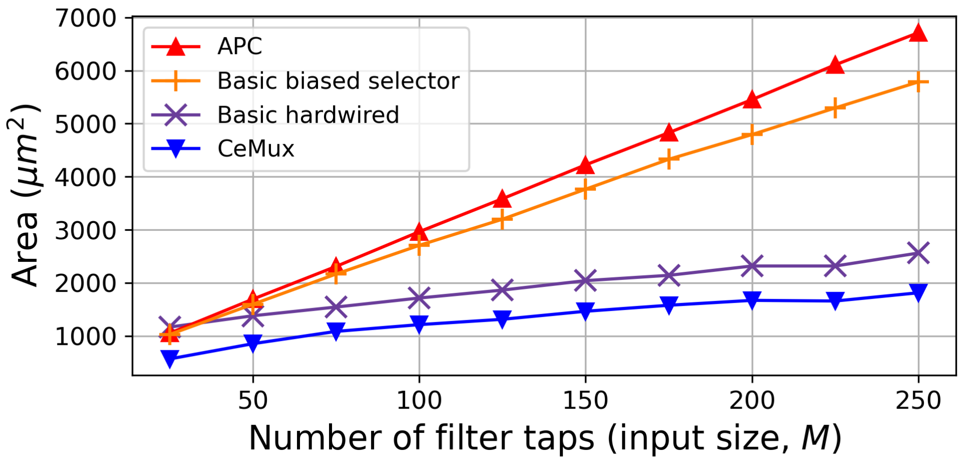


Figure 5.16: Area vs. input size for various SC digital filter designs.

5.2.1 Area Comparison

Next, Synopsys Design Compiler is used with the Nangate 45nm open cell library [85] to synthesize SC filters and estimate the area of their weighted addition datapath. We do not consider the memory used to store prior signal values since each design requires the same amount of memory. Figure 5.16 plots the circuit area versus number of filter taps M for $n = 10$ bit precision. CeMux occupies 30% to 75% less area than other SC designs because it replaces costly SNGs for weights or mux select inputs with a simple but precise sampling counter. Figure 5.16 also shows that the two largest designs are the conventional biased selector mux adder and the APC. These designs have higher area because they use more SNGs and because, unlike the other designs, they are flexible in their ability to update filter coefficients stored in an external memory whose cost is not considered here. While flexibility in updating filter coefficients is a convenient feature, FIR designs for resource limited applications like ECG filtering [9] and hearing aids [25] often assume and benefit from fixed filter coefficients. If programmable weights are desired, a CeMux-like design built around biased selector mux tree can be used [14].

The focus of our analysis has been on the analysis and improvement of SC mux adders. We have demonstrated that CeMux is the best mux-based SC adder in terms of accuracy (Figure 5.15) and area (Figure 5.16). For completeness, we also give a brief comparison with a conventional binary design. We compare a 10-bit CeMux filter with a traditional sequential binary (SB) filter designed using MATLAB's Filter Design HDL coder. The SB design is synthesized assuming the filter coefficients are fixed, and the SB design employs standard optimizations like the exploitation of symmetric coefficients which greatly reduces multiplier count. Note that the designs' precision levels are chosen to give them similar accuracy, CeMux uses 10-bit precision while the SB design

	10-bit CeMux Filter			**10-bit Sequential Binary Filter**		
Filter size M	**Area** (μm^2)	**Power** (μW)	**Root BMSE** ($\times 10^{-3}$)	**Area** (μm^2)	**Power** (μW)	**Root BMSE** ($\times 10^{-3}$)
25	566	13.16	4.17	1158	20.67	5.06
50	853	20.16	4.47	1385	23.45	3.49
75	1085	25.71	4.57	1603	28.51	3.99
100	1212	28.82	4.87	1761	29.64	3.61
125	1314	31.42	5.16	1855	30.83	3.13
150	1465	35.52	5.41	2066	36.58	4.69
175	1578	37.50	5.62	2119	37.99	4.85
200	1670	39.79	5.64	2293	39.71	3.55
225	1658	39.44	6.26	2324	39.70	4.19
250	1813	43.20	6.05	2477	41.70	4.12

Table 5.1: Cost and Performance of a CeMux Filter and a Sequential Binary (SB) Filter

uses 8-bit precision. Both designs are also configured to operate in real-time which requires each one to process digitized ECG samples at the sampling rate of 360 Hz.

Table 5.1 shows the area, power and RMSE of the CeMux and SB filters as the filter size M is varied from 25 to 250. CeMux's area is 49% to 73% lower than the SB design's area due to the use of cheap SC computational units. The CeMux design must process the entire 1024-bit SN at a rate of 360 Hz to meet the real-time latency constraint. Consequently, CeMux's digital clock frequency is set to be faster than the SB's design digital clock frequency which results in both designs having similar power despite CeMux having lower area. Finally, the SB design has slightly better root BMSE, especially as input size grows. Based on the data presented in Table 5.1, we conclude that CeMux, a mux-based SC adder, has the potential of being a lower-cost alternative to conventional binary designs. Besides being smaller, SC designs also offer greater fault tolerance [7] which is one avenue for future exploration with CeMux.

As a closing example, we compare the performance of SC filter designs in the case of a noisy ECG signal filtered by an $M = 100$ tap filter with precision n set to 10. As Figure 5.17 shows, the CeMux-based filter produces the smoothest, most noise-free curves, another reflection of CeMux's superior accuracy. In general, we have seen that CeMux is the best mux adder design developed so far for large weighted-adder networks, in terms of both accuracy and area trade-offs. These properties result from three key design features: full correlation, precise sampling and a Sobol RNS. CeMux can thus be considered a major step towards practical implementation of many-input, compact adders for a variety of SC applications

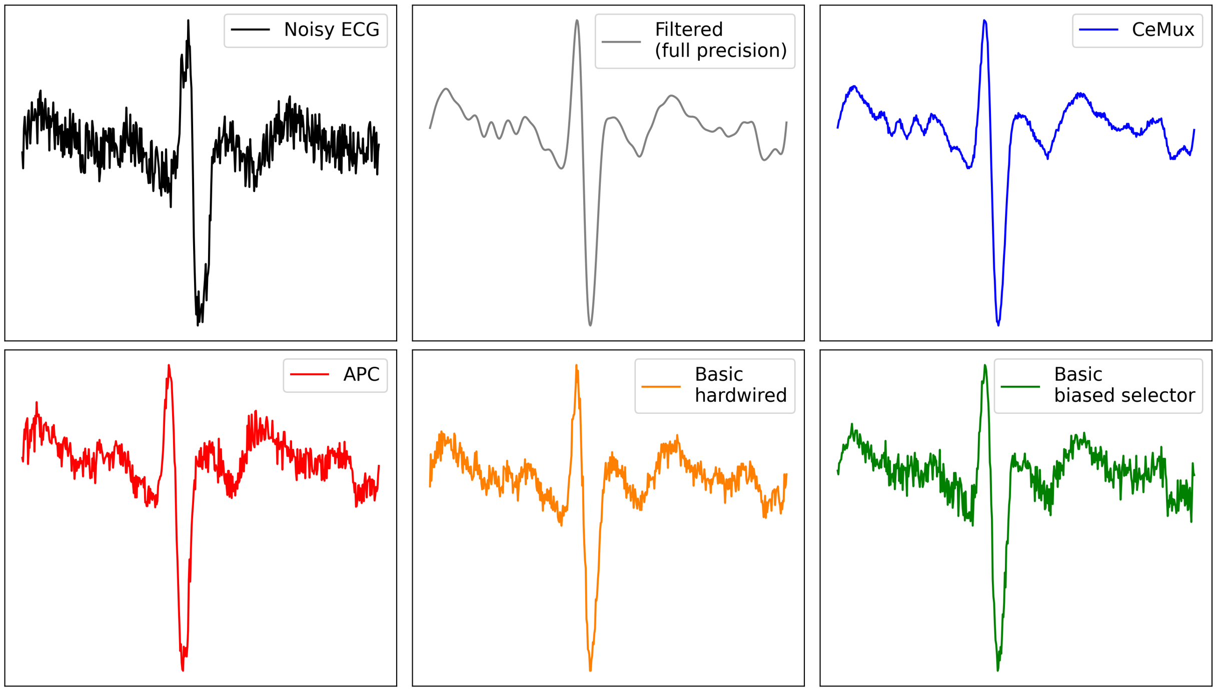


Figure 5.17: Noisy ECG waveform (top-left) filtered by a full precision design (top-middle) and various 10-bit precision SC designs.

5.3 Parallel Sampling Adders

Although CeMux is a major improvement over prior mux adder designs and even outperforms APCs for ECG filtering, mux adders are generally less accurate than APCs. The latency of CeMux is also fundamentally limited by its sampling approach. Specifically, if CeMux has M inputs, then the SN length L must be at least M clock cycles so that each input is sampled at least once. If $L < M$, then some inputs are never sampled by CeMux which usually results in high error. For example, a fully connected input layer of a network used to classify Fashion-MNIST clothing items [95] employs inner products with 784 inputs each. In that case, CeMux would require an SN length of at least 784 bits (and likely more) to achieve good image classification accuracy. In contrast, an APC may needs only 16 bits for good classification performance [14, 45]. A major drawback of APCs, however, is that they often dominate the area of SC-based neurons due to their large size [45, 66].

In short, mux adders like CeMux sample one bit from one input per clock cycle, thus limiting their accuracy. On the other hand, an M-input APC adds the bits from all input SNs and, in some sense, samples all M inputs each clock cycle resulting in high area. In between these two extremes lies an opportunity for a new flexible design methodology, our parallel sampling adders (PSAs) [14]. Each clock cycle, an M-input PSA samples between 1 and M input SNs. The

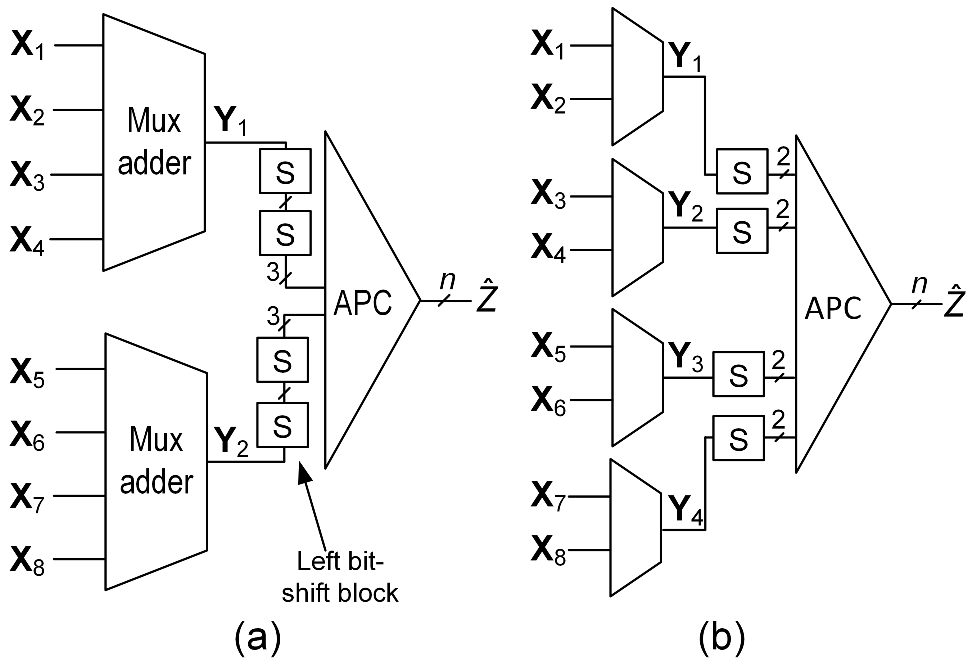


Figure 5.18: Basic PSA designs with group size: (a) $G = 4$; (b) $G = 2$.

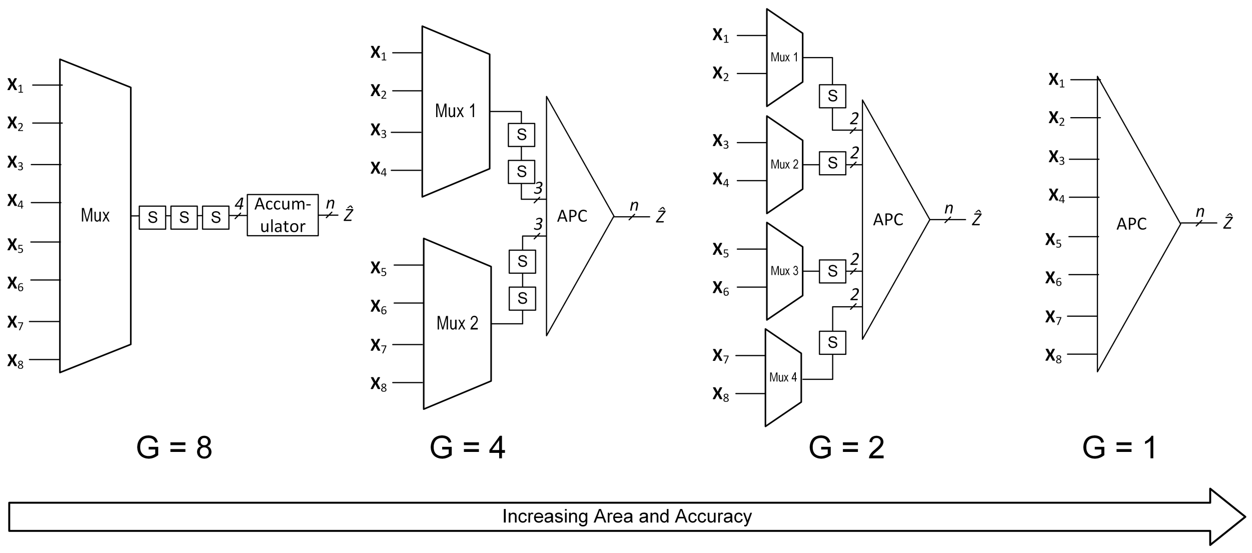


Figure 5.19: All 8-input PSA designs.

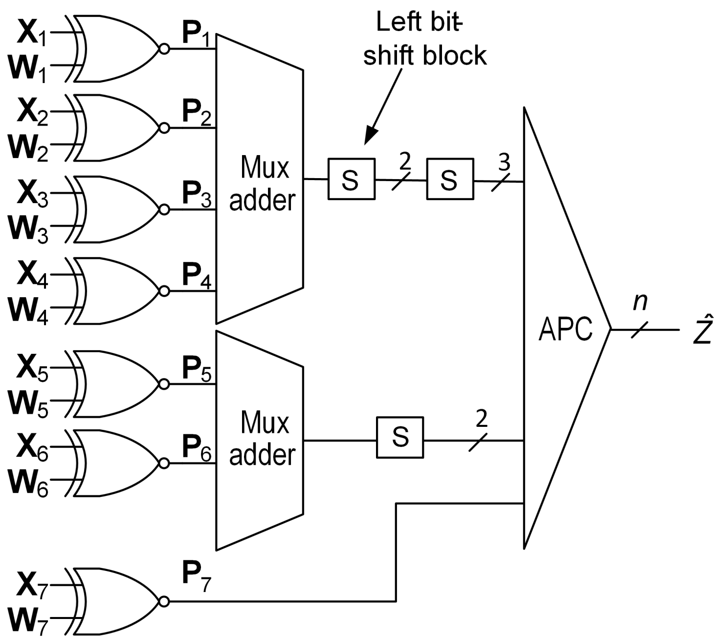


Figure 5.20: Weighted PSA with group size $G = 4$.

PSA framework generalizes the mux and APC adders and includes hybrid mux-APC designs that are more accurate than mux adders, but less costly than APCs. Most usefully, PSAs exhibit an accuracy-area trade-off that can be adapted based on the application and its requirements.

PSAs combine mux adders and APCs into a single flexible design as shown in Figure 5.18. In a PSA, input SNs are arranged into disjoint groups of size G, each of which is applied to a separate mux adder. The outputs of all muxes are then accumulated by an APC. For example, Figure 5.18a shows an 8-input PSA with two groups of size $G = 4$. Both groups are summed separately by muxes with outputs $\mathbf{Y}_1$ and $\mathbf{Y}_2$ that have value $Y_1 = 0.25\sum_{i=1}^{4} X_i$ and $Y_2 = 0.25\sum_{i=5}^{8} X_i$, respectively. Left bit-shifts are used to remove the 0.25 scale factors from the mux output values before $\mathbf{Y}_1$ and $\mathbf{Y}_2$ are accumulated by a small APC whose output value is $Z = \sum_{i=1}^{8} X_i$.

We parameterize a PSA by its group size G which is the maximum input size of any mux adder used in the PSA. For example, the PSA in Figure 5.18a has $G = 4$ while the PSA in Figure 5.18b has $G = 2$. Group size represents the degree of approximation of the PSA – higher G means a more aggressive approximation and typically lower area and accuracy. For an M-input PSA, G takes values between 1 and M and is restricted to be a power-of-two so that scale factors introduced by the mux adders can always be removed by bit-shifts. For example, all 8-input PSAs are shown in Figure 5.19. All muxes in a PSA are implemented as CeMux designs because it is the smallest and most accurate mux adder [13]. APCs and CeMux are special cases of the PSA design corresponding to group size $G = 1$ and $G = M$, respectively.

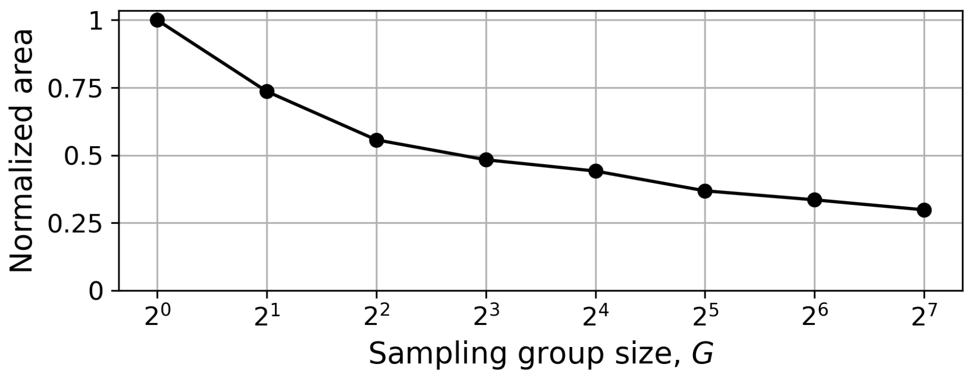


Figure 5.21: Unipolar PSA area trade-offs. Area is normalized by dividing by an APC's area.

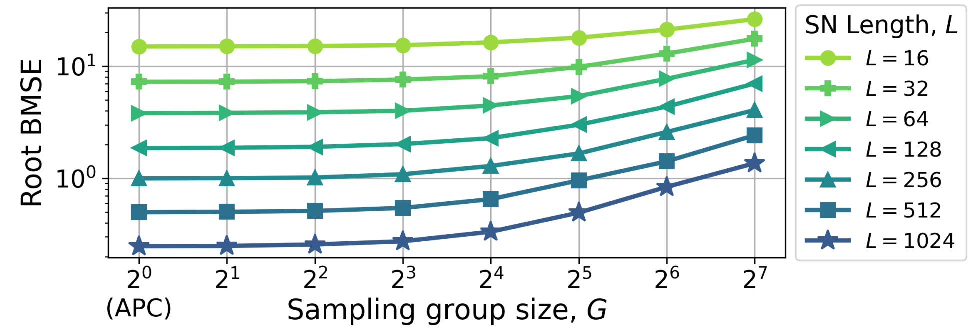
Like APCs, the basic PSA designs in Figures 5.18 and 5.19 can be extended to implement general weighted addition. Figure 5.20 displays a bipolar weighted PSA with $M = 7$ inputs and $G = 4$ group size. It is designed in the following manner. First, inputs and weight SNs are multiplied using XNOR gates to produce 7 product SNs $\mathbf{P}_i$ with value $P_i = W_i X_i$. The product SNs are then grouped into sets of size $G = 4$. In this case, one group is constructed: $\{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4\}$. Three SNs remain that cannot form another four-element group, so G is cut in half and one group of size 2 is constructed: $\{\mathbf{P}_5, \mathbf{P}_6\}$. As one SN remains, the group size is again cut in half to size one and a final group $\{\mathbf{P}_7\}$ is formed. Each group is input to a separate mux and all mux outputs are summed with an APC to produce the final output $\hat{Z}$.

One viewpoint on PSA design is that it uses mux adders to approximate and reduce the size of a pure APC design. PSAs may therefore seem superficially similar to approximate APC designs like AxPC [55] and SUC adders [62], but they actually differ in fundamental ways. Unlike AxPC and SUC adders, PSAs can be used with bipolar SNs. PSA weights also do not need to be partitioned into disjoint subsets that sum to 1 as in SUC adders. Lastly, PSA weights are programmable, whereas changing weights in an SUC adder requires redesigning hardwire interconnections.

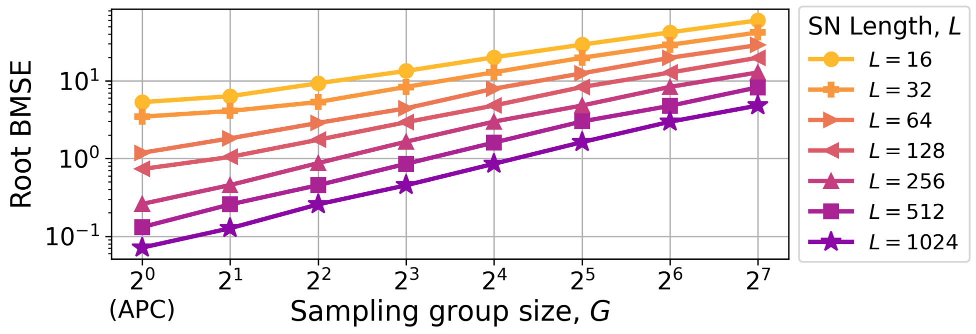
5.3.1 Accuracy Area Trade-offs

An M-input PSA's area and accuracy depend on the PSA's group size $G \in \{1, 2, 4, 8, ..., M\}$. When $G = 1$, the PSA is equivalent to an APC. Larger G implies a higher level of approximation compared to an APC which leads to lower area and accuracy. A PSA's G is best taken into context with its input size M. For example, when $M = 512$, a group size of $G = 16$ implies a relatively low amount of approximation. In contrast, when $M = 16$, a group size of $G = 16$ is the highest level of approximation possible.

To determine how area changes with G, Synopsys Design Compiler (DC) with the Nangate



(a)



(b)

Figure 5.22: Accuracy trade-off of a: (a) unipolar PSA; (b) bipolar PSA.

45nm library [85] is used to synthesize PSAs with various sampling group sizes, input sizes, and SN lengths. Figure 5.21 shows the area of a PSA configured to perform weighted addition on 512 unipolar SNs of length 128. Here, area varies roughly linearly with the log of the group size G. When $G = 1$, the PSA is equivalent to a pure APC adder and its area is about 70% higher than the smallest PSA design with $G = 512$. These results make intuitive sense: as sampling group size increases, the size of the APC in the PSA shrinks proportionally thus decreasing area cost in a consistent manner. Other SN lengths and input sizes were tested and yielded similar normalized area results.

The impact of group size G on accuracy is investigated by simulating a 512-input PSA with SN length L using both unipolar and bipolar SC. For each pair of values G and L, the PSA is simulated with uniformly random input and weight values: $X_i, W_i \in [0, 1]$ for unipolar SC and $X_i, W_i \in [-1, 1]$ for bipolar SC. As before root BMSE is estimated over $R = 10,000$ simulation runs with Equation (5.7).

Figure 5.22a plots the unipolar SC simulation results. Interestingly, all PSAs with $G \leq 16$ have nearly the same root BMSE as an APC, which is equivalent to a PSA with $G = 1$. This finding is significant because a PSA with $G = 16$ has 50% less synthesized area than an APC, as shown in Figure 5.21. For $G \geq 32$, the log of root BMSE increases roughly proportionally with the log of G implying that area can be traded for accuracy by adjusting G. The SN length L only affects the magnitude of error but does not affect the shape of the root BMSE vs. group size curve. Thus, the accuracy trade-off is very consistent across different SN lengths.

Next, Figure 5.22b plots the bipolar SC simulation results. As in the unipolar case, SN length L only impacts the magnitude of root BMSE, but not how it varies with PSA group size G. However, unlike the unipolar case, the accuracy does not saturate as G decreases. Instead, root BMSE exhibits a consistent power law relationship with G for all tested SN lengths L. The error saturates in the unipolar case, but not the bipolar due to complicated interactions between the Sobol RNS used in the CeMux adders and the multipliers in the weighted PSA design. Nonetheless, PSA still offers a useful area-accuracy trade-off for bipolar SC whose value is demonstrated in the following case study.

5.4 Binarized Neural Network Case Study

Next, PSAs are applied to neural network image classification. Large-scale SC adders have been used to design low-power neural network ASICs [66] which rely heavily on the weighted summation operation. One such design [45] uses SC to implement the first layer of a binarized neural network (BNN). BNNs are a class of neural networks where all weights and activations are binarized to take values ± 1 during the inference (but not the training) phase. This extreme quantization facilitates the design of low-cost NNs while maintaining high classification accuracy on image classification benchmarks.

When used for image processing, BNNs often do not binarize the raw (grayscale) pixel inputs because too much information would be lost. Thus, pixel values are kept at their original 8-bit precision implying that the first NN layer will require fixed-point multipliers and adders. Instead of employing traditional fixed-point computing, implementing the first BNN layer with SC and popcount circuits (i.e., APCs) leads to a 62% reduction in overall area for the two-layer network in [10]. Here we investigate the extent to which PSAs can be used to further improve hardware efficiency while maintaining classification accuracy.

As in [45], we apply a binarized multi-layer perceptron with two 1,024-neuron hidden layers to the Fashion-MNIST benchmark [95]. Fashion-MNIST consists of 28x28 grayscale images of fashion items that fall into one of ten classes like “coat” or “sneaker” – examples are shown in Figure 5.23. The hybrid SC-BNN network employs PSA adders in the first layer and the iterative training procedure from [45] is used to train the network.

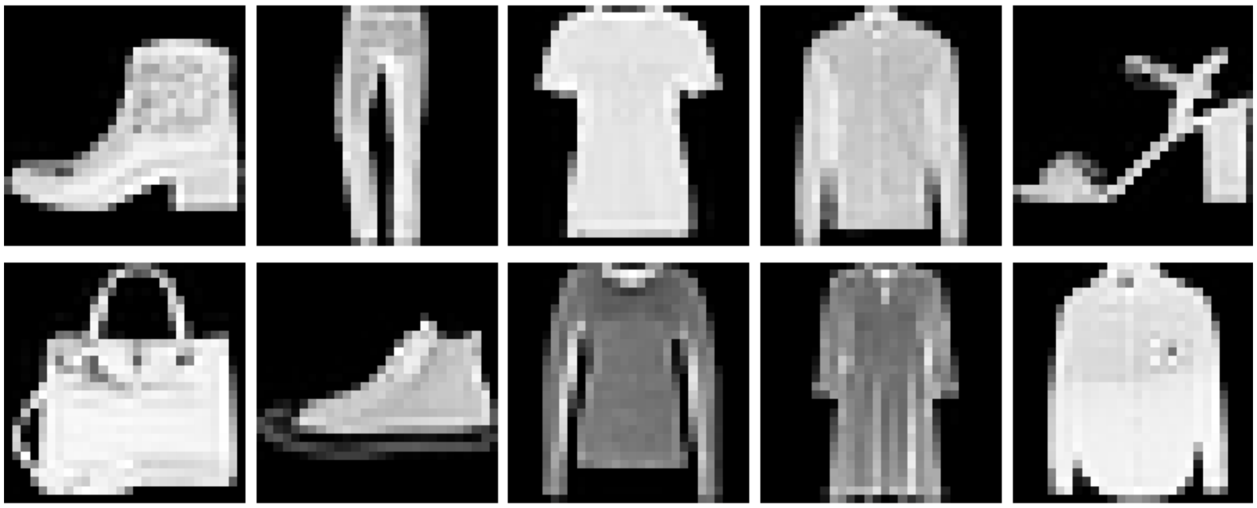


Figure 5.23: Example images from each class of the Fashion-MNIST classification benchmark [95].

After training, the network is evaluated on the 10,000 test images from the Fashion-MNIST dataset. The SN length is set to 16 bits which is the shortest length that gave the same image classification accuracy as a fixed-point version of the network. The area of a first-layer neuron is measured for each value of G using Synopsys DC with the Nangate 45nm library [85] and area is normalized by dividing by the area of a neuron that employs an APC adder. Since SNGs can be shared across many neurons, normalized neuron area is given both with and without SNG cost included.

Alongside area, the network classification accuracy and PSA neuron accuracy are also measured as G is varied. Neuron accuracy is characterized in two ways. First the PSA's root BMSE relative to the fixed-point version of the network is measured. Second, the percentage of neurons with correct[3] activations are reported since the accuracy of the PSA only matters insofar as the binarized neuron has a correct output of either $+1$ or -1. For example, a neuron with very high PSA error could still output a correct activation of, say, $+1$. In that case, the high PSA error turns out to be inconsequential. Likewise, a neuron with very low PSA error could have an incorrect neuron activation of, say, -1. In that case, the low PSA error is consequential since the neuron's activation flipped compared to a fixed-point implementation. In general, higher PSA error will result in a higher likelihood of incorrect neuron activation value. Then, in turn, more neurons with incorrect activations will increase the likelihood of network image mis-classification.

Table 5.2 shows the experimental results. The PSA root BMSE increases roughly in proportion with group size G. This increasing PSA error is reflected in the decreasing accuracy of neuron activation values. However, the network is robust to some neuron activation inaccuracy. Compared

[3]Correct with respect to a non stochastic version of the BNN.

Group size, G	**PSA root BMSE**	**First layer neuron activation accuracy**	**Classification accuracy**	**Norm. area w/ SNGs**	**Norm. area w/o SNGs**
1 (APC)	1.83	95.72%	87.26%	1.00	1.00
2	2.23	93.99%	87.86%	0.73	0.74
4	3.93	89.10%	87.00%	0.56	0.50
8	7.43	81.50%	85.66%	0.47	0.35
16	13.26	71.08%	84.22%	0.45	0.24

Table 5.2: PSA-Based Binarized Neural Network Performance

to using an APC ($G = 1$), the neuron activation inaccuracy is up to 6% higher for $G = 2$ or 4, but the overall image classification accuracy varies only slightly. Using a PSA with $G = 4$ in place of an APC reduces the neuron area by half while only reducing image classification accuracy by 0.26%. For larger group sizes like $G = 8$ and $G = 16$, the classification accuracy drops by 1.6% and 3% in exchange for further area savings. Group sizes beyond $G = 16$ lead to poor results since the SN length is only 16 bits.[4] A hybrid SC-BNN network that employs pure CeMux adders in the first layer is also evaluated. The network's classification accuracy is only 25% because 16-bit SN length is very short relative to the number of neuron inputs (784). In contrast, the PSA avoids inaccuracy problems by employing several small mux adders which operate in parallel to sample multiple SNs each clock cycle.

5.5 Summary

SC offers the promise of low-cost large-scale weighted adders. However, traditional mux adders are inaccurate while alternative adders like APCs are costly. We first addressed mux inaccuracy by introducing a novel mux adder named CeMux which, for an ECG filtering application, was around 9 times more accurate than conventional mux adders while also occupying up to 75% less area. CeMux uses correlation-related optimizations to improve its accuracy and area. These optimizations were directly inspired by our error modeling work presented in Chapters 3 and 4. Thus, statistical models like those not only improve circuit understanding, but also lead to tangible design improvements like CeMux.

When used for ECG filtering, CeMux was demonstrated to be more accurate than an APC. This result underpins another important conclusion from Chapters 3 and 4: an application's input value distribution (IVD) can drastically affect a circuit's accuracy performance. In this case, the ECG

[4]Recall that group size G is the size of the largest mux adder used in the PSA. When G is greater than the SN length, some mux input SNs are never sampled which usually leads to high inaccuracy.

task's IVD was particularly favorable for CeMux. Compared to a random baseline test, CeMux's error was up to 130 times lower on the ECG task. Meanwhile, the APC suffered from correlation-related error which we addressed with an inversion-based decorrelation technique presented in [14]. However, even after decorrelation, the APC's error was still up to 3.8 times higher than CeMux's error for the ECG task. These results are atypical, but now can be understood by using the error framework and statistical models of Chapters 3 and 4.

Despite its success with ECG filtering, CeMux does not perform well with neural networks where the number of summands is several hundred or thousand. Compared to an APC, CeMux needs to use a bitstream length that is significantly longer to reach similar network classification accuracy. Although APCs are more efficient than CeMux for this task, APCs tend to dominate neuron area which motivated our parallel sampling adder (PSA) design. PSAs combine mux and APC adders into a single flexible design framework that can trade area for accuracy depending on the application needs. PSAs generalize mux and APC adders and we showed how they can be used to attain a 50% reduction in SC neuron area without significant classification accuracy loss.

CHAPTER 6

Probability Conversion Circuit Design

Although computing with SNs is cheap, encoding non-stochastic data into SNs is very costly as SNGs often account for 70% or more of the total circuit area [61]. Addressing the high overhead of SNGs is an important area of ongoing research. This chapter investigates how the design of PCCs, a core SNG component, influences circuit area, accuracy and correlation. PCC area varies by as much as 60% between designs and we demonstrate that existing low-area PCC types cannot consistently generate highly correlated SNs. To address correlation-area limitations, we introduce multiplexer-majority chains (MMCs), a new PCC design framework that has an exploitable area-correlation trade-off. The work in this chapter is partly published in [15] and the remainder is being prepared for publication. For ease of reference, Table 6.1 summarizes the abbreviations used in this chapter.

Abbreviation	**Meaning**	**Note**
SNG	Stochastic number generator	Used to create SNs. Consists of an RNS and PCC.
RNS	Random number source	Supplies necessary pseudo-randomness for SN generation.
PCC	Probability conversion circuit	Converts the RNS output into an SN with specified probability.
CMP	Comparator	The most common PCC type.
WBG	Weighted binary generator	Another widely used PCC type.
MMC	Multiplexer-majority chain	Our novel PCC type.

Table 6.1: Major SNG-related Abbreviations

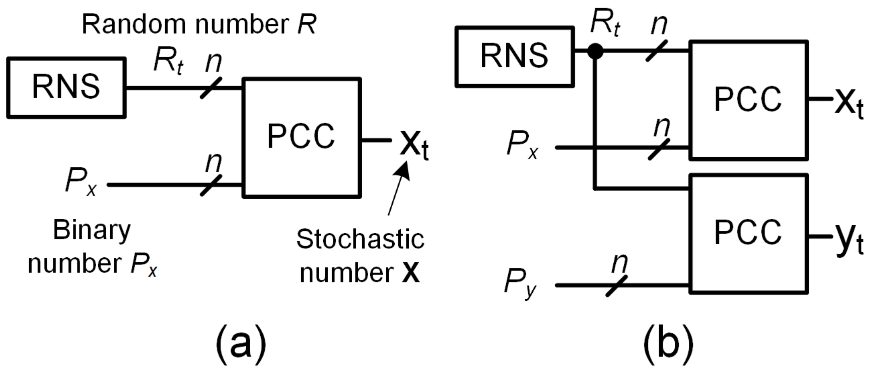


Figure 6.1: SNG design: (a) generic SNG design; (b) SNGs with shared RNS.

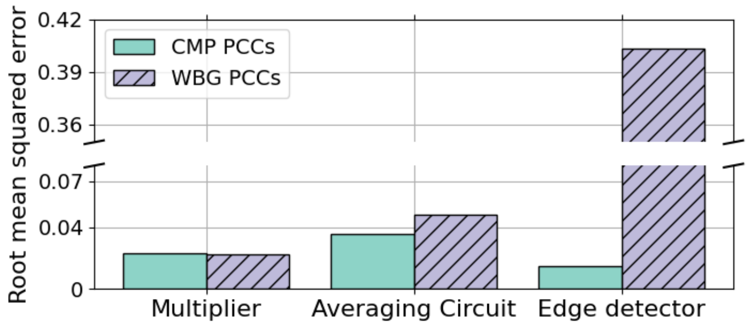


Figure 6.2: Comparison of output error for various circuit types with differing PCCs.

6.1 RNS Sharing and PCCs

A typical SNG consists of an RNS and a PCC as shown in Figure 6.1a. The RNS supplies a sequence of random numbers which the PCC converts one-by-one to a probabilistic sequence of bits forming an SN $\mathbf{X}$. Partly in response to high SNG area, there has been growing interest in stochastic circuits that share a single RNS across two or more SNGs as in Figure 6.1b. Examples are found in image processing [1, 19], neural networks [2, 31] and our CeMux design [13]. RNS-sharing saves area, but correlates the input SNs which often causes inaccuracies [23, 51], but other times is beneficial as in CeMux [3, 13]. In either case, optimizing PCC design can greatly reduce area since each distinct SN still requires its own PCC when RNS-sharing is used. For example, replacing traditional comparator (CMP) PCCs with weighted binary generators (WBGs) reduces overall circuit area by 50-60% in SC digital filter designs [13, 97].

In addition to circuit size, PCC design can also significantly affect circuit accuracy, a fact that has been noticed, but has received little attention [13, 80]. For example, Figure 6.2 shows the error level for three diverse circuit types [3, 13]. In each case, the average output error is measured when using either CMP or WBG PCCs. As can be seen, the PCC type's impact on accuracy varies

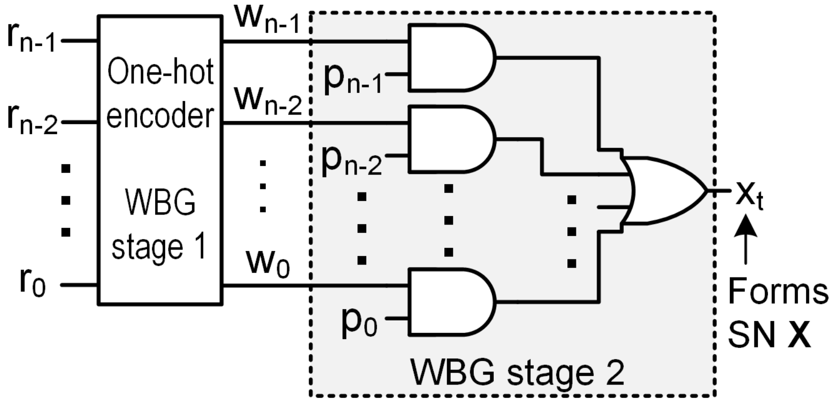


Figure 6.3: Weighted binary generator PCC design.

considerably. A key motivation of our PCC analysis is to explain behavior like that depicted in Figure 6.2 and show when and how PCC choice influences accuracy and area.

Before exploring further, we develop a formal definition for PCCs. An n-bit PCC has an n-bit random input R_t and n-bit value input P_x. The PCC output is a generated SN bit x_t with

$$\mathbb{P}(x_t = 1) = P_x \tag{6.1}$$

For example, CMP that outputs $x_t = R_t < P_x$ is a common PCC type. Since R_t is uniformly distributed, $\mathbb{P}(x_t = 1) = \mathbb{P}(R_t < P_x) = P_x$ confirming that the CMP satisfies Equation (6.1) and is therefore a valid PCC. Another PCC type is the weighted binary generator (WBG) [41] whose two-stage implementation is illustrated in Figure 6.3. The WBG's first stage first creates a one-hot encoding $w_{n-1}w_{n-2}...w_0$ which then determines which bit of $P_x = p_{n-1}p_{n-2}...p_0$ is propagated to the output. Overall, the WBG also satisfies the defining PCC equation (6.1) and is therefore a valid PCC type [15, 41].

6.2 Multiplexer-Majority Chains

This section introduces a novel and flexible PCC design style based on systematically combining mux and maj gates. The new design has a useful area-correlation trade-off and is helpful in analyzing the WBG and CMP PCCs. Figure 6.4a shows a 5-bit version of our proposed multiplexer-majority chain (MMC) framework for PCCs. Like all 5-bit PCCs, an MMC has a random input $R_t = r_4r_3r_2r_1r_0$ and a value input $P_x = p_4p_3p_2p_1p_0$; it outputs a single SN bit x_t. Each M-block

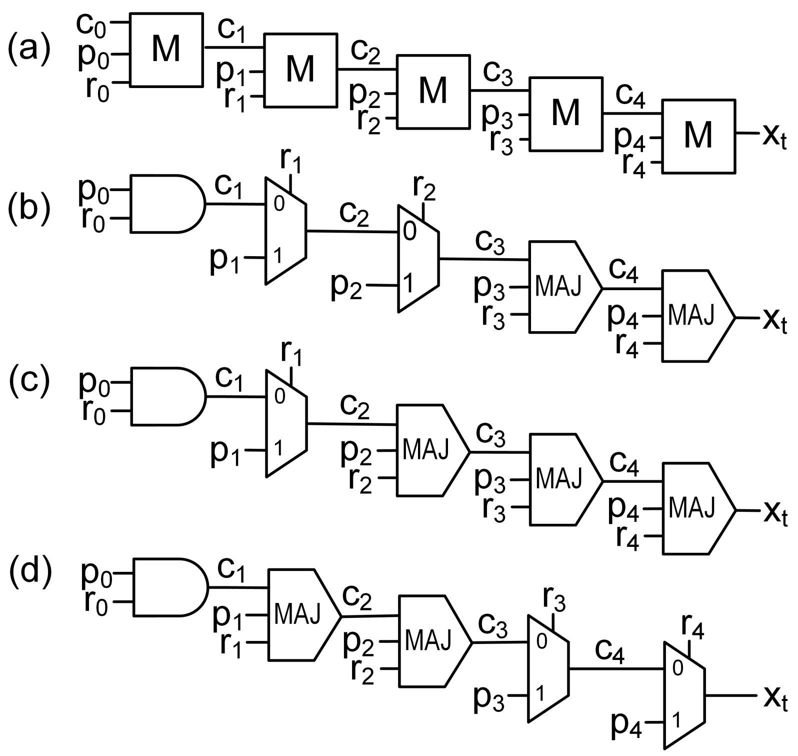


Figure 6.4: 5-bit MMC PCC design framework: (a) generic design; (b-c) MMC examples; (d) reversed MMC example.

in the MMC performs the same stochastic operation:

$$\mathbb{P}(c_{i+1} = 1) = \frac{1}{2}(p_i + \mathbb{P}(c_i = 1)) \tag{6.2}$$

The chain's output $x_t = c_5$ is seen to satisfy the PCC equation (6.1) by recursively applying Equation (6.2) and setting $c_0 = 0$. In particular,

$$\mathbb{P}(x_t = 1) = \frac{1}{2}(p_4 + \frac{1}{2}(p_3 + \frac{1}{2}(p_2 + \frac{1}{2}(p_1 + \frac{1}{2}(p_0 + 0))))) \tag{6.3}$$

when expanded yields

$$\mathbb{P}(x_t = 1) = \frac{1}{2}p_4 + \frac{1}{4}p_3 + \frac{1}{8}p_2 + \frac{1}{16}p_1 + \frac{1}{32}p_0 = P_x \tag{6.4}$$

proving that the defining PCC equation (6.1) is satisfied and the MMC is a valid PCC.

The stochastic operation (Equation (6.2)) of each M-block is equivalent to a basic SN scaled addition operation. In a similar vein as SN addition, each M-block can be implemented by either

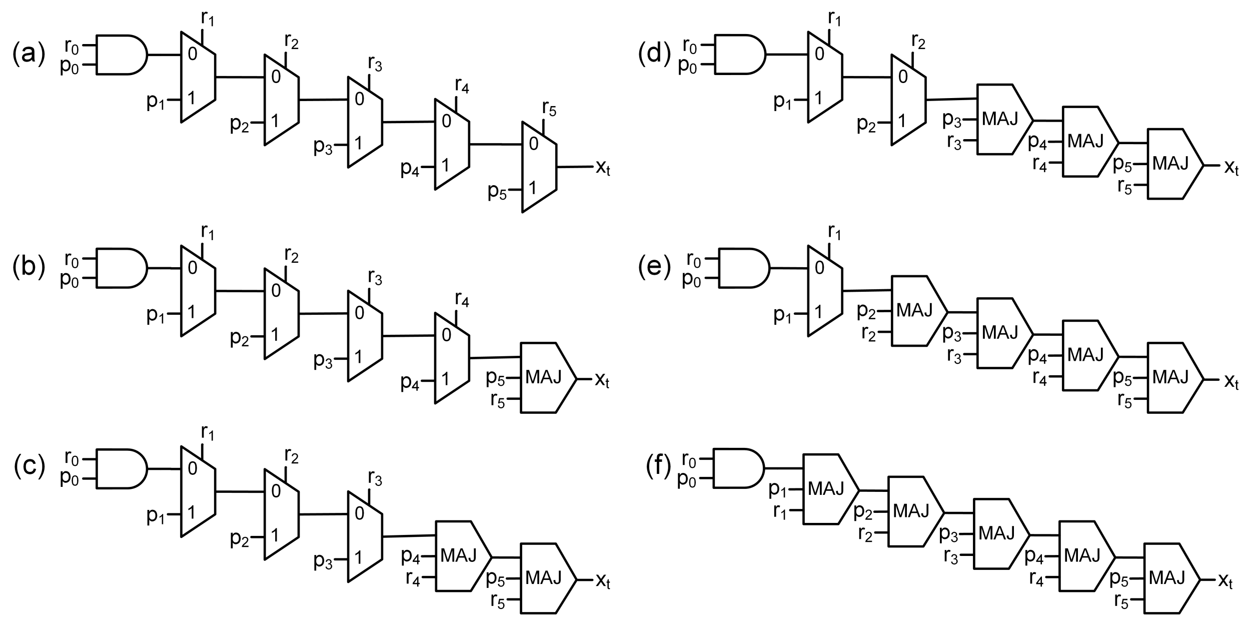


Figure 6.5: All 6-bit MMCs with various γ. (a) $\gamma = 0$ (equivalent to WBG); (b) $\gamma = 1/5$; (c) $\gamma = 2/5$; (d) $\gamma = 3/5$; (e) $\gamma = 4/5$; (f) $\gamma = 1$ (similar to CMP).

a mux or a maj gate [24].[1] For example, Figure 6.4b is an MMC that uses two mux and two maj gates while Figure 6.4c is an MMC that uses use one mux and three maj gates. Figures 6.4b, 6.4c, and 6.4d also explicitly show that the leftmost M-block always simplifies to an AND gate because $c_0 = 0$.

Any combination of mux and maj gates can be assigned to the M-blocks in Figure 6.4a to produce a valid PCC. However, a special restriction imposed on MMCs is that the chain must be all mux gates followed by as maj gates like in Figures 6.4b and 6.4c. Figure 6.4d is a valid PCC, but is not an MMC because a maj gates precede the mux gates in the chain. The advantage of MMCs over other possible designs is that MMCs have a useful area-correlation control trade-off as explained in the following section.

For a given PCC bitwidth n, there are n distinct MMCs. For example, Figure 6.5 shows all 6-bit MMCs. A specific MMC can be identified by the number of maj gates it contains, $k \in \{0, 1, \ldots, n-1\}$. An MMC's k parameter can be normalized to give $\gamma = k/(n-1)$ which is the proportion of maj gates in the MMC. Characterizing an MMC in terms of γ is particularly insightful because MMCs with different bitwidth n, but similar γ will have similar correlation properties.

When $\gamma = 0$, each M-block in an MMC is implemented as a mux gate as in Figure 6.5a. The

[1]Maj gates implement scaled addition like mux gates, but are larger than mux gates. When used for SN addition, maj gates seem to offer no advantage over multiplexers so they are rarely used in practice.

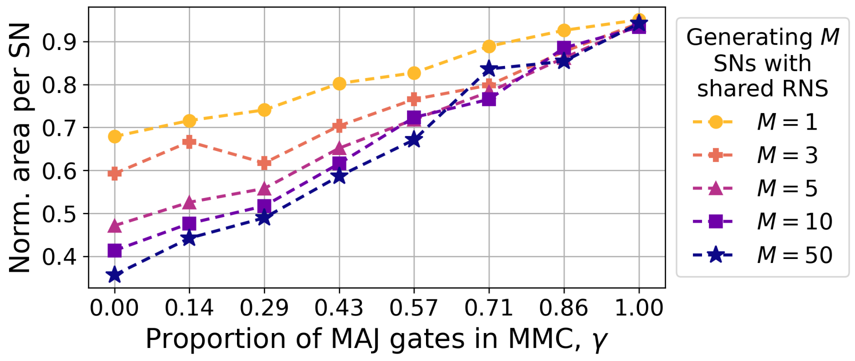


Figure 6.6: Synthesized 8-bit MMC area per SN. Area is normalized by dividing by the PCC area of one CMP-type PCC.

resulting PCC is a chain of mux gates which has appeared before [18] and is logically equivalent to a WBG. Thus, the WBG is a special case of an MMC. Likewise, when $\gamma = 1$, each M-block in an MMC is implemented as a maj gate as in Figure 6.5f. The resulting maj chain design has also appeared before [90] but is rarely used. MMCs containing both mux and maj gates like those of Figures 6.4b-c and 6.5b-e are novel designs with $0 < \gamma < 1$.

Interestingly, a chain of n maj gates forms the carry (or borrow) logic of an n-bit ripple subtractor and can be shown to compute $\bar{R}_t < P_x$ where $\bar{R}_t = \bar{r}_{n-1}\bar{r}_{n-2}...\bar{r}_0$. Since $\mathbb{P}(r_i = 1) = P(r_i = 0) = 0.5$, the inversions on $\bar{R}_t$'s bits can be absorbed into the RNS allowing us to henceforth consider a chain of maj gates as implementing a standard CMP that computes $R_t < P_x$. Thus, both WBGs and CMPs are special cases of MMCs where MMCs with $\gamma = 0$ are logically equivalent to WBGs and MMCs with $\gamma = 1$ behave like CMPs. The MMC framework also generalizes to novel PCC designs like those in Figures 6.5b-e which are implemented with a mix of mux and maj gates.

6.3 PCC Design Trade-offs

This section discusses the useful area-correlation trade-off supported by the MMC design.

6.3.1 Area Analysis

SNGs often dominate the area of a stochastic circuit [13, 61]. Thus, minimizing PCC cost is crucial to meeting SC's promise of low-cost, yet computationally powerful circuits. Here, we determine the relationship between PCC design and area by using Synopsys Design Compiler (DC) with the Nangate 45nm cell library [85] to synthesize MMCs with different proportions of maj gates γ. The analysis includes the CMP and WBG designs since they are special cases of the MMC. Synopsys

DC is given a Verilog description of the MMC's logic and the final synthesized area is reported. The objective of this area analysis is to characterize the PCC area based on its Boolean function; the objective is not to compare chain implementations against non-chain implementations of PCCs.

The $M = 1$ curve in Figure 6.6 shows that a single 8-bit MMC's area increases linearly with γ. When $\gamma = 1$, the MMC is like a CMP and area is at its highest. When $\gamma = 0$, the MMC is a WBG, and has an area 28% lower than that of a CMP. We also synthesized 6-bit through 12-bit MMCs and found similar results (which are omitted here for brevity).

When generating two or more SNs with a shared RNS as in Figure 6.1b, considerable area can be saved by sharing part of the WBG alongside the shared RNS [96, 97]. Consequently, the area cost per SN is lower when WBGs are used to generate SNs with a shared RNS than when WBGs are used with separate RNSs. In contrast to WBGs, no portion of a CMP can be shared alongside a shared RNS implying CMP area efficiency does not improve with RNS-sharing. Here, we extend the work of [97] by characterizing the per-SN area efficiency of MMCs.

Figure 6.6 shows the area-per-SN of using MMCs to generate $M = 3, 5, 10$ or 50 SNs with a shared RNS. For each M, the MMC area increases linearly with γ as in the single $M = 1$ SN case, but the area efficiency is much better. For example, when $M = 50$ the area-per-SN for WBGs is 62% lower than for CMPs. This 62% difference is much better than the 28% difference noted in the single SN case, highlighting the improved area efficiency of WBGs. When $0 < \gamma < 1$, the area-per-SN of the MMCs is also lower than the single $M = 1$ SN case. Thus, like WBGs, MMCs consisting of mux and maj gates are also more area efficient when using RNS-sharing, especially when γ is small. Note that M in Figure 6.6 is determined by the application, i.e., how many input SNs that share an RNS are needed.

6.3.2 Correlation Analysis

Recent work has demonstrated that correlation amongst input SNs can sometimes drastically change a circuit's function [3, 19] or improve its accuracy [13]. Owing to the importance of correlation, we investigate the ability of different PCCs to generate correlated SNs. The most frequent correlation level required by correlation exploiting designs is a pairwise SCC of $+1$ between all or most input SNs. CeMux is an example of such a design. The correlated SN generator in Figure 6.1b is often used to generate SNs with $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = 1$, however the actual SCC of the generated SNs can vary wildly depending on the PCC used, as we show next.

Figure 6.7 shows $\mathrm{SCC}(\mathbf{X}, \mathbf{Y})$ as a function of P_x, P_y when using a correlated SN generator. Figure 6.7h corresponds to using CMP PCCs and shows $\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = 1$ for all P_x, P_y as desired. In contrast, Figure 6.7a corresponds to using WBGs and shows that SCC($\mathbf{X}$,$\mathbf{Y}$) varies greatly with P_x, P_y and that SCC($\mathbf{X}$,$\mathbf{Y}$) often takes negative values which is antithetical to the goal of

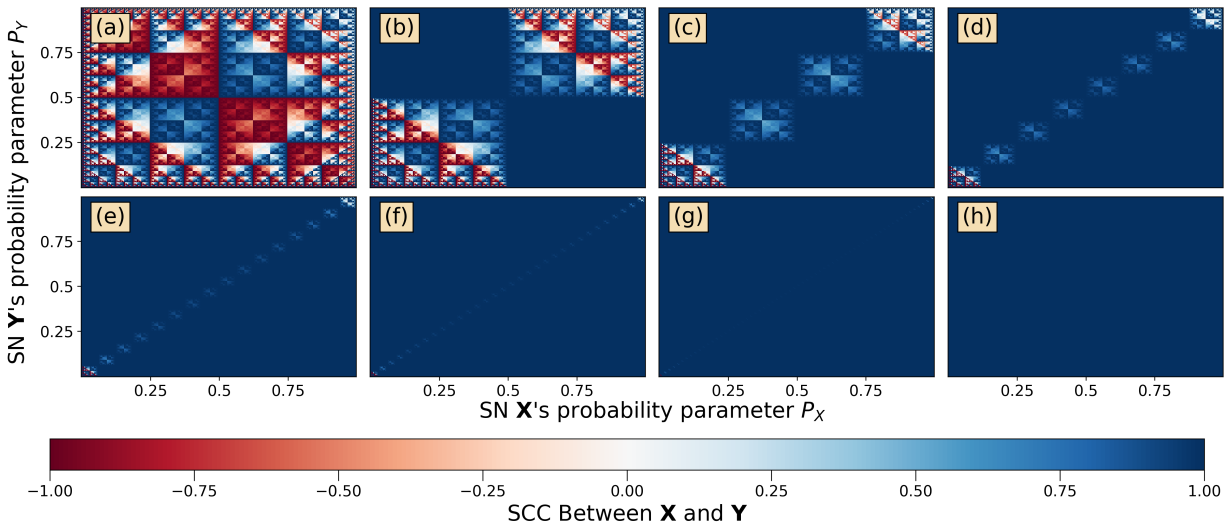


Figure 6.7: SCC of SNs generated with a shared RNS and MMCs with (a) $\gamma = 0$ (WBG); (b) $\gamma = 1/7$; (c) $\gamma = 2/7$; (d) $\gamma = 3/7$; (e) $\gamma = 4/7$; (f) $\gamma = 5/7$; (g) $\gamma = 6/7$; (h) $\gamma = 1$ (CMP)

$\mathrm{SCC}(\mathbf{X}, \mathbf{Y}) = +1$. Thus, although WBGs are very area efficient at producing SNs with a shared RNS (Figure 6.6), WBGs are unable to consistently generate maximally correlated SNs. The WBG's failure to generate maximally correlated SNs implies that WBGs are not suitable for use in correlation-reliant designs and is a key conclusion of our analysis.

Next, Figures 6.7b to 6.7g show how $\mathrm{SCC}(\mathbf{X}, \mathbf{Y})$ varies with P_x, P_y, for an 8-bit MMC with other γ values. When $\gamma = 1/7$, Figure 6.7b shows that the overall correlation is much higher than in the WBG case although the proportion of maj gates γ has only increased from 0 to 1/7. When γ is increased further to $\gamma = 4/7$, Figure 6.7e shows that SCC is +1 or close to +1 for many more values of P_x, P_y. Thus, novel MMC designs where $0 < \gamma < 1$ generate significantly more correlated SNs than WBGs, but at a lower area cost than CMPs. The correlation trend in Figure 6.7 is summarized by calculating the average SCC as a function of an MMC's parameter γ.

$$\mathrm{SCC}_{\mathrm{avg}}(\gamma) = \int_0^1 \int_0^1 f_{x,y}(p_x, p_y)\, \mathrm{SCC}(\mathbf{X}, \mathbf{Y} | p_X, p_Y, \gamma)\, dp_x\, dp_y \tag{6.5}$$

where $f_{x,y}$ is the input value distribution for P_x, P_y and $\mathrm{SCC}(\mathbf{X}, \mathbf{Y} | P_x, P_y, \gamma)$ is the SCC between $\mathbf{X}$ and $\mathbf{Y}$ given P_x, P_y and γ. Note that $\mathrm{SCC}(\mathbf{X}, \mathbf{Y} | P_x, P_y, \gamma)$ is visualized in Figure 6.7 for various γ values.

Figure 6.8 plots $\mathrm{SCC}_{\mathrm{avg}}$ for the correlated SN generator in the general case where P_x and P_y are uniformly and independently distributed, $f_{x,y} = 1$. When $\gamma = 0$, the MMC is equivalent to a WBG and the average SCC is nearly 0. As γ increases, the average SCC quickly rises until it hits a maximum value 1 when $\gamma = 1$ and the MMC is similar to a CMP. The reason for the rapid

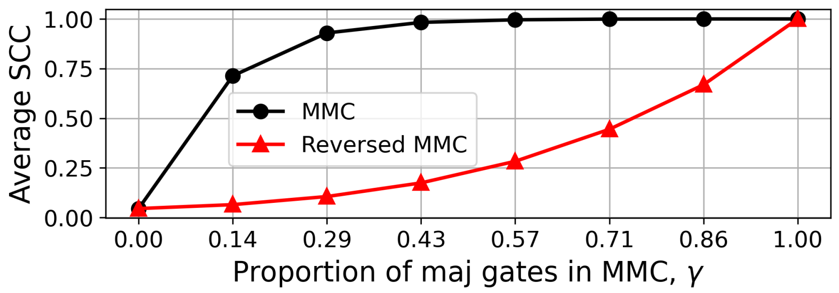


Figure 6.8: Average SCC when using 8-bit MMCs or reversed MMCs in a correlated SN generator.

growth of correlation in Figure 6.8 is that the maj gates always act on the MSBs of the MMCs' value inputs P_x and P_y. Consequently, the maj gates have significant influence on the MMC output and cause the correlation level to reflect that of a CMP more so than that of a WBG.

The desired rapid rise in correlation magnitude in Figure 6.8 is precisely why MMCs are singled out from the space of all possible adder chain designs. For example, consider a "reversed MMC" which is similar to a plain MMC, but with the reverse ordering of mux and maj gates. In other words, the mux gates in a reversed MMC act on the MSBs of P_x or P_y while maj gates act on the LSBs; Figure 6.4d is an example of a reversed MMC. For reversed MMCs, the correlation level per maj gate is much lower than for regular MMCs as shown in Figure 6.8.

Overall, MMCs offer a very favorable area-correlation trade-off when generating correlated or anti-correlated SNs. The trade-off is controlled by γ, the number of maj gates in the MMC. As γ increases Figure 6.6 shows that the synthesized area of the MMC increases linearly while Figure 6.8 shows that the correlation control rises faster than linearly. Our analysis showed that WBGs (i.e., MMCs with $\gamma = 0$) have very poor correlation control but slightly increasing γ beyond 0 to, for example $\gamma = 1/7$, greatly increases correlation levels of generated SNs for very little area overhead.

6.4 Case Studies

The foregoing analysis shows that PCC designs vary in the amount of area they occupy and in their ability to generate correlated SNs. When RNS-sharing is not used, the input SNs are uncorrelated and the main considerations for PCC design are area and power. In that case, MMCs with $\gamma = 0$ or, equivalently, WBGs are the preferred PCC type since they are the most hardware efficient (Figure 6.6). However, when RNS-sharing is used, the PCC's correlation properties can impact circuit accuracy to various degrees. The relationship between PCC design and accuracy depends on why the circuit employs RNS-sharing.

Here, four categories of RNS-sharing circuits are introduced and summarized in Table 6.2:

	PCC impact on			
Correlation category	Bias	Variance	Overall accuracy	Example design
Agnostic	None	None	None	BNN Neuron [45]
Reliant	High	Moderate	High	Edge detector [5]
Optional	None	Moderate	Moderate	CeMux [13]
Averse	High	Moderate	High	AND Multiplier [81]

Table 6.2: RNS-Sharing Circuit Categorization

correlation-agnostic, correlation-reliant, correlation-optional and correlation-averse. Correlation-agnostic circuits are designs whose accuracy is unaffected by correlation. In contrast, correlation-reliant circuits use correlation to change gate function. These circuits require correlated inputs to avoid high error. Next, correlation-optional designs are those like CeMux that use correlation to improve accuracy, but not alter gate function. Finally, correlation-averse designs are those that share an RNS, but require uncorrelated inputs to avoid high error. Although such circuits could use separate RNSs to achieve uncorrelated inputs, RNS-sharing is used to save area and power, usually at the expense of some accuracy [51, 80].

The RNS-sharing circuit categorization of Table 6.2 is useful because the impact of PCC design on accuracy is consistent within a category, but differs between categories as demonstrated by the following case studies. These applications represent a variety of high-performing SC designs that use RNS-sharing for applications in neural networks [45, 66], digital filtering [13], and image processing [19]. The area and accuracy of the four representative designs are measured as PCC type is varied. The results will demonstrate how PCC design impacts these important metrics for different types of RNS-sharing circuits.

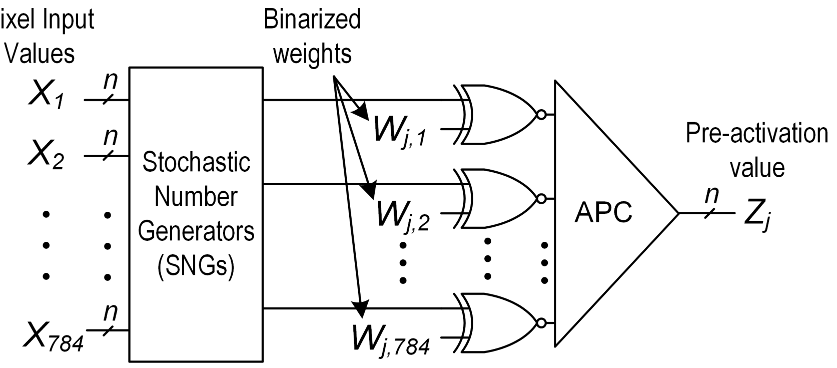


Figure 6.9: SC implementation of a first-layer BNN neuron [45].

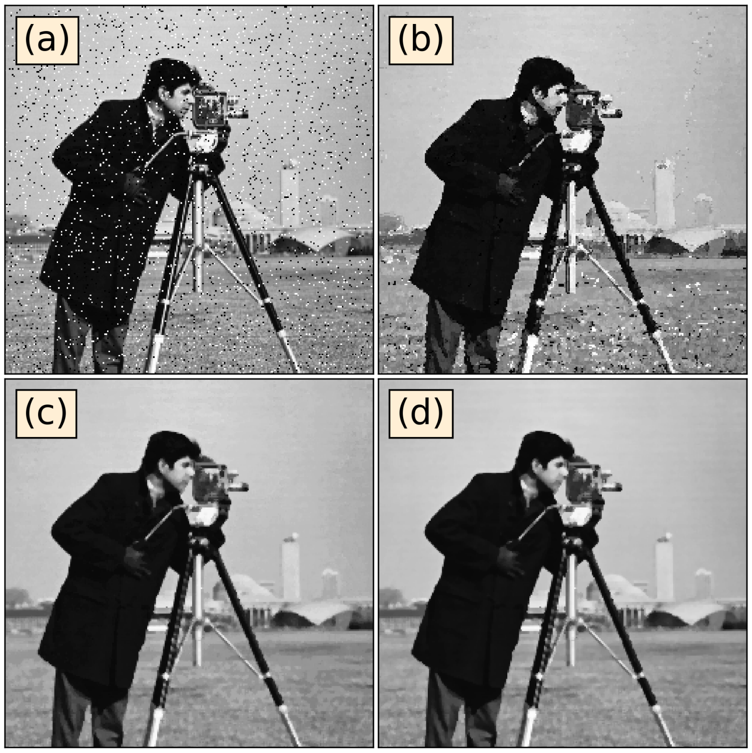


Figure 6.10: Median filter application. (a) Image corrupted with salt-and-pepper noise. SC median filter output when PCCs are: (b) WBGs; (c) MMCs with $\gamma = 4/7$; (d) CMPs.

6.4.1 Applications

The first application is image classification with a binarized neural network (BNN). This application was also explored earlier when analyzing the PSA design in Section 5.4. An SC implementation of a BNN neuron is shown in Figure 6.9. Since weights in a BNN are binarized and held fixed at logical 1 (to represent $+1$) or logical 0 (to represent -1), correlation amongst the pixel input SNs does not affect accuracy. Thus, the BNN neuron design is an example of a correlation-agnostic circuit where a single RNS is shared amongst all SNGs. For our experiments, input images are derived from the Fashion-MNIST image classification benchmark [95] while input weights are from a trained BNN consisting of two hidden layers with 1,024 neurons each [45]. The error is measured between the multiplier outputs in Figure 6.9 and target output $W_{j,i}X_i$.

The second application is electrocardiogram denoising with finite impulse response (FIR) filters. This application was also explored earlier when analyzing CeMux and other adders in Section 5.2. In that experiment, CeMux was shown to be the most accurate and smallest FIR filter design. Here, we run a similar experiment and study the extent to which MMCs can be used to further improve CeMux's area efficiency. Since CeMux uses correlation to reduce variance, but not change gate function, it is a correlation-optional design. For the following experiments, CeMux's target

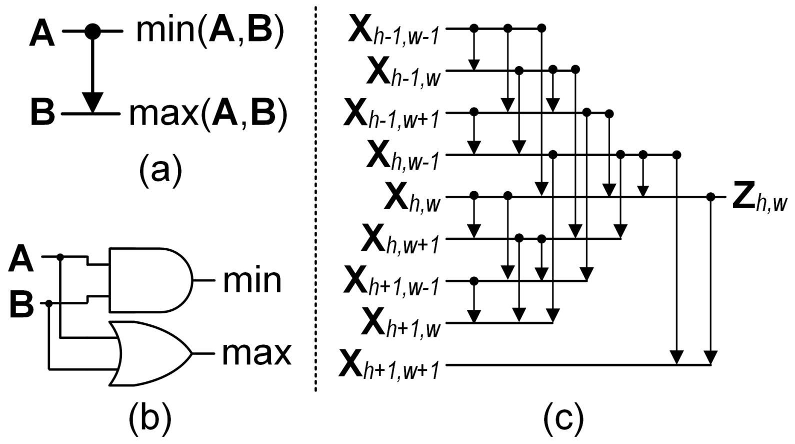


Figure 6.11: SC median filter design: (a) compare and swap block symbol; (b) SC compare and swap block implementation; (c) median filter built from 19 compare and swap blocks.

input is derived from a noisy ECG signal [36, 70] and the error is measured between CeMux's output and the target filtered output value.

The third application is image denoising with median filters which are effective at filtering out impulse noise types. For example, the image in Figure 6.10a that is corrupted by salt-and-pepper noise can be mostly recovered by convolving the noisy image with a 3×3 median filter as shown by the filtered image in Figure 6.10d. A 3×3 median filter replaces each noisy image pixel $X_{h,w}$ with the median value of itself and the surrounding 8 noisy pixels

$$Z_{h,w} = \text{Median}(\{\, X_{h-i,w-j} \mid -1 \leq i, j, \leq 1 \,\}) \tag{6.6}$$

where $Z_{h,w}$ is the filter's output pixel. In SC, a median filter can be implemented very efficiently by using AND and OR gates to implement a series of MIN and MAX operations on correlated SNs. The arrangement of the AND and OR gates that constitutes the SN median filter design is shown in Figure 6.11 [7, 19]. SC median filters require that their input SNs be maximally correlated with a pairwise SCC of +1 to alter the functions of AND and OR to MIN and MAX, respectively. Thus, the median filter is a correlation-reliant design.

For out experiments, the SC median filter's target inputs are pixel values of grayscale test images from the MATLAB image processing toolbox. The images are corrupted with random salt-and-pepper noise where each image pixel is assigned a 5% chance of becoming corrupted to all-black or all-white. The cameraman photo of Figure 6.10 is an example of one of the test images. The target output of the median filter is $Z_{h,w}$ (Equation (6.6)). In total, the ten images constitute over 1 million kernel placements and thus yield statistically significant accuracy estimates.

The fourth application is image edge detection. The Roberts Cross edge detector (RCED) used

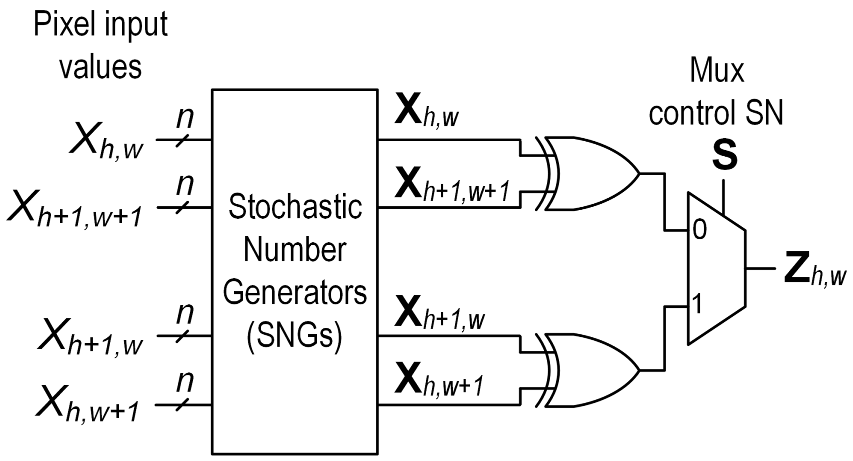


Figure 6.12: SC Roberts cross edge detector design [3].

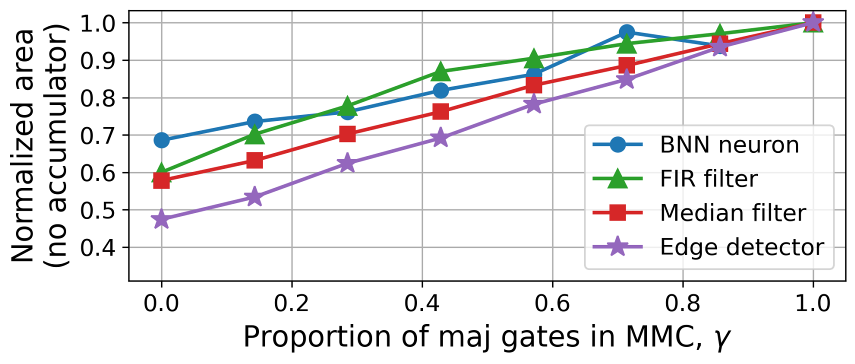
in image processing can be implemented efficiently with SC using the correlation-exploiting circuit of Figure 6.12 [3]. The SC RCED implements

$$Z_{h,w} = \frac{1}{2}\big(|X_{h,w} - X_{h+1,w+1}| + |X_{h+1,w} - X_{h,w+1}|\big) \tag{6.7}$$

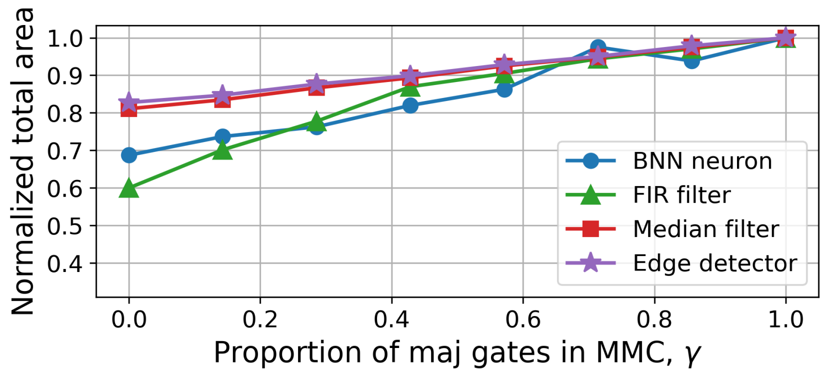
where $Z_{h,w}$ is a pixel of the edge detector output and $X_{h,w}$ is a pixel of the input image. To accurately implement Equation (6.7), the SC RCED requires that its inputs are correlated with an SCC of +1 to alter the function of XOR gates which implement the absolute differences in Equation (6.7). Thus, the edge detector is also a correlation-reliant design. For our experiments, the circuit's target inputs are derived from non-corrupted versions of the same 10 MATLAB image processing toolbox images used for the median filter experiments; the target output for error measurements is $Z_{h,w}$ defined in Equation (6.7).

6.4.2 Experiments and Results

For our evaluation, the four application circuits follow a similar structure. The SNG precision is set to 8 bits while the SN length is 256 bits. All designs use a single RNS since they benefit from RNS-sharing. A Sobol sequence generator is used as the RNS because it gives the best accuracy and has low cost as it can be implemented as the bitwise reverse state of a counter [61]. All designs employ 8-bit MMC PCCs with γ proportion of maj gates and the focus of the following experiments is to determine how overall circuit area and accuracy varies with γ. The widely-used WBG and CMP designs are included in the analysis since they are special cases of the MMC design with $\gamma = 0$ and $\gamma = 1$, respectively. Area is quantified by synthesizing designs using Synopsys DC with the Nangate 45nm cell library [85] while accuracy is quantified with BMSE estimated by bitstream



(a)



(b)

Figure 6.13: MMC area trade-off for various applications implemented in an: (a) open-ended SC system; (b) end-to-end SC system.

simulation.

The relationship between circuit area and MMC γ is investigated for two cases. The first case is an open-ended SC system where the output SN is kept as a bitstream so that it can be routed to another SC design. The second case is an end-to-end SC system where the output SN feeds into an accumulator (i.e., counter) that estimates its value as a fixed-point number. The difference between these two cases is that the area of the open-ended system does not include the cost of the output counter found in the end-to-end system. Both systems include the cost of the SNGs and arithmetic circuits. For ease of comparison, area is normalized by dividing by the area of the corresponding design that employs MMCs with $\gamma = 1$.

All four designs exhibit a linear relationship between γ and circuit area. For open-ended systems, Figure 6.13a illustrates that each design is 30% to 50% smaller when $\gamma = 0$ relative to $\gamma = 1$ For end-to-end SC systems, Figure 6.13b shows that the area reduction at $\gamma = 0$ relative to $\gamma = 1$ is now only 20% for the edge detector and median filter. This 20% reduction is less than the 40%

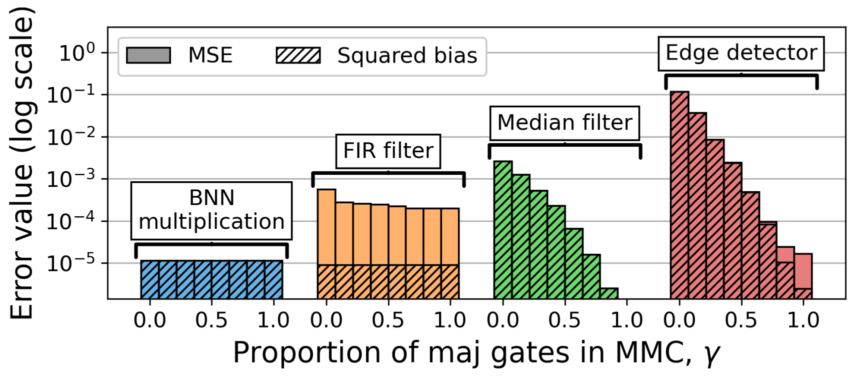
reduction in the open-ended system because the end-to-end system's output accumulator occupies a considerable portion of these relatively small circuits. Consequently, PCC area has less influence on overall area. In contrast, the area trade-off is the same in both cases for the BNN neuron and FIR filter because these are large circuits where the added cost of an output accumulator hardly affects its total area.

The results of Figure 6.13 highlight two important conclusions. First, the slope of γ's linear relationship with *total* circuit area implies that PCC design has a notable influence on overall area for all four circuits. Second, for end-to-end SC systems, the PCC design influences total area more when the circuit has many inputs because PCCs occupy a higher percentage of total circuit area and because MMC area varies considerably when RNS-sharing is used as shown earlier in Figure 6.6. PCC choice should therefore be carefully considered by SC designers, especially in view of the increasing use of RNS-sharing and correlation in recent work [1, 2, 3, 7, 13, 19, 23, 31, 59, 80, 97].

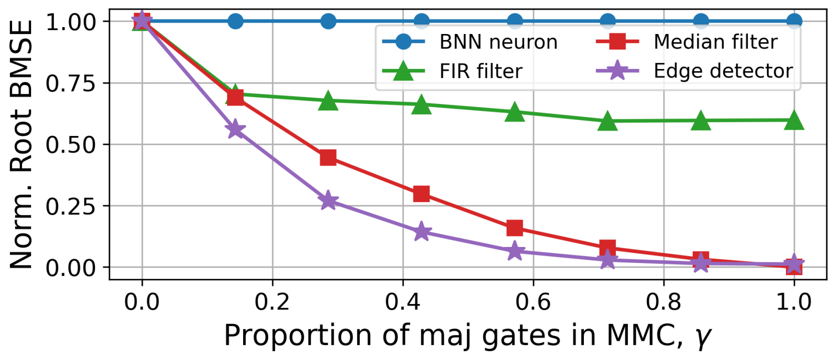
When RNS-sharing is used, PCC design influences input SN correlation which can impact circuit accuracy [23]. Figure 6.14a plots each application circuit's BMSE on a log scale while Figure 6.14b shows the normalized root BMSE on a linear scale. The PCC design has no impact on BNN multiplier accuracy, a small impact on CeMux FIR filtering accuracy and a very large effect on median filtering and edge detection accuracy. Specifically, comparing MMCs with $\gamma = 0$ to $\gamma = 1$, the BNN multiplier's BMSE is the same while the FIR filter's BMSE is 2.8 times higher for $\gamma = 0$. In contrast, the edge detector's BMSE is over 7000 times times higher for $\gamma = 0$ and the median filter's BMSE is 1000 times higher when $\gamma = 0$ compared to $\gamma = 6/7$ (the median filter has 0 BMSE when $\gamma = 1$).

The PCC design's widely varying impact on BMSE relates to our proposed categorization of RNS-sharing designs. The BNN neuron is a correlation-agnostic design so PCC design does not affect accuracy. Meanwhile, the FIR filter is a correlation-optional design where PCC design has a moderate effect on accuracy. In contrast, the median filter and edge detector are correlation-reliant designs where PCC design has a drastic impact on accuracy because correlation control is central to these designs.

The bias-variance decomposition used in our BASE framework also sheds light on how error can be understood in these designs. Recall that BMSE can be expressed as a sum of bias squared and variance where bias quantifies systematic error and variance quantifies random error. For correlation-agnostic circuits, correlation does not affect bias or variance so BMSE is unchanged. For correlation-optional circuits, correlation only affects variance which impacts MSE a moderate amount. For correlation-reliant and correlation-agnostic designs, correlation has some effect on variance, but a very large effect on bias which results in drastically high BMSE when correlation requirements are not met. Importantly, this high bias is a systematic error that cannot be mitigated by increasing SN length.



(a)



(b)

Figure 6.14: Impact of PCC design on: (a) BMSE and squared bias (log-scale); (b) root BMSE normalized by dividing the value when $\gamma = 0$.

As an example, the edge detector's expected output value is the desired Roberts cross edge detection formula (Equation (6.7)) when the pairwise SCC between all input SNs $+1$. However, if instead the pairwise SCC is 0, the expected output value is

$$Z_{h,w} = \frac{1}{2}(X_{h,w} + X_{h+1,w+1} - 2X_{h,w}X_{h+1,w+1} + X_{h+1,w} + X_{h,w+1} - 2X_{h+1,w}X_{h,w+1}) \quad (6.8)$$

which is systematically different than Equation (6.7). Ignoring quantization error for simplicity, the bias of the edge detector when the inputs have $\text{SCC} = 0$ rather than $\text{SCC} = 1$ is the difference between the desired Roberts cross formula (Equation (6.7)) and Equation (6.8). This bias is relatively large and is not improved by increasing SN length.

As part of the experiment, bias was calculated for each design by combining the methodologies of [23] and our BMSE framework [11]. The results are illustrated in Figure 6.14a. Each bar representing BMSE in Figure 6.14a is partially hatched to show the squared bias. For the BNN

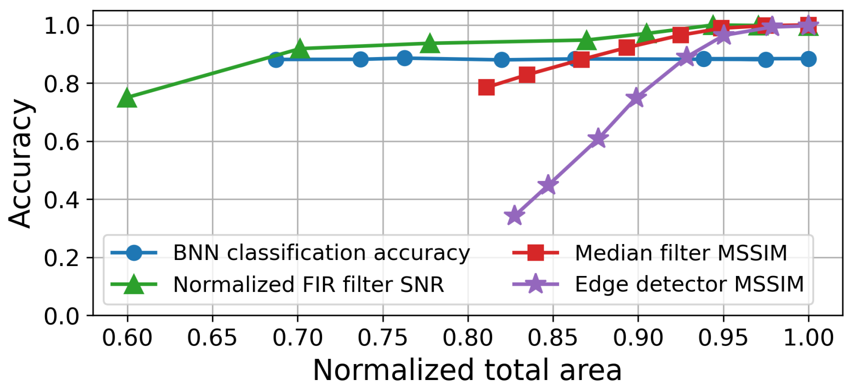


Figure 6.15: MMC accuracy-area trade-off. Each data point corresponds to using a different MMC γ where left-more points correspond to lower γ.

and FIR filter, bias is small and constant across MMC γ because correlation has no affect on bias. Bias only arises in these designs due to quantization error. In contrast, the median filter's and edge detector's bias varies greatly because these correlation-reliant designs are biased when MMC $\gamma < 1$. The bias arises because the required input correlation of $\text{SCC} = +1$ is not met.

In addition to BMSE, application-specific accuracy metrics are measured to better capture the practical performance of each design. For image classification, the overall BNN classification accuracy is tracked. For FIR filtering, the signal-to-noise ratio (SNR) is measured as

$$\text{SNR} = 10 \log_{10} \frac{\mathbb{E}[Z_t^2]}{\text{BMSE}} \tag{6.9}$$

where $\mathbb{E}[Z_t^2]$ is the average signal power and BMSE quantifies the average noise power. SNR is commonly used in signal processing and usefully puts the error in context with the target output value Z_t.

For the median filter and edge detector, the mean structural similarity index (MSSIM) [92] between the output image and target image $\mathbf{Y}$ is evaluated. MSSIM is a useful measure because two distorted images having the same MSE relative to a reference image can vary substantially in terms of perceived visual quality [92]. MSSIM was designed to address this problem by capturing the perceptual differences between images better than measures like MSE. MSSIM is based on the hypothesis that the human visual system is adapted to perceive large-scale structural features of images rather than low-level details or differences in luminance and contrast. MSSIM varies between 0 and 1 where high values indicate the two images are more visually similar.

Figure 6.15 shows the accuracy-area trade-off for each design achieved by varying the MMC's γ parameter. Curves that are more horizontal, i.e., with lower slope, indicate a more favorable accuracy-area trade-off for that design. The BNN's classification rate is a constant 89% across all area values because it is a correlation-agnostic design. Relative to using MMCs with $\gamma = 1$, using MMCs with $\gamma = 0$ leads to an area reduction of 30% and no loss in classification accuracy. For FIR filtering, using $\gamma = 0$ yields a 40% area reduction in exchange for 25% lower SNR. The trade-off becomes much better at $\gamma = 1/7$ where a 30% area reduction is achieved in exchange for only an 8% drop in SNR. The reason for the better trade-off is that MMCs with $\gamma = 1/7$ generate much more correlated SNs than MMCs with $\gamma = 0$ (Figure 6.8) and, although correlation is not maximized, it is high enough to achieve most of the variance-reduction benefits.

In the case of edge detector and median filtering, area-accuracy trade-off is less favorable than in the BNN or FIR filter case because these image processing circuits are correlation-reliant designs. Both designs can save between 5% and 10% area in exchange for sacrificing a similar percentage of MSSIM by using MMCs with $\gamma = 4/7, 5/7, 6/7$ in place of MMCs with $\gamma = 1$. In contrast to the BNN and FIR filter, the edge detector and median filter designs generally require that MMC γ be high because these designs are sensitive to even small deviations in input correlation.

6.5 Summary

MMCs are parameterized by $\gamma \in [0, 1]$, the proportion of maj gates used in the design. Section 6.3 showed that higher γ corresponds to higher area and higher correlation in the generated SNs when RNS-sharing is used. The widely used WBG and CMP PCC designs are special cases of MMCs, having $\gamma = 0$ or $\gamma = 1$, respectively. Thus, MMCs and adder chains generalize these PCC designs and offer new correlation-area trade-offs.

When RNS-sharing is used, PCC design impacts input correlation which affects accuracy to varying degrees. Table 6.2 summarizes four categories of RNS-sharing circuits that are differentiated based on how correlation affects the circuit's bias and variance. PCC design impacts overall accuracy differently for each category, but PCC design has a consistent impact on circuit accuracy within a category. In contrast, PCC impact on area is consistent for all RNS-sharing circuits and tends to depend on the number of circuit inputs.

The analysis in Section 6.4 investigated three of the four RNS sharing circuit types. The accuracy of correlation-agnostic circuits is not affected by PCC type and so these circuits should use MMCs with $\gamma = 0$ because they are the smallest PCC. Meanwhile, the accuracy of correlation-reliant circuits is extremely sensitive to PCC type. In general, these designs should use MMCs with $\gamma \geq 0.5$ to avoid high bias. The accuracy of correlation-optional circuits is somewhat sensitive to PCC type. Such designs can use MMCs with γ just above 0 to strike a balance between circuit accuracy and area. For correlation-optional and correlation-reliant circuits, higher γ can be used to improve accuracy at the cost of higher area.

Overall, PCC design has a substantial impact on circuit area, correlation and accuracy, especially when RNS-sharing is used. Poor PCC design can increase error by over 7000 times for correlation-reliant circuits while effective design can reduce total circuit area by 30% for correlation-agnostic designs. The novel MMC design framework can be used to flexibly trade area for accuracy and offers a new avenue to circuit optimization.

CHAPTER 7

Concluding Remarks

SC's probabilistic approach to computing enables its low-cost datapaths and high fault tolerance. However, its unique encoding also comes with challenges, some of which were studied and addressed in this thesis. This section summarizes our contributions and points to some promising directions for future work on SC.

7.1 Summary of Contributions

This thesis demonstrates the value of leveraging statistical models in stochastic circuit error analysis. Statistical analysis yields a better understanding of circuit behavior and complements existing simulation approaches. Insights gleaned from iterating between theoretical analysis and experimental simulation was shown to yield more accurate and smaller circuit designs for important operations like subtraction and many-input summation. Our major contributions are as follows:

1. Bayesian analysis of stochastic errors (BASE), a comprehensive framework for analyzing stochastic circuit accuracy that can account for SC's many error types.

2. New statistical models were described for analyzing circuit variance, bias, and input value distribution. Analysis led to important new insights into SN adder and generator designs.

3. Correlation-enhanced multiplexer (CeMux), a smaller and significantly more accurate mux adder that was demonstrated to have high performance for FIR filtering.

4. Parallel sampling adders (PSAs), a new large-scale adder design style that addresses challenges in many-input SN adder design and can improve the area of SC-based neural networks.

5. Multiplexer-majority chains (MMCs), a new probability conversion circuit design style that has a favorable area-correlation trade-off and provides new insights into SN generation and correlation.

In Chapter 3, we proposed BASE, a comprehensive framework that can account for the effect of SC's many errors sources. BASE is built on statistical estimation theory [54] and highlights three key quantities determine a circuit's accuracy: the input value distribution (IVD), bias, and variance. The IVD is determined by the application. Different IVDs will result in different expected circuit accuracy, highlighting the application-dependence of a design's performance. Bias quantifies the circuit's systematic error from sources like approximation, quantization, and correlation while variance quantifies the circuit's random error from sources like SN stochasticity and transient faults. Decomposing total error into bias and variance reveals how to address high error levels. For example, increasing SN length will address high variance, but not high bias.

BASE provides a framework for error analysis where its three key quantities, variance, bias, and IVD must be derived or estimated for a given circuit and application. BASE is compatible with existing simulation approaches as well as methods for variance analysis [67, 74] and for correlation error analysis [23]. However, existing analysis methods sometimes fail to accurately represent the behavior of stochastic circuits such as those with LFSRs and correlation. Chapter 4 introduced the new hypergeometric SN model which can be used to analyze the variance of such designs. Other new models were also presented, such as a Markov chain model for quantifying autocorrelation error, and a beta mixture model for representing an IVD. We demonstrated the usefulness of our models and the BASE framework for a neural network classification task in Section 4.5.

Overall Chapters 3 and 4 stress the value that statistical models bring to SC. By introducing concepts from probability theory and statistics like the hypergeometric and beta distributions, bias-variance decomposition of MSE, higher-order Markov chains, and estimator dominance, we derived new methods for stochastic circuit error analysis. Then insights from accurate statistical models led to better designs. For example, our hypergeometric SN model led to the CAM subtractor which was shown to be 39% more accurate than an existing subtractor design when tested on random input data. Chapters 5 and 6 took these insights further to address the challenges in many-input SN adder design and in SN generator design.

Chapter 5 applied the hypergeometric SN model to many-input mux adder design which led to two new design concepts, full correlation and precise sampling. Together these optimizations reduce mux variance by 67% on average when applied to random input data. Full correlation and precise sampling were combined with a Sobol generator RNS to create our CeMux adder design. CeMux is up to 12 times more accurate [13] and 30% to 75% smaller than existing SC adder when applied to ECG filtering with FIR filters. CeMux's accuracy was found to be better than an APC for ECG filtering because the application's IVD is particularly favorable for CeMux.

While CeMux performs well for FIR filtering, its accuracy or latency is poor when used in applications like neural networks. Due to mux inaccuracy, other adders like APCs are usually preferred for SC neural network design [29, 45, 66]. Although using APCs leads to highly accurate classification performance with reasonable SN lengths, APCs tend to dominate the area of an SC-neuron. In Chapter 5, we introduced PSA, a new adder circuit which combines the mux and APC adders into a single design supporting a flexible area-accuracy trade-off. Compared to using APCs, we showed that using PSAs can save half the area of a SC-based neuron without sacrificing significant network classification accuracy.

This thesis and many recent works exploit correlation to save area, change gate function, and improve accuracy [2, 7, 12, 13, 31, 49, 80]. Correlated bitstreams are usually obtained by sharing an RNS between one or more SNGs. The area savings from RNS-sharing is higher when using WBG PCCs alongside the shared RNS than when using traditional comparator PCCs [13, 96, 97]. However, in Chapter 6, we showed that using WBGs with a shared RNS leads to inconsistently correlated input bitstreams. This results in very high inaccuracy for designs that rely on correlation, but has little effect on other designs that are less correlation sensitive. Thus, we concluded that the area efficiency of WBGs can only be only exploited in some circuits like CeMux, but not others like edge detectors.

To investigate area, accuracy, and correlation trade-offs in PCC design further, we introduced MMCs in Chapter 6. MMCs unify the comparator and WBG designs into a single PCC design framework that has a favorable area-correlation trade-off. Our analysis with MMCs inspired a new classification system for RNS-sharing circuits that can be used to minimize PCC overhead while maintaining accuracy. For example, we showed that MMCs can reduce CeMux's area by 25% while lowering SNR by only 7% for an ECG filtering application. Analysis also revealed that correlation-reliant designs like the median filter and edge detector should use higher-area PCCs like the comparator that can generate highly correlated bitstreams and avoid inaccuracy. Meanwhile, correlation-agnostic designs like binarized NN multipliers should use WBGs to minimize area because the accuracy of such designs are unaffected by input correlation levels.

7.2 Future Directions

Overall, our work demonstrates the value of using statistical models to analyze stochastic circuit accuracy. Many of our designs were developed out of insights gleaned from iterating between statistical analysis and simulation. The BASE framework provides a starting point for the analysis of any stochastic circuit, but new models may be required to analyze new designs. One promising direction along these lines would be to focus on SNs derived from Sobol sequences. Our hypergeometric SN model was developed for LFSR-generated SNs and tends to overestimate the error of circuits that use Sobol sequences for SN generation. A new SN model for Sobol or other low-discrepancy SNs may reveal new ways to improve circuit design just as the hypergeometric SN model revealed insights for improving mux adder design.

Another promising direction for research is the use of SC's probabilistic encoding with emerging devices like memristors [7, 44, 56] and phase change memories [20]. Such devices are subject to random errors due to device variability, temperature fluctuations and electronics noise. Thus, SC's fault tolerant encoding is especially useful as it provides much higher robustness than conventional fixed-point format. Moreover, If the device noise is characterized, then our BASE framework could be used to analyze how device noise affects variance and overall accuracy.

Additionally, SC's small multipliers are also useful for enabling near-memory computing with emerging devices. Combining SC with memristors, for example, can minimize data movement which tends to consume a large portion of the overall energy used by a neural network [44]. Our new SC designs could possibly improve the performance of these designs further. For example, our PSA design saves a significant amount of area over using an APC for SC neuron design [14].

SC's performance on benchmark machine learning applications like MNIST, CIFAR-10, and Fashion-MNIST has steadily improved. For example, the SN length and circuit area required for accurate classification has been reduced by using correlation, Sobol sequences and new adder designs like the PSA [14, 29, 31, 62, 66]. A promising future step would be leverage these advances and target specific applications of neural networks and other algorithms on small devices like wearables, medical implants, and heart monitors. For example, SC can be used for gesture recognition using data from an inertial measurement unit [42] and our work on filtering demonstrates the potential of using SC in hearing aid design [16]. In short, more "real-world" applications of SC NNs and other designs like FIR filters would be a valuable contribution.

SC's probabilistic encoding is similar to encodings used in other areas. For example, SC's bipolar format is similar to the binarization used in binarized neural networks and our work on built on others that show SC's synergy with these networks [14, 45]. Likewise, spiking neural networks encode values using neural spikes similar to SC's unipolar format and recent work combines SC, binarized NNs and, spiking NNs into an efficient design deployed on FPGAs [35, 86]. Another related area is stack filtering which uses a 'threshold decomposition' similar to SN generation [22, 33]. Combining insights and advances in different fields could yield even further design improvements and insights.

Overall, SC's future is promising. Interest in the field has been steady growing [40] and its probabilistic encoding enables new directions for overcoming the challenges facing integrated circuit design. We hope that this thesis can help enable the development of better SC theory and new SC designs such that SC may continue to see more widespread use in the future.

APPENDIX A

Derivation of Mux Variance Formulas

Here we derive analytic expressions for mux adder variance and various adder configurations using the binomial and hypergeometric SN models. Expressions derived with the hypergeometric SN model match the simulated variance of circuits that employ LFSR SNGs but overestimate the variance of CeMux which uses a low discrepancy random number source.

Normally, a bipolar SN $\mathbf{X} = x_1, x_2, ..., x_L$ has expected value $\mathbb{E}[\hat{X}] = 2\left(\frac{1}{L}\sum_{i=1}^{L} x_i\right) - 1$ and expected bit value $\mathbb{E}[x_i] = \frac{X+1}{2}$ where $X = 2P_x - 1$ is $\mathbf{X}$'s bipolar value. In the following derivations, however, a bipolar SN's bits are defined to take values $\{-1, 1\}$ rather than $\{0, 1\}$ where -1 acts as logical 0. Consequently, a bipolar $\mathbf{X}$ has estimated value $\frac{1}{L}\sum_{i=1}^{L} x_i$ and expected bit value $\mathbb{E}[x_i] = X$, both of which match the unipolar case. Ultimately, changing the definition of bipolar bits in this way allows the following derivation to simultaneously apply to both unipolar and bipolar SC.

We assume that all input SN bits are identically distributed which is the case in both the binomial and hypergeometric SN models, but not true for low discrepancy SNs. When bits are identically distributed[1], they become statistically indistinguishable which allows one to focus on reasoning about an arbitrary bit x_i. For example, $\sum_{i=1}^{L} \mathrm{Var}(x_i)$ becomes $L\,\mathrm{Var}(x_i)$ when the x_i's are identically distributed. The following analysis also makes extensive use of the fact that the expectation operator, $\mathbb{E}[\cdot]$, is linear.

A.1 Mux Variance Decomposition

Consider a mux tree with M data input SNs, $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_M$ that have values $X_1, X_2, ..., X_M$ and length L. The mux tree has a select input S which is not treated as an SN, but rather as a stream of identically distributed random words. S's value during clock cycle j, S_j, determines which data input is selected during that clock cycle. For example, $S_5 = 3$ indicates that SN $\mathbf{X}_3$ is sampled

[1]Note that the concept of identically distributed is separate from the concept of independence. A set of bits can be identically distributed and also correlated.

during the 5th clock cycle. Let the normalized weight $|\widetilde{W_i}|$ be the probability that $\mathbf{X}_i$ is sampled during any given clock cycle and let the output of the mux tree be SN $\mathbf{Z} = z_1, z_2, ..., z_L$. $\mathbf{Z}$'s estimated value $\hat{Z} = \frac{1}{L}\sum_{i=1}^{L} z_i$ is the mux's output value. By definition, the variance of the mux tree output estimate is

$$\mathrm{Var}(\hat{Z}) = \mathbb{E}\left[\left(\hat{Z} - \mathbb{E}[\hat{Z}]\right)^2\right] \tag{A.1}$$

Let C_i be a random variable representing the number of times that $\mathbf{X}_i$ is sampled by the mux tree. Since the input SN bits are identically distributed, the output SN's estimated value can be expressed as follows.

$$\hat{Z} = \frac{1}{L}\sum_{i=1}^{L} z_i = \frac{1}{L}\sum_{i=1}^{M}\sum_{j=1}^{C_i} x_{i,j} \tag{A.2}$$

where $x_{i,j}$ is defined to represent $\mathbf{X}_i$'s bit when it is sampled by the mux tree for the j^{th} time rather than $\mathbf{X}_i$'s j^{th} bit. This re-definition of $x_{i,j}$ is justified at the end of the derivation.

Next, the expected output value can be found.

$$\mathbb{E}[\hat{Z}] = \frac{1}{L}\sum_{i=1}^{M}\mathbb{E}\left[\sum_{j=1}^{C_i} x_{i,j}\right] \tag{A.3}$$

Since C_i is a random variable, $\mathbb{E}\left[\sum_{j=1}^{C_i} x_{i,j}\right]$ is a random sum of random variables that evaluates to $\mathbb{E}[C_i][x_{i,j}]$. Further, since C_i is the number of times that $\mathbf{X}_i$ is sampled and $|\widetilde{W_i}|$ is the probability that $\mathbf{X}_i$ is sampled during any given clock cycle, we have $\mathbb{E}[C_i] = |\widetilde{W_i}|L$. Putting together these notions yields

$$\mathbb{E}\left[\sum_{j=1}^{C_i} x_{i,j}\right] = \mathbb{E}[C_i]\mathbb{E}[x_{i,j}] = |\widetilde{W_i}|LX_i \tag{A.4}$$

Thus, Equation (A.3) becomes

$$\mathbb{E}[\hat{Z}] = \sum_{i=1}^{M} |\widetilde{W_i}|X_i \tag{A.5}$$

In words, the mux's expected output value is a weighted sum of its input values as is the known operation of the mux. The output variance can now be written as

$$\mathrm{Var}(\hat{Z}) = \mathbb{E}\left[\left(\frac{1}{L}\sum_{i=1}^{M}\sum_{j=1}^{C_i} x_{i,j} - \sum_{i=1}^{M}|\widetilde{W_i}|X_i\right)^2\right] \tag{A.6}$$

by combining Equations (A.1), (A.2), and (A.5).

Let $\epsilon_i = \frac{1}{L}\sum_{j=1}^{C_i} x_{i,j} - |\widetilde{W}_i|X_i$. Then

$$\mathrm{Var}(\hat{Z}) = \mathbb{E}\left[\left(\sum_{i=1}^{M}\epsilon_i\right)^2\right] = \mathbb{E}\left[\sum_{i=1}^{M}\sum_{j=1}^{M}\epsilon_i\epsilon_j\right] \tag{A.7}$$

$$\mathrm{Var}(\hat{Z}) = \sum_{i=1}^{M}\mathbb{E}\big[\epsilon_i^2\big] + \sum_{i=1}^{M}\sum_{\substack{j=1\\ j\neq i}}^{M}\mathbb{E}\big[\epsilon_i\epsilon_j\big] \tag{A.8}$$

Next, the two sums of Equation (A.8) will be re-expressed one at a time. Noting Equation (A.4) and the definition of variance, the term in the first summation of Equation (A.8) can be rewritten.

$$\mathbb{E}\big[\epsilon_i^2\big] = \frac{1}{L^2}\mathbb{E}\Bigg[\bigg(\sum_{j=1}^{C_i} x_{i,j} - \widetilde{W}_i X_i L\bigg)^2\Bigg] \tag{A.9}$$

$$\mathbb{E}\big[\epsilon_i^2\big] = \frac{1}{L^2}\,\mathrm{Var}\left(\sum_{j=1}^{C_i} x_{i,j}\right) \tag{A.10}$$

As before, $\sum_{j=1}^{C_i} x_{i,j}$ is a random sum of random variables. Since the $x_{i,j}$'s are identically distributed and independent of C_i, it can be shown that

$$\mathbb{E}\big[\epsilon_i^2\big] = \frac{1}{L^2}\left(\mathrm{Var}\left(\sum_{j=1}^{\mathbb{E}[C_i]} x_{i,j}\right) + \mathrm{Var}(C_i)\mathbb{E}[x_{i,j}x_{i,k}]\right) \tag{A.11}$$

Next, the terms in the second summation of Equation (A.8) are re-expressed.

$$\mathbb{E}[\epsilon_i\epsilon_j] = \mathbb{E}\left[\left(\frac{1}{L}\sum_{k=1}^{C_i} x_{i,k} - |\widetilde{W}_i|X_i\right)\left(\frac{1}{L}\sum_{l=1}^{C_j} x_{j,l} - |\widetilde{W}_j|X_j\right)\right] \tag{A.12}$$

Expanding yields

$$\mathbb{E}[\epsilon_i\epsilon_j] = \mathbb{E}\left[\frac{1}{L^2}\sum_{k=1}^{C_i} x_{i,k}\sum_{l=1}^{C_j} x_{j,l} - \frac{|\widetilde{W}_j|X_j}{L}\sum_{k=1}^{C_i} x_{i,k} - \frac{|\widetilde{W}_i|X_i}{L}\sum_{l=1}^{C_j} x_{j,l} + |\widetilde{W}_i||\widetilde{W}_j|X_iX_j\right] \tag{A.13}$$

which can be simplified using Equation (A.4)

$$\mathbb{E}[\epsilon_i\epsilon_j] = \frac{1}{L^2}\mathbb{E}\left[\sum_{k=1}^{C_i}\sum_{l=1}^{C_j} x_{i,k}x_{j,l} - L^2|\widetilde{W}_i||\widetilde{W}_j|X_iX_j\right] \tag{A.14}$$

Noting that $\sum_{k=1}^{C_i} \sum_{l=1}^{C_j} x_{i,k}x_{j,l}$ is a random sum of random variables, that $\mathbb{E}[C_i] = |\widetilde{W}_i|L$, and that $\mathbb{E}[x_{i,j}] = X_i$ yields

$$\mathbb{E}[\epsilon_i\epsilon_j] = \frac{1}{L^2}\Big[\mathbb{E}[C_iC_j]\mathbb{E}[x_{i,k}x_{j,l}] - \mathbb{E}[C_i]\mathbb{E}[C_j]\mathbb{E}[x_{i,k}]\mathbb{E}[x_{j,l}]\Big] \tag{A.15}$$

Given that the covariance of two random variables A and B is $\mathrm{Cov}(A, B) = \mathbb{E}[AB] - \mathbb{E}[A]\mathbb{E}[B]$, Equation (A.15) can be rewritten as

$$\mathbb{E}[\epsilon_i\epsilon_j] = \frac{1}{L^2}\Big[\,\mathrm{Cov}(C_i, C_j)\mathbb{E}[x_{i,k}x_{j,l}] - \mathbb{E}[C_i]\mathbb{E}[C_j]\,\mathrm{Cov}(x_{i,k}, x_{j,l})\Big] \tag{A.16}$$

Putting Equations (A.8), (A.11) and (A.16) together yields

$$\mathrm{Var}(\hat{Z}) = \frac{1}{L^2}\left[\sum_{i=1}^{M}\mathrm{Var}\left(\sum_{j=1}^{\mathbb{E}[C_i]} x_{i,j}\right) + \sum_{i=1}^{M}\sum_{j=1}^{M}\mathrm{Cov}(C_i, C_j)\mathbb{E}[x_{i,k}, x_{j,l}] + \sum_{i=1}^{M}\sum_{\substack{j=1\\ j\neq i}}^{M}\mathbb{E}[C_i]\mathbb{E}[C_j]\,\mathrm{Cov}(x_{i,k}, x_{j,l})\right] \tag{A.17}$$

Define

$$\epsilon_{\text{noise}} = \frac{1}{L^2}\sum_{i=1}^{M}\mathrm{Var}\left(\sum_{j=1}^{\mathbb{E}[C_i]} x_{i,j}\right) \tag{A.18}$$

$$\epsilon_{\text{samp}} = \frac{1}{L^2}\sum_{i=1}^{M}\sum_{j=1}^{M}\mathrm{Cov}(C_i, C_j)\mathbb{E}[x_{i,k}, x_{j,l}] \tag{A.19}$$

$$\epsilon_{\text{corr}} = \frac{1}{L^2}\sum_{i=1}^{M}\sum_{\substack{j=1\\ j\neq i}}^{M}\mathbb{E}[C_i]\mathbb{E}[C_j]\,\mathrm{Cov}(x_{i,k}, x_{j,l}) \tag{A.20}$$

Thus, the mux output estimate's variance is

$$\mathrm{Var}(\hat{Z}) = \epsilon_{\text{noise}} + \epsilon_{\text{samp}} + \epsilon_{\text{corr}} \tag{A.21}$$

To summarize, the mux has M input SNs $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_M$ and output $\mathbf{Z}$. C_i is the number of times $\mathbf{X}_i$ is sampled by the mux and $x_{i,k}$ is the k-th *sampled* bit of $\mathbf{X}_i$ and not the k-th bit of $\mathbf{X}_i$. This redefinition of $x_{i,k}$ is permitted because both the binomial and hypergeometric SN models stipulate that SN bits are identically distributed. Further, the analysis assumes that mux sampling, which is directed by its select inputs, correctly implement the quantized, normalized addend weights: $|\widetilde{W}_i| = \mathbb{P}(\mathbf{X}_i \text{ is sampled})$. Equations (A.18-A.21) apply when modeling the $\mathbf{X}_i$'s as binomial or hypergeometric SNs and apply for both the unipolar SN and bipolar SC cases. However, in the bipolar SN case, the SN bits take value $\{-1, 1\}$ instead of $\{0, 1\}$ where -1 acts as logical 0. This redefinition is important when evaluating the expectations and covariances of the SN bits $x_{i,k}$ in Equations (A.18), (A.19) and (A.20). Finally, if an XOR array is used before the mux tree as is usual in bipolar mux adders, then Equations (A.18-A.21) still apply, but $x_{i,k}$ is re-defined to be $\text{sign}(W_i)x_{i,k}$ where $\text{sign}(W_i) = 1$ if $W_i \geq 0$ and $\text{sign}(W_i) = -1$ otherwise.

Some intuitions can be gained from Equations (A.18-A.20). ϵ_{noise} only depends on the variance of the input SNs which is determined by the SN model (i.e., binomial or hypergeometric). ϵ_{samp} depends mainly on the covariance of the number of times each input is sampled which is determined by the sampling method (noisy or precise). ϵ_{corr} is a function of the covariance between sampled bits of two input SNs which depends on the SN model and on the correlation between input SN bits.

A.2 Mux Variance Formulas

Here we list expressions for $\text{Var}(\hat{Z})$ when the mux tree has bipolar inputs and when an XOR array is used before the mux tree. To derive such expressions, ϵ_{noise}, ϵ_{samp}, and ϵ_{corr} are re-expressed according to which SN model (binomial or hypergeometric), sampling method (noisy or precise), and input correlation level ($\text{SCC} = 0$ or $\text{SCC} = +1$) is used. Then ϵ_{noise}, ϵ_{samp}, and ϵ_{corr} are summed together and the expression simplified and reported in Table A.1. For precise sampling, we assume that all $\widetilde{W}_i L$ are integers, which is always the case if SN length L is a power of 2 and a hardwired mux tree is used. Input correlation of $\text{SCC} = 0$ in Table A.1 means the pairwise SCC between all mux tree inputs is 0. Likewise, $\text{SCC} = +1$ in Table A.1 means the pairwise SCC between all mux tree inputs is +1 (as is the case when full correlation is achieved). Note that these derived equations were experimentally validated by simulating the stochastic circuits in which they correspond to.

For the $\text{SCC} = +1$ case, it is helpful to define the order statistics [27] of the mux tree input. Let $s_i = \text{sign}(W_i)$ and let $\boldsymbol{\mathcal{A}} = \{s_1X_1, s_2X_1, ..., s_MX_M\}$ be the values of the SN inputs to the mux tree. Then $s_{(i)}X_{(i)}$ is defined to be the i-th order statistic of $\boldsymbol{\mathcal{A}}$. For instance, $s_{(1)}X_{(1)}$ is the minimum element in $\boldsymbol{\mathcal{A}}$, $s_{(M)}X_{(M)}$ is the maximum element in $\boldsymbol{\mathcal{A}}$ and, in general, $s_{(i)}X_{(i)}$ is the i-th smallest element in $\boldsymbol{\mathcal{A}}$.

Of the six equations presented in Table A.1, only the first corresponding to the binomial model with noisy sampling has appeared before [74, 91]. Inspecting the equations in Table A.1 reveals that circuits that use hypergeometric SNs dominate[2] those that use binomial input SNs. Thus, LFSR RNSs that generate hypergeometric-like SNs should be preferred over other RNS that generate binomial-like SNs. Mux adders that use precise sampling dominate those that use noisy sampling and mux adders that use correlated inputs dominate those that use independent inputs except when the inputs are binomial SNs. In that case, correlation does not affect variance so there is no circuit dominance. Thus, according to the hypergeometric and binomial SN models, using precise sampling and achieving full correlation are always beneficial.

The most interesting aspect of the equation in Table A.1 is the equations corresponding $\mathrm{SCC} = +1$. Normally, variance equations in SC are functions of the input values. For example, an AND multiplier with hypergeometric inputs has variance $XY(1-X)(1-Y)/L$ where L is SN length and X and Y are the input values. However, the mux variance equations for the $\mathrm{SCC} = +1$ case are different. For those equations, the mux variance is a function of the *difference* of input values rather than a function of the input values themselves. In the extreme case that all input values are the same value, the mux variance would be 0 for the $\mathrm{SCC} = 1$ case. This fact is shown in the example of Figure 5.6. In general, when the input values are similar (but not exactly the same), the mux variance is low for the $\mathrm{SCC} = 1$ case. This fact is why CeMux performs so well on ECG filtering: the ECG signal values that serve as the mux adder inputs have very similar value. Thus, the ECG filtering application has a favorable input value distribution for CeMux.

[2]Recall that a circuit A dominates a circuit B when A is more accurate than B for at least one set of input values and no less accurate than B for all other sets of input values.

SN model	**Sampling method**	**Input SCC**	**Derived Mux Variance**
Binomial	Noisy	Any	$\frac{1-\left(\sum_{i=1}^{M}\widetilde{W}_i X_i\right)^2}{L}$
Binomial	Precise	Any	$\frac{1-\sum_{i=1}^{M}\lvert\widetilde{W}_i\rvert X_i^2}{L}$
Hypergeometric	Noisy	0	$\frac{1-\left(\sum_{i=1}^{M}\widetilde{W}_i X_i\right)^2-\sum_{i=1}^{M}\widetilde{W}_i^2\left(1-X_i^2\right)}{L}$
Hypergeometric	Noisy	1	$\frac{\sum_{i=1}^{M}\sum_{j=1}^{i-1}\lvert\widetilde{W}_i\rvert\lvert\widetilde{W}_j\rvert\left(s_{(j)}X_{(j)}-s_{(i)}X_{(i)}\right)}{L}$
Hypergeometric	Precise	0	$\frac{\sum_{i=1}^{M}\lvert\widetilde{W}_i\rvert\left(1-\lvert\widetilde{W}_i\rvert\right)\left(1-X_i^2\right)}{L-1}$
Hypergeometric	Precise	1	$\frac{\sum_{i=1}^{M}\sum_{j=1}^{i-1}\lvert\widetilde{W}_i\rvert\lvert\widetilde{W}_j\rvert\left(\left(s_{(j)}X_{(j)}-s_{(i)}X_{(i)}\right)-\left(s_{(j)}X_{(j)}-s_{(i)}X_{(i)}\right)^2\right)}{L-1}$

Table A.1: Mux Adder Variance Formulas

BIBLIOGRAPHY

[1] Hamdan Abdellatef, Mohamed K. Hani, and Nasir Shaikh-Husin. Accurate and compact stochastic computations by exploiting correlation. *Turkish Journal of Electrical Engineering and Computer Sciences*, 27(1):547–564, 2019.

[2] Hamdan Abdellatef, Mohamed K. Hani, Nasir Shaikh-Husin, and Sayed O. Ayat. Stochastic computing correlation utilization in convolutional neural network basic functions. *Telecommunication Computing Electronics and Control (TELKOMNIKA)*, 16(6):2835–2843, 2018.

[3] Armin Alaghi and John P. Hayes. Exploiting correlation in stochastic circuit design. In *Proceedings IEEE International Conference on Computer Design (ICCD)*, pages 39–46, 2013.

[4] Armin Alaghi and John P. Hayes. Fast and accurate computation using stochastic circuits. In *Proceedings Design, Automation and Test in Europe Conference (DATE)*, pages 1–4, 2014.

[5] Armin Alaghi, Cheng Li, and John P. Hayes. Stochastic circuits for real-time image-processing applications. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2013.

[6] Armin Alaghi, Weikang Qian, and John P. Hayes. The promise and challenge of stochastic computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(8):1515–1531, 2017.

[7] Mohsen R. Alam, M. Hassan Najafi, and Nima TaheriNejad. Sorting in memristive memory. *ACM Journal on Emerging Technologies in Computing Systems*, 2022.

[8] Benjamin Arazi. On the synthesis of de-Bruijn sequences. *Information and Control*, 49(2):81–90, 1981.

[9] Shadnaz Asgari and Alireza Mehrnia. A novel low-complexity digital filter design for wearable ECG devices. *PloS one*, 12(4), 2017.

[10] Timothy J. Baker and John P. Hayes. Impact of autocorrelation on stochastic circuit accuracy. In *Proceedings IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 271–277, 2019.

[11] Timothy J. Baker and John P. Hayes. Bayesian accuracy analysis of stochastic circuits. In *Proceedings IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2020.

[12] Timothy J. Baker and John P. Hayes. The hypergeometric distribution as a more accurate model for stochastic computing. In *Proceedings Design, Automation and Test in Europe Conference (DATE)*, pages 592–597, 2020.

[13] Timothy J. Baker and John P. Hayes. CeMux: Maximizing the accuracy of stochastic mux adders and an application to filter design. *ACM Transactions on Design Automation of Electronic Systems*, 27(3), Jan 2022.

[14] Timothy J. Baker and John P. Hayes. Design of large-scale stochastic computing adders and their anomalous behavior. In *Proceedings Design, Automation and Test in Europe Conference (DATE)*, 2023. (to appear).

[15] Timothy J. Baker, Owen Hoffend, and John P. Hayes. Multiplexer-majority chains: Managing correlation and cost in stochastic number generation. In *Proceedings IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 1–6, 2022.

[16] Timothy J. Baker, Yiqiu Sun, and John P. Hayes. Benefits of stochastic computing in hearing aid filterbank design. In *Proceedings IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–5, 2021.

[17] Petrus M.T. Broersen. *Automatic Autocorrelation and Spectral Analysis*. Springer Science & Business Media, 2006.

[18] Bradley D. Brown and Howard C. Card. Stochastic neural computation. I. Computational elements. *IEEE Transactions on Computers*, 50(9):891–905, 2001.

[19] Rahul K. Budhwani, Rengarajan Ragavan, and Olivier Sentieys. Taking advantage of correlation in stochastic computing. In *Proceedings IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2017.

[20] Raphael Cardoso, Clément Zrounba, Mohab Abdalla, Paul Jimenez, Mauricio Gomes, Benoît Charbonnier, Fabio Pavenello, Ian O'Connor, and Sébastien Le Beux. Towards a robust multiply-accumulate cell in photonics using phase-change materials. In *Proceedings Design, Automation and Test in Europe Conference (DATE)*, 2023. (to appear).

[21] Yun-Nan Chang and Keshab K. Parhi. Architectures for digital filters using stochastic computing. In *Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 2697–2701, 2013.

[22] Keping Chen. Bit-serial realizations of a class of nonlinear filters based on positive Boolean functions. *IEEE Transactions on Circuits and Systems*, 36(6):785–794, 1989.

[23] Te-Hsuan Chen and John P. Hayes. Analyzing and controlling accuracy in stochastic circuits. In *Proceedings IEEE International Conference on Computer Design (ICCD)*, pages 367–373, 2014.

[24] Te-Hsuan Chen and John P. Hayes. Equivalence among stochastic logic circuits and its application. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015.

[25] Kwen-Siong Chong, Bah-Hwee Gwee, and Joseph S. Chang. A 16-channel low-power nonuniform spaced filter bank core for digital hearing aids. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 53(9):853–857, 2006.

[26] Kyle Daruwalla, Heng Zhuo, Carly Schulz, and Mikko Lipasti. BitBench: A benchmark for bitstream computing. In *Proceedings International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 177–187. ACM, 2019.

[27] Herbert A David and Haikady N Nagaraja. *Order Statistics*. John Wiley & Sons, 2004.

[28] Marvin Faix, Raphael Laurent, Pierre Bessière, Emmanuel Mazer, and Jacques Droulez. Design of stochastic machines dedicated to approximate bayesian inferences. *IEEE Transactions on Emerging Topics in Computing*, 7(1):60–66, 2016.

[29] Rasoul Faraji, M. Hassan Najafi, Bingzhe Li, David J. Lilja, and Kia Bazargan. Energy-efficient convolutional neural networks with deterministic bit-stream processing. In *Proceedings Design, Automation and Test in Europe Conference (DATE)*, pages 1757–1762, 2019.

[30] David Fick, Gyouho Kim, Allan Wang, David Blaauw, and Dennis Sylvester. Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2D edge detection and noise filtering. In *Proceedings IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4, 2014.

[31] Christiam F. Frasser, Pablo Linares-Serrano, Alejandro Morán, Joan Font-Rosselló, Vicent Canals, Miquel Roca, Teresa Serrano-Gotarredona, and Josep L Rosselló. Exploiting correlation in stochastic computing based deep neural networks. In *Proceedings Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6. IEEE, 2021.

[32] Gary M Friesen, Thomas C Jannett, Manal Afify Jadallah, Stanford L Yates, Stephen R Quint, and H Troy Nagle. A comparison of the noise sensitivity of nine QRS detection algorithms. *IEEE Transactions on Biomedical Engineering*, 37(1):85–98, 1990.

[33] Moncef Gabbouj, Edward J. Coyle, and Neal C. Gallagher. An overview of median and stack filtering. *Circuits, Systems and Signal Processing*, 11:7–45, 1992.

[34] Brian .R. Gaines. Stochastic computing systems. In *Advances in Information Systems Science: Volume 2*, pages 37–172. Springer US, Boston, MA, 1969.

[35] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Spiking neural networks. *International Journal of Neural Systems*, 19(04):295–308, 2009.

[36] Ary L. Goldberger, Luis A.N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):215–220, 2000.

[37] Solomon W. Golomb and Guang Gong. *Signal Design for Good Correlation: For Wireless Communication, Cryptography, and Radar*. Cambridge University Press, 2005.

[38] Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability*. American Mathematical Society, 2012.

[39] Warren J. Gross, Vincent C. Gaudet, and Aaron Milner. Stochastic implementation of LDPC decoders. In *Proceedings Asilomar Conference on Signals, Systems, and Computers*, pages 713–717. IEEE, 2005.

[40] Warren J. Gross and Vincent C. Gaudet (eds.). *Stochastic Computing: Techniques and Applications*. Springer, 2019.

[41] Prabhat K. Gupta and Ramdas Kumaresan. Binary multiplication with PN sequences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(4):603–606, 1988.

[42] Kamel-Eddine Harabi, Tifenn Hirtzlin, Clément Turck, Elisa Vianello, Raphaël Laurent, Jacques Droulez, Pierre Bessière, Jean-Michel Portal, Marc Bocquet, and Damien Querlioz. A memristor-based bayesian machine. *Nature Electronics*, 6(1):52–63, 2023.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[44] Tifenn Hirtzlin. *Digital Implementation of Neuromorphic systems using Emerging Memory devices*. PhD thesis, Université Paris-Saclay, 2020.

[45] Tifenn Hirtzlin, Bogdan Penkovsky, Marc Bocquet, Jacques-Olivier Klein, Jean-Michel Portal, and Damien Querlioz. Stochastic computing for hardware implementation of binarized neural networks. *IEEE Access*, 7:76394–76403, 2019.

[46] Peter D. Hortensius, Robert D. McLeod, and Howard C. Card. Parallel random number generation for VLSI systems using cellular automata. *IEEE Transactions on Computers*, 38(10):1466–1473, 1989.

[47] Matthew B. Hoy. Alexa, siri, cortana, and more: An introduction to voice assistants. *Medical Reference Services Quarterly*, 37(1):81–88, 2018.

[48] Hsuan Hsiao, Joshua S. Miguel, Yuko Hara-Azumi, and Jason Anderson. Zero correlation error: A metric for finite-length bitstream independence in stochastic computing. In *Proceedings Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 260–265, 2021.

[49] Shuai Hu, Kaining Han, Fujie Wang, and Jianhao Hu. Hybrid stochastic LDPC decoder with fully correlated stochastic computation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2022.

[50] Hideyuki Ichihara, Motoi Fukuda, Tsuyoshi Iwagaki, and Tomoo Inoue. State assignment for fault tolerant stochastic computing with linear finite state machines. In *Proceedings International Test Conference in Asia (ITC-Asia)*, pages 156–161. IEEE, 2017.

[51] Hideyuki Ichihara, Tatsuyoshi Sugino, Shota Ishii, Tsuyoshi Iwagaki, and Tomoo Inoue. Compact and accurate digital filters based on stochastic computing. *IEEE Transactions on Emerging Topics in Computing*, 7(1):31–43, 2019.

[52] Apple Inc. About Face ID advanced technology. https://support.apple.com/en-us/HT208108. Accessed: 2023-04-26.

[53] Apple Inc. The future is here: iphone x. https://www.apple.com/newsroom/2017/09/the-future-is-here-iphone-x/. Accessed: 2023-04-29.

[54] Steven M Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice-Hall, 1993.

[55] Kyounghoon Kim, Jongeun Lee, and Kiyoung Choi. Approximate de-randomizer for stochastic circuits. In *Proceedings International SoC Design Conference (ISOCC)*, pages 123–124. IEEE, 2015.

[56] Phil Knag, Wei Lu, and Zhengya Zhang. A native stochastic computing architecture enabled by memristors. *IEEE Transactions on Nanotechnology*, 13(2):283–293, 2014.

[57] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[58] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[59] Vincent T. Lee, Armin Alaghi, and Luis Ceze. Correlation manipulating circuits for stochastic computing. In *Proceedings Design, Automation and Test in Europe Conference (DATE)*, pages 1417–1422, 2018.

[60] Vincent T. Lee, Armin Alaghi, John P. Hayes, Visvesh Sathe, and Luis Ceze. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *Proceedings Design, Automation and Test in Europe Conference (DATE)*, pages 13–18, 2017.

[61] Vincent T. Lee, Armin Alaghi, Rajesh Pamula, Visvesh S. Sathe, Luis Ceze, and Mark Oskin. Architecture considerations for stochastic computing accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2277–2289, 2018.

[62] Bingzhe Li, M. Hassan Najafi, and David J. Lilja. Low-cost stochastic hybrid multiplier for quantized neural networks. *ACM Journal on Emerging Technologies in Computing Systems*, 15(2):1–19, 2019.

[63] Peng Li, David J. Lilja, Weikang Qian, Kia Bazargan, and Marc D. Riedel. Computation on stochastic bit streams digital image processing case studies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(3):449–462, 2014.

[64] Peng Li, David J. Lilja, Weikang Qian, Marc D. Riedel, and Kia Bazargan. Logical computation on stochastic bit streams with linear finite-state machines. *IEEE Transactions on Computers*, 63(6):1474–1486, 2012.

[65] Siting Liu and Jie Han. Toward energy-efficient stochastic circuits using parallel Sobol sequences. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(7):1326–1339, 2018.

[66] Yidong Liu, Siting Liu, Yanzhi Wang, Fabrizio Lombardi, and Jie Han. A survey of stochastic computing neural networks for machine learning applications. *IEEE Transactions on Neural Networks and Learning Systems*, 32(7):2809–2824, 2020.

[67] Chengguang Ma, Shunan Zhong, and Hua Dang. Understanding variance propagation in stochastic computing systems. In *Proceedings IEEE International Conference on Computer Design (ICCD)*, pages 213–218. IEEE, 2012.

[68] Cong Ma, Peng Li, and David J. Lilja. Autocorrelation study for finite-state machine-based stochastic computing elements. In *Proceedings International Workshop on Logic Synthesis (IWLS)*, 2013.

[69] Alberto Marchisio, Muhammad A. Hanif, Faiq Khalid, George Plastiras, Christos Kyrkou, Theocharis Theocharides, and Muhammad Shafique. Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges. In *Proceedings IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 553–559, 2019.

[70] George B. Moody, W. Muldrow, and Roger G. Mark. A noise stress test for arrhythmia detectors. In *Proceedings Computers in Cardiology (CinC)*, pages 381–384, 1984.

[71] Bert Moons and Marian Verhelst. Energy and accuracy in multi-stage stochastic computing. In *Proceedings IEEE International New Circuits and Systems Conference (NEWCAS)*, pages 197–200, 2014.

[72] M. Hassan Najafi, David J. Lilja, and Marc Riedel. Deterministic methods for stochastic computing using low-discrepancy sequences. In *Proceedings IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2018.

[73] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

[74] Florian Neugebauer, Ilia Polian, and John P. Hayes. Framework for quantifying and managing accuracy in stochastic circuit design. *ACM Journal on Emerging Technologies in Computing Systems*, 14(2):1–21, 2018.

[75] Florian Neugebauer, Ilia Polian, and John P. Hayes. S-box-based random number generation for stochastic computing. *Microprocessors and Microsystems*, 61:316–326, 2018.

[76] Naoya Onizawa, Daisaku Katagiri, Kazumichi Matsumiya, Warren J. Gross, and Takahiro Hanyu. An accuracy/energy-flexible configurable Gabor-filter chip based on stochastic computation with dynamic voltage-frequency-length scaling. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(3):444–453, 2018.

[77] World Health Organization. *World report on hearing*. World Health Organization, 2021.

[78] Behrooz Parhami and Chi-Hsiang Yeh. Accumulative parallel counters. In *Proceedings Asilomar Conference on Signals, Systems, and Computers*, pages 966–970. IEEE, 1995.

[79] Weikang Qian, Xin Li, Marc D. Riedel, Kia Bazargan, and David J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Transactions on Computers*, 60(1):93–105, 2011.

[80] Sayed A. Salehi. Low-correlation low-cost stochastic number generators for stochastic computing. In *Proceedings IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1–5, 2019.

[81] Sayed A. Salehi. Low-cost stochastic number generators for stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(4):992–1001, 2020.

[82] Christopher Schröder and Sven Rahmann. A hybrid parameter estimation algorithm for beta mixtures and applications to methylation state classification. *Algorithms for Molecular Biology*, 12(1):1–12, 2017.

[83] Mohamed Adel Serhani, Hadeel T. El Kassabi, Heba Ismail, and Alramzana Nujum Navaz. ECG monitoring systems: Review, architecture, processes, and key challenges. *Sensors*, 20(6):1796, 2020.

[84] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[85] James E. Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W. Rhett Davis, Paul D. Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, et al. FreePDK: An open-source variation-aware design kit. In *Proceedings IEEE International Conference on Microelectronic Systems Education (MSE)*, pages 173–174. IEEE, 2007.

[86] Chengcheng Tang and Jie Han. Hardware efficient weight-binarized spiking neural networks. In *Proceedings Design, Automation and Test in Europe Conference (DATE)*, 2023. (to appear).

[87] Pai-Shun Ting and John P. Hayes. Isolation-based decorrelation of stochastic circuits. In *Proceedings IEEE International Conference on Computer Design (ICCD)*, pages 88–95, 2016.

[88] Paishun Ting and John P. Hayes. On the role of sequential circuits in stochastic computing. In *Proceedings Great Lakes Symposium on VLSI (GLSVLSI)*, pages 475–478, 2017.

[89] Paishun Ting and John P. Hayes. Exploiting randomness in stochastic computing. In *Proceedings IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–6, 2019.

[90] Max van Daalen, Pete Jeavons, John Shawe-Taylor, and Dave Cohen. Device for generating binary sequences for stochastic computing. *Electronics Letters*, 29(80-81):22, 1993.

[91] Ran Wang, Jie Han, Bruce F. Cockburn, and Duncan G. Elliott. Design, evaluation and fault-tolerance analysis of stochastic FIR filters. *Microelectronics Reliability*, 57:111–127, 2016.

[92] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[93] World Health Organization (WHO). Cardiovascular diseases (CVDs). https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds). Accessed: 2023-02-20.

[94] Ming Ming Wong, Dennis Wong, Cishen Zhang, and Ismat Hijazin. Stochastic inner product core for digital FIR filters. *WSEAS Transactions on Systems and Control*, 12:246–252, 2017.

[95] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[96] Meng Yang, Bingzhe Li, David J. Lilja, Bo Yuan, and Weikang Qian. Towards theoretical cost limit of stochastic number generators for stochastic computing. In *Proceedings IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 154–159, 2018.

[97] Kuncai Zhong, Meng Yang, and Weikang Qian. Optimizing stochastic computing-based FIR filters. In *Proceedings IEEE International Conference on Digital Signal Processing (DSP)*, pages 1–5, 2018.

[98] Christian Zibreg. What is Apple's neural engine and how does it work? https://www.makeuseof.com/what-is-a-neural-engine-how-does-it-work/. Accessed: 2023-04-26.