Towards Practical Stochastic Computing Architectures for Emerging Applications

Vincent T. Lee

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Luis H. Ceze, Chair

Mark H. Oskin

Armin Alaghi

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

Towards Practical Stochastic Computing Architectures for Emerging Applications

Vincent T. Lee

Chair of the Supervisory Committee:
Professor Luis H. Ceze
Paul G. Allen School of Computer Science & Engineering

The end of Dennard scaling and demands for energy efficient, low power, and high density computing solutions over the past decade has forced exploration of new computing technologies. Stochastic computing is one of these alternative computing technologies which has enjoyed renewed interest and is the primary focus of this dissertation. Stochastic computing is a form of approximate computing which encodes values as probabilistic bitstreams where the ratio of 1s and 0s determines the encoded value. This representation allows stochastic computing to achieve lower operating power, higher computational density, and better error resilience compared to conventional binary-encoded circuits. In its current form, stochastic computing presents a number of challenges before it can become a practical replacement for conventional binary-encoded computing. First, there is little prior work detailing design methodologies to guide effective implementation and integration of stochastic computing into accelerator architectures. Second, the application space where stochastic computing yields compelling gains is far from obvious and has only seen limited exploration. Third, stochastic arithmetic circuits are unintuitive to design because they require careful consideration of correlation and quantization effects.

This thesis focuses on new circuit components, applications, architectural considerations, and design techniques to improve the practicality of stochastic computing accelerators. I first propose novel stochastic circuits to improve the accuracy of stochastic computations and augment the range of implementable functions. I then evaluate the viability of stochastic computing with a design

space exploration of end-to-end stochastic computing accelerator architectures. In this exploration, I evaluate under what design parameters and conditions stochastic computing accelerators are competitive alternatives to their binary-encoded counterparts. Using these guidelines, I use these results to establish a set of architecture design guidelines to help designers identify when and why they should consider stochastic computing. I then evaluate codesign opportunities and empirically measuring power, area, and energy efficiency for emerging applications. I also propose borrowing techniques from program synthesis such as stochastic synthesis and mixed integer linear programming to automatically synthesize novel stochastic circuits. Finally, I conclude with future directions for further improving the practicality of stochastic computing as well as additional research directions beyond stochastic computing.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would first like to thank my advisors Luis Ceze and Mark Oskin for serving as both technical and non-technical mentors during the course of my graduate school career. Without their help, advise, and support, this dissertation would not have been possible. There are also numerous advisors who also generously provided both research guidance and mentorship to make this dissertation possible. In particular, I would like to thank my collaborator and mentor Armin Alaghi for his technical advice for my stochastic computing research, and John P. Hayes at the University of Michigan for his research guidance and advice for many of the projects in this dissertation. In addition, I would like to thank Rastislav Bodik and Alvin Cheung for providing their insightful advise to help expose me to and navigate program synthesis techniques.

I would like to acknowledge Armin Alaghi, Visvesh Sathe, Rajesh Pamula, and Sam Elliot for their key technical contributions to several components of this dissertation. First, I would like to thank Armin Alaghi for his key contributions to the hybrid-stochastic binary neural network project where he contributed the new stochastic adder and multiplier design in addition to the novel neural network activation units in Section 3.1, Section 3.2, and Section 5.3. Second, I would like to thank Rajesh Pamula, Armin Alaghi, and Visvesh Sathe for providing their expertise and contributing the fully fabricated ASIC prototype used to evaluate the impact of voltage overscaling on stochastic circuits in Section 4.3. Finally, I would like to thank Sam Elliot for helping implement and refine the program synthesis formulation used to design number generator sequences in Section 6.2.

I would also like to thank my many collaborators and colleagues for their support and contributions throughout my time at the University of Washington. In particular, I would like to thank my research collaborators Amrita Mazumdar, Thierry Moreau, Meghan Cowan, Max Willsey, Emily Furst, Jacob Nelson, Samuel Elliot, Justin Kotalik, and Carlo del Mundo. Much of the research work

that I did during my graduate school career could not have been done without their help, insights, and support. Special thanks to James Bornholt for accompanying me to Aladdin's outings which helped keep the research and non-research ideas flowing (and of course for sharing his expertise in program synthesis). I would also like to thank all of my senior colleagues in the University of Washington - Sampa Group for providing guidance, mentorship, and advice in the early years of my Ph.D career: Brandon Myers, Brandon Holt, Djordje Jevdjic, Adrian Sampson, Ben Ransford, Ben Wood, and Jacob Nelson. I would also like to thank Ming Liu, Liang Luo, Mark Wyse, Sung Kim, Luis Vega, and Kendall Stewart for their support and help throughout my time here whether it was over coffee, tea, lunch, dinner, or beers.

I would like to also thank my mentors outside of the University of Washington. I want to first thank my external research collaborators Sungpack Hong, Hassan Chafi, Shrikumar Hariharasubramanian, and Michael Delorimier at Oracle Labs who were instrumental in providing practical concerns and feedback for my early research. In addition, I would like to thank Eric Chung, Blake Pelton, Logan Adams, and Shlomi Alkalay at Microsoft Research for advising me while I was there.

I would also like to thank my colleagues Shumo Chu, Ignacio Cano, Alex Mariakakis, Shrainik Jain, and Daniel Miller for both their professional and moral support throughout the Ph.D. process and making my six years here especially memorable both inside and outside of Computer Science and Engineering.

Finally, the most important person I would like to thank is my wife Hiroko Kawana for putting up with me all these years while working towards this dissertation and degree.

DEDICATION

to my dear wife, Hiroko

Chapter 1

INTRODUCTION

Stochastic computing (SC) is an approximate computing technique which in recent years has witnessed a resurgence in interest as an alternative to conventional binary-encoded computation. This chapter presents motivation for stochastic computing and a preliminary outline of the practical challenges for stochastic computation that will be addressed in this dissertation. It then summarizes the primary contributions of this dissertation in the context of these key challenges.

1.1 Motivation

The computing landscape in recent decades has witnessed unprecedented challenges with the end of Dennard scaling. Energy efficiency and performance gains from smaller transistors sizes have rapidly diminished to the point where they are no longer commonplace. This has left behind a performance and energy efficiency gap which has forced innovation in specialized computing and alternative computing techniques. At the same time, emerging applications across a wide range of domains continue to demand increasing compute, memory, and storage resources. These include applications from traditional domains like multimedia and signal processing to rapidly maturing spaces such as artificial intelligence, computer vision, and genomics. While performance remains the primary design consideration, other constraints in area and fabrication cost, power limitations, and energy budgets have emerged as increasingly important design constraints. Embedded systems in particular have continued to demand highly energy efficient computation with limited power and area budgets to minimize overall device cost [68]. To satisfy these new design constraints, computing technologies have rapidly diversified to encompass new computing architecture, techniques, and encodings to bridge the widening gap between application demands and the capabilities of existing silicon.

Stochastic computing is one such technique which promises better power, area, energy efficiency, and error resilience and is the subject of this dissertation. Stochastic computing is a form of approximate or inexact computing which trades computation accuracy for improved power, area, energy efficiency, and error resilience. Unlike conventional binary-encoded arithmetic, stochastic computing encodes values as a time series of 1s and 0s where the number of 1s and 0s, and length of the bitstream determines the encoded value. The encoded value is calculated as the total weight of the bits in the bitstream divided by the total length. For instance, a stochastic bitstream $X = 10101010$ encodes the value $p_X = 0.5$. This is because each offset is weighted at face value (i.e., 1s are weighted as 1 and 0s are weighted as 0) and the bitstream length is 8. The encoded value is calculated as $\frac{1}{N}\sum_{i=0}^{N} X[i]$ where $X[i]$ denotes the ith index of the bitstream.

Stochastic computing is a promising technique because it offers compact and low power implementations of arithmetic operations. In stochastic computing, a multiplication is implemented as a single two-input AND gate (Figure 1.1). Given two bitstreams $X = 10101010$ and $Y = 00111111$ with values $p_X = 0.5$ and $p_Y = 0.75$ respectively, the resulting output bitstream $Z = 00101010$ with value $p_Z = 0.375$ encodes the product of the input bitstreams. Unlike conventional binary-encoded arithmetic, the stochastic multiplier operates on a 1-bit datapath which results in a smaller and lower power circuit. The key tradeoff is longer run time because of the time-multiplexed encoding. In addition, the stochastic multiplier requires the input bitstreams to be uncorrelated (i.e. $p(X \wedge Y) = p(X)p(Y)$) otherwise the multiplication results in errors. This correlation consideration is one of the key design challenges in stochastic computing and is a potential source of many errors.

While the stochastic computing literature dates back to the 1960s [38], it has yet to become a practical replacement for binary-encoded computation. In this dissertation, I present new circuit components, architectural considerations, application codesign, and automated synthesis techniques to improve the practicality of stochastic computing.

1.2 Practicality Challenges for Stochastic Computing

Stochastic computing faces a number of challenges to resolve before it can become a practical alternative to conventional binary-encoded computation.

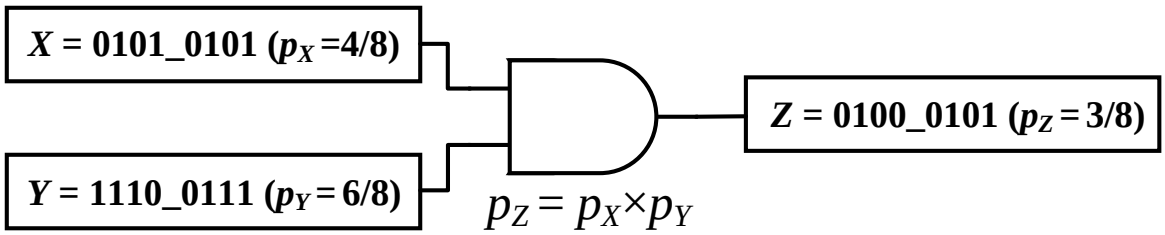


Figure 1.1: A stochastic multiplier implemented as a two-input AND gate.

The key primary challenges for stochastic computing are three-fold:

1. Understanding the tradeoffs, and when and why stochastic computing can provide compelling performance, area, power, and energy efficiency gains.

2. Designing new stochastic circuit components to augment the range of computation that stochastic computing can support and to improve the accuracy of existing stochastic operations.

3. Identifying application domains and codesign strategies to maximize power, area, and energy efficiency gains of stochastic computing and minimize application accuracy losses.

While there have been many advances in circuit implementation techniques for stochastic computing, there are few unifying guidelines for when applications should leverage stochastic computing. This is especially apparent when trying to quantify the energy efficiency advantages of stochastic computing accelerators against binary-encoded ones. In particular, there has been comparatively little work understanding when and why end-to-end stochastic computing accelerators can achieve compelling energy efficiency gains. Thus, a key challenge is understanding the design space and parameters under which stochastic computing accelerators are more energy efficient than binary-encoded accelerators. Establishing such guidelines is critical to facilitate general adoption of stochastic computing in accelerator architectures.

Stochastic computing is limited in the arithmetic operations that it can implement accurately. Accurate implementations of arithmetic operations in stochastic computing are not available for all arithmetic operations which restricts the space of applications that can be supported. As a result, identifying and designing new stochastic circuits for previously unrealized functionality is important. This in turn also improves the practicality of stochastic circuits since the augmented range of functionality enables support for new application domains or improves the accuracy of existing ones. However, the design of new stochastic circuits and accelerator architectures has historically been a challenging and arcane design problem. The primary design challenge when engineering stochastic circuits lies in the fact that designers must properly engineer and consider correlation effects. These correlation considerations significantly complicate the design process of stochastic circuits because they are unintuitive. While there exist a number of automated design methodologies for synthesizing specific classes of stochastic circuits such as polynomial evaluation, there are no general methodologies for designing arbitrary stochastic circuits.

The third key challenge is identifying applications and codesign strategies to maximize power, area, and energy efficiency gains of stochastic computing accelerators. Prior work has demonstrated the utility of stochastic computing for applications such as low density parity checks, image processing, and signal processing. Despite progress in these application domains, there remains a large set of emerging applications where the power, area, and energy efficiency gains of stochastic computing are not well known. Identifying additional application domains and codesign opportunities where stochastic computing can yield compelling power, area, and energy efficiency gains would improve the practicality of stochastic computing.

1.3 Summary of This Dissertation

This dissertation focuses on improving the practicality of stochastic computation by addressing the challenges outlined in the previous section. In Chapter 2, I start with basic definitions, a background of stochastic computing, a summary of the basic tradeoffs in stochastic computing, and definitions of stochastic circuit properties.

Chapter 3 presents a set of novel stochastic circuits to improve that implementation cost and accuracy of stochastic computation. I first present a novel correlation insensitive adder design which improves the accuracy of the conventional stochastic adder. Next, I present an improved multiplier design which uses more accurate random number generator combinations to yield more accurate multiplication. Finally, I present a new set of correlation manipulating circuits which can be used to improve the accuracy of arithmetic circuits by injecting correlation between two bitstreams. Together these novel circuits dramatically improve the tradeoffs of stochastic arithmetic operations.

Chapter 4 focuses on comparing the energy efficiency tradeoffs and gains of stochastic computing accelerators against binary-encoded ones with a design space exploration. Through this design space exploration, I find that individual stochastic computing operations are not necessarily more energy efficient, lower power, and higher density per operation than binary-encoded ones. Despite this negative result, I find that after considering end-to-end accelerator architecture overheads stochastic computing accelerators can still achieve reasonable energy efficiency gains over binary-encoded ones. I then provide a set of architectural design guidelines to help facilitate the process of identifying when an application may be compatible with stochastic computing and achieves compelling energy efficiency gains.

Chapter 5 evaluates a set of emerging applications based on the application guidelines identified in Chapter 4. I evaluate three sets of applications - similarity search, support vectors machines, and neural networks - and identify codesign opportunities for each case study. In each case, I evaluate the viability of the application by exploring codesign opportunities and comparing them against binary-encoded accelerators to gauge practicality. In the case of neural networks, I evaluate hybrid stochastic and binary-encoded accelerator implementations where neural network layers are partitioned between stochastic and binary-encoded operating domains. Furthermore, I show that retraining the model based on the error models of stochastic computing can provide superior application accuracy over models that are not retrained.

Chapter 6 introduces novel design techniques for stochastic circuits to automate the design burden of identifying and engineering new stochastic circuits. I first introduce a general technique for designing new stochastic circuits using stochastic synthesis. I show that unlike previous design

methods in stochastic computing, stochastic synthesis is not limited to any particular class of circuits but trades off scalability; the technique is limited to synthesizing smaller circuits. In lieu of an exact solution, I show that stochastic synthesis can also identify circuits which realize reasonable approximations of functions that may not have an exactly accurate solution. I then introduce a new technique using mixed integer linear programming to synthesize optimally correlated number sequences that can be used to improve the accuracy of stochastic arithmetic circuits. I show that this technique can be used to improve the accuracy of operations such as multiplication and squaring. Finally, I briefly outline how the stochastic synthesis and integer linear programming formulation can be combined to synthesize new stochastic circuits for future work.

Finally, Chapter 7 outlines immediate future directions to enhance the practicality of stochastic computing. I first explore the need for reconfigurability to enhance the practicality of stochastic computing accelerators. I also highlight potential avenues of exploration in alternative computing substrates to change the calculus in favor of stochastic computing. Finally, I outline future directions towards generalizing the stochastic computing paradigm and how this dissertation can be leveraged for such work.

Chapter 2

BACKGROUND ON STOCHASTIC COMPUTING

Stochastic computing (SC) is a half-century old technique that relies on unary encoded bitstreams to encode values and a small number of elementary logic gates to perform arithmetic operations [38, 85]. In recent years, stochastic computing has enjoyed renewed interest as an alternative computing technique to binary-encoded (BE) computation. In this chapter, I provide background on stochastic computing, summarize the design tradeoffs, and explore the properties of stochastic computing circuits.

2.1 Stochastic Computing Basics

Stochastic computing (SC) uses unary bitstreams (time series of 1s and 0s) to represent values. The value of a bitstream is encoded by the number of 1s and 0s, and the precision is determined by the bitstream length N. Stochastic bitstreams are often referred to as *stochastic numbers* (SNs) and typically use either unipolar or bipolar encodings.

In unipolar-encoded SNs, 1s have a weight of +1 and 0s have a weight of 0. The encoded value is the sum of each position in the SN divided by the SN length N, and is limited to the range [0, 1]. For instance, the SN $X = 01100001$ has the value $p_X = 3/8$ since there are three 1s and the SN length is $N = 8$. Alternatively, bipolar encodings weight 1s as +1 and 0s as −1 allowing them to encode both negative and positive values in the range [−1, +1]. For example, the same SN $X = 01100001$ has the value $p_X = -1/4$ under bipolar encodings.

The key strength of stochastic circuits is that they are extremely dense, low power, and error tolerant. For instance, a stochastic multiplication can be implemented by a single two-input AND gate (Fig. 2.1a) since, given two SNs X and Y with encoded values p_X and p_Y respectively, the bitstream $Z = X\&Y$ has value $p_Z = p_X p_Y$. Notice that the SNs X and Y are assumed to be

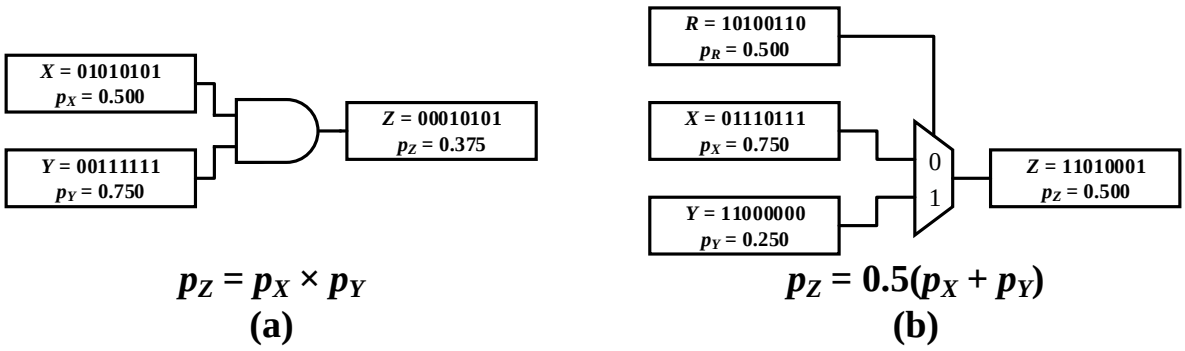


Figure 2.1: Example operation of stochastic computing: (a) multiplication and (b) addition.

uncorrelated, otherwise the fidelity of the stochastic multiplication breaks down. Stochastic addition is implemented using a multiplexor with input SN operands as the data inputs, and an uncorrelated auxiliary SN R with value $p_R = 0.5$ as the select signal. The stochastic adder effectively samples each input SN with equal probability resulting in a scaled sum $p_Z = 0.5(p_X + p_Y)$. An example of stochastic addition is shown in Fig. 2.1b.

Notice that in SC, all bits have equal weight which means the equivalent fixed-point precision of a SN with length N is approximately $log_2(N)$ bits since it can represent the values $\{0/N, 1/N, 2/N, ..., (N-1)/N, N/N\}$. Consequently, the input and output SNs for all arithmetic operations must have the *same* precision. This highlights one source of error in SC: quantization errors due to precision reduction. For instance, consider stochastic addition which has a scale factor of 0.5 in the resulting sum. This forces the output SN precision to be the same as the input SN precision and as a byproduct drops the least significant bit of the true sum. As a result, quantization errors can compound and severely reduce computation accuracy if not properly considered. One way to mitigate against compounding quantization errors is to increase the SN length. This increases the run time since SN length scales exponentially with increasing precision. An alternative solution is to use higher precision conversion circuits such as accumulative parallel counters (APC) [108] which can yield higher precision results by aggregating results over multiple streams. The APC however forces a conversion between the SC and BE domain.

To generate SNs, SC uses digital-to-stochastic (D/S) converters which takes a BE fixed-point value $x \in [0, 1]$, and a random integer value $r \in [0, 1]$ produced by a random number generator (RNG). At each clock cycle, the RNG value r is compared against x to produce the desired SN with value $p_X = x$. The choice of RNG is vitally important since correlated or uncorrelated RNGs can be used to generate correlated or uncorrelated SNs respectively. To convert an SN back to the BE domain, designers use a stochastic-to-digital (S/D) converter (Fig. 2.6f) which is implemented as a counter that sums up each bit in the SN. These S/D converters, D/S converters, and RNGs are overheads unique to operating in the stochastic domain. They are also much larger and use more power than stochastic arithmetic circuits. However, these overheads can be amortized over many stochastic operations by judiciously exploiting application data reuse and composing multiple operations together (evaluated in Chapter 4).

A full end-to-end example of a stochastic multiplication for a bitstream length N = 8 is shown in Figure 2.2. Values are first encoded into stochastic bitstreams (Figure 2.2a) using number generators and D/S converters. Once bitstreams are generated, they are passed through stochastic arithmetic circuits (Figure 2.2b). For downstream processing or storage, the stochastic bitstreams are then converted back to BE values using an S/D converter (Figure 2.2c).

2.1.1 The Role of Correlation

Correlation between SNs is one of the principle sources of errors in SC. For instance, the stochastic multiplier introduced earlier has an affinity for uncorrelated SNs, otherwise the assumption that $p_Z = p_X \wedge p_Y = p_X p_Y$ does not hold and the computation results in errors. Correlation between bitstreams is measured using the stochastic computing correlation (SCC) [6]. For two bitstreams X and Y, the SCC is defined as.

$$\mathtt{SCC}(X, Y) = \begin{cases} \frac{ad-bc}{N\times\mathtt{min}(a+b,a+c)-(a+b)(a+c)} & ad > bc \\ \frac{ad-bc}{(a+b)(a+c)-N\times\mathtt{max}(a-d,0)} & \mathtt{else} \end{cases} \tag{2.1}$$

In this definition, a is the number of positions where X and Y are both 1, b is the number of positions where X is 1 and Y is 0, c is the number of positions where X is 0 and Y is 1, and d

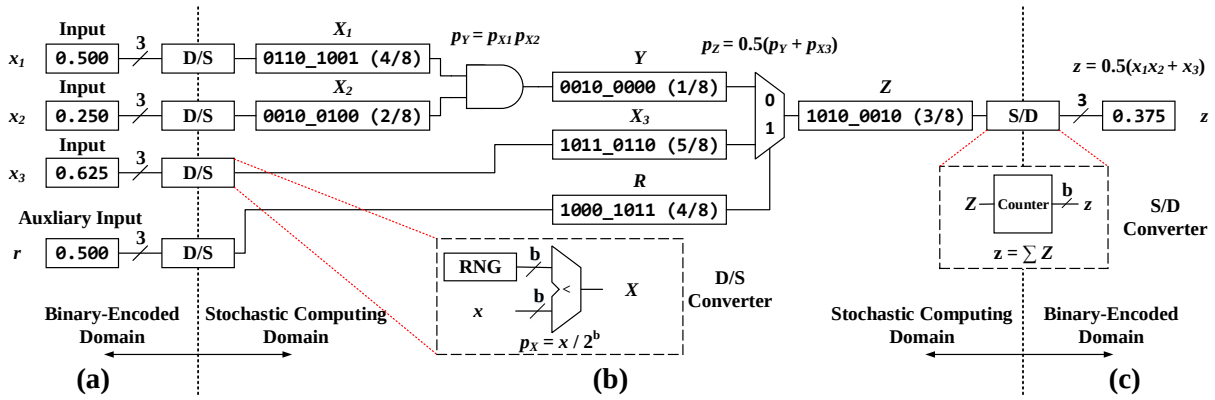


Figure 2.2: End-to-end example of stochastic multiplication. (a) Values are converted to bitstreams. (b) Bitstreams are passed through stochastic arithmetic units. (c) Bitstreams are converted back to binary-encoded values.

is the number of positions where X and Y are both 0. A SCC = 0 means the SNs X and Y are perfectly uncorrelated, while a SCC = +1 and SCC = −1 indicates maximal positive and negative correlation respectively. Intuitively, maximally positively correlated bitstreams (SCC = +1) will have almost all overlapping 1s (occurring on the same cycle) between the two bitstreams. On the other hand, negatively correlated bitstreams (SCC = −1) will have few if any overlapping 1s between the two bitstreams. Many circuits in SC have an optimal correlation under which they operate most accurately. As a result, the closer the correlation of the input bitstreams are to the optimal SCC, the more accurate the computation will be.

Managing and mitigating the impact of unwanted correlation between SNs is a key design challenge in SC. There are three principle methods for controlling correlation in SC: (1) using correlation insensitive circuits [13], (2) using correlation manipulating circuits [53], and (3) judiciously selecting number sequences for SNGs. The first method - using correlation insensitive circuits - relies on special variants of each arithmetic circuit which are immune to correlation levels. Given two input SNs with values p_X and p_Y, a correlation insensitive circuit will always yield the same value $f(p_X, p_Y)$ regardless of the SCC between the SNs X and Y. One key drawback of

correlation insensitive circuits is that not all operations in SC have correlation insensitive variants. The known set of correlation insensitive circuits are also larger and consume more power than then their equivalent correlation sensitive counterparts.

The second correlation engineering technique is to insert correlation manipulating circuits between arithmetic operations. Correlation between SNs is often distorted over successive stochastic computations. Unfortunately, the quantitative impact of how each stochastic arithmetic operation changes the SN correlation with respect to other SNs is not well understood. As a result, it is sometimes difficult or impractical to completely guarantee correlated or uncorrelated input SNs across many operations. To mitigate against undesirable changes in correlation at intermediary stages of stochastic computation, designers use correlation manipulating circuits. Correlation manipulating circuits are a class of circuits that alter the correlation between two or more SNs. The correlation levels are usually adjusted towards the desired correlation required for downstream stochastic circuits.

Known correlation manipulating circuits include isolators [110], synchronizers, desynchronizers, decorrelators [53], and regenerators [107]. Regenerators mitigate the impact of computation-induced correlation by converting all SNs back to the BE domain using S/D converters and re-encoding the converted values back to SNs using D/S converters [110]. This technique is known as *regeneration* since it regenerates SNs to reset any correlation that may have existed or did not exist between SNs. However, regeneration is expensive to execute since S/D converters and D/S converters consume one to two orders of magnitude more power and area than stochastic arithmetic circuits. A smaller and lower power alternative to regeneration is inserting isolators [110]. Isolators temporally shift two SNs relative to each other which should reduce correlation if SNs are generated from truly Bernoulli random processes. However, isolators do not modify the order of bits in a SN. As a result, autocorrelation between bits in the SN is preserved which ultimately limits the impact on the overall SCC.

The third correlation engineering technique is to judiciously select number sequences for SNGs. Not all number sequence combinations achieve the same accuracy so it is important to choose properly correlated number sequences or number sequences with favorable properties. Unfortunately,

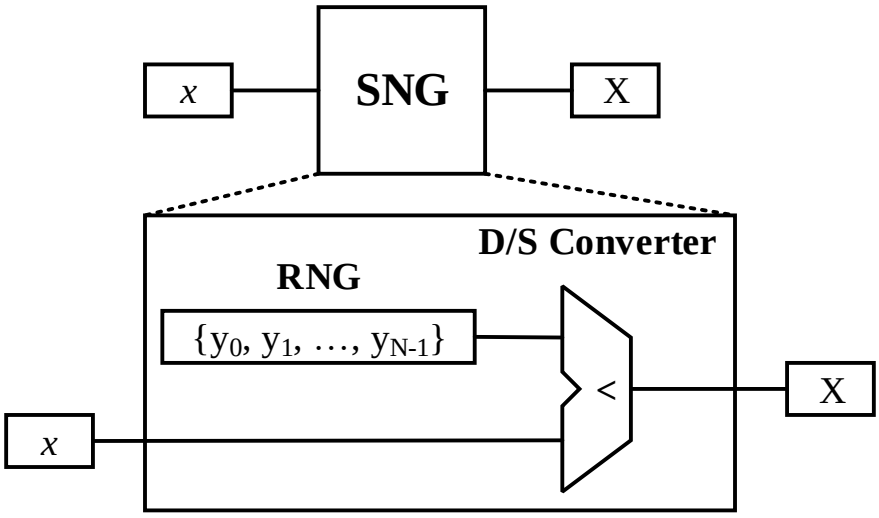


Figure 2.3: A stochastic number generator (SNG) is composed of a digital-to-stochastic (D/S) converter and random number generator.

it is impractical to use distinct uncorrelated RNGs to generate all SNs since RNGs are significantly larger and higher power than individual SC operations [42]. As a result, most stochastic computation amortizes the cost of RNGs by using each RNG to generate many SNs. A key limitation of this technique is that correlation between SNs can only be induced during D/S conversion. This is in contrast to correlation manipulating circuits which can operate on intermediary SNs.

2.1.2 The Role of Random Number Generators

Random number generators are one of the key components in SC because they are used to generate bitstreams with digital-to-stochastic converters (Figure 2.3). More importantly, they are one of the primary means of engineering proper correlation between bitstreams. The choice of number sequence generator across different bitstreams is a key design parameter which governs the fidelity of the computation.

The number sequence that drives the random number generator (RNG) comes in several variants: truly random, pseudorandom, and deterministic. Truly random number sequence generators are

drawn from natural random noise source phenomena. Recent work has proposed used nanoscale devices [116, 118] to realize practical implementations of truly random noise sources. Random noise sources yield nondeterministic results since they yield different concrete SN results across executions.

Pseudorandom number sequence generators rely on components like linear feedback shift registers (LFSRs) to drive SN generation. LFSRs are attractive noise sources because they have compact implementations. Deterministic number sequences on the other hand use preset, repeating number sequences to generate SNs. Commonly used deterministic number sequences include low discrepancy sequences such as Van der Corput, Halton [8], and Sobol [65] sequences which have desirable correlation properties.

Random, pseudorandom, and deterministic number sequences each have their own strengths and weaknesses. Random noise sources can often be less expensive to implement in terms of power, area, and energy by exploiting naturally occurring sources of randomness. However, since they are nondeterministic, circuits that use random noise sources on average have lower accuracy. Deterministic number sequences are often significantly larger since they are implemented using standard CMOS components. On the other hand, deterministic number sequences yield deterministic computation results and allow better bounds on the computation error. As a result, many recent work that target CMOS implementations uses deterministic number sequences to drive D/S conversions. Unless otherwise noted, I will primarily use deterministic number sequences in my evaluations in this dissertation.

2.1.3 Relative Ordering Invariance

The deterministic number sequences chosen to drive SNGs can be rotated or have number positions swapped to yield equivalently correlated bitstreams and identical computation accuracy for combinational stochastic circuits. For instance, two number sequences $S_X = 0, 1, 2, 3$ and $S_Y = 0, 3, 1, 2$ would produce *iso-accurate* results as the number sequences $S'_X = 3, 0, 1, 2$ and $S'_Y = 2, 0, 3, 1$ (rotated versions). The number positions can also be swapped to obtain equivalently accurate results. For instance, $S''_X = 0, 2, 1, 3$ and $S''_Y = 0, 1, 3, 2$ would also yield equivalently accurate results as

the original number sequences. As long as the relative position of numbers between the number sequences are preserved, the sequences produce iso-accurate results since correlation between the number sequences is preserved.

More precisely, consider two arbitrary number sequences $S_X = \{x_0, x_1, ..., x_{N-1}\}$ and $S_Y = \{y_0, y_1, ..., y_{N-1}\}$ of length N. Encoding the values x and y yields the bitstreams $X = \{x < x_0, x < x_1, ..., x < x_{N-1}\}$ and $Y = \{y < y_0, y < y_1, ..., y < y_{N-1}\}$ respectively. Given a combinational stochastic circuit, I can define a function $h(x, y)$ which describes the circuit behavior and computes the output bit given two input bits; note that this is only possible because there is no state. The output of the stochastic circuit yields a bitstream $Z = \{h(x < x_0, y < y_0), ..., h(x < x_{N-1}, y < y_{N-1})\}$ which after S/D conversion produces $\frac{1}{N}\sum_{n=0}^{N-1} h(x < x_n, y < y_n)$.

If S_X and S_Y are rotated by the same offset m, the resulting number sequences $S'_X = \{x_{m\%N}, x_{(m+1)\%N}, ..., x_{(m+N-1)\%}$ and $S'_Y = \{y_{m\%N}, y_{(m+1)\%N}, ..., y_{(m+N-1)\%N}\}$. The key insight is that the same numbers in each sequence after this transformation occur at the same offset. Swapping the positions of two numbers within the number sequences has the same effect. As a result, the resulting bitstream is a rotated version of the original output bitstream $Z' = \{h(x < x_{m\%N}, y < y_{m\%N}), ..., h(x < x_{m+N-1\%N}, y < y_{m+N-1\%N})\}$. Since the S/D conversion is commutative, the encoded value of the output bitstream is equivalent to that of the original computation. In other words, the resulting value $\frac{1}{N}\sum_{n=0}^{N-1} h(x < x_n, y < y_n)$ is agnostic to the ordering and will still yield the same value.

I refer to this property as *relative ordering invariance* between two deterministic number sequences which will become important when synthesizing number sequences (Section 6.2). Two examples of this property are illustrated in Figure 2.4 where exploiting relative ordering invariance yields two different instances of equivalently accurate multiplication. Notice that this property does not hold for sequential circuits since state elements are sensitive to autocorrelation which is not preserved under these transformations.

2.1.4 Error Tolerance

Stochastic computing encodings are more error tolerant than binary-encodings. Unlike binary-encodings, the stochastic encoding is redundant in that multiple unique SNs can map to the same

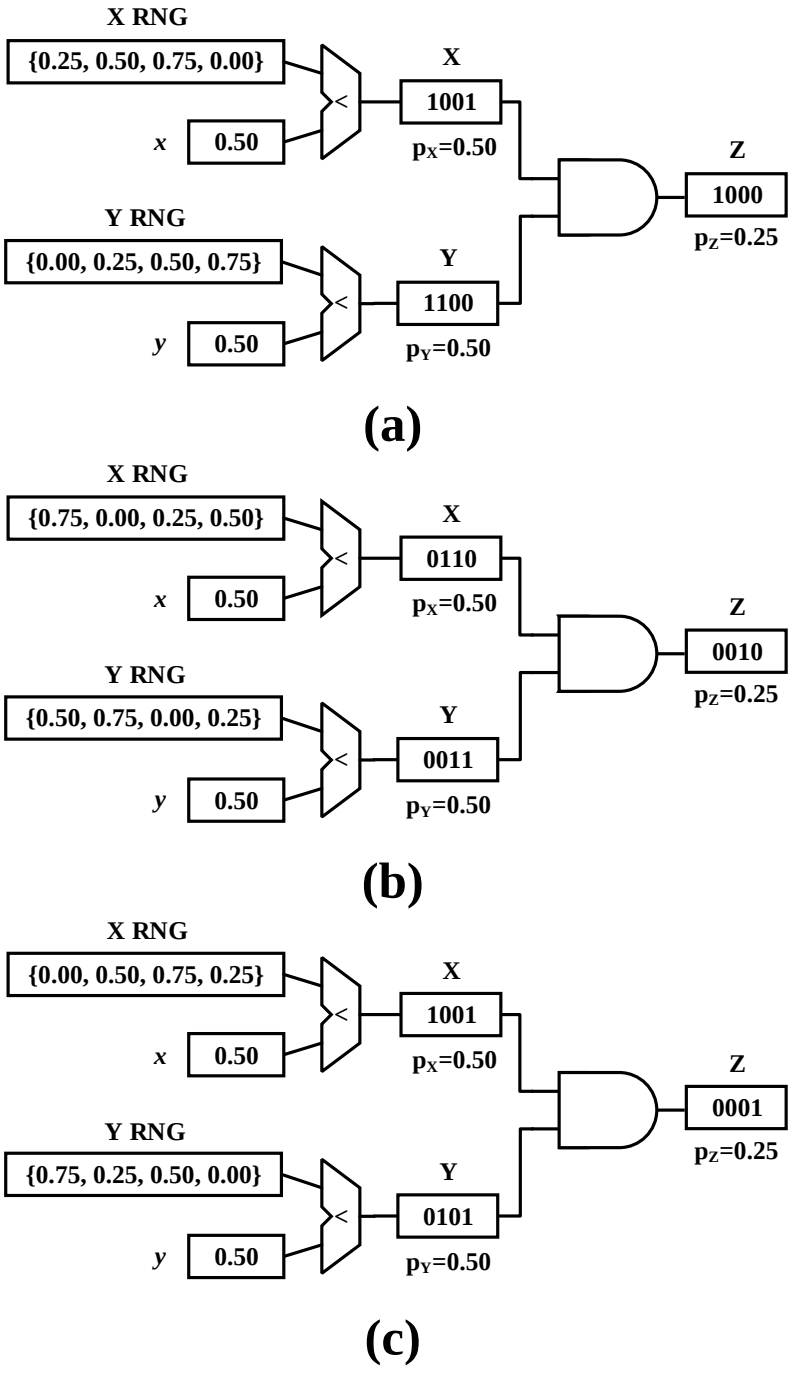


Figure 2.4: Relative ordering invariance of number sequences allows (a) the original set of number sequences (b) to be rotated or (c) swap number positions to yield equivalently accurate computation.

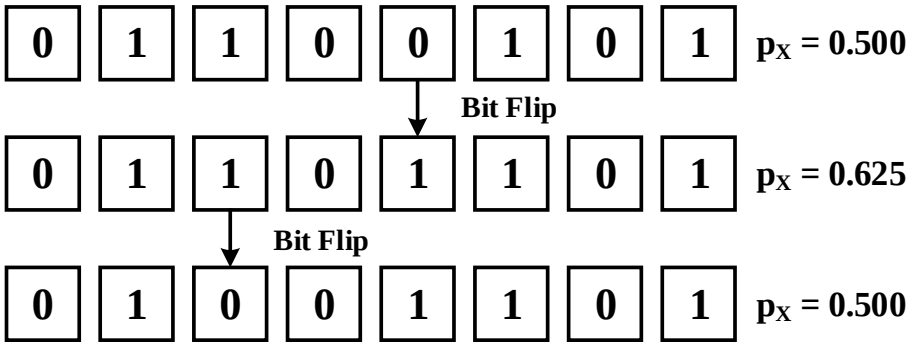


Figure 2.5: Example of stochastic encoding error tolerance. A bit flip in a stochastic bitstream impacts the encoded value minimally. Bit flips may even cancel out.

value. For instance, the SN $X = 01101100$ and $X' = 1111000$ both have an encoded value of 0.5. This redundant encoding enables better error tolerance against bit flips in the computation.

Bit flips in the encoding only distort the encoded value minimally since all bits have equal weight. In binary-encodings, a bit flip could alter the encoded value by up to half the range of the encoding. Furthermore, bit flips in stochastic encodings may even cancel out (e.g. a 0-to-1 bit flip and a 1-to-0 bit flip occur) regardless of where the bit flips occur in the bitstream. This is in contrast to BE values where bits carry different weights which makes this self-correcting bit flip behavior significantly less likely.

The inherent error tolerance is best illustrated with an example. Figure 2.5 shows the impact of individual bit flips on the encoded value in SC for a bitstream length N = 8. The first bit flip in this example is a 0-to-1 error which increases the encoded value $p_X = 0.500$ by 0.125 to $p_X = 0.625$. Notice that regardless of whether the error is a 0-to-1 or 1-to-0 error, the worst-case absolute error than can occur from a single bit flip is 1/N = 0.125. The second bit flip that occurs is a 1-to-0 error which decreases the new encoded value $p_X = 0.625$ back to its original value $p_X = 0.500$. Notice that the resulting bitstream is a redundant instance of the original bitstream and encodes the same value.

2.2 Properties of Stochastic Computing Circuits

Unlike conventional BE computation, SC has a diverse set of circuit properties. Each property yields different accuracy, energy, power, area, and throughput tradeoffs. To better understand the tradeoffs offered by each property, this section provides a taxonomization of known stochastic arithmetic circuits. The full taxonomy summarizing existing stochastic circuits is shown in Table A.1.

2.2.1 Correlation Affinity

Managing and manipulating correlation between SNs in stochastic computation is critical for producing accurate results. Figure 2.6 shows a subset of known arithmetic correlation sensitive circuits. Note how some stochastic operations are more accurate when their input SNs are correlated. As previously defined, the correlation between two SNs X and Y are quantified using the stochastic computing correlation (SCC) defined in [6]. I define the correlation conditions under which a particular arithmetic unit yields most accurate results as the *correlation affinity*. A correlation affinity for uncorrelated (0) inputs means the stochastic circuit functions most accurately with uncorrelated input bitstreams. Similarly, a correlation affinity of positive (+) or negative (−) expresses that the stochastic circuit works most accurately when input bitstreams are positively or negatively correlated respectively. Stochastic circuits can also be correlation insensitive (*) if they are iso-accurate under arbitrary correlation levels. Figure 2.6 shows several stochastic circuits and their correlation affinities from prior work.

2.2.2 Exact Computation

In conventional stochastic circuit operation, the input precision and output precision are always the same since the input and output SN lengths must be the same. As a result, operations such as multiplication where the true result requires higher output precision than input precision cannot by implemented without some loss in accuracy. In other words, these operations cannot be implemented

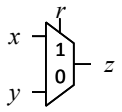
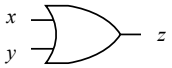
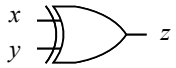
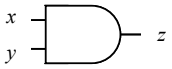
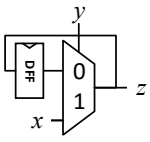
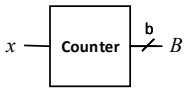
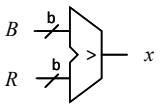
Operation	Circuit	Reference	Operand Correlation
(a) Addition	 $p_Z = 0.5(p_X + p_Y)$	[37]	Uncorrelated with *r*
(b) Saturating Addition	 $p_Z = \min(1, p_X + p_Y)$	[6]	Negative
(c) Subtraction	 $p_Z = \lvert p_X - p_Y \rvert$	[9]	Positive
(d) Multiplication	 $p_Z = p_X \times p_Y$	[37]	Uncorrelated
(e) Division	 $p_Z = p_X / p_Y$	[27]	Positive
(f) S/D Conversion	 $B = p_X \times 2^b$		Insensitive
(g) D/S Conversion	 $p_X = B / 2^b$		N/A

Figure 2.6: Correlation sensitive stochastic circuits and converter units: (a) scaled addition, (b) saturating addition, (c) subtraction, (d) multiplication, (e) division, (f) S/D converter, and (g) D/S converter.

exactly in conventional stochastic circuit operation.[1] There do exist a handful of stochastic circuits which yield exactly correct results (no accuracy loss). For instance, the subtractor proposed in [6] implemented using a two-input XOR gate produces exact results when input SNs are positively correlated. I classify such functional units which can yield completely accurate results under proper correlation conditions as *exact*. Functional units which are exact are desirable because they do not compromise accuracy. Unfortunately, exact implementations do not exist for all stochastic operations.

2.2.3 Quantization Error and Direction

Unlike BE circuits, many stochastic circuits suffer from quantization errors across successive arithmetic operations. In SC, the input precision of a functional unit is the same as the output precision of the functional unit since the input and output SNs must be of the same length. As a result, any computation which requires higher output precision than input precision suffers from quantization errors. Most quantization errors in SC truncate the lower bits of the true result. I classify these stochastic circuits as *most significant bit* (MSB) preserving. For instance, stochastic multiplication is an example of a MSB preserving circuit since the result only maintains the high bits of the true result. There are a few instances where SC truncates the higher bits of the true result. I classify these computations as *least significant bit* (LSB) preserving. An example of this is the saturating addition proposed in [6].

Quantization errors are a familiar challenge to the approximate computing design space [72, 73]. One way to mitigate the impact of forced quantization errors is to increase the precision by increasing the length of the SN. In SC, increasing the SN length trades accuracy for run time and energy efficiency. In addition, every extra bit of operating precision in SC doubles the SN length. This results in the unfavorable run time and energy efficiency scaling behavior with higher operating precision. As a result, to maximize energy efficiency in SC, it is desirable to use the minimum viable arithmetic precision or SN length since this yields shorter run times and better energy efficiency.

---

[1] Higher operating precisions can be constructed by executing a lower operating precision circuit multiple times and aggregating the result.

2.2.4 Progressive Precision and Pseudostochastic Functional Units

Many stochastic circuits have the unique property in that different operating precisions can be implemented with the same functional unit implementation. Recall that BE functional units require wider or narrower datapaths to support arithmetic at different precisions. In SC, it is possible to use the same functional unit but change the length of the SN or run time to realize different precisions. For instance, the same two-input AND gate used for stochastic multiplication generalizes to arbitrary SN length since, given two SNs X and Y, the product Z will always be $p_Z = p_X p_Y$. This property is known as *progressive precision* since the circuit implementation generalizes to arbitrary SN length. Circuits with the progressive precision property are desirable because they enable the flexibility of operating at different precisions as application demands vary without incurring additional power or area cost.

A handful of functional units use BE elements such as counters or integrators into their implementation. As a result, they are not true SC elements since a component of the functional unit operates in the BE regime. The distinction is important because the BE elements make the overall functional unit larger as the power and area of these BE elements scale with operating precision. These BE elements are also not as error resilient because they employ BE values. Having these BE elements also disqualify these functional units from having progressive precision since they require different width BE elements to support different operating precisions. True stochastic circuits where the same implementation generalizes across operating precisions have constant power and area. I classify stochastic circuits which employ BE elements as *pseudostochastic* functional units since they require components that operate in both the stochastic and BE domains. An example of a pseudostochastic element is the stochastic maximum proposed in [88] which requires several BE counters that scale with each bit of operating precision. Conversion units such as the S/D converter, D/S converter, and accumulative parallel counter [108] are also pseudostochastic since they convert between stochastic and BE values or vice versa. These operations necessarily require both stochastic and BE elements.

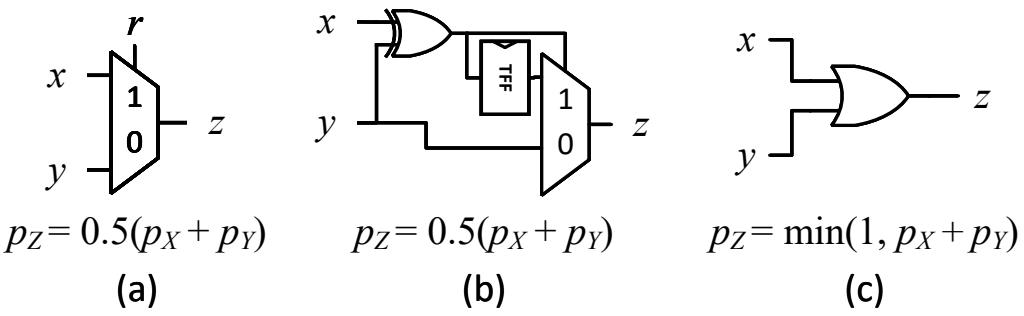


Figure 2.7: Stochastic adder variants: (a) original scaled adder, (b) correlation insensitive scaled adder, and (c) saturating adder.

2.2.5 Discrepancy Sensitivity

The discrepancy of a SN is defined as the deviation of the value of a continuous subset of a SN from the actual SN value [8]. Intuitively, the discrepancy of a SN is a measure of how evenly distributed 1s and 0s are throughout the SN. For instance, a SN X = 10101010 (p_X = 0.5) has low discrepancy because a continuous subset of the SN A = 01 (p_A = 0.5), B = 1010 (p_B = 0.5) or C = 10101 (p_C = 0.6) does not deviate significantly from the SN value p_X = 0.5. However, a SN Y = 11110000 has higher discrepancy because certain continuous subsets of the SN can deviate significantly from the SN value such as D = 1111(p_D = 1.0), E = 000(p_E = 0.0), F = 1000(p_F = 0.25). The discrepancy of a SN is important as it can impact the accuracy of the computation and behavior of progressive precision. For instance, two SNs can be uncorrelated but individually have high discrepancy. An example of this is the multiplier proposed in [54] using a high discrepancy ramp sequence and low discrepancy Van der Corput sequence as RNGs. Unfortunately, there are no formal methodologies for determining whether a functional unit is discrepancy sensitive other than enumerating its functionality. As a result, I define discrepancy sensitivity as the property that the circuit accuracy changes with the discrepancy of the individual operands. Correlation and discrepancy sensitivity are not orthogonal but the exact relationship is not well understood and requires additional work to reconcile in SC.

Table 2.1: Summary of stochastic circuit properties and their impact on stochastic computing's tradeoffs when properly exploited.

Property	**Values**	**Description**
Correlation Affinity	positive (+), negative (−), uncorrelated (0), agnostic (∗), autocorrelated (+†), not autocorrelated (0†)	The optimal correlation of input SNs under which functional unit has least average accuracy error. Improves accuracy.
Quantization	MSB, LSB, None	The bits that are preserved when the precision of the true output is higher than the input. Reduces accuracy.
Progressive Precision	Yes (✓), No (✗)	Functional unit implementation generalizes to all SN lengths without modification.
Correlation Manipulating	Yes (✓), No (✗)	Functional unit's primary purpose is altering correlation between SNs or autocorrelation of an SN. Improves accuracy. Reduces energy efficiency.
Multifunctional	Yes (✓), No (✗)	Functional unit can implement different functionality under different input SN correlation conditions.
Exact	Yes (✓), No (✗)	Functional unit achieves no arithmetic accuracy loss from true value under optimal input SN correlation conditions. Improves accuracy.
Pseudostochastic	Yes (✓), No (✗)	Contains both stochastic and binary-encoded components. Reduces power, area, and energy efficiency.
Discrepancy Sensitive	Yes (✓), No (✗)	Accuracy of functional unit under optimal correlation conditions affects accuracy or efficacy of functional unit. Reduces accuracy.
Encoding Agnostic	Yes (✓), No (✗)	Same functional unit implementation can be re-used across unipolar and bipolar encodings.

2.3 Definitions and Quantifying Metrics

Throughout this dissertation, I will evaluate and compare the accuracy, power, area, throughput, and energy efficiency of stochastic circuits and accelerators. This section defines each of the metrics and briefly justifies their utility.

2.3.1 Precision and Accuracy

I define precision as the number of representable values that are encodable and measured as bits of precision. For instance, if a computation has 8-bit precision, the number of encodable values is 2^8 values. Conversely, a representation of a value which can encode N values has a precision of $log_2(N)$ bits. In the context of SC, this means a bitstream with length N will have an effective precision of $log_2(N)$ bits. Throughput this dissertation, I will make iso-precision comparisons between stochastic and binary-encoded computation.

To measure the accuracy of a given computation $f(x)$, I define error as the distance between the floating point computation value $f'(x)$ and $f(x)$. In this dissertation, I will use average absolute error, mean squared error (MSE), and peak signal-to-noise ratio (PSNR). For stochastic computations, I will measure the error over all possible input combinations for a particular arithmetic circuit. For instance, given a two-input stochastic circuit which implements some function $f(p_x, p_y)$ using bitstream length N, the accuracy of the circuit can be evaluated using the average absolute error over (Equation 2.2) or the mean squared error (Equation 2.3) over all possible combinations of p_x and p_y:

$$\frac{1}{(N+1)^2}\sum_{n=0}^{N}\sum_{m=0}^{N}|f(n/N, m/N) - f'(n/N, m/N)| \tag{2.2}$$

$$\frac{1}{(N+1)^2}\sum_{n=0}^{N}\sum_{m=0}^{N}(f(n/N, m/N) - f'(n/N, m/N))^2 \tag{2.3}$$

PSNR on the other hand is defined as:

$$PSNR = 20\log(MAX_I) - 10\log(MSE)$$

where MAX_I is defined as the max intensity value.

2.3.2 Power and Area

In embedded settings, a large power supply may not be readily available which limits the operating power. As a result, power consumption is an important quantity which can govern the viability of an accelerator design. In addition, accelerator designs which require substantial silicon area quickly become expensive to fabricate since area roughly correlates to the cost of ASIC fabrication. Thus, the total area of a design is also an important quantity that loosely informs economical viability. For evaluations going forward, I will report the total power and area estimates but also rely on other normalized metrics discussed next.

2.3.3 Energy Efficiency

Energy efficiency is a throughput-normalized metric which is agnostic to variations in run time and operating power. Energy efficiency is defined as the units of work accomplished per unit of energy and is often reported in Joules per unit of work. Since accelerator designs may be of dramatically different sizes, comparing their raw throughput may be of limited utility. For instance, consider two accelerators: (1) an accelerator A which performs 1 unit of work per second while operating at 1 Watt and (2) an accelerator B that performs 10 units of work per second while operating at 100 Watts. Accelerator B is clearly faster because it performs more work per second but uses more energy per unit of work (i.e. 10 units of work / s expends 100 J / s so energy efficiency is 10 J / unit of work). In contrast, accelerator A is more energy efficient since it only requires 1 J per unit of work.

Since many embedded systems operate on battery units, energy budgets are often a limiting factor that govern the available computation capabilities of these systems. Thus, in many embedded systems, achieving maximal energy efficiency is a key design consideration in order to maximize operating durations. For my evaluations, I will use energy efficiency to provide a normalized comparison of stochastic circuits again BE ones since each may have different operating power and run times.

2.3.4 Area-Normalized Throughput

To measure throughput, I use area-normalized throughput which is agnostic to variations in total accelerator area and run time. Area-normalized throughput is a measure of work per second per unit area. Actual accelerators may have different numbers of cores or processing units which makes a direct throughput comparison of limited utility. Area-normalized throughput provides a metric of how efficiently the silicon area is being utilized.

Area-normalized throughput is especially important when comparing SC against BE accelerators. This is because stochastic accelerators will have dramatically different run times due to its time multiplexed encoding and areas because of their high-density arithmetic units. As a result, area-normalized comparisons are more informative and will complement direct area comparisons in my evaluations going forward.

2.4 Summary of Challenges and Opportunities

Stochastic computing remains an emerging technology with significant design challenges that require managing the competing influences of many design tradeoffs. A summary of the tradeoffs outlined in this section is tabulated in Table 2.2. While stochastic circuits are usually smaller and more power efficient, whether they are decisively more energy efficient than BE arithmetic is an open question. This is because operations on stochastic circuits take more cycles to execute unlike single cycle arithmetic operations.

Stochastic computing also presents opportunities to exploit that allow it to gain competitive advantages over BE circuits. For instance, they are more error tolerant which allow them to operate in potentially harsher error prone environments or save energy by reducing the operating voltage while exploiting the encoding's error tolerance. An empirical analysis of the extent to which error tolerance can be exploited to improve energy is discussed in Section 4.3.

Finally, the accelerator design space of SC accelerators is not well explored and the design points at which stochastic computing offers superior energy efficient, area, and throughput remains an open question for many application domains. Furthermore, there are few studies which directly

Table 2.2: Summary of tradeoffs in stochastic circuits relative to binary-encoded ones.

Quantity	Tradeoff	Reason
Throughput	Lower	Exponential longer bitstream lengths to increase precision
Density	Higher	1-bit datapaths
Power	Lower	1-bit datapaths
Accuracy	Usually lower	Quantization errors and correlation considerations
Error Resilience	Higher	Redundant encoding and bit flips only minimally change value
Energy Efficiency	Sometimes Better	Lower power but longer execution time

compare BE and SC, and expose the architectural tradeoffs to consider when building energy optimal accelerators. A key challenge to performing these types of studies is the number of design parameters which can be changed while maintaining an informative comparison. Since stochastic and binary-encoded circuits do not have the same throughput, area, and power, they require normalized metrics to provide a fair comparison between technologies. In the following chapters, I will explore and evaluate the impact of these tradeoffs.

Chapter 3

NOVEL STOCHASTIC CIRCUIT COMPONENTS

A key challenge with stochastic circuits are errors due to correlation or quantization which can reduce accuracy compared to equivalent BE circuits operating at the same precision. Reducing errors due to quantization effects is undesirable since this can only be achieved by increasing the bitstream length exponentially with the target operating precision. However, there are many opportunities to reduce errors due to correlation by improving the number generator sequences, injecting desirable correlation, or using correlation insensitive stochastic circuits.

Improving the accuracy of stochastic circuits has immediate practical benefits since iso-accurate computation with BE arithmetic can be done with shorter bitstreams. This ultimately improves the throughput, run time, and energy efficiency of the computation. The accuracy of stochastic circuits are typically improved in two ways: (1) a novel circuit design or (2) improved random number generator sequences. In this chapter, I present both novel stochastic circuits and new number sequence combinations to improve the accuracy of existing stochastic operations.

3.1 A New Adder Design

Addition is a ubiquitous arithmetic primitive which mandates an accurate solution to make most applications viable. Recall that the input and output precision of stochastic circuits must be the same so either the most significant or least significant bit of an addition is lost due to quantization. As a result, there are two variants of addition: a MSB-preserving scaled addition proposed in [38] which realizes $f(p_X, p_Y) = \lfloor N \times \frac{p_X + p_Y}{2} \rfloor / N$, and a LSB-preserving saturating addition proposed in [6] which realizes $f(p_X, p_Y) = min(p_X + p_Y, 1.0)$.

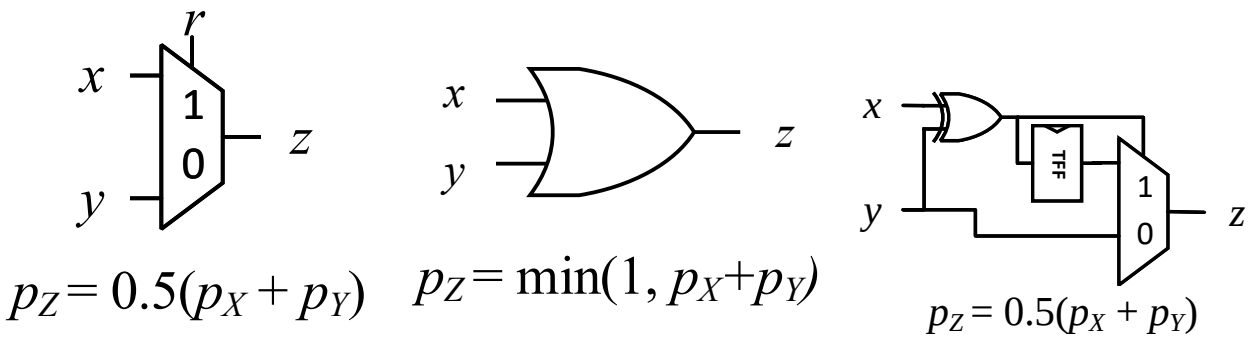
The conventional scaled addition is implemented as a two-input multiplexor where the auxiliary select bit of the multiplexor is driven by a bitstream with value 0.5 that is uncorrelated with the

other two inputs (Figure 3.1(a)). This effectively yields a circuit that randomly samples each bitstream input with equal probability yielding the functionality $f(p_X, p_Y) = \lfloor N \times \frac{p_X+p_Y}{2} \rfloor / N$ where $\lfloor x \rfloor$ denotes the floor function. In contrast, the saturating adder design (Figure 3.1(b)) requires negatively correlated inputs. The key idea is to misalign as many bits in the input bitstreams as possible (negatively correlated). This allows the bits to spread out across the bitstream maximally so that the resulting value after the OR gate will encode the maximum number of 1s possible in the output bitstream.

Both previous adder designs rely on properly correlated input bitstreams which cannot be guaranteed in practical application settings where undesirable correlation may compound over successive computations. To improve the accuracy of stochastic scaled addition, I evaluate a new scaled stochastic addition circuit which is agnostic to correlation levels. The new scaled adder design is shown in Figure 3.1(c) and is a modification of the multiplexor-based scaled adder. The key difference is the addition of a T-flip-flop (TFF) which functions as a 1-bit saturating counter and replaces the auxiliary input to always pass a bit when the counter saturates. Since the saturation condition occurs only every two bits, the circuit achieves the desired scale factor of 2 and emits a bitstream with a value that is exactly $\lfloor N \times \frac{p_X+p_Y}{2} \rfloor / N$. This scaled addition yields the correct result regardless of the correlation of the input bitstreams.

Note that a residual bit may end up saved in the TFF after computation has concluded (hence the floor function). For instance, if the TFF is initialized to zero, the resulting computations on average will yield negatively biased bitstreams because resulting bitstreams will round down (i.e. a residual bit can remain in the TFF after computation). If the TFF is initialized to one, the resulting computation will yield a positively biased bitstream value on average. In this case, the initialized TFF will force the residual bit to round up during computation and yield the value $\lceil N \times \frac{p_X+p_Y}{2} \rceil / N$.

To compare the accuracy of the correlation insensitive scaled adder with existing designs, I exhaustively calculate the MSE as described in Subsection 2.3.1. Table 3.1 compares the accuracy of the new proposed adder against existing adder designs using previous RNG combinations. In terms of accuracy, the new design is optimally accurate because it will only ever yield results that



(a) Scaled Addition (b) Saturating Addition (c) New Correlation Insensitive Scaled Addition

Figure 3.1: Comparison of stochastic adder designs.

Table 3.1: MSE of new stochastic adder for different RNG combinations (lower is better).

Design	X RNG	Y RNG	MSE N = 256	MSE N = 16
Old Adder (Figure 3.1(a))	Random	LFSR	3.24×10^{-4}	5.55×10^{-3}
	Random	TFF	5.49×10^{-4}	5.49×10^{-3}
	LFSR	TFF	1.06×10^{-4}	2.66×10^{-3}
New Adder (Figure 3.1(c))	Any	Any	1.91×10^{-6}	4.88×10^{-4}

are off due to rounding issues. I show how this improve adder can be used to improve the accuracy of applications in Section 5.3.

3.2 Improving Multiplier Accuracy

I now propose a more accurate stochastic multiplier by improving the RNG number sequence combination. Previous work uses combinations of LFSRs and low discrepancy sequences to achieve highly uncorrelated bitstreams in order to achieve accurate multiplication. In my work, I propose using a ramp sequence [35] and Van der Corput sequence to drive the SN generation for the

multiplier. A ramp sequence is the number sequence generated by a counter that increments by one each cycle.

Table 3.2: MSE of stochastic multiplier for different RNG methods (lower is better).

X RNG	Y RNG	8-bit MSE	4-bit MSE
LFSR	Shifted LFSR	2.78×10^{-3}	2.99×10^{-3}
LFSR	LFSR	2.57×10^{-4}	1.60×10^{-3}
Van der Corput	Halton3	1.28×10^{-5}	1.01×10^{-3}
Van der Corput	Ramp [35]	8.66×10^{-6}	7.21×10^{-4}

I compare the proposed configuration against several configurations of number generators which have been used in prior work. The first configuration uses an LFSR of appropriate length with a rotated version of itself. The second configuration uses two LFSRs with different tap configurations. The third configuration uses a Van der Corput sequence combined with a Halton sequence with base 3 which I will refer to as Halton3. For each configuration, I exhaustively evaluate all possible input values and calculate the mean squared error over all cases.The accuracy results are shown in Table 3.2.

Compared to the configurations in prior work, the new proposed configuration using a Van der Corput and ramp sequence achieves better mean squared error. LFSR combinations due not yield as accurate results because the sequences generated by rotated LFSRs or LFSRs with different tap configurations still remain weakly correlated. I will later show how the more accurate multiplication can be used to improve the overall application accuracy in the context of a neural network in Section 5.3.

3.3 Correlation Manipulating Circuits

A common design challenge when designing multilevel stochastic computation logic is compounding correlation levels across successive computations. In addition, many stochastic arithmetic circuits

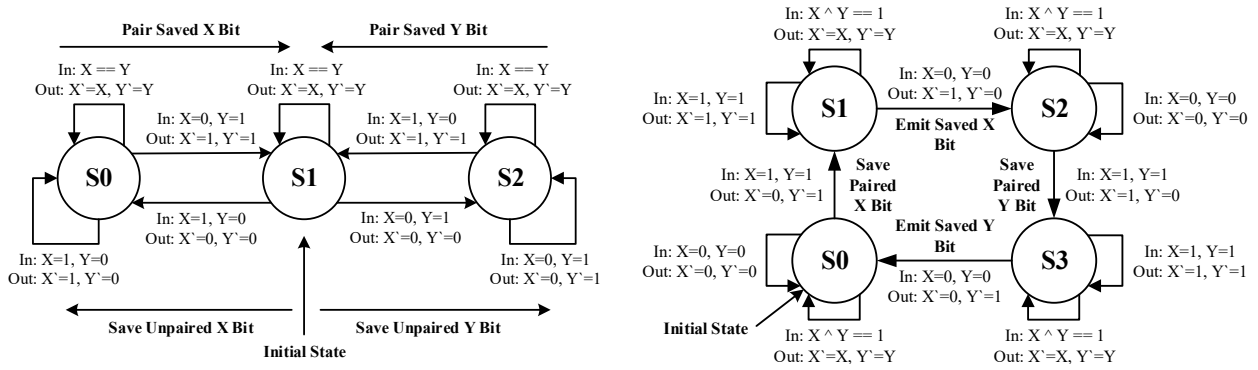
have optimal correlations under which they operate optimally under and yield significant errors under other correlation conditions. To inject proper levels of correlation between bitstreams between stochastic circuits, designers rely on correlation manipulating circuits. In this section, I propose several new correlation manipulating circuits which can be used to inject correlation between two bitstreams, and are less expensive to implement or more effective than prior techniques.

3.3.1 Synchronizer and Desynchronizer

Many stochastic circuits such as the subtractor require positively correlated input operands to operate accurately while others such as the saturating adder require negatively correlated input operands to operate accurately. Furthermore, these operations may occur after successive computations where the correlation between SNs has been distorted by prior operations. Prior techniques such as regeneration require expensive conversions and run time overheads to execute. Other correlation manipulating circuits such as isolators are primarily used to engineer uncorrelated SNs. As a result, there is an important need to provide correlation manipulating circuits that can inject positive or negative correlation on existing SNs. In this section, I propose a new synchronizer circuit for injecting positive correlation between SNs and a desynchronizer for injecting negative correlation between SNs. I also show how the designs can be generalized and composed to improve their efficacy, and how they can be used to improve the accuracy of existing SC operations.

The finite state machine specification for the synchronizer circuit design is shown in Figure 3.2(a). Given two input SNs X and Y, the synchronizer produces two SNs X' and Y' which are more positively correlated and have the same value as the input SNs X and Y respectively. The key idea is to dynamically pair up 1s and 0s from the two input streams as often as possible between the two input SNs. When X and Y inputs are the same, the synchronizer simply passes them to the outputs. If X and Y are different, the synchronizer "saves" the unpaired bit to be paired with a complement at a later SN index. Pairing up bits as often as possible effectively induces positive correlation between the two SNs.

The desynchronizer relies on a similar principle to increase negative correlation between two input SNs (Figure 3.2(b)). Instead of trying to dynamically pair up bits, the desynchronizer



(a) Synchronizer instance with save depth $D = 1$.

(b) Desynchronizer instance with save depth $D = 1$.

Figure 3.2: Correlation inducing stochastic circuit designs: (a) synchronizer and (b) desynchronizer.

deliberately tries to *unpair* bits and emit as many unpaired bits as possible. To do this, when the X and Y inputs are both 1 the desynchronizer saves one of the bits and passes the other. If the inputs are both zero, it takes a saved bit if available and overrides one of the zeros. Finally, when input bits are different, the desynchronizer simply passes them since they are already unpaired. By unpairing as many bits as possible, the resulting SNs become negatively correlated.

To quantify their efficacy, I measure average SCC before and after each circuit, and the average *bias*. Bias is defined as the average deviation in the output values from the input values. Ideally, the bias should be zero since the circuits should only alter the SN correlation and not SN value. Table 3.3 shows the initial and resulting SCCs, and average bias of result SNs for several RNG configurations averaged over all possible input values.

To measure induced correlation, I deliberately choose RNGs that initialize input SNs to be minimally correlated for the synchronizer and desynchronizer. I also provide one RNG configuration where the inputs are initially positively correlated to test the desynchronizer. In general, I find that the synchronizer and desynchronizer designs work well across all configurations. I also observe stronger induced correlation when the input SNs have low discrepancy (i.e. generated with Halton and VDC RNGs). The results also show that on average there is a slight negative bias in the resulting SNs. This is due to saved bits getting "stuck" in the FSM at the end of the computation.

Table 3.3: Average SN correlation before and after correlation manipulating circuits (N = 256).

Design	X RNG	Y RNG	Input SCC	Output SCC	X' Bias	Y' Bias
Synchronizer (Figure 3.2(a))	VDC	Halton	-0.048	0.996	-0.001	-0.002
	LFSR	VDC	-0.062	0.903	-0.002	-0.001
	Halton	Halton	0.984	0.992	-0.002	-0.002
Desynchronizer (Figure 3.2(b))	VDC	Halton	-0.048	-0.981	-0.002	0
	LFSR	VDC	-0.062	-0.788	-0.002	0
	Halton	Halton	0.984	-0.930	-0.003	0
Decorrelator (Figure 3.3a)	LFSR	LFSR	0.992	0.249	0.000	-0.004
	VDC	VDC	0.992	0.168	0.001	0.003
	Halton	Halton	0.984	0.067	0.001	0.002
Isolator Insertion	LFSR	LFSR	0.992	0.600	-0.002	0.000
	VDC	VDC	0.992	-0.637	-0.004	0.000
	Halton	Halton	0.984	-0.353	0.002	0.000
Tracking Forecast Memory [103]	LFSR	LFSR	0.992	0.654	-0.014	-0.051
	VDC	VDC	0.992	0.779	0.246	0.363
	Halton	Halton	0.984	0.353	-0.005	-0.007

3.3.2 Decorrelator

I now introduce a decorrelator circuit which is designed to take two SNs and emit two less correlated SNs. The decorrelator circuit is composed of two shuffle buffers (Figure 3.3b). A shuffle buffer is a small memory which randomly replaces and emits bits. At each cycle, the shuffle buffer will either pass the current input or store the current input and emit a previously stored input. The depth D of the memory can be parametrized; intuitively, a deeper memory allows the shuffle buffer to scramble bits across longer segments of the SN. To determine which bit to store and emit at a given cycle, the

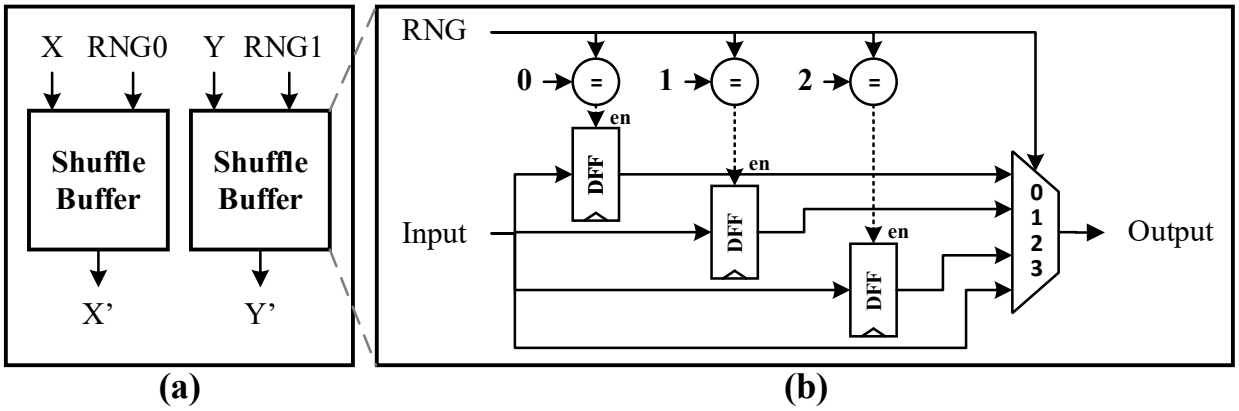


Figure 3.3: (a) Decorrelator design with (b) shuffle buffer depth $D = 4$.

decorrelator requires an auxiliary RNG input. To decorrelate two different SNs, I use two shuffle buffers with different RNGs.

The efficacy of the decorrelator design for two SNs is shown in Table 3.3 for several correlated RNG configurations. On average, I find that the decorrelator significantly reduces the correlation between SN operands. To reduce bias errors, I initialize half of the buffer in the design to 1s and the others to 0s so that on average fewer 1s from the input SNs will get "stuck" in the buffer. Like the synchronizer and desynchronizer, decorrelator designs can be composed in series to further reduce correlation. I also evaluate isolators and tracking forecast memories [103] for decorrelating SNs, and find they are both less effective than the proposed decorrelator. Finally, the decorrelator is a streaming unit which does not need to capture the entire SN before emitting the new SNs. This is unlike regeneration where the entire bitstream must first be accumulated before reconstructing a new SNs which effectively halves the throughput of regeneration compared to the decorrelator circuit.

3.3.3 Generalized Designs and Composition

The synchronizer and desynchronizer designs can both be generalized or composed to improve the strength of the induced correlation. To generalize these designs, the key idea is to extend the FSMs

to save additional bits. For the synchronizer, this is equivalent to adding an equal number of states to the left and right of the FSM to track how many bits from X and Y have been saved. For the desynchronizer, I add additional FSM states and transitions to the FSM cycle to represent other possible saved bit configurations. I refer to the number of bits that a synchronizer and desynchronizer can save as the save depth D. The key idea is that by having a larger save depth, the designs will be more resilient to runs of 1s and 0s which reduce their efficacy.

These extensions present their own challenges: for designs with large D, it is possible for the saved bits to get "stuck" in the FSM before they can all be paired or unpaired. In adversarial cases, this can result in larger biases from the original value if the saved bits in the FSM are not emitted before the end of the SN. To mitigate this issue, one could add an optional FSM flush functionality which requires keeping track of the current offset into the bitstream t. If the number of remaining saved bits in the FSM is greater than or equal to t, it forces the FSM to emit the saved bits regardless of saved state. However, these modifications require additional overheads and can become tremendously expensive for large save depth D.

An alternative way to enhance correlation strength is to compose multiple synchronizers or desynchronizers with minimal depth $D = 1$ together in series. Each synchronizer or desynchronizer will improve the correlation albeit with diminishing returns. In the limit, output SNs will eventually become maximally correlated. However, like the FSM extensions, it is still possible for residual bits to remain in each FSM resulting in compounding bias errors. To address this issue, the FSMs can be modified to start with a saved X or Y bit by adjusting the initial state.

3.3.4 Improved Maximum, Minimum, and Saturating Add

I now show how to improve stochastic arithmetic operations using the proposed correlation manipulating circuits in the context of a maximum, minimum, and saturating addition. The improved stochastic maximum combines the synchronizer design with an OR gate (Figure 3.4(a)) while the stochastic minimum combines the synchronizer with an AND gate (Figure 3.4(b)). For the stochastic maximum, the key insight is that for input SNs with maximal, positive correlation, the larger SN will exactly mask the smaller SN when taking the OR of the two SNs. The OR gate will

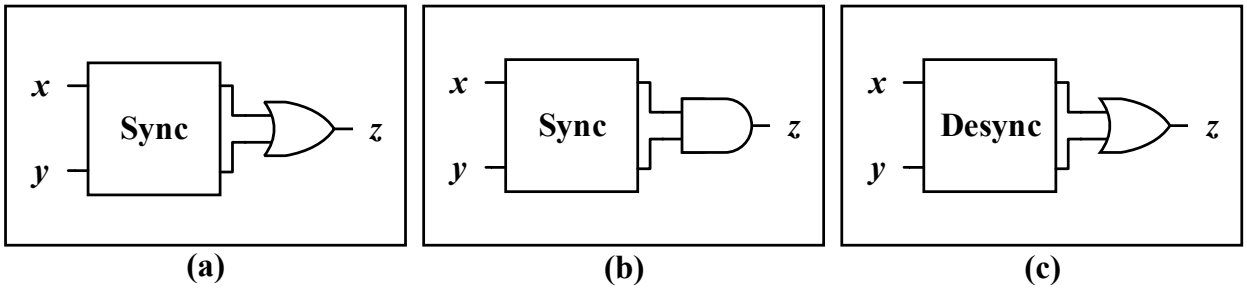


Figure 3.4: Improved stochastic circuits: (a) maximum, (b) minimum, and (c) saturating add.

also propagate any extra 1s in the larger SN. If input SNs are not positively correlated, the resulting SN will be less accurate and have a value strictly *greater than or equal to* the larger input SN. Thus, I use the new synchronizer design to induce positive correlation between the two input SNs before feeding the them to the OR gate.

A similar concept is used to calculate the minimum between two SNs. Instead of an OR gate, I use an AND gate to pass *at most* the maximum number of masked 1s between the SNs; this results in a SN with the minimum value. Finally, I prepend a desynchronizer to the OR gate to improve the stochastic saturating adder which requires negatively correlated inputs (Fig. 3.4c).

To evaluate accuracy, I exhaustively generate all possible inputs using two uncorrelated RNGs - a VDC sequence and Halton3[1] - and report average absolute error. I also evaluate the design power, area, energy, and accuracy across several maximum designs: (1) single OR gate, (2) the improved synchronizer-based maximum, and (3) the correlation insensitive maximum (CI maximum) in [88]. For stochastic minimum, I compare (1) a single AND gate and (2) the synchronizer-based minimum.

Table 3.4 compares the improved synchronizer-based maximum and minimum against prior work, and shows that the proposed designs are more accurate than just using an OR gate without correlation manipulation. Compared to the correlation insensitive maximum, the synchronizer-based design is 5.2× smaller and 11.6× more energy efficient with minimal accuracy loss. Using a deeper save depth can improve accuracy but also increases power, area, and energy relative to other designs.

[1]Halton sequence with base 3

As with many aspects of SC, this illustrates a design tradeoff: more accurate stochastic circuits are larger and consume more energy. I also observe similar results for the synchronizer-based minimum design.

Table 3.4: Average absolute error, bias, area, power, and energy for stochastic maximum and minimum designs for N = 256.

Design	Abs. Error	Avg. Bias	Area (μm^2/op)	Power (μW/op)	Energy (pJ/op)
OR Max.	0.087	0.087	2.16	0.26	165
CI Max.	0.006	0.001	252.36	56.7	36288
Sync. Max.	0.003	0.003	48.6	4.89	3130
AND Min.	0.082	-0.082	2.16	0.25	158
Sync. Min.	0.005	0.005	45.0	8.38	5363

3.3.5 Improving Application Accuracy and Energy Efficiency

I evaluate the power, area, accuracy, and energy of a Gaussian blur (GB) followed by a Roberts cross edge detector (ED) [14] stochastic accelerator for several configurations. This image processing pipeline illustrates the role of correlation manipulation since the stochastic Gaussian blur requires inputs to be uncorrelated while the edge detector requires positively correlated inputs. The proposed accelerator architecture expects the input image to be tiled and processes each tile individually one at a time. All outputs within each tile are computed in parallel before moving on to the next tile. For this evaluation, I use a 10×10 input tile size.

I evaluate several stochastic accelerator variants: (1) an accelerator with no correlation manipulating circuits between GB and ED, (2) an accelerator that uses regeneration between the GB and ED, and (3) an accelerator that uses synchronizers to manipulate correlation between GB and ED. For each design, I synthesize, place, and route each accelerator using a TSMC 65nm library

with Synopsys Design Compiler, IC Compiler, and PrimeTime. For power measurements, I use post-placement and route simulation using random traces to make measurements data agnostic. To model quality, I use a cycle-level simulator which uses models that have been verified against RTL simulation traces. I report accuracy in terms of the average absolute error of the SC result compared to a floating-point baseline image.

Table 3.5: Image results for Gaussian Blur followed by Roberts Cross Edge Detector. Using synchronizers is more energy efficient than using regeneration.

Design	Floating Point (Baseline)	Stochastic No Correlation Correction	Stochastic Using Regeneration	Stochastic Using Synchronizer
Image Result				
Area	–	24313 μm^2	34802 μm^2	36202 μm^2
Energy	–	1383 nJ/frame	1971 nJ/frame	1505 nJ/frame
Abs. Error	0	0.076	0.019	0.020

Table 3.5 shows the quality, energy, and area results for the combined GB and ED accelerator. I find that the image quality and average absolute error is markedly better when using regeneration or synchronizers between image processing kernels than the results generated by the design without correlation manipulating circuits. In terms of energy, the synchronizer-based design improves total accelerator energy consumption by 24% over the design using regeneration. I also find the average absolute error difference between the design using regeneration versus the new design is negligible.

I also compare the overheads of synchronizer circuits and regeneration. To do this, I tabulate the power break down for S/D converters, D/S converters, compute circuits, RNGs, and synchronizers. I then aggregate the costs associated only with correlation manipulation. The two designs require different numbers of S/D and S/D converters, and synchronizers to accomplish the same task. In this case, the synchronizer-based design requires 2× more synchronizers than the number of S/D and D/S converters used by regeneration. This is because synchronizers only induce correlation between two SNs while using regeneration induces correlation between *all* SNs. Fortunately, for many applications like the ED kernel, it is sufficient to induce correlation between pairs of SNs. As a result, I find that the overhead of correlation manipulation using synchronizers is 3.0× more energy efficient than when using regeneration.

3.4 Related Work

This is not the first work to exploit correlation manipulating circuits to improve the fidelity of stochastic computation. Ting and Hayes [110] introduce an algorithm for placing isolator circuits to decorrelate SNs. However, isolators are limited in that they only shift bits by a fixed offset and do not scramble relative bit order. The decorrelator proposed in my work can scramble bit order across larger segments of the SN. Tehrani et al. [103, 105] propose edge memories and tracking forecast memories (TFM) for improving SN correlation in low-density parity-check (LDPC) decoding. Since TFMs were designed specifically for LDPC decoding, they do not generalize as well as the new decorrelator design. TFMs are also larger since portions of the TFM require BE arithmetic. Finally, Parhi et al. [81] propose a technique for synthesizing correlated and uncorrelated SNs but do not propose generic correlation manipulating circuits.

3.5 Summary

This chapter proposes new stochastic circuits and configurations to improve the implementation efficiency of SC operations and improve the accuracy of existing stochastic circuits. Unlike BE arithmetic, SC does not support all arithmetic operations equally and in some cases can restrict the types of computations that can be accurately implemented. Improving the implementation cost

of operations and capabilities of stochastic computation thus improves the practicality of SC by enabling implementations of new applications. More precisely, new circuits can improve upon existing arithmetic units by making them more accurate, more energy efficient, consume less power, or reduce their size.

My work proposes several new circuits that improves the power, area, accuracy, and/or energy of existing circuits. I first presented an improved stochastic adder which is correlation insensitive. This new adder is larger and consumes more power, but provides superior accuracy compared to previous adder implementations. In addition, this chapter proposed an improved multiplication RNG configuration using Van der Corput and ramp sequences. I also proposed several new correlation manipulating circuits which can be used to inject or correct correlation of intermediary bitstreams to accurately implement operations with specific correlation affinities. I then showed how these correlation manipulating circuits can be used to improve the accuracy, power, and area of existing arithmetic units such as maximum, minimum, and saturating addition. Together, these new circuits improve the practicality of SC by improving the implementation cost and providing either more accurate, more energy efficient, smaller, or lower power solutions than existing solutions.

Chapter 4

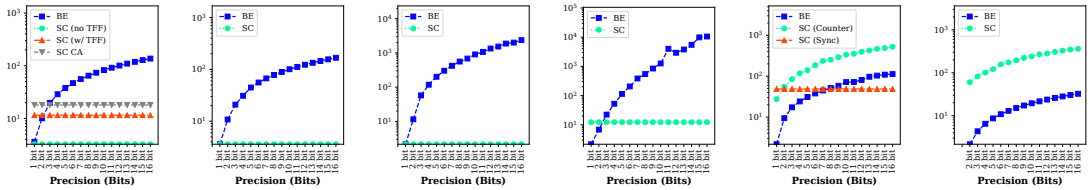
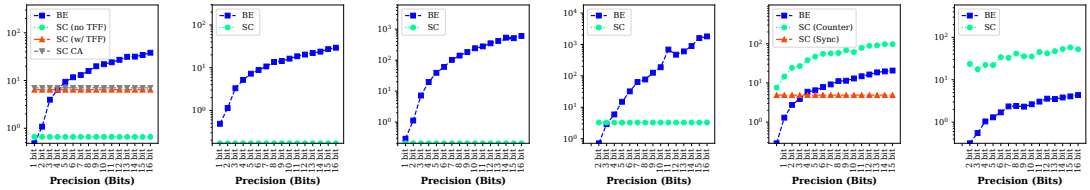
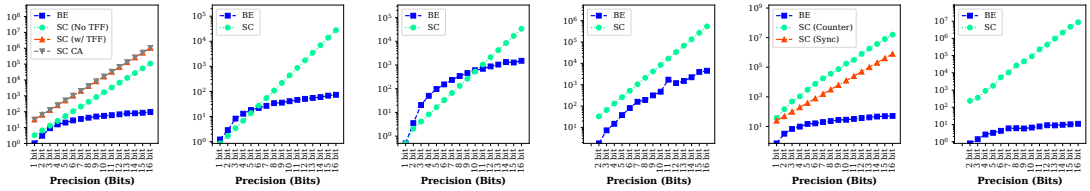
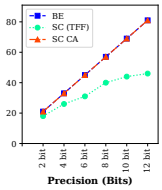
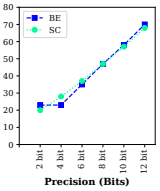
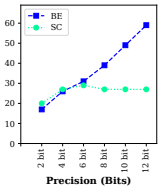
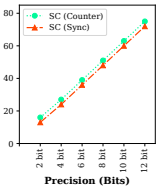
ARCHITECTURAL CONSIDERATIONS FOR STOCHASTIC COMPUTING

To understand the practical application spaces for SC accelerators, it is important to first understand the architectural design space where SC provides benefits in power, area, and/or energy efficiency over BE computation. In this chapter, I evaluate a microbenchmark of well-known stochastic arithmetic circuits to tabulate the power, area, and energy cost per operation. I then conduct a larger end-to-end accelerator design space exploration and present empirical measurements to highlight the tradeoffs of SC accelerators. The architectural characterizations and lessons exposed by these evaluations are then used to provide better insight as to *when* and *why* SC accelerators can yield compelling benefits over BE accelerators. These results are synthesized as design guidelines which serve as a blueprint to guide accelerator designs for potential applications.

4.1 Cost Per Operation Characterization

To understand the architectural tradeoffs of SC accelerators, I first evaluate the intrinsic cost per operation in stochastic circuits and compare it against equivalent BE circuits. To do this, I formulate a set of hardware microbenchmarks to measure energy efficiency, power, accuracy, and area at different input precisions (Table 4.1). In this analysis, I compare results using binary-encoded and stochastic circuits operating at the same precision (iso-precision). For instance, an m-bit BE circuit is compared against an stochastic circuit using SN length $N = 2^m$. I refer to both designs as m-bit designs as they both operate at effectively m-bits of precision. For each benchmark, I measure post-placement and route, area, and power using Synopsys Design Compiler, IC Compiler, and PrimeTime using a TSMC 65nm library. I use a 400 MHz frequency target and random input values to generate activity traces for power measurement. To measure PSNR, I calculated MSE over every

Table 4.1: Iso-precision power, area, energy efficiency, and accuracy comparison of common binary-encoded and stochastic arithmetic circuits operating at 0.8V, 25C. Not all stochastic circuits are smaller and lower power than binary-encoded equivalents. Most stochastic circuits are not more energy efficient than binary-encoded equivalents.

	Adder [37, 54]	Subtractor [9]	Multiplier [37]	Divider [27]	Maximum [6, 88]	ReLU [59]
RNG	LFSR/LFSR /TFF	VDC/VDC	VDC/Ramp	LFSR/LFSR	VDC/VDC	VDC
$f(p_X, p_Y)$	$\frac{p_X+p_Y}{2}$	$p_X - p_Y$	$p_X p_Y$	$\lfloor p_X/p_Y \rfloor$	$max(p_X, p_Y)$	$max(p_X, 0)$
Area (μm^2)						
Power (uW)						
Energy (fJ/op)						
PSNR (dB)		Exact BE/SC PSNR = ∞				Exact BE/SC PSNR = ∞

set of inputs using well-known RNG configurations. I use deterministic RNGs such as Van der Corput (VDC), Halton (HLT), linear feedback shift registers (LFSR), and ramp sequences. Using deterministic RNGs means the output result will not fluctuate across executions as opposed to using truly random number sequences where different executions will yield different results for the same input.

I first compare the impact of numerical precision on energy efficiency for commonly used arithmetic operations: addition and multiplication. For BE arithmetic units, I use the circuit microarchitectures synthesized by the Synopsys DC Compiler. For stochastic addition, I evaluate the multiplexor-based stochastic adder in Figure 2.1(b) with and without a T-flip-flop (TFF) select signal, and the correlation insensitive adder [54]. The results show that the *individual energy cost per operation for stochastic addition is actually worse than equivalent binary-encoded circuits*. At best, the margin between stochastic and binary-encoded addition circuits is only 1.44× at 3-bit precision but quickly grows to 10× at 8 bits of precision. I also find that the more accurate correlation insensitive adder is up to 10.6× less energy efficient than the multiplexor-based stochastic adder because it contains a TFF that requires a state element. Stochastic multiplication on the other hand exhibits better energy per operation gains over BE multiplication between 2 and 10 bits of precision which is roughly in line with the findings of [32].

I also evaluate several other stochastic circuits and compare their power, area, and energy savings relative to BE circuits in Table 4.1. I find that arithmetic operations like multiplication and division are able to achieve better power and area overheads compared to their BE counterparts. However, the results generally show *individual stochastic circuits are not more energy efficient per operation than their binary-encoded counterparts (except for multiplication)*. I also show that certain stochastic circuits such as the counter-based stochastic maximum [88], and stochastic rectified linear unit (ReLU) [59] are neither more energy efficient, lower power, nor denser than their equivalent BE circuits. Finally, the accuracy results show that different stochastic operators realize significantly different accuracies.

In addition to compute units, stochastic circuits have additional overheads in D/S and S/D conversion circuits (Figure 2.1(a) and Figure 2.1(c)). Many prior works do not report the magnitude

of these overheads or ignore the costs of these circuits all together. On average, I find S/D converters require 8.14× and 4.41× more power and area respectively than D/S converters. Relative to a stochastic multiplication, S/D converters are 13.8× larger and require 5× more power at 8-bit precision. While stochastic accelerators can amortize these overheads over many arithmetic operations, the power and area costs of these conversions cannot be completely ignored as I will show later.

These observations do not necessarily imply that stochastic accelerators are always less energy efficient than BE accelerators. Actual accelerator architectures are composed of many other elements such as control units, buffers, and pipeline registers which is evaluated in next.

4.2 End-to-End Accelerator Architecture Design Space Exploration

To fully understand the architecture design tradeoffs of SC accelerators, I evaluate end-to-end accelerator designs including all necessarily overheads such as conversion units, pipelining, and random number generators. I evaluate accelerator implementation for six application workloads commonly used in image processing, machine learning, and computer vision in prior works [7, 53, 109]: a 3×3 Gaussian blur (GB), a 2×2 geometric interpolation (INT), a modified Roberts cross edge detection (ED), a 5×5 general convolution (CONV), a modified Sobel kernel (SBL), and a matrix-vector multiplication (MV). These applications contain abundant parallelism which provides a good opportunity to evaluate the architectural design space and tradeoffs for stochastic accelerators.

In terms of evaluation metrics, I measure power, area, and run time. I also further breakdown power and area measurements into different accelerator components like computation and conversion overhead. In addition, I compare normalized metrics such as energy efficiency and area-normalized throughput (throughput/area) which are agnostic to design parameters (Section 2.3). This is because iso-precision stochastic and binary-encoded accelerators can have different operating power, area footprint, and number of functional units. Each of these metrics also takes into account stochastic and binary-encoded accelerator overheads such as conversion circuits, RNGs, pipelining registers, buffers, and control circuitry.

For each binary-encoded and stochastic accelerator design, I evaluate design points with 1 to 16 bits of input precision. For RNGs, I use deterministic sources such as VDC, Halton3, and ramp sequences. For kernels with multipliers, I use Van der Corput and ramp sequences as proposed in [54]. For kernels which only require positively or negatively correlated inputs, I use VDC to generate both input SNs. To evaluate accelerator area, I synthesize, place, and route each stochastic and binary-encoded accelerator design using a TSMC 65nm technology library, Synopsys Design Compiler, and Synopsys IC Compiler. To obtain data agnostic power results, I use random data traces using the Synopsys PrimeTime power estimator. The accuracy results of each workload are calculated using a custom-built cycle-level model for stochastic circuits. The models were verified against cycle-level RTL hardware simulations.

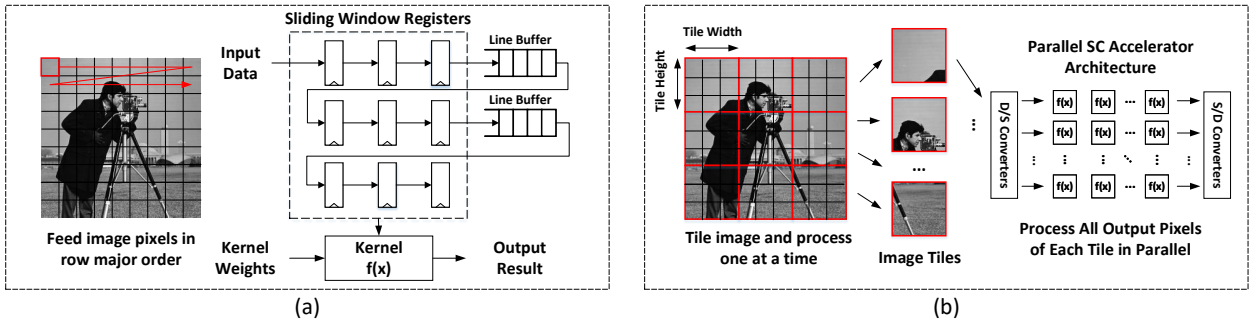
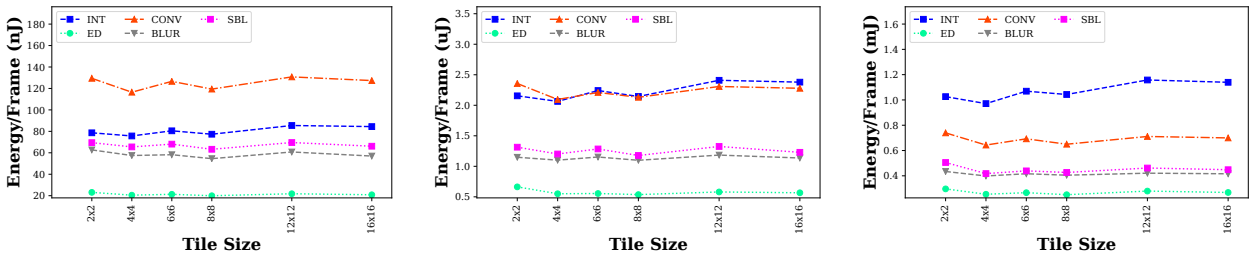


Figure 4.1: Image processing accelerator architectures: (a) binary-encoded sliding window architecture and (b) tiled stochastic computing architecture.

The architectures for both stochastic and binary-encoded accelerators are shown in Figure 4.1. The BE image processing accelerators use a line-buffered, sliding window architecture similar to the architectures in [41]. In this architecture, image values are fed into the accelerator serially in row major order and line-buffered to maximize data reuse; this results in a compact accelerator architecture. For BE matrix-vector multiplication, I use a simple SIMD-like architecture with several multiply-accumulate units processing matrix rows in parallel. The input matrix is batched by row and fed into the accelerator in row major along with the kernel.

In contrast, the SC accelerator architectures for this evaluation have more parallel datapaths since SC's inherent density enables many more arithmetic units to be instantiated per unit area than BE circuits. To exploit this parallelism, the SC accelerator architectures tile the input image or matrix, and computes each output within that tile in parallel. For instance, for the 3×3 Gaussian blur, the SC accelerator would process a 10×10 input tile of the image and compute all 8×8 output pixels in parallel. This allows SC to also exploit maximal data reuse which is critical to amortize D/S data conversion costs since each pixel in the input tile is used by multiple convolution windows.



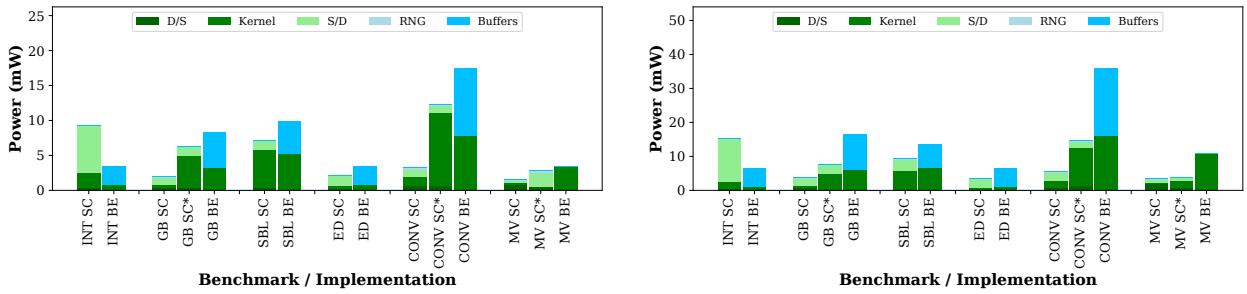
(a) N=16 (4-bit precision) (b) N=256 (8-bit precision) (c) N=65536 (16-bit precision)

Figure 4.2: Energy per frame for different stochastic computing accelerator tile sizes.

I determine the energy optimal tile width for SC across benchmarks by sweeping tile sizes 2, 4, 6, 8, 12, and 16 for SN lengths 16, 256, and 65536. The results in Figure 4.2 show that most energy optimal points across benchmarks occur at a 8x8 tile size followed by a 4x4 tile size. Thus, for my evaluations I compare SC accelerator architectures with an output window size of 8×8. Finally, for GB, CONV, and MV, I evaluate two accelerator design points: (1) with the smaller conventional stochastic scaled adder, and (2) with the larger correlation insensitive stochastic adder.

4.2.1 Evaluation Results

This section presents empirical measurements comparing each of the BE and SC accelerators architectures operating at different precisions.



(a) 4-bit precision

(b) 8-bit precision

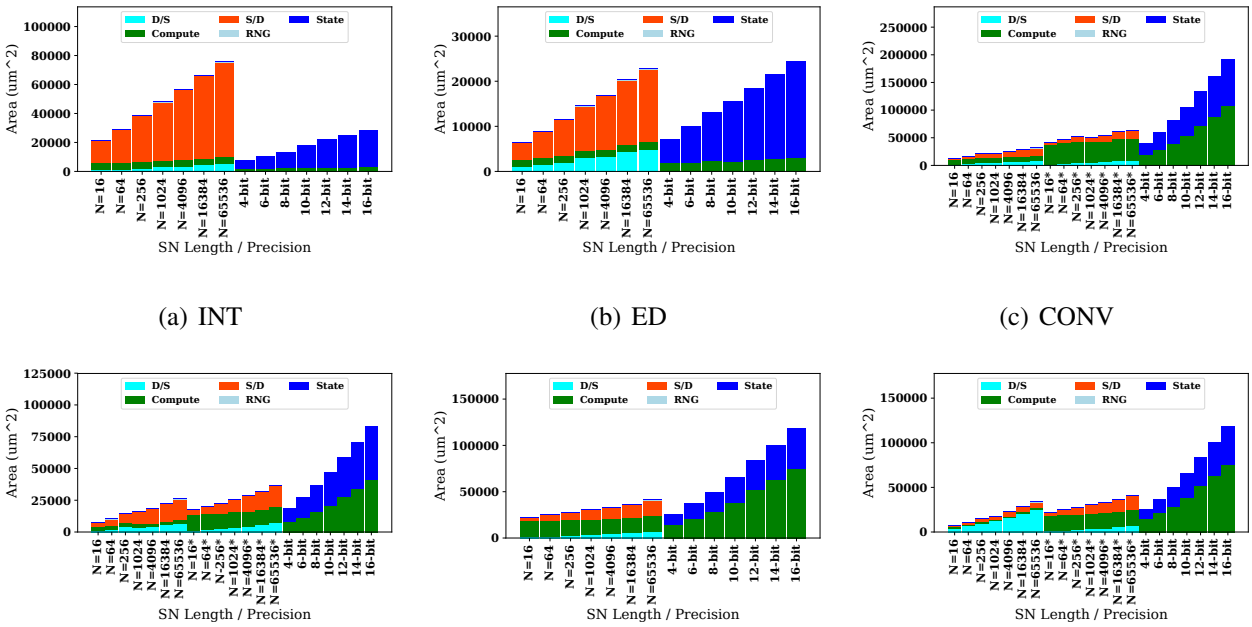
Figure 4.3: Power breakdown by component of stochastic computing accelerator designs.

Power

Figure 4.3 shows the breakdown of power usage decomposed into their individual components at 4-bit (N=16) and 8-bit (N=256) precision: S/D converters, D/S converters, RNGs, kernel computation, and buffers. I generally find that the power breakdown is similar across different precisions within the same design. For BE accelerators, results show that buffer overheads used to manage data reuse makes up a hefty fraction of the power usage relative to kernel computation. For SC accelerators, the results show that S/D and D/S conversion overheads are non-trivial. For example, the interpolation kernel which has 4× as many S/D converters as other kernels devotes a painfully high fraction of its power budget to conversion overheads. This ultimately results in poor energy efficiency gains (discussed later). On the other hand, these conversion overheads are relatively small for compute-heavy kernels like convolution which better amortize them. Finally, the results show RNG overheads are small compared to cross domain conversion overheads and compute kernels. This is because their costs can be amortized over many D/S conversion circuits.

Area and Density

In terms of density, results show that SC accelerators have an effective arithmetic operator density (arithmetic units / unit area) up to 212× higher than BE accelerators (not shown). In terms of total



(a) INT (b) ED (c) CONV

(d) BLUR (e) SBL (f) MV

Figure 4.4: Accelerator area breakdown of stochastic computing accelerators. Designs denoted with * use correlation insensitive adder.

accelerator area (Figure 4.4), SC accelerators are roughly the same size as BE accelerators despite having one to two orders of magnitude more arithmetic units. This operator density allows SC accelerator architectures to instantiate many more datapaths in parallel. The additional parallel datapaths also allows SC accelerators to improve throughput to make up for longer execution times per operation due to SN length. I therefore conclude, it is not only feasible but necessary for SC accelerators to capitalize on multiple parallel datapaths to efficiently support computation.

Throughput and Run Time

Unlike BE accelerators where run times are agnostic to precision, SC bitstream lengths are exponentially shorter at each bit of lower precision. Figure 4.6(a) shows the run time of the tiled architectures for different SC accelerator tile widths. For both SC and BE architectures, the run time is mostly

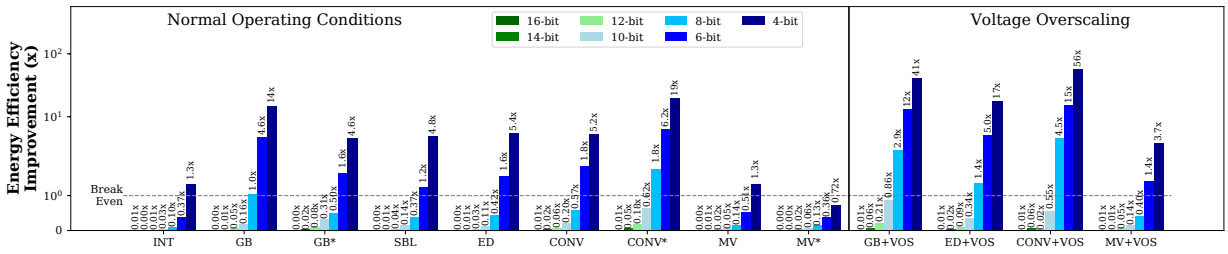
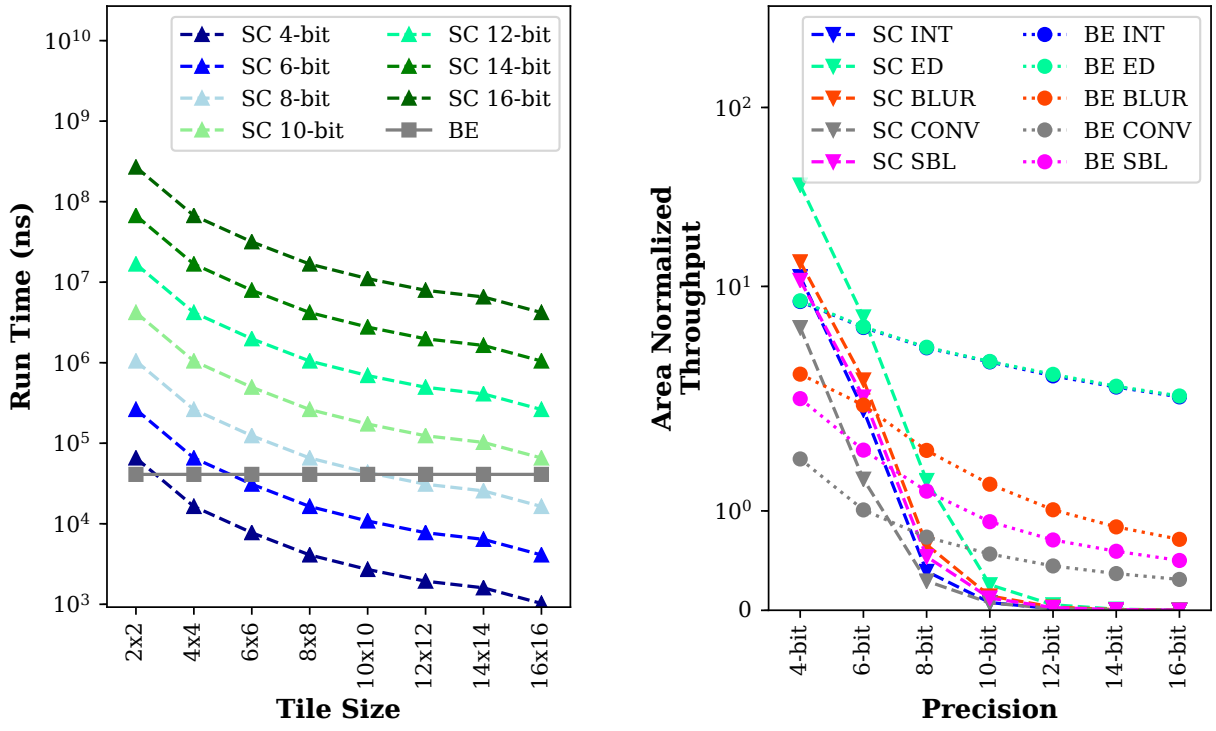


Figure 4.5: Energy efficiency improvement of stochastic computing accelerators over binary-encoded ones (higher is better). Stochastic computing accelerators operating at normal voltage break even with binary-encoded designs at 8-bits. Designs with * use the correlation insensitive adder. Voltage overscaled (VOS) designs are discussed in Section 4.3.

agnostic of the kernel so curves are combined for simplicity. For a 8×8 tile size, results show that SC image processing accelerators are 4× slower at 8-bit precision, break even at 6-bit precision, and are faster at 5-bit precision or lower. However, a run time comparison does not properly account for differences in accelerator area so I also estimate the area-normalized throughput by scaling the stochastic computing run time by binary-encoded over SC accelerator area (Figure 4.6(b)). In terms of area-normalized throughput, I find that SC accelerators achieve parity between 6 and 8 bits of operating precision and can be up to 10.9× more efficient at 4 bits of precision.

Energy Efficiency

Figure 4.5 shows the relative energy efficiency improvement of SC accelerators over their equivalent BE accelerators. In terms of energy efficiency, the results show that SC accelerators under normal operating conditions are able to achieve better energy efficiency at 8 or fewer bits of precision, and are less energy efficient at higher precision. Note that SC accelerator energy efficiency decreases exponentially for each bit of increased precision since SNs become exponentially longer. The results also show that kernels such as the Gaussian blur and convolution with many arithmetic operations tend to achieve better energy efficiency gains. In these particular kernels, the power and area expenditures of each distinct S/D and D/S conversion is amortized by 6.6 to 18.2 arithmetic operations because of data locality and reuse. In contrast, kernels like interpolation only have 1.11



(a) Run Time (ns)

(b) Area-Normalized Throughput

Figure 4.6: (a) Run time (lower is better) of different tile size stochastic computing accelerator designs, and (b) area-normalized throughput (higher is better) for different operating precision for stochastic computing accelerators using a 8x8 tile size.

Table 4.2: Image processing results and PSNR at 8-bit precision.

	Gaussian Blur (GB)	Edge Detection (ED)	Interpolate (INT)	Sobel (SBL)
Fixed Point				
	46.05 dB	55.41 dB	55.42 dB	50.92 dB
Stochastic				
	24.04 dB	56.26 dB	34.89 dB	12.45 dB

arithmetic operations per conversion, and achieve poor energy savings because of high conversion overheads. As a result, I conclude that workloads which have a significantly higher proportion of arithmetic operations to conversion circuits are more likely to have better energy savings in SC over BE computation.

The energy results also illustrate several other trends. First, SC accelerators using multiplexor-based adders achieve energy efficiency parity with BE accelerators at 8 bits of precision while accelerator designs using the correlation insensitive adder break even at 7 bits. Second, the results show that stochastic matrix-vector multiplication performs poorly relative to the BE SIMD accelerator. This is because the BE SIMD accelerator is a primarily compute-dominated accelerator and requires minimal state elements. Since individual BE arithmetic operators are more energy

Table 4.3: Support vector machine classification accuracy for UCI Machine Learning Repository datasets [64].

Dataset	Fixed-point		Stochastic	
	(8-bit)	(4-bit)	(8-bit)	(4-bit)
Breast Cancer	96.1%	96.9%	96.4%	96.0%
Diabetes	75.9%	77.3%	75.8%	71.7%
House Votes	95.3%	96.3%	94.1%	92.7%
Liver Disorder	73.1%	73.1%	71.9%	71.5%
Four Class	73.6%	73.6%	73.6%	74.2%

efficient (shown in Section 4.1), the BE matrix-vector multiplication accelerator performs better than the stochastic one.

Quality

To evaluate quality, I use cycle-level simulation models to evaluate the image processing workloads using single precision floating point, 8-bit fixed point, and $N = 256$ bitstream length. The results generally show PSNR is within 20 dB of fixed point baselines between the three scenarios except for the Sobel kernel (Table 4.2). For applications like ED, SC can achieve *identical* PSNR to fixed-point baselines since the underlying stochastic circuits produce exact results. For matrix multiplication, I evaluate linear support vector machine classifiers trained on several datasets from the UCI Machine Learning Repository [64] shown in Table 4.3. I perform cross-validation over each dataset and report the average accuracy. Compared with fixed-point implementations, SC classifiers are within 1.1% of the BE classifier implementation. In some cases, classification accuracy for SC is still within 1% of the fixed-point implementation at 4-bits of precision. More detailed evaluation of support vectors machines is presented in Section 5.2.

4.2.2 Summary

In this evaluation, I found that a useful application quantity in the context of SC accelerator architectures is the compute over conversion ratio. Higher compute-to-conversion ratio is likely to allow SC to achieve better energy efficiency gains. The results show that SC is viable for low precision arithmetic and breaks even in terms of energy efficiency between 2-8 bits. Despite this precision limitation, there are still applications which can operate well in this range which are highlighted later in Chapter 5.

4.3 Maximizing Energy Efficiency With Voltage Scaling

This section evaluates the error tolerance limits of SC by evaluating the impact of voltage overscaling on a fabricated ASIC prototype. Voltage overscaling (VOS) is a technique which scales the circuit supply voltage beyond its critical point. This introduces timing violations and potential bit flips in the computation but improves power and energy consumption. Intuitively, because SC uses a sparse encoding, it should be more error tolerant to these bit errors compared to BE arithmetic. Recall bit errors in SC only change the encoded value by one and may cancel out. For example, a 0-to-1 and 1-to-0 bit error result in no net error in the encoded value. On the other hand, bit errors in BE values may prove to be catastrophic as not all bits are weighted equally. Prior work has shown how this error tolerance can be exploited in the context of stochastic circuit VOS [4, 83] but only uses error models and simulations to measure the effects of VOS. Such models and simulations do not completely capture the complexities and actual behavior of fabricated silicon since voltage scaling is a hybrid analog-digital technique.

To do this, I evaluate VOS on a fabricated ASIC prototype for a handful of stochastic circuits and their BE counterparts. The chip prototype is fabricated using a 65nm TSMC technology node and includes stochastic and BE implementations of the Roberts cross edge detector kernel (ED) and a dot product unit (DP) using the correlation insensitive adder. Figure 4.7 shows the chip micrograph and layout of the stochastic and BE portions. The chip uses deterministic RNGs: for ED I use positively correlated ramp RNGs while for DP I use a VDC and a ramp RNG.

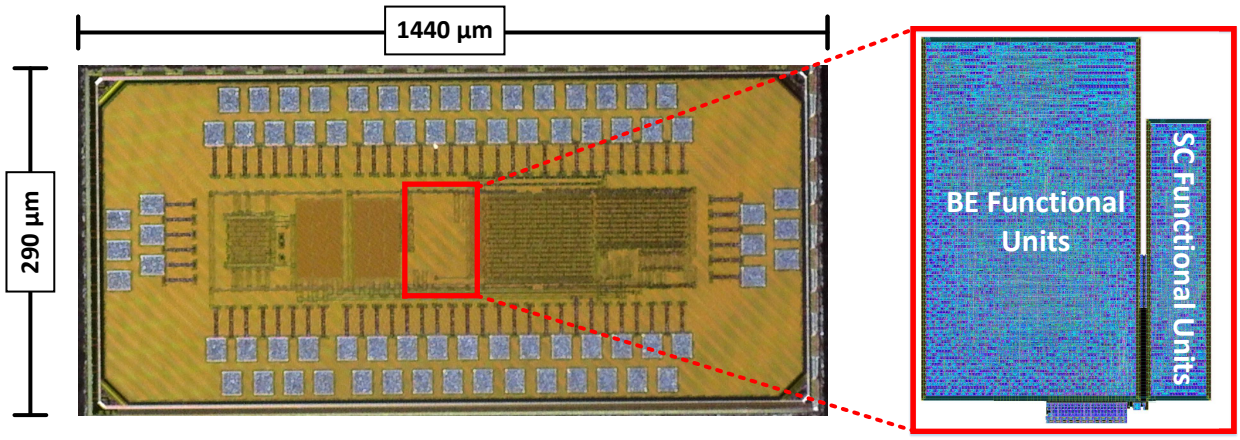


Figure 4.7: Micrograph of test chip used to evaluate voltage overscaling for stochastic and binary-encoded circuits.

The experiments are run using a 500 MHz operating frequency. Operating voltages are swept from 1.0 V (normal operation), where both SC and BE arithmetic work correctly, down to 0.5 V at which both SC and BE implementations fail. For each operating voltage, I measure the average absolute error for SC and BE functional units using *randomly* generated inputs. Figure 4.9 shows the error versus voltage results for all tested functional units. Notice that the critical operating voltage at which timing violations appear is different for each functional block. For BE circuits, timing violations immediately prove fatal to the computation leaving no margin for supply voltage reduction. In contrast, stochastic circuits tolerate timing violations well beyond the critical operating voltage, and can yield large energy savings.

I also test the edge detector using real image data for a Roberts Cross edge detector kernel. Figure 4.8 compares the edge detector accuracy for the BE and SC kernel at different operating voltages. Unlike the BE edge detector, the SC edge detector suffers minimal accuracy losses despite the introduction of timing violations below 1V down to 0.55V. This means, SC can yield an estimated 3.3× energy improvement bringing the total energy savings of SC at 8-bit and 4-bit precision to 1.38× and 17.8× respectively. This is equivalent to pushing the energy efficiency breakeven precision of SC compared to BE computation up by one to two bits. The results also show

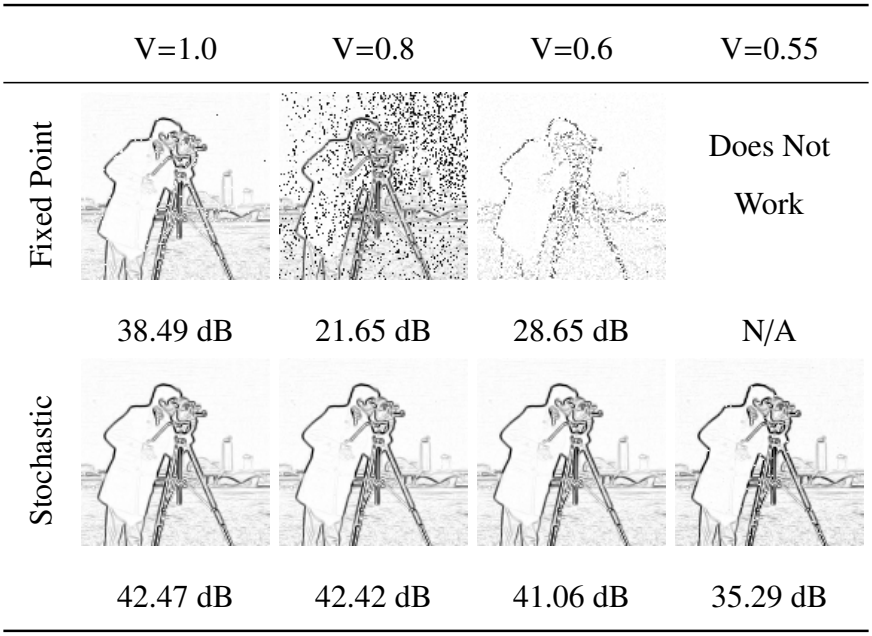


Figure 4.8: Estimated voltage overscaling results for edge detector on the ASIC prototype operating at 8-bit precision.

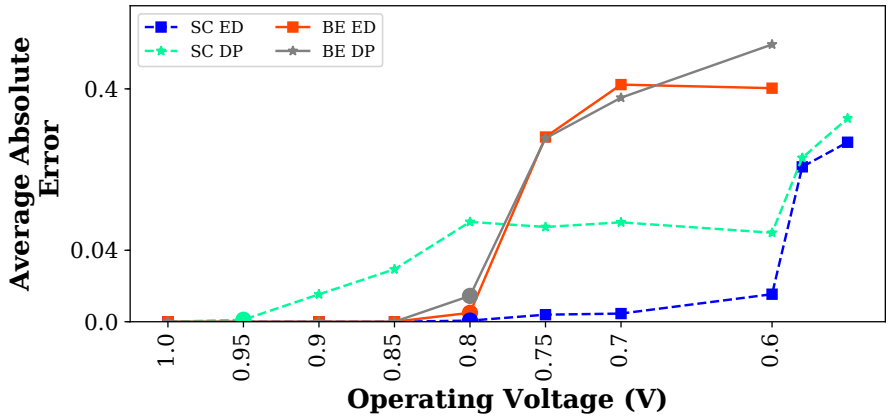


Figure 4.9: Accuracy versus voltage for voltage overscaled stochastic and binary-encoded circuits for *random* data. Circles indicate voltage where timing violations are first observed.

that the SC DP can reduce voltage down to 0.6V and still exhibit reasonable error rates. Combined with the results in Section 4.2 these results can yield up to 4.45× better energy efficiency at 8-bit precision, and up to 56× better energy efficiency at 4-bit precision (Figure 4.5) for kernels like convolution.

There is a vast body of existing work demonstrating the merits of voltage scaling to save energy [47]. However, this work is the first to empirically evaluate the error resilience and energy efficiency tradeoff of SC on a fully fabricated chip prototype. While the voltage scaling evaluation in this work is simplistic, it provides a good empirical estimate of the energy efficiency yields available to SC accelerators. I leave a more thorough exploration against more advanced voltage scaling techniques such as those proposed by [47] to future work.

4.4 Architectural Design Guidelines

I now synthesize the results from the design space exploration to formulate general design guidelines for stochastic computing accelerators.

4.4.1 Design Guideline #1: Higher computation-to-conversion ratio better amortizes stochastic computing overheads

Unlike BE accelerators, SC accelerators must pay for overheads such as S/D and D/S conversions, and RNGs. As a result, it is important for SC accelerator designers to also quantify SC overheads when measuring SC accelerator gains which is not always the case in previous work. While RNG costs can be amortized and shared across stochastic operators, the evaluation results have shown that the cost of S/D and D/S conversions can be painfully high relative to that of the computation. This is because every unique value encoded and decoded from the SC domain incurs a conversion cost. From an architectural perspective, I have found that a useful quantity to tabulate is the ratio of arithmetic operations over distinct conversions which can help better gauge whether an application is a good candidate for acceleration with SC. Actual energy efficiency gains will be governed by the exact arithmetic operation mix, but generally *computations with a higher ratio of compute*

operations per conversion will achieve better power, area, and energy efficiency improvements by better amortizing stochastic computing overheads.

4.4.2 Design Guideline #2: Limited viable operating precision demands judicious application codesign

A fundamental challenge with SC is that the performance and energy costs scale exponentially with precision. As a result, the viable operating precision range in which SC is more energy efficient than BE computation is limited to between 2 and 8 bits of precision. Voltage scaling and application compositions can improve the energy efficiency breakeven point by 1 or 2 bits but not significantly beyond that. This limited viable operating precision demands judicious application codesign to minimize application accuracy losses. Such precision restriction is clearly fatal to many classes of computation which require exact results. However, there are many applications which are resilient enough to precision reduction errors such as classification, neural networks [49], and image processing tasks. In these cases, SC can achieve gains with reasonable quality degradation.

Identifying and gauging when to exploit SC therefore requires judicious selection of applications or partitioning of error tolerant parts of an application. A common characteristic among applications that perform well in SC in prior work and this work is that they are *threshold-based computations* where the overall application accuracy is not affected by small errors. Instead the accuracy depends on whether the result value surpasses a threshold. For instance, the precise output value of an classifier inference is not important as long as the classifier model yields a value that surpasses the threshold it will yield the correct classification label. Thus, I conclude that *codesigning stochastic computing accelerators with threshold-based applications is more likely to yield better energy/accuracy tradeoffs than applications which require exact numerical accuracy.*

4.4.3 Design Guideline #3: Exploiting data reuse and parallelism reduces power and area overheads

A key strength of stochastic circuits are their exceptional density and 1-bit datapaths with fewer levels of logic. This lends them to more parallel architectures with less pipelining. For instance, the SC accelerators evaluated in this work have one to two orders of magnitude more stochastic arithmetic units than the BE circuit; but the final SC accelerators were roughly the same size as the BE accelerator. In other words, the operator density enabled SC accelerator architectures to eliminate the need to buffer data by instantiating many parallel datapaths to process each window of data. In contrast, the BE accelerators require pipelining elements and buffering to maximize data reuse and achieve a compact accelerator size. Notice that a parallel architecture for SC was only possible because the application itself contained abundant parallelism.

As a result, an important consideration when designing SC accelerators is exploiting application data reuse. Recall that S/D converters and D/S converters are many times larger and consume more power than individual arithmetic units. However, designs only require one S/D converter per *unique* value generated in the computation. Thus, a single S/D converter can be shared to generate an SN which can be reused multiple times which amortizes its power and area cost. For instance, applications such as convolution or matrix multiplication can reuse values generated for the weights across multiple dot products. As a result, I conclude that *applications with significant data reuse and abundant parallelism are better suited to reap power, density, and ultimately energy efficiency gains in stochastic computing*.

4.4.4 Summary

A common theme throughout each of these design guidelines is that they all depend heavily on application considerations or judicious codesign. As shown in the evaluation results, certain applications fail to achieve compelling energy efficiency at acceptable accuracies while others can achieve large gains. This suggests that the choice of application and the application codesign process is critically important to make SC accelerators viable alternatives to BE ones.

4.5 Related Work

This section describes related work in the context of stochastic accelerator design. While there is a rich body of work that explores the utility of SC, circuit synthesis techniques, and limit studies, few provide a holistic evaluation of end-to-end accelerator architectures.

4.5.1 Design Techniques

While there are no established SC architecture design methodologies, there are techniques for synthesizing individual stochastic circuits and managing their accuracy. Alaghi et al. [5, 10] show how to use spectral transformations to synthesize stochastic circuits using inverted bipolar encodings. Similarly, Qian et al. [87] propose a technique for synthesizing polynomial evaluation operations in SC while Li et al [63] show how to synthesize small FSM-based stochastic circuits. Both [61] and [89] also show how to synthesize sequential element stochastic operations. Chen et al. [29] introduce the concept of stochastic equivalence classes to reason about equivalent stochastic circuits. Finally, Neugebauer et al [77] propose a general framework for managing the accuracy degradation for a SC design.

4.5.2 Application-Driven Stochastic Computing Accelerators

There is also a substantial body of work that evaluates the application-specific merits of SC. Recent work has proposed SC for neural networks [16, 19, 22, 23, 49, 54, 88, 97, 98] which have been shown to be inherently robust to low precision and systematic computation errors. Canals et al. [24] even propose a new encoding - extended stochastic logic - to better support neural network computation. SC can achieve compelling results for image processing [14, 60], error-correcting codes [33, 39, 57, 80, 91, 92, 106], signal processing [14, 25, 26, 45, 62, 66, 90], matrix operators [109], data mining [30], machine learning [58], and discrete cosine transforms [71]. However, not all prior work considers the cost of SC overheads like D/S and D/S conversion nor do they always compare against BE ASIC implementations. Both aspects are crucial when trying to evaluate the viability of SC. My work not only quantifies these SC overheads, but also provides an evaluation of the power,

area, and energy efficiency breakdown per operation, and presents guidelines for the design of SC accelerators.

4.5.3 Limit Studies

Only a handful of studies quantifying the limitations of stochastic circuits exist. Manohar [70] uses an analytical model to postulate that under iso-accuracy conditions, SC will never be more energy efficient than BE computation. However, this work assumes purely random RNGs which is not the case in recent SC work and does not holistically consider end-to-end architectural tradeoffs. Aguiar et al. [32] quantify the energy efficiency gains of SC multipliers over BE ones under various iso-metric conditions. However, these works do not consider architectural SC and BE design overheads like state elements, control circuitry, pipelining, correlation manipulating circuits, and RNGs. I find that these overheads are non-trivial; but despite that, SC accelerators can still provide energy efficiency gains once both BE and SC overheads are all fully considered.

4.6 Summary

In this chapter, I provided a characterization of individual SC operations and end-to-end accelerator architectures to better understand the codesign opportunities and viable design space for SC accelerators. This study showed that individual SC operations are generally *not* more energy efficient than BE operations. However, I also found that when evaluated in the context of end-to-end application accelerators which considers all architectural overheads like pipelining and control structures, SC can still be more energy efficient at low precision. In order to maximize energy efficiency gains and amortize SC overheads, I find that the most compatible applications will have plentiful data reuse to amortize SC overheads, and inherent error tolerance to errors due to quantization and correlation. Successfully accomplishing this requires judicious characterization and selection of target applications, and careful application and hardware codesign to maximally exploit these opportunities. In the next chapter, I evaluate several emerging applications in the context of SC accelerators and analyze their tradeoffs.

Chapter 5

APPLICATION CODESIGN FOR STOCHASTIC COMPUTING

Hardware and application codesign is a critical design principal for SC accelerators. Hardware and application codesign is the practice of using application characteristics to inform underlying architectural design and vice versa. For instance, deterministic errors in the hardware accelerator may require modifications to the application algorithm to tolerate such errors. Similarly, repeated computational patterns in an application inform the integration of specialized hardware units to better support such operations. Such synergistic modifications require careful consideration of both hardware and application properties but can yield significant gains in throughput and energy efficiency when combined.

As shown in Chapter 4, application characteristics like available data reuse and computation size are important when considering whether an application is suitable for SC. Tailoring applications for SC requires careful consideration of quantization effects and correlation errors. To reduce the impact of these errors, I leverage hardware and application codesign to propose modifications to the algorithms. In this section, I present several case studies of hardware and application codesign for SC accelerators. In particular, I focus on how SC can be harnessed to accelerate support vectors machines, neural networks, and similarity search. I present empirical evaluations which highlight their viability and discuss why each application yields compelling energy efficiency gains or not.

5.1 Similarity Search

Similarity search is a core computation component of many larger application pipelines. In this section, I discuss and evaluate the practicality of a SC in the context of similarity search which manifests as k-nearest neighbors (kNN). The kNN algorithm is a threshold-based computation and can tolerate slight deviations in the numerical values of the results. It also contains abundant

application parallelism and opportunities for data reuse making it a potentially promising candidate for SC acceleration.

5.1.1 Similarity Search Basics

Similarity search is a key computational primitive found in a wide range of applications, such as computer graphics [86], image and video retrieval [44, 99], data mining [117], and computer vision [112]. Similarity search manifests as a simple algorithm: k-nearest neighbors. The k-nearest neighbors algorithm takes a query vector and compares it against a database of vectors to find the most similar one. Similarity metrics typically include Euclidean distance, cosine similarity, and Hamming distance (for binarized feature vectors). Vectors are typically generated by feature extractors that convert raw data to an intermediary representation. Raw data is typically pre-processed to generate the database but the feature vector for the query data is extracted online.

The k-nearest neighbors algorithm is composed of two primary computations (Figure 5.1): (1) a pair-wise distance calculation between each query vector q in a query set Q and each dataset vector v in a dataset D, and (2) a k-selection which reduces the distances to the best k results. The pairwise distance calculations compute the similarity between the query and each candidate vector using similarity metrics such as Euclidean distance. During k-selection, the top-k vectors with lowest distance are filtered and the resulting set of neighbors R is returned as the result.

In terms of computational resources, given q queries and n database vectors where each vector is d dimensions, the distance calculation cost scales at $\mathtt{O}(qnd)$ while the k-selection cost scales at $qn\mathtt{log}(k)$. To reduce the computational cost, it is possible to apply both algorithmic approximations and architectural approximation techniques. In kNN, accuracy is defined as the Jaccard Index $S_E \cap S_A/|S_E|$, where S_E is the true set of neighbors returned by exact floating point linear kNN search, and S_A is the set of neighbors returned by approximate kNN.

Algorithmic approximations typically involve using indexing structures such as randomized kd-trees, hierarchical k-means, and multiprobe locality sensitive hashes. The key idea is that indexing structures organize the dataset into buckets of vectors where each bucket contains vectors which are similar. During query time, these indexing structures are traversed to prune away portions

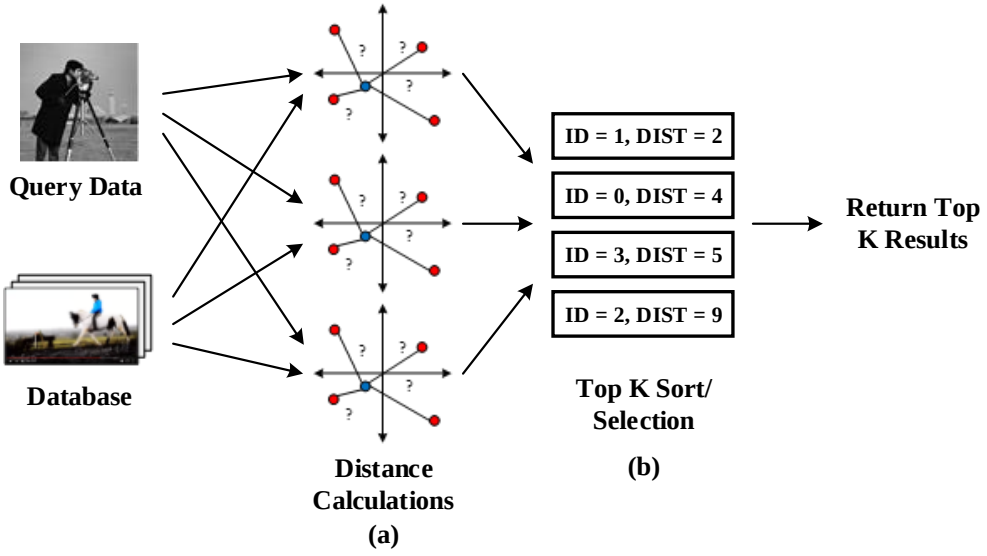


Figure 5.1: The kNN algorithm consists of (a) pairwise distance calculations and (b) k-selection.

of the search space and reduce the size of the dataset to search. Multiple buckets can be traversed to improve the quality of the search. Once a search reaches a bucket, it backtracks up the indexing structure to search additional "close by" buckets. Each of these approximate kNN algorithms trade accuracy for enhanced throughput; since the entire dataset is not scanned, some of the closest neighbors may not be searched. In practice, the degree of kNN accuracy loss is governed by the indexing data structure and can be tuned by adjusting the duration of the traversal. Allowing approximate kNN variants to search more of the dataset generally improves search accuracy.

In contrast, architectural approximations involve using potentially inexact functional units such as stochastic circuits or reduced precision fixed-point arithmetic units. Architectural approximations primarily yield enhanced power, area, and energy efficiency gains. For example, substituting fixed-point arithmetic units for floating-point units reduces the precision of the computation but improves the power usage, area footprint, and energy efficiency. Many architectural approximation techniques such as reduced precision arithmetic or inexact circuits can be applied orthogonally to algorithmic approximations. However, the accuracy degradations will compound so approximations must be carefully balanced to maintain reasonable application accuracy.

5.1.2 Approximation Opportunities

Stochastic computing is an architectural approximation technique so to expose potential opportunities with minimal impact on application accuracy I need to determine opportunities in the application where approximation is tolerable. The primary opportunity for approximation in kNN is approximating the pairwise distance calculations between vectors. Distance calculations can be approximated in kNN because the ultimate result that is returned to the user is the set top-k neighbors regardless of the precise numerical values of the distances ascribed to each neighbor. In other words, it is sufficient to only compute whether the distance of a vector is in the top-k results or not. As a result, distance calculations can be approximated and have no impact on the resulting accuracy as long as they preserve the relative ordering of distances between neighbors. Distance calculations can also corrupt the relative ordering of the results as long as the returned set of neighbors is equivalent. This allows for additional approximation errors in the distance calculations which may reorder the resulting neighbors. It is this error tolerance that I will exploit and explore when I evaluate kNN on SC accelerators.

5.1.3 Accelerator Architecture

The potential kNN accelerator architectures for the Euclidean distance calculation are shown in Figure 5.2. Because inputs to the subtract circuit must be positively correlated, I use the same number generator to generate each input SN. For the squaring unit, input SNs need to be uncorrelated so there are several options to generating the uncorrelated complement SN. The design options are shown in Figure 5.2: (1) insert an isolator to generate a less correlated SN, (2) insert a decorrelator to generate a less correlated SN, or (3) duplicate the subtraction unit using different RNGs to generate an uncorrelated SN. Isolators and decorrelators in prior work have shown mixed results as they do not provide guarantees on how uncorrelated their output SNs are. To evaluate the isolator and decorrelator configurations, I use Van der Corput or Halton sequences (base 3) to evaluate accuracy.

The duplicated subtraction configuration (Figure 5.2c) operates by using two independent RNGs to generate the SNs for two copies of the subtraction unit. The resulting SNs from each subtraction

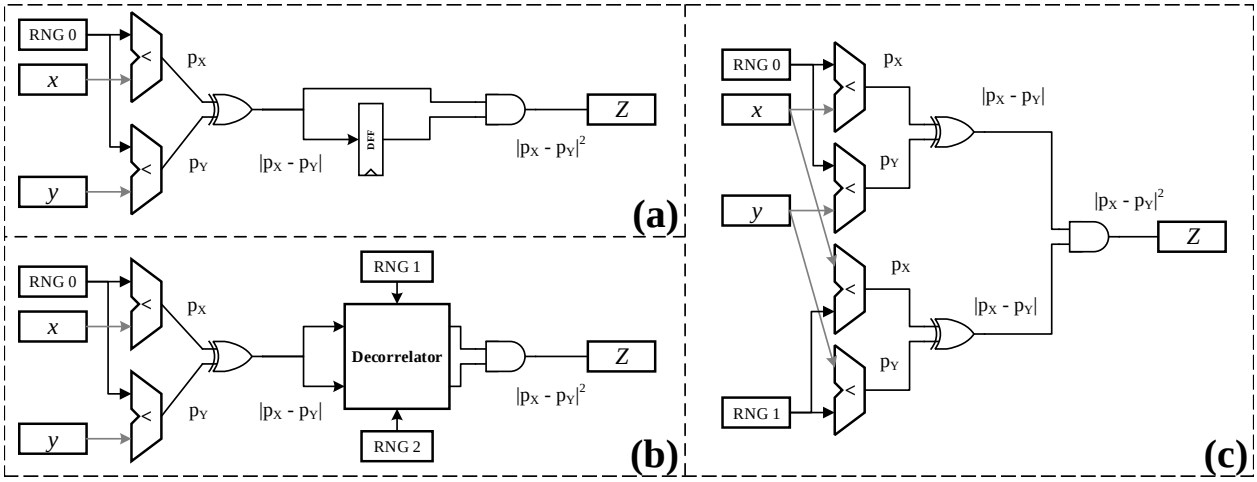


Figure 5.2: Design space for stochastic Euclidean distance function unit. Euclidean distance unit using (a) isolator-based decorrelation, (b) decorrelator, and (c) replicated RNGs.

can be used as input to the multiplier. In this evaluation, I use the RNG configuration from [54] - a ramp compare and Van der Corput sequence - to generate the input SNs for the two subtractors. The key insight is that the correlation between these two RNG configurations achieves accurate multiplication so they should also produce similar results for the Euclidean distance evaluated here.

The add reduction for the Euclidean distance is implemented as a balanced tree reduction. All adders in the reduction tree are implemented using the correlation insensitive adder proposed in Section 3.1. For the S/D conversions, I evaluate two configurations: (1) a standard S/D converter which preserves the high bits of the add reduction and (2) an accumulative parallel counter (APC) [108]. The APC is a hybrid stochastic-binary circuit which takes multiple parallel bitstreams and performs a S/D conversion. Evaluation results (discussed later) show that using the APC yields superior results because of the additional precision.

The kNN algorithm also contains significant application data reuse. For instance, the query vector is reused over multiple pairwise distance calculations with each dataset vector. This allows the D/S converter which generate the query vector SNs to be shared across multiple vectors by processing multiple dataset vectors in parallel. To tabulate the savings that this affords, I evaluate

accelerator architectures which process varying number of dataset vectors in parallel. I assume that a priority queue handles the k-selection that follows the S/D conversion.

5.1.4 Methodology

To evaluate the application accuracy tradeoffs, I evaluate three commonly used datasets for measuring kNN accuracy: (1) the Global Vectors for Word Representation (GloVe) dataset [82], (2) the scale invariant feature transform (SIFT) dataset [43, 67], and (3) the GIST descriptor dataset [34, 43]. Each dataset is separated into a dataset of database vectors and a query set of query vectors. Query vectors are used to measure accuracy and are separate from the original dataset. Unfortunately, simulating stochastic circuits is extremely slow for large datasets because of the long SN lengths. To make the simulation times tractable, I reduce the database size and query set size to 1000 and 100 vectors respectively.

In order to maximally exploit operating precision, dataset and query values were re-scaled. For stochastic kNN, the datasets were shifted and scaled to the range [0, 1] to make them compatible with unipolar encodings. Note that the shift and scale does not affect the relative distance between vectors. For fixed point computation, the datasets were shifted and scaled to the range [0, 1], and then scaled by 2^p where p is the fixed-point operating precision. For example, 8-bit computation would scale the dataset and query set range to $[0, 2^8]$

To measure power, area, and energy efficiency, I prototype the functional units in SystemVerilog and use Synopsys Design Compiler using a 32 nm generic standard cell library. Energy efficiency measurements are estimated as the average energy required to process each query. I compare the SC results against an equivalent BE fixed point unit.

5.1.5 Evaluation

I first evaluate the accuracy for the fixed point and stochastic kNN implementations. Figure 5.3 and Figure 5.4 shows the accuracy and energy efficiency respectively results for different operating precisions. The number of nearest neighbors is set to $k = 8$.

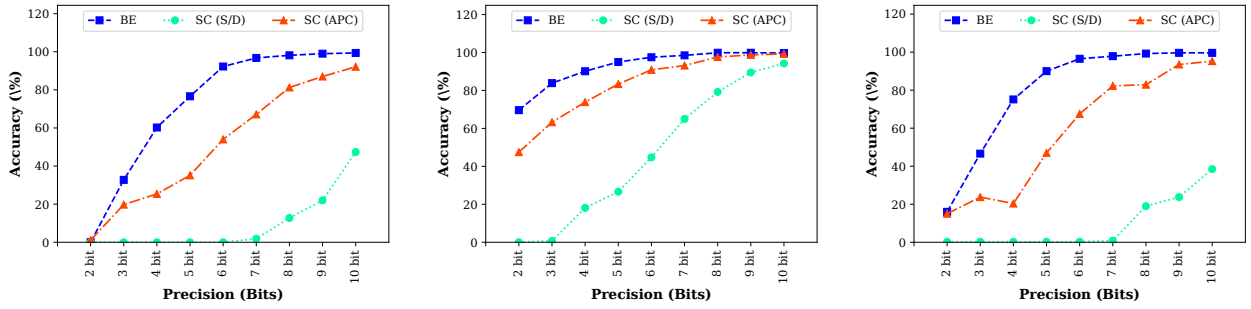
Across each dataset, the results generally show that the fixed point accuracy degrades more gracefully than the stochastic computing solution at equivalent precision. For fixed-point calculations, the results show that accuracy minimally degrades for operating precisions higher than 5-bits ($> 90\%$ accuracy). Stochastic computing results show that using the APC reduction circuit provides superior accuracy over using the S/D converter. This because the higher bits of the add reduction in kNN tend to be sparse and so the lower bits are more important when distinguishing between distances during k-selection.

In general, the results show that SC is not particularly compelling for accelerating distance calculations in kNN. In all cases, the accuracy-energy efficiency curve of the fixed-point accelerator significantly outperforms the results from the SC accelerator. As a result, accelerating kNN in SC is likely not a practical since the accuracy loss and energy efficiency gains are not compelling (Figure 5.4).

This is to be expected as the stochastic implementation introduces non-monotonically decreasing errors when executing the squaring function. For instance, the square of 0.5 may not necessarily be less than the square of 0.55 due to random fluctuations in the SN [111]. This is in contrast to fixed point quantization where the quantization error resulting from the square function increases or decreases monotonically with the inputs.

5.1.6 Analysis and Discussion

Tabulating the compute-to-conversion ratio provides better insight into why kNN does not yield as compelling energy efficiency gains. A single d-dimensional Euclidean distance requires $2d + 1$ conversion units: $2d$ D/S converters for the input operands and a single S/D converter to recover the resulting value. In terms of compute units, a d-dimensional Euclidean distance requires $\approx 3d$ total computations: d x subtractions, d x squaring units, and $d - 1$ addition units. This yields a compute-to-conversion ratio of $\approx 3d/2d = 1.5$ which is significantly smaller than that of the convolution kernel evaluated in Section 4.2. This conversion ratio is more in line with the geometric interpolation kernel which performs poorly as a SC accelerator.



(a) GloVe Results (b) SIFT Results (c) GIST Results

Figure 5.3: Search accuracy at different operating precisions for fixed point and stochastic kNN for GloVe, SIFT, and GIST datasets at varying precisions (up and to the left is better).

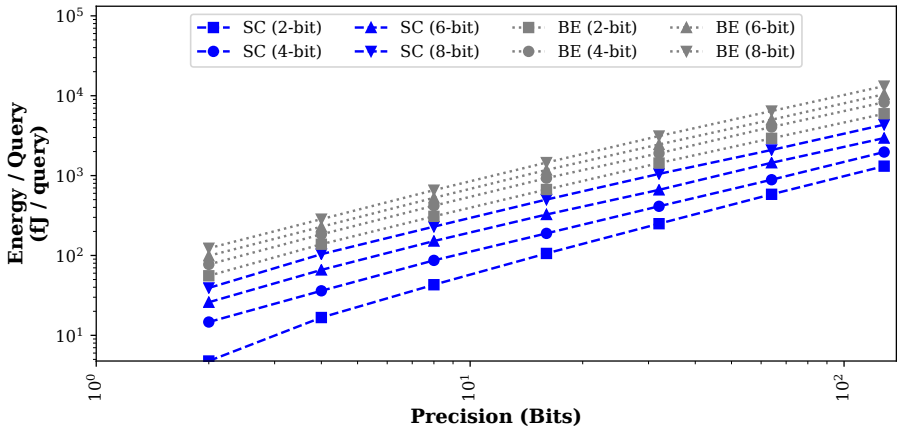


Figure 5.4: Energy efficiency comparison of stochastic computing and binary-encoded Euclidean distance unit for different operating precisions.

In addition, an efficient implementation of the k-selection in stochastic computation to distill the top k-nearest neighbors is not known. As a result, implementing the priority queue or equivalent sorting unit requires conversion to the BE domain. The BE computation required to finish the computation thus imposes additional Amdahl's limitations upon the possible gains yielded by accelerating the distance calculations in SC.

Finally, the analysis presented in this section does not account for additional complexity introduced by the indexing structures used in approximate variants of kNN. Indexing structures such as hierarchical k-means [46, 74], multiprobe locality sensitive hashes [69], and randomized kd-trees [96] are often used to reduce the search space. Accelerating pointer chasing computations which are needed to traverse indexing structures is not well-explored with stochastic computation. In particular, because index traversals are data dependent they do not contain data parallelism which can be exploited by stochastic computation accelerators.

5.2 Support Vector Machines

Support vector machines (SVMs) are a popular class of machine learning models used for classification tasks. More specifically, in the simplest case a SVM is used to make a binary decision whether an input is or is not part of a class of objects. SVMs therefore are relatively small and computationally inexpensive models, and as a result tend to be fast and require small implementations. This makes them ideal for execution in highly constrained environments such as sensor nodes or mobile devices. SVMs are error tolerant because they are a threshold-based computation and the training process can be made self-healing in the presence of systematic errors. In this section, I explore how SVM models can be cotrained with SC and analyze the potential gains of SVMs in SC.

5.2.1 Support Vector Machine Classification

Support vector machines (SVM) are a class of supervised machine learning algorithms that build a model for classifying feature vectors. SVM models come in many different forms such as linear, polynomial, and radial basis function kernels. In this work, I focus on training linear classifiers

but it is possible to extend the proposed methodology to other SVM models. SVM classification is separated into two parts: training and inference. During the training phase, the SVM algorithm takes a set of labeled data observations as input and trains the classifier model. During inference, the trained classifier model is used to predict the classification of previously unseen data. The quality of the trained model is measured by classification accuracy which is defined as the number of correct inferences over the total number of inferences. For a given dataset, it is common practice to separate it into a training and test dataset. The training dataset is used to train the classifier model and the test dataset is used to represent unseen values to test the generality of the resulting model.

Data points in a dataset are high dimensional feature vector representations of raw data. Each d-dimensional dataset vector $\overrightarrow{x_i} \in \mathbb{R}^d$ is ascribed a class label y_i where a label of $+1$ indicates the vector is of one class and a label of -1 indicates it is of the other class. The goal of the SVM training process is to find a hyperplane model with weights $\overrightarrow{w}$ and bias b such that dataset vectors with $y_i = +1$ fall on one side the hyperplane while vectors with label $y_i = -1$ fall on the other. Given a model $\overrightarrow{w}$ with bias b, the inferred label of a dataset vector $\overrightarrow{x_i}$ is computed by calculating $f(\overrightarrow{w}, \overrightarrow{x_i}, b) = \overrightarrow{w}^T\overrightarrow{x_i} + b$ and taking the sign of $f(x)$. Notice that the sign function is agnostic to the magnitude of $f(\overrightarrow{w}, \overrightarrow{x_i}, b)$ which is the threshold-based computation behavior desirable for SC. The threshold-based computation provides a margin for error tolerance which can be exploited to reduce the impact of stochastic computation errors.

Training the SVM model is typically done using gradient descent or equivalent methods. For linear SVMs, it is typical to use the hinge loss (L1 loss) or the square of the hinge loss (L2 loss) function with an added L2 regularizer which helps prevent overfitting. The value λ effectively is used to adjust how strictly the training process should penalize misclassified samples.

$$L_1(\overrightarrow{x_i}, b, y_i) = max(0, 1 - y_i(f(\overrightarrow{w}, \overrightarrow{x_i}, b))) + \frac{1}{2}\lambda\overrightarrow{w}^T\overrightarrow{w}$$

$$L_2(\overrightarrow{x_i}, b, y_i) = max(0, 1 - y_i(f(\overrightarrow{w}, \overrightarrow{x_i}, b)))^2 + \frac{1}{2}\lambda\overrightarrow{w}^T\overrightarrow{w}$$

The gradient descent formulation attempts to minimize the error function by iterating over the training dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_{|D|}, y_{|D|})\}$, evaluating the error of the current model, and

adjusting the weights and bias towards more optimal solutions by subtracting the gradient. The fraction of the gradient subtracted from the current solution is given by the learning rate α. The learning rate effectively controls the rate at which the model descends the gradient towards more optimal solutions. For this work, I use an adaptive learning rate $\alpha = 1/(\lambda(t+1))$ proposed in the Pegosas algorithm [95] where t is defined as the number of times the model has been updated.

$$\overrightarrow{w'} \leftarrow \overrightarrow{w} - \frac{\alpha}{|D|} \sum_{i=1}^{|D|} \frac{\partial L_1}{\partial \overrightarrow{w}}$$

$$\overrightarrow{w'} \leftarrow \overrightarrow{w} - \frac{\alpha}{|D|} \sum_{i=1}^{|D|} \frac{\partial L_2}{\partial \overrightarrow{w}}$$

In standard gradient descent, the training process computes the error and partial gradient over each vector in the training dataset. One scan over the entire dataset is known as an epoch. The weights and bias are then updated by averaging the partial gradients together. This process can be slow and may result in long convergence times to the final model. To increase the convergence speed of the training process, it is common practice to use stochastic gradient descent. Stochastic gradient descent partitions the training dataset into batches and updates the model using the average of the partial gradients for only the training vectors in a batch. This allows for multiple iterations per scan of the dataset allowing for faster convergence times.

Once a model is trained, the quality of the model is evaluated on the test dataset. Each point in the test set has an expected class label y_i which is compared against the inferred class label returned by the SVM model. For a linear SVM classifier, this is equivalent to resolving which side of the hyperplane the test point resides on using a dot product and returning the class label on that side of the hyperplane. If the expected class label and inferred class label match, the inference is marked as correct. Notice that the label inference process does not require exact computation as long as the inferred label is ultimately correct.

5.2.2 Support Vector Machine Cotraining Formulation

I now described the cotraining and retraining formulation which I use to cotrain SVM models with SC. In order to expose the training process to underlying SC errors, I use cycle-level simulation models of stochastic circuits when evaluating the predicted label for each training vector. Cycle-level models are necessary to faithfully model errors that manifest in the computation due to SC and correlation effects between SNs. I then replace the precise BE computation in the training procedure with the results of the same computation executed in SC. Most notably, I modify the inference function $f(\overrightarrow{w}, \overrightarrow{x_i}, b)$ to execute using stochastic arithmetic. This effectively integrates SC error models into the training process so that the training process can compensate for the systematic errors introduced by SC.

Recall that SC suffers from both quantization errors and systematic errors (Figure 5.5). Quantization errors also occur in equivalent BE fixed-point computations and is the result of rounding to the nearest representable value. The key idea behind cotraining and retraining is that these errors impact the weight and bias updates which leads to a model which can correct for these errors. I define cotraining as the process of training a model using SC error models for the entire training process. Retraining on the other hand bootstraps the classifier training process by first training a floating point model. The trained floating point model is then quantized and retrained for several iterations using SC error models to correct for these errors. By bootstrapping the training process with a floating point model, the cotraining process only needs to optimize the floating point model within the vicinity of the local minimum instead of looking for an entire new local minimum from scratch.

5.2.3 Evaluation Methodology

I evaluate the accuracy for several different BE and stochastic linear SVM models to compare the efficacy of different training methods. I first train a baseline model which uses single precision floating point and use this as the baseline accuracy. For BE fixed-point models, I quantize the floating point weights and bias to the nearest value, and also quantize the test dataset. I then use the

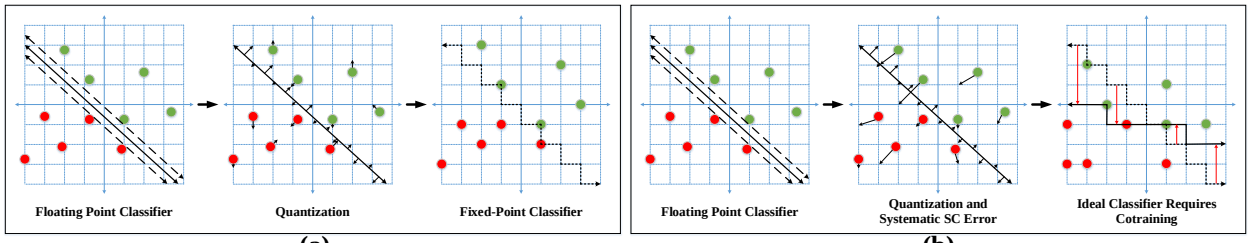


Figure 5.5: Impact of quantization and systematic stochastic computing errors on SVM classification. (a) Quantization effects have limited effect on classifier accuracy. (b) Quantization and systematic stochastic computing errors require cotraining or retraining of SVM classifier.

quantized dataset, weights, and bias to predict the labels of the test set vectors using BE arithmetic. For SC, I evaluate three models: (1) the unmodified quantized fixed-point model executed in SC, (2) a retrained model for SC starting from the quantized fixed-point model, and (3) a cotrained model for SC starting from a randomly initialized model. The retrained model is initialized with the trained floating point model, and executes additional epochs of training with SC error models. The cotrained model attempts to train the model with SC error models without any initialization using the same initialization configuration as the original floating point model.

To test the generality of the SC cotraining techniques, I use nine real datasets shown in Table 5.1 taken from the UCI Machine Learning repository [64]. For each dataset, I partition the data points into training and test sets, perform 3-fold cross validation, and report the average classification accuracy for each training methodology. The baseline floating point models are trained using a custom implementation of the Pegosas SVM training algorithm [95].

I compare power, area, and energy for several binary-encoded and SC design variants operating at the same precision. For BE fixed-point designs, I instantiate several fused-multiply accumulate units in a SIMD fashion. SC designs use a similar parallel architecture and instantiate so that the binary-encoded and stochastic computing designs have roughly the same area usage. I use Synopsys Design Compiler, IC Compiler, and PrimeTime to synthesize, place-and-route, and perform power

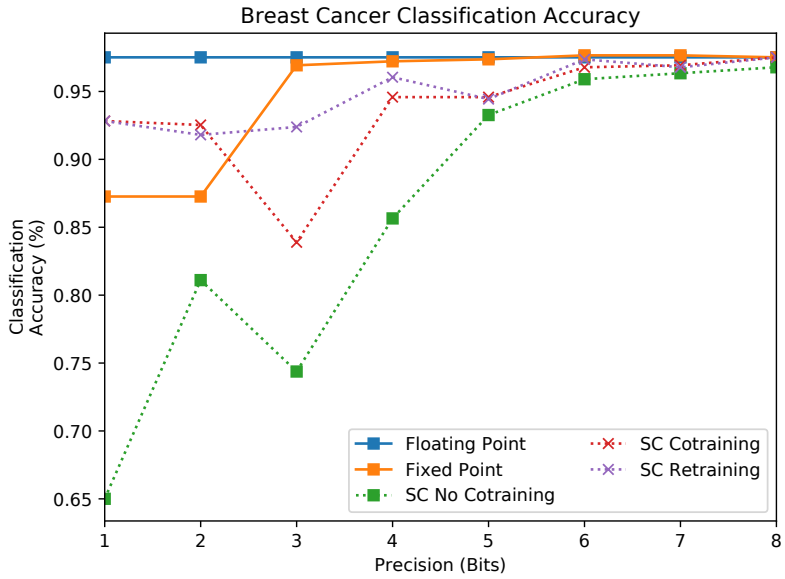


Figure 5.6: Comparison of SVM classification results for breast cancer dataset. Cotraining and retraining improves accuracy.

estimation using a TSMC 65 nm library. Post-placement and route power estimation is evaluated using real dataset traces.

5.2.4 Cotraining Accuracy Results

I now present classification results for binary-encoded and SC SVM classifiers. Table 5.1 compares the classification accuracies for a full floating point classifier and the accuracy losses associated with each encoding and training methodology. In general, I find quantized fixed-point models perform well for 8-bit and 6-bit precision, and show that the SVM models are resilient against quantization errors.

For stochastic computing models without any cotraining or retraining, I find that they suffer significant accuracy losses. For cotrained and retrained SC models, the results show that in some cases the resulting model can yield similar accuracies as fixed point models (in some cases with no accuracy loss). For instance, the accuracy results for the breast cancer dataset (Figure 5.6) shows

only minimal accuracy degradation compared to the baseline floating point models. The results also show that retraining is also effective for recovering accuracy losses in many cases. However, the results do not provide conclusive evidence as to when retraining should be used over cotraining. For other cases, the cotraining and retraining formulations are unable to recover to comparable accuracies.

5.2.5 Power, Area, and Energy Results

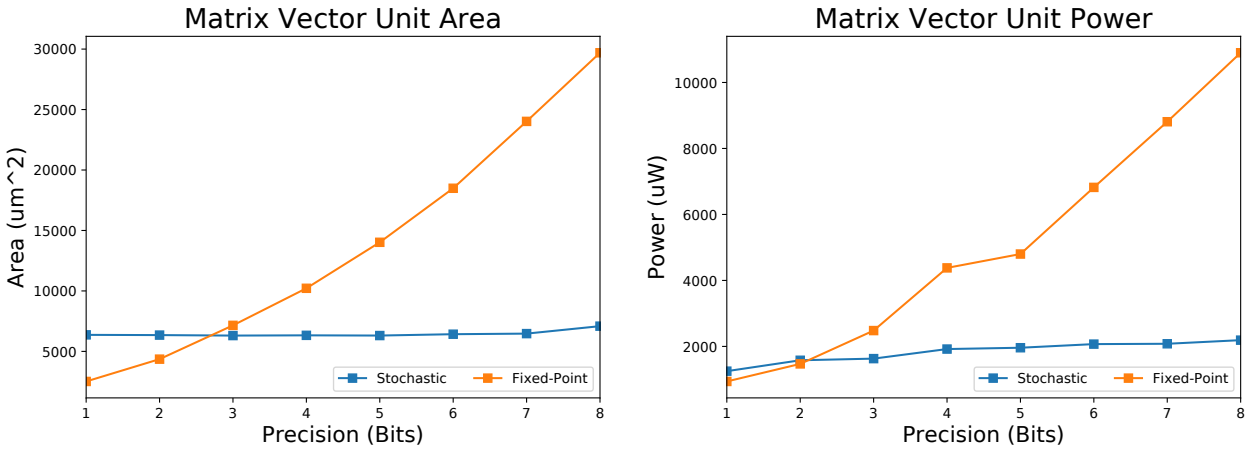
I now briefly compare the power, area, and energy efficiency of a SC SVM accelerator to those of a BE equivalent accelerator. For the BE design, I compare against a SIMD-like matrix vector product unit. For SC designs, I evaluate a spatial accelerator design where all compute units are instantiated in parallel. I opt to use a spatial architecture because it allows the SC architecture to better amortize overheads like D/S converters and RNGs by exploiting data reuse. The spatial architecture is also possible because of the density of the individual compute units.

To measure power and area, I place and route each SC design for 1-bit to 8-bit precision or N=2 to N=256 length bitstreams using Synopsys Design Compiler, and IC Compiler using a TSMC 65 nm library. Power measurements were estimated using post-placement and route simulation traces. For BE accelerators, the power and area of the designs increase with the datapath width or precision of the computation (Figure 5.7). In contrast, the compute area of SC accelerators scales sublinearly with the equivalent BE precision because the width of the compute datapaths remains constant. In terms of compute logic area, SC accelerators fare significantly better than BE compute logic.

However, this does not tell the entire story; there are still significant overheads associated with both SC and BE designs such as pipelining and conversion overheads. Conversion overheads in SC can exhaust any of the gains if not properly accounted for. For instance, the fraction of logic required for D/S conversion, S/D conversion, and RNG logic for matrix multiplication increases to up to 50% for 4-bit precision and 65% at 8-bit precision. Pipelining for matrix multiplication on the other hand also increases the design overhead (not evaluated). With the conversion overheads and additional units, I expect the results to align with those tabulated in Section 4.2.

Table 5.1: L1 SVM classification accuracy changes (higher is better) using binary-encoded fixed-point classifiers and stochastic computing classifiers with different training configurations. Cotraining or retraining SC classifiers is significantly more accurate than without cotraining.

Dataset	Floating Point	BE Fixed-Point (Accuracy Change)			SC No Cotraining (Accuracy Change)			SC with Cotraining (Accuracy Change)			SC with Retraining (Accuracy Change)		
		8-bit	6-bit	4-bit	8-bit	6-bit	4-bit	8-bit	6-bit	4-bit	8-bit	6-bit	4-bit
Breast Cancer	97.51	0.00	0.15	-0.15	-0.59	-62.52	-62.52	-0.29	-0.15	-14.36	-0.29	-0.15	-10.99
Diabetes	73.31	0.13	-0.78	0.91	1.17	-5.08	-8.20	3.13	-6.25	-8.33	3.13	-4.82	-8.07
Four Class	71.92	-0.12	-1.28	-4.87	-0.35	-2.44	-7.31	0.00	-2.44	-7.31	0.00	-2.44	-7.31
Adult 1	70.16	0.13	0.31	1.81	5.23	5.23	5.23	5.23	5.23	5.23	5.23	5.23	5.23
Heart Disease	82.22	0.37	0.00	0.00	0.74	-1.48	-24.44	-0.74	-12.59	-25.93	-1.48	-13.33	-35.93
Liver Disorder	69.67	0.00	0.00	-2.10	0.67	-3.46	0.62	-1.39	0.67	0.64	0.00	6.14	0.64
Mushrooms	96.44	-0.16	0.35	-6.00	-48.24	-48.24	-48.24	-48.24	-48.24	-48.24	-41.74	-46.05	-48.24
Sonar	83.20	0.00	0.49	-6.73	-18.77	-13.95	-36.54	-9.61	-5.78	-36.54	-11.05	-11.06	-36.54
Splice	82.40	0.40	0.70	-2.10	-28.10	-30.70	-30.70	-16.60	-24.50	-30.00	-16.30	-30.60	-30.70



(a) Area

(b) Power

Figure 5.7: Power and area comparison of compute logic area for binary-encoded and stochastic matrix-vector multiplication accelerators.

5.2.6 Summary

In this section, I proposed a cotraining and retraining formulation for support vector machine classifiers to compensate for errors introduced by SC. I found that by cotraining and retraining support vector machines models, the accuracy losses in many cases can be reduced. The power and area results in particular show that the SC accelerator is better than binary-encoded ones. However, a tabulation of the energy efficiency shows that SC is not very competitive against the equivalent BE accelerator. The evaluation presented in this section does not consider exploiting application data reuse or batching to amortize overheads and increase the compute to conversion ratio. These considerations could further improve the gains realized by SC SVMs and is the subject of future work. In the next section, I will explore neural network applications which are similar to support vector machines but contain significantly more computation per conversion.

5.3 Hybrid Stochastic-Binary Encoded Neural Networks

Neural networks are a popular class of machine learning applications which are used for a wide variety of classification and feature extraction tasks. Recent work has established their utility in a wide range of tasks across computer vision, robotics, and artificial intelligence. Neural network applications are ideal candidates for SC because: (1) they can operate at low-precision, (2) are tolerant of computation errors, and (3) contain abundant parallelism and data reuse.

5.3.1 Background: Neural Networks

Neural networks come in a wide range of network topologies, and generally consist of an input layer, an output layer, and a number of hidden layers in between [78]. A layer is composed of neurons, each of which has a set of inputs, an output, and an activation function $f(x)$ (e.g., a rectified linear unit). Each neuron is connected to neurons in the previous layer; a connection is defined by a weight that is multiplied by the previous neurons output. These values are summed with other connections' outputs and passed to an activation function. For instance, given a neuron y that is connected to k neurons in the previous layer with output values $\overrightarrow{x} = x_0, x_1, , x_{k-1}$ and connection weights $\overrightarrow{w} = w_0, w_1, , w_{k-1}$ respectively, the output of neuron y is defined as $y_{\text{out}} = f(\sum_{i=0}^{k-1} x_i w_i)$.

Neuron connection topologies can either be fully connected or locally connected to the previous layer. In fully connected layers, each neuron is connected to every neuron of the previous layer. In the locally connected case, neurons are connected to a subset of neurons in the previous layer. Locally connected layers are often referred to as convolutional layers because their connections from the previous layer take the form of a window. The resulting operation is mathematically equivalent to a convolution where the convolutional kernel is summarized as a matrix of the connection weights. Finally, neural networks also may have max pooling layers, which are locally connected layers that subsample a window in the previous layer and output the maximum value.

To determine the weights for each layer, neural networks are trained over an input training set using backpropagation [78]. This is a technique that iterates over the training dataset and gradually adjusts the weights based on the gradient of the error in the neural networks output

function. The error metric varies across applications but a commonly used one for neural network classification is the cross-entropy loss. One iteration over the entire training set is known as an epoch. Training is often supplemented by dropout which is a training technique that randomly removes connections during the training process at certain layers to prevent overfitting. Once the training process converges to a set of weights, a test set is used to evaluate the quality of the neural network model. The quality metric varies across applications but a commonly used metric is classification accuracy based on the outputs of the neural network model.

Using SC for neural networks has a well-established history [22, 50] dating back to the 1990s. Recent work proposes fully SC neural network designs using FPGA fabrics and full custom ASICs [49]. Similarly, Ardakani et al. [19] propose a SC neural network for digit recognition which outperforms BE designs by using shorter bitstreams (down to length 16). Unlike my approach, prior SC work uses older, fully connected neural network topologies with only two hidden layers which are smaller and less accurate than current state-of-the art neural network topologies like LeNet-5 (used in this evaluation). Finally, fully SC neural networks need longer bitstreams (N = 256 to 1024) to achieve reasonable accuracy. In contrast, this work does not execute the entire neural network in the SC domain. Instead, I execute the first layer using SC, then allow higher precision BE or floating point units to finish the neural network calculation.

5.3.2 Retraining Hybrid Stochastic-Binary Encoded Models

I now present the stochastic-binary hybrid design for near-sensor neural network computation. Figure 5.8 gives an overview of the proposed neural network layer and system design. To evaluate its utility, I will use it to implement the first layer of the LeNet-5 neural network topology [51].

Signal Acquisition

Image sensors capture light intensity and convert it to analog signals, which are converted to digital numbers for processing. In this work, I use parts of a ramp-compare analog-to-digital converter (ADC) to convert the analog signal to the SC domain. The conversion circuit shown in Figure 5.8

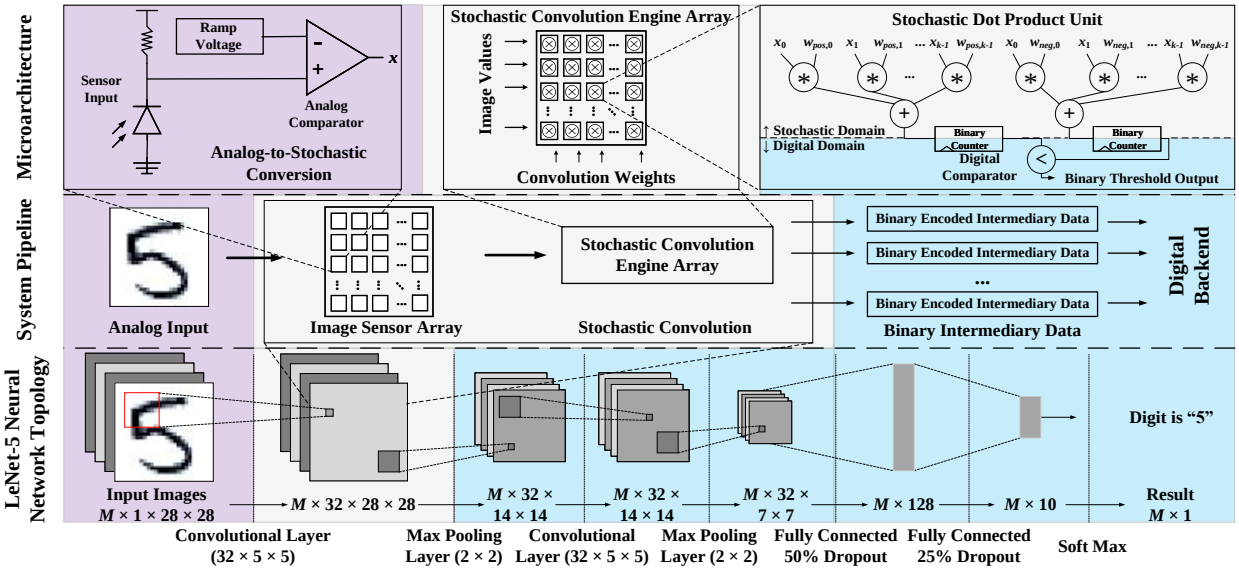


Figure 5.8: System diagram of the proposed near-sensor SC neural network. Bottom: LeNet-5 neural network topology. Middle: system pipeline. Top: microarchitecture. Purple, grey, and blue regions denote analog, SC, and BE operating domains, respectively.

is functionally equivalent to a SNG, with some modifications: (i) the inputs are analog, and (ii) a ramp signal is applied to the second input of the comparator rather than a random number generator. Despite becoming heavily auto-correlated, the bitstream generated by this conversion circuit is still usable for the SC design, because the correlation insensitive adders proposed in Section 3.1. Previous work has shown such analog-to-stochastic converters are comparable, in terms of cost and performance to regular ADCs [15, 35]. Furthermore, prior work [114] has shown such conversions operate on the order of 100 pJ, which is much lower than the energy consumed by computation (100s of nJ/image). Thus, I do not include the cost of sensor data conversion in the evaluations.

Stochastic Computing Convolutional Neural Network Layer

The SC neural network layer consists of 784 SC dot-product units shown in Figure 5.8 which process the sensor input in parallel. Because there are 32 different first layer kernels, I perform 32 parallel

convolutions per image. The convolution engines perform a basic dot-product operation followed by stochastic-to-binary conversion and an activation function. More precisely, each convolution engine implements:

$$g\left(\overrightarrow{x}, \overrightarrow{w}\right) = sign(\overrightarrow{x} \cdot \overrightarrow{w})$$

where $\overrightarrow{x}$ and $\overrightarrow{w}$ denote the input window and kernel weights respectively, and $\cdot$ denotes the dot-product operation ($\overrightarrow{x} \cdot \overrightarrow{w} = \sum_{i=0}^{k-1} x_i w_i$). The activation function outputs the sign of the dot product results and outputs either -1, 0, or $+1$. The weight inputs are shared among all convolution engines, so the cost of generating them is amortized across all units.

Since the computation involves negative numbers, the bipolar SC domain [1, +1] is a natural choice [22]. However, by employing bipolar SC, the decision point of activation functions maps to bitstreams with maximum fluctuation (i.e., unipolar value 0.5). This increases power usage and decreases accuracy. Therefore, I adopt a different approach which uses only unipolar operations by dividing the weights into positive and negative bitstreams $\overrightarrow{w}_{\text{pos}}$ and $\overrightarrow{w}_{\text{neg}}$. I then perform two unipolar dot product operations, $g_{\text{pos}} = \overrightarrow{x} \cdot \overrightarrow{w}_{\text{pos}}$ and $g_{\text{neg}} = \overrightarrow{x} \cdot \overrightarrow{w}_{\text{neg}}$, followed by two asynchronous counters to convert the results g_{pos} and g_{neg} to the BE domain. Finally, the BE activation function is implemented by a comparator. As shown in Figure 5.8, the rest of the neural network operates in the BE domain.

5.3.3 Experimental Methodology

I use the MNIST database [52], a standard machine learning benchmark for handwritten digit recognition, to evaluate accuracy. The benchmark consists of M = 70,000 images of handwritten digits (0 to 9); each image uses a 28×28 8-bit greyscale encoding. A subset of 60,000 images are used to train the neural network, while the remaining 10,000 images are used to test its accuracy. *Classification accuracy* is defined as the ratio of correctly classified test images to the total number of test images. The *misclassification rate* is defined as one minus the classification accuracy. These metrics are often multiplied by 100 and reported as a percentage. All neural network training was

performed using the TensorFlow framework [3] and the Keras library [31] using a NVIDIA Titan X GPU. For each SC design, I built a custom C++ model to evaluate its accuracy.

Previous work on SC neural networks [19, 49] evaluates neural network topologies with only fully connected layers and achieves misclassification rates between 1.95% and 2.41%. On the other hand, this work uses the LeNet-5 topology which has both convolutional and fully connected layers, and achieves misclassification rates around 1%. In practice, the number of convolutional layer kernels and the size of the kernels used in LeNet-5 vary. For this evaluation, I use a variant provided by the Keras library which has the topology shown in Figure 5.8.

A key tradeoff in SC is reducing precision to enhance performance. To quantify the impact of reduced precision on classification accuracy, I build separate neural network models which execute the first layer of LeNet-5 at different precision levels (2 to 8 bits). I also replace the standard rectified linear activation function with a sign function, which does not impose a significant accuracy loss, but has a much simpler implementation in SC. I do not execute subsequent layers in the SC domain since precision losses would compound and require longer bitstreams to achieve accurate results.

For comparison, I evaluate how precision reduction affects the fully BE implementation. The experimental results show that only quantizing the first layer weights and replacing the activation function with sign detection reduces classification accuracy by several percentage points (up to 6.85% misclassification rate for 4-bit precision). However, by retraining the rest of the neural network weights, the neural network model is able to recover from the noise introduced by losses in precision and the new activation function (Table 5.2). Interestingly, I find that I can reduce precision down to 3 or 4 bits and still achieve excellent misclassification rates (below 1%) after retraining. Since the training process is also noisy, the classification accuracy does not always exhibit monotonically decreasing behavior as precision is reduced.

Bit reduction of SC designs exhibits similar accuracy losses but leads to exponential run time reduction and energy savings. However, SC convolutions present unique challenges. SC can be inexact at near-zero input values, and output values are sensitive to errors. Prior work [17] shows that a non-trivial percentage of neural network values are near zero, so I use weight scaling and soft thresholding as proposed by Kim et al. [49] to mitigate these errors. Weight scaling normalizes

Table 5.2: Misclassification rates for fully binary-encoded and hybrid stochastic-binary designs, and throughput-normalized power, energy efficiency, and area results for binary-encoded and SC convolution designs.

	Design	8 Bits	7 Bits	6 Bits	5 Bits	4 Bits	3 Bits	2 Bits
Misclassification Rate (%)	Binary	0.89	0.86	0.89	0.74	0.79	0.79	1.30
	Old SC	2.22	3.91	1.30	1.55	1.63	2.71	4.89
	This Work	0.94	0.99	1.04	1.12	1.04	2.20	43.82
Normalized Power (mW)	BE	40.95	72.80	121.52	204.96	325.36	501.76	683.20
	This Work	33.17	33.55	33.26	33.01	33.20	29.96	28.35
Energy Efficiency (nJ / frame)	BE	670.92	596.38	497.74	419.76	333.17	256.90	174.90
	This Work	543.42	274.82	136.22	67.60	34.00	15.34	7.26
Area (mm^2)	BE	1.313	1.094	0.891	0.710	0.543	0.391	0.255
	This Work	1.321	1.282	1.240	1.200	1.166	1.110	1.057

the values of each convolution kernel to use the full dynamic range [1,+1] while soft thresholding forces a result to zero if it is within some small threshold. Finally, I also employ the retraining techniques introduced earlier in the BE domain of the design.

5.3.4 Evaluation Results

I now compare the resulting classification accuracy using SC with the new correlation insensitive adder and new multiplier (Chapter 3), and the conventional adder and multipliers used in prior work. Table 5.2 shows misclassification rates (lower is better) for each design. The results indicate that the new adder and multiplier generally achieve lower misclassification rates than those in prior SC work (up to 2.92% better). The hybrid-stochastic binary neural network design is also able to achieve misclassification rates which are within 0.05% and 0.25% of the BE design for 8-bit and

4-bit precision respectively. Furthermore, the results show that retraining the neural network model can compensate for errors introduced by both precision reduction and SC. In particular, using the more accurate adder and multiplication scheme there is less error that the retraining process must compensate for than the old adder. Finally, the results confirm that there is significant opportunity for precision reduction in SC, which translates to exponential reductions in bitstream lengths and better run times.

I synthesize, place-and-route, and measure power using Synopsys Design Compiler, IC Compiler, and PrimeTime for the SC and BE designs using a 65 nm TSMC library. For comparison, I evaluate a sliding window convolution engine as the BE baseline design similar to the stencil computation accelerators in [41]. Activity factors for power measurement are recorded using traces based on MNIST test images and weights from the TensorFlow model.

Table 5.2 shows the throughput-normalized power, energy efficiency, and design area for both SC and BE convolution designs. Power measurements are throughput-normalized relative to the SC design. For instance, a BE design operating at 0.25× the throughput and 2× the power relative to a SC design would have a throughput-normalized power of 8× relative to the SC design. Since run times of SC designs decrease exponentially with lower precision, I find that the BE design must operate at exponentially higher frequency and power to match the increase in throughput. Finally, I find the area and energy costs of the SC number generators are higher than a single SC dot product unit, but the cost is shared and amortized over many units.

Since the actual operating frequency will vary across application demands, I contrast the throughput-normalized power between the SC and BE designs. Throughput-normalized power is more representative of energy efficiency since it is more agnostic to the differences in frequency and number of parallel units in the design. In terms of energy efficiency, the proposed design breaks even with BE designs at 8-bit precision and is 9.8× more energy efficient at 4-bit precision. Furthermore, it achieves these gains with better classification accuracy than prior work [49].

Finally, I see that the SC convolution design achieves reasonable area overhead relative to the BE one. The SC convolution engine exhibits virtually no change in resource utilization since precision in SC only affects the length of the bitstreams. However, BE designs benefit from linear area

reductions since reduced precision narrows the datapath. I find that the proposed design achieves roughly the same area as the BE design at 8-bit precision but is 2× larger than the BE design at 4-bit precision.

5.3.5 Summary

In this section, I showed how SC can be integrated into a hybrid stochastic and BE accelerator for neural network applications. The key contribution was to expose the underlying errors in stochastic computation to the training process. This allows the training process for the neural network model to adjust the model to account for the behavior of stochastic circuits. These adjustments ultimately reduces accuracy losses due to quantization and SC errors while maintaining significant energy efficiency gains.

5.4 Summary

This section evaluated several codesign opportunities across emerging applications: neural networks, support vector machine classifiers, and similarity search. Based on the architectural considerations outlined in Section 4.4 each of these applications provides some or all of the desirable codesign opportunities for SC. For hybrid stochastic-binary neural networks, I found that depending on the operating precision convolutional neural network layers can reap up to one order of magnitude improvement. I also find that the energy efficiency breakeven point with an equivalent BE accelerator is roughly 8-bits of operating precision. This result is roughly in line with the energy efficiency gains outlined in Chapter 4.

For other applications such as support vectors machines and similarity search, I find that the gains achievable by end-to-end SC accelerators are not compelling. For instance, when compared to an equivalent BE accelerator, both the accuracy and energy efficiency yields of the SC accelerator were not as good. In the case of similarity search, I concluded that while kNN has some desirable properties which makes it compatible with SC (ex. error tolerance) it does not contain sufficient

computation relative to the number of conversions required as overhead. As a result, the conversion overheads dominated power and area consumption resulting in reduced energy efficiency gains.

I conclude that SC accelerators show promising results for applications such as neural networks but do not fare as well for others. In order to yield compelling energy efficiency gains, applications need both compatibility with the limited viable precision range, high application data reuse, and high compute to conversion ratio. I expect that future work will continue to explore other application domains to identify such promising application candidates for SC.

Chapter 6

SYNTHESIS TECHNIQUES FOR STOCHASTIC COMPUTING

In Chapter 3, I showed new circuits that relied on designer insight and intuition to design to improve the accuracy of SC. Automating the process of designing new circuits is desirable as it can remove some or all of the engineering burden. Unfortunately, the design space of stochastic circuits often defies human intuition because of considerations like correlation and the time-multiplexed encoding. Most design techniques lack generality and are limited to a certain class of circuit. Thus a key challenge for stochastic computing is providing circuit design techniques for targeting arbitrary functionality as well as generating the accompanying number sequences for RNGs.

To address these design challenges, I propose borrowing techniques from program synthesis. I will show how these techniques can aid the design process of new stochastic circuits and supplement designer efforts by automating tedious or unintuitive design tasks. This section highlights two such efforts at automating design tasks for SC: (1) designing new circuits with stochastic synthesis and (2) engineering random number sequences with mixed integer linear programming.

6.1 Stochastic Synthesis for Stochastic Circuits

While there have been many recent advances in SC theory and technology, there are few general methodologies for designing new stochastic circuits for operations that do not have well-known solutions. For instance, there are no known implementations or good approximations of the trigonometric functions sine, square root, exponentiation, cosine, and tangent in SC. While there are a handful of methods for synthesizing stochastic circuits such as STRAUSS [10], they are limited to certain classes of functionality like polynomial evaluation circuits. In this work, I propose using stochastic synthesis for automating the design process of stochastic circuits for a target functionality.

6.1.1 Background: Stochastic Synthesis

Stochastic synthesis is a program synthesis technique used for superoptimization of program binaries and compiling to idiosyncratic instruction sets or architectures [84, 93, 94]. Stochastic synthesis is an instance of Markov chain Monte Carlo where the space of programs is treated as a high-dimensional space. Each program P is ascribed a cost calculated by a user-defined function $C(P)$ which captures correctness and/or optimality. The synthesizer than iteratively traverses the space of programs towards lower cost programs similar to gradient descent algorithms. Intuitively, this procedure effectively samples promising sectors of the program space since exhaustive enumeration is prohibitively expensive.

A summary of the stochastic synthesis algorithm is shown in Figure 6.1 for a target program of length I. The initial program in the stochastic search is randomly generated. From this initial program, the search iteratively generates proposals for better programs by randomly applying one of several rewrite rules $R(P)$. The set of rewrite rules must be ergodic which guarantees that given infinite resources the search will eventually explore all possible programs. Given a current program P and a candidate program $P' = R(P)$, the search either accepts or rejects the candidate program. If a program proposal is accepted, the current program becomes the proposed program. If the candidate is rejected, the current program remains the same. The candidate process is than repeated until an optimal program is found or the computational budget is expended.

The probability of a proposed program being accepted or rejected is based on its cost $C(P')$ relative to the cost of the current program $C(P)$ and is computed using the Metropolis ratio:

$$\alpha(P, P') = min(1, \texttt{exp}(-\beta(C(P') - C(P)))$$

Intuitively, this probability distribution forces the search to always accept a proposal with better cost while allowing the search to still accept less optimal programs. Accepting less optimal, higher cost programs during the search is crucial for enabling the search to escape local minima in the program space. The value for β is tuned experimentally, similar to how the learning rate is tuned for machine learning applications.

```
1:  procedure SYNTHESIS(I, β, C(X))
2:      P ← random program of length I
3:      B ← P // Initialize best program
4:      while compute budget not exhausted do
5:          R ← random rewrite rule
6:          P′ ← R(P) // Generate proposal program
7:          α ← min(1, exp(−β(C(P′) − C(P)))) // Evaluate cost
8:          if random_number(0, 1) < α then
9:              P ← P′ // Accept proposal
10:         else
11:             pass // Reject proposal
12:         end if
13:         if C(P′) < C(B) then
14:             B ← P′ // Update best program
15:         end if
16:     end while
17:     return B
18: end procedure
```

Figure 6.1: High level stochastic synthesis algorithm.

Stochastic circuits are an ideal candidate for stochastic synthesis for several reasons. First, many known stochastic circuits use a handful of gates, which limits the search space to small programs. Stochastic synthesis is notorious for poor scalability with increasing program size so confining the search to small programs significantly improves the chance of success. Second, precision or SN length in SC does not affect circuit functionality. This allows the same stochastic circuit synthesized with one SN length to generalize to longer SNs. On the other hand, BE computation requires different circuits to process different precision values. Third, relative to large software instruction sets, the number of hardware primitives is small, which reduces the search space significantly. All together, these considerations significantly reduce the search space when compared to software formulations of stochastic synthesis which have larger and more complex program spaces.

6.1.2 Synthesis Formulation

This section outlines the synthesis formulation I use to design new stochastic circuits.

Instruction Set and Program Constraints

Stochastic synthesis can be directly leveraged to synthesize circuits by abstracting circuits as dataflow programs. Existing software formulations of stochastic synthesis target software instruction sets like x86 and are agnostic to the notion of cycle count or time. Hardware design on the other hand must incorporate the notion of cycle count or time into the formulation to expose the semantics of state elements. The hardware instruction set and semantics for the stochastic synthesis formulation are shown in Table 6.1. The instruction set is reminiscent of the primitives provided by structural Verilog, and includes primitive gates (ex. AND, OR, XOR) in addition to well-known primitives for SC (ex. T-flip flop (TFF), multiplexor (MUX)). Each instruction is composed of an opcode indicating its operation, input operands, and output operands.

Unlike software formulations of stochastic synthesis, a hardware program is considered invalid if the same destination register is assigned multiple times (doubly driven wire) or the program forms a combinational loop. In this formulation, I express a circuit as a dataflow program of hardware

instructions. Programs that realize invalid circuits are ascribed maximal cost ($C(P) = 1.0$) to discourage the search from these areas. To prevent doubly driven or undriven wires, I impose single static assignment over the program and prevent rewrite rules from overriding assigned destination registers. I also require the user to specify the target program length. Finally, it is important to note that the spatial nature of hardware makes programs agnostic to instruction order.

Table 6.1: Hardware program instruction set for stochastic synthesis.

Instruction	Semantics
AND src, trg, dst	dst[n] $\leftarrow$ src[n] & trg[n]
OR src, trg, dst	dst[n] $\leftarrow$ src[n] | trg[n]
XOR src, trg, dst	dst[n] $\leftarrow$ src[n] $\oplus$ trg[n]
NOT src, dst	dst[n] $\leftarrow \neg$ src[n]
PASS src, dst	dst[n] $\leftarrow$ src[n]
DFF src, dst	dst[n] $\leftarrow$ src[n-1]
TFF src, dst	dst[n] $\leftarrow$ dst[n-1] $\oplus$ src[n-1]
MUX src, trg, sel, dst	dst[n] $\leftarrow$ src[n] if sel[n] else trg[n]

Specification and Cost Function

The input specification to the synthesizer is a set of test cases and their target output values. A test case is defined as a set of input bindings to input operands and desired output SN value. The user must specify the number of input operands which can be derived from the target function to synthesize. For a given program P, I define the cost of a program as the average absolute error between the expected output SN value and the result SN value produced by the circuit. The result SN value is calculated by simulating the circuit for each set of input bindings. I define the total cost $C(P)$ of the program as the average absolute error over all test cases. This ensures that the cost function is agnostic to SN length and test case count which reduces how often I need to tune β.

To generate test cases, I select from LFSR, Van der Corput, or Halton (base = 3) sequences for generating input SN operands. Test case selection and coverage directly impacts the cost function and ultimately the behavior of the synthesized circuit. Test case selection can also be manipulated to express the conditions under which the desired stochastic circuit will operate. For instance, test cases can be intentionally generated with correlated or uncorrelated inputs to tell the synthesis process to find a circuit that operates correctly with correlated or uncorrelated operands respectively.

If the optimal operating conditions are unknown, the user can ask the synthesizer to determine what the optimal correlation should be. To do this, the user supplies duplicate operands to the synthesizer and lets the synthesizer determine which ones to use. For example, if I were to try and synthesize a stochastic subtractor but did not know whether the input operands needed to be correlated or uncorrelated, I would supply three input SNs X, X', and Y, where X and X' have the same value. Test cases would be generated such that X and Y are uncorrelated and X' and Y are positively correlated. The synthesizer will then figure out whether the uncorrelated or correlated inputs are unnecessary and will leave one disconnected if necessary in the synthesized result. The key drawback of this technique is that it increases the search space of potential programs by introducing additional input operands.

Program Generation and Rewrite Rules

Candidate programs are generated by randomly selecting from a set of rewrite rules. The set of rewrite rules is typically a combination of rules which locally perturb the program and rules which impose more global modifications. Each rewrite rule is assigned a selection probability which governs how often it is used in the search. I generally find that ascribing higher selection probability to local rewrite rules and lower probabilities to more global rewrite rules works well. Intuitively, this allows the search to spend sufficient time exploring local minimum before moving on to another local minimum.

The set of rewrite rules used by the stochastic synthesizer is shown in Table 6.2. Unlike traditional software program synthesis, I do not have a swap instruction rewrite rule since hardware dataflow programs are agnostic to program instruction order. I also add a swap all operands rewrite

Table 6.2: Program rewrite rules and selection probabilities. Minor rewrite rules like operand replacement have higher selection probability than major rewrite rules like random restart.

Rewrite Rule	Prob.	Description
Replace Operand	0.817	Replace a random instruction's input register with new random operand.
Replace Opcode	0.091	Replace a random instruction's opcode with a new opcode of the same arity.
Replace Instruction	0.045	Replace the entirety of a random instruction with a new randomly generated instruction.
Swap All Operands	0.045	Randomly selects two registers *ra* and *rb*. Replaces every instance of *ra* with *rb* and *rb* with *ra* in the program.
Random Restart	0.001	Replaces the entire program with a new random program.

rule which randomly selects two operands *ra* and *rb*, and replaces every instance of *ra* with *rb* and *rb* with *ra*. This effectively swaps the connectivity of two wires in the circuit netlist without requiring the synthesizer to traverse potentially many high cost intermediary circuits. Finally, I also employ random restarts [40] which is a technique that reinitializes the search to a random program. This effectively forces the synthesizer to explore a different portion of the program space and potentially rescues it from getting stuck in deep local minima in the search space.

In experiments, I find that the parameter $\beta = 2$ in the Metropolis ratio produces good results. Since I ascribe a maximum cost of 1.0 to invalid circuits, $\beta = 2$ allows invalid circuits to be accepted with small but non-negligible probability ($\approx 13.5\%$). I still want the search to explore such invalid programs since it is often necessary to traverse invalid programs to reach new valid ones. To improve cost evaluation performance, the synthesizer performs combinational loop checks and dead code elimination.

Table 6.3: Stochastic circuit synthesis benchmarks. The stochastic synthesizer can synthesize existing stochastic circuit designs as well as new ones.

Operation	Design	Program Length	Function $f(p_X, p_Y)$	Result Size	Abs. Error	Correct
Scaled Adder	[54]	3	$\frac{(p_X+p_Y)}{2}$	3	0.027	Yes
Subtractor	[6]	1	$\lvert p_X - p_Y \rvert$	1	0	Yes
Uncorrelated Multiplier	[38]	1	$p_X p_Y$	2	0.021	Yes
Division	[28]	2	p_X / p_Y	2	0.038	Yes
Scale $\times 1/4$	[111]	4	$0.25 p_X$	5	0	Yes
Scale $\times 1/3$	[111]	4	$0.33 p_X$	5	0	Yes
Scale $\times 1/2$	[111]	2	$0.5 p_X$	2	0	Yes
Scaled ReLU	[59]	11	$max(0.5, p_X)$	16	0	Yes
Correlated Multiplier		N/A	$p_X p_Y$	4	0.035	N/A
Square Root	[38]	N/A	$\sqrt{p_X}$	5	0.024	N/A
Sine	[38]	N/A	$\frac{sin(2\pi p_X)+1}{2}$	8	0.068	N/A
Exponentiation	[22]	N/A	$p_X^{p_Y}$	7	0.031	N/A
Cosine	[38]	N/A	$\frac{cos(2\pi p_X)+1)}{2}$	10	0.15	N/A

6.1.3 Evaluation Methodology

I evaluate the stochastic synthesis formulation for both well-known arithmetic operations in addition to operations which have known but inefficient implementations. In the evaluation, I limit the set of synthesis benchmarks to unipolar stochastic circuits, but I expect the synthesis formulation to generalize to bipolar and other SC representations. Table 6.3 shows the set of synthesis benchmarks I use to evaluate the capabilities of stochastic synthesis. I deem a synthesized circuit as correct if it is logically equivalent or has the same cost or better than known solutions in prior work.

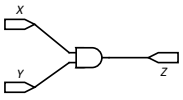
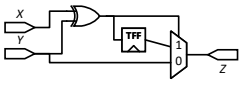
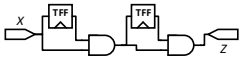
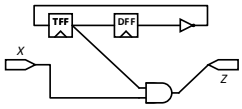
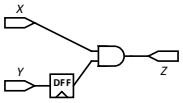
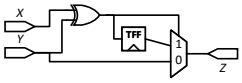
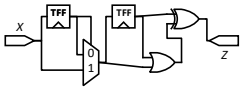
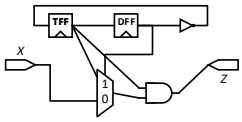
I am particularly interested in benchmarks which involve state elements and require feedback since such solutions are more likely to defeat human design intuition and existing synthesis techniques. For each benchmark, I execute the synthesizer for at least 1 million proposals, which takes less than 10 minutes, and return the solution with best cost. I increase the initial instruction count and number of proposals as needed for operations that have previously unknown solutions. Note that the number of evaluated proposals corresponds directly to the compute budget in Figure 6.1. If the synthesizer encounters an exact solution (a zero-cost solution has no error), it immediately terminates the search and returns that solution.

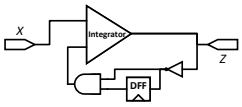
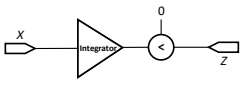
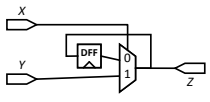
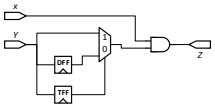
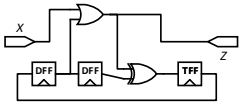
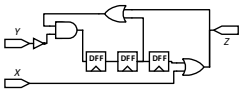
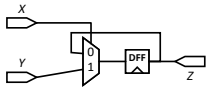
A key strength of stochastic synthesis is that even if the synthesizer fails to reach an exact solution, it will still return a solution that approximates the target function. This makes stochastic synthesis an ideal solution for attempting to synthesize SC implementations of functions which elude known SC synthesis techniques and currently have inefficient solutions. Examples of such functions include square root and exponentiation, and trigonometric functions such as sine, and cosine which can be implemented using Adaptive Digital Elements (ADDIE) [38]. However, ADDIE circuits are expensive and inefficient because they often require counters and additional auxiliary inputs. For evaluated synthesis benchmarks, I modify the target sine and cosine functions such that their range is in the unipolar domain (i.e. [0,1]).

6.1.4 Experimental Results

This section presents synthesis results and quantifies the efficacy of the synthesis formulation.

Table 6.4: Synthesis results for uncorrelated multiplication, scaled addition, constant scaling by 0.25, constant scaling by 0.33, square root, exponentiation, correlated multiplication, and polynomial evaluation.

Operation	Uncorrelated Multiplier	Scaled Addition	Scale by 1/4	Scale by 1/3
Known Solution				
Synthesized Solution				

Operation	Correlated Multiplier	Square Root	Exponentiation	Divider
Known Solution	No Known Solution			
Synthesized Solution				

Synthesis Results

In general, the stochastic synthesizer is able to quickly synthesize correct implementations of known stochastic circuits such as the subtractor, uncorrelated multiplier, and scaled adder. Note that existing tools are incapable of designing the subtractor and correlation insensitive adder which exploit or manage correlation. On average, I find the synthesizer evaluates about 2000 program proposals per second[1]. Interestingly, I find that for the uncorrelated multiply benchmark the synthesizer finds a solution that correctly includes an isolator (DFF) before the AND gate. Similar experiments indicate that the synthesizer is able to perform isolator insertion as it tries to improve the circuit. I also attempt to synthesize a stochastic multiplier to handle cases where input SNs are correlated (the original stochastic multiplier assumes uncorrelated SNs). As shown in Table 6.4, the synthesizer is able find a circuit which uses a T-flip flop, D-flip flop, and multiplexor to break the correlation between the two input SNs before feeding the SNs into an AND gate to perform the multiplication. For scaled addition, the synthesizer is able to correctly identify the correlation insensitive adder proposed in [54].

I also find that the synthesizer can discover correct implementations (Table 6.4) of scaling by constant circuits equivalent to those generated by CEASE [111]. For these particular benchmarks the synthesizer produces solutions which are suboptimal in terms of number of resources (cost function does not optimize for circuit size) but are logically equivalent to correct implementations. For instance, the synthesized solution for scale by 1/4 can reduce the multiplexor to an AND gate, and the solution for the scale by 1/3 can optimize away the multiplexor. Fortunately, standard logic reduction techniques can be employed to reduce such synthesized solutions to their smaller, more optimal implementations. Interestingly, the synthesizer finds the correlation insensitive variant for the scale by 1/4 benchmark and the correlation sensitive variant for scale by 1/3 benchmark [111]. More importantly, this result shows that the synthesis formulation can discover circuits with both sequential elements and feedback paths which is not possible with some techniques in prior works.

[1]Tool is implemented in Python. C++ implementations will be faster.

The synthesizer is also able to find approximate implementations of functions that are difficult to analyze and manually design solutions for. For instance, I was able to synthesize a new approximate square root and exponentiation ($f(x, y) = x^y$) circuit as shown in Table 6.4 which do not require expensive integrators or auxiliary SNGs as used in previous work [38]. Unlike many stochastic circuits, the synthesized square root and exponentiation circuits shown in Table 6.4 cannot be synthesized by existing SC synthesis techniques. Interestingly, I find that these synthesized solutions use a set of sequential elements reminiscent of the modulo counters used in scaling by constant circuits. Unlike the scaling by constant circuits, the feedback loop also takes p_X and p_Y as input.

For other more difficult benchmarks like scaled sine and cosine, I find the synthesizer is not able to find as optimal solutions. The synthesizer is still able to find a reasonable sawtooth wave approximation for Van der Corput generated inputs, but these solutions are far from ideal. Interestingly, the synthesized sine and cosine solutions reduce to finite state machines (FSMs) which shows the synthesis formulation is able to find approximate implementations when appropriate. The failure of the synthesis formulation to find a good solution does not preclude the existence of a better solution nor guarantee a better solution exists and alludes to some of its limitations discussed later.

For larger circuits, I find that instantiating the program with more instructions than the known solution size improves the search result. These extra instructions serve as extra degrees of freedom and many are often deleted during dead code elimination since they do not drive any part of the circuit. But by increasing the program size, it allows the search to find larger but logically equivalent variations of the known solution. This increases the number of potential solutions and hence the number of optimal local minima in the program space.

Generality of Synthesized Circuits

I find that synthesized circuits generalize to arbitrary SN length and validates the assumption that it is sufficient to synthesize a general SC solution using a fixed SN length. An example of synthesized circuits that generalize to arbitrary SN length are the constant scaling circuits. In this case, the synthesizer finds both the modulo counter-based implementation and correlation insensitive implementations which generalize to arbitrary SN length.

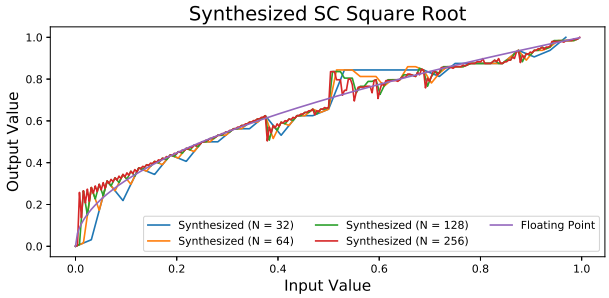


Figure 6.2: Synthesized approximate stochastic square root function. The synthesized circuit generalizes to different SN lengths.

Another instance of this precision generalization is the synthesized approximate square root circuit. Figure 6.2 compares the actual result generated by the synthesized square root circuit for several SN lengths and compares them against the expected floating point function. As the evaluation results show, the synthesized circuit behavior remains the same across all SN lengths. The circuit was synthesized using SNs using Van der Corput sequences but also works (albeit with modest errors) for SNs generated with LFSR and Halton (base = 3) sequences.

6.1.5 Summary

In this section, I proposed a new synthesis formulation using stochastic synthesis for designing new stochastic circuits. Unlike prior work, my synthesis formulation takes a functional specification and a set of testcases expressing correlation conditions as input, and generates a stochastic circuit (or reasonable approximation) as output. The key idea is to use stochastic synthesis to automate the search through the space of candidate circuits. I show that the proposed formulation is able to both identify existing stochastic circuit solutions as well as new approximations for functions like square root or polynomial evaluation. In the next section, I will explore how to to automate the search for stochastic number generators to drive stochastic circuits.

6.2 Stochastic Number Generator Synthesis With Mixed Integer Linear Programs

Number generators for SNGs which drive the generation of bitstreams are vitally important since they are one of the primary means of engineering correlation between bitstreams. The correlation between bitstreams governs the functionality and accuracy of arithmetic operations in SC, and many arithmetic operations in SC have a correlation under which they are most accurate. As a result, the selection of number sequences for SNGs is important since correlated or uncorrelated number sequences will generate correlated or uncorrected bitstreams respectively.

The key challenge is that manually engineering optimally correlated SNGs requires exploration of an exponentially large design space. For instance, exhaustively searching through all number sequences of length of 16 would mandate evaluating $16! \approx 20$ trillion potential number sequences. As a result, existing work relies on a handful of number sequences with desirable correlation properties and reasonable implementation costs such as low discrepancy sequences [8], linear feedback shift registers (LFSRs), and pulse-width modulated analog signals [75]. This leaves a large space of number sequences which may yield more accurate results than known number sequence combinations.

In this section, I propose a number sequence synthesis formulation using mixed integer linear programs to synthesize accurate arithmetic operations [56]. The synthesis formulation proposed in this section is able to generate optimally accurate number sequences. More importantly, this technique eliminates the design burden of selecting properly correlated number sequences for stochastic circuits.

6.2.1 Problem Formulation

Linear Programming (LP) is an optimization technique that models problems as a set of linear constraints over symbolic variables, and a linear objective function to minimize with. In conventional LP, variables can take any real value that satisfies the constraints. Variants such as integer linear programming (ILP) restrict variables to only take integral values. I use a variant known as mixed integer linear programming (MILP), where variables can be either integral or real-valued. The job

of the LP solver is to assign values for all variables so that they satisfy the constraints. Feasible solutions are those that satisfy all the constraints while optimal solutions are feasible solutions that are minimize the objective function. If no feasible solution exists, the problem has no solution or is unfeasible.

The proposed synthesis formulation defines a MILP problem that takes two input specifications: (1) a real-valued function specification, $f(p_X, p_Y)$, and (2) a hardware specification $h(X, Y)$. The function specification $f(p_X, p_Y)$ defines the expected value of the output SN; the inputs and output of the function specification must be within the bounds of the SC encoding (i.e. unipolar or bipolar operating ranges). In contrast, the hardware specification, $h(X, Y)$, specifies the behavior of the underlying hardware circuit. Given these specifications, the goal of the synthesis formulation is to produce two integer number sequences $S_X = \{x_1, ..., x_N\}$ and $S_Y = \{y_1, ..., y_N\}$ for the SNGs of X and Y respectively. Each $x_i, y_i : \mathbb{N}$ is within the range $[0, N)$ and is unique within its sequence. The goal is to have S_X and S_Y to approximate $f(p_X, p_Y)$ when used to generate the SNs X and Y for the hardware circuit described by $h(X, Y)$. For instance, to synthesize the optimal number sequences for stochastic multiplication using a two-input AND gate, I would specify $f(p_X, p_Y) = p_X p_Y$ and $h(X, Y) = X\&Y$. The proposed synthesis formulation does not actually synthesize the circuit itself but rather tries to fit the SNG number sequences around the specified hardware circuit to best approximate the function specification.

6.2.2 Synthesis Formulation Constraints

I now define the MILP constraints used in the synthesis formulation to generate number sequences. Instead of directly synthesizing the number sequence itself, I synthesize the actual SNs that correspond to each value. To encode the number sequences S_X and S_Y, I define two symbolic matrices of indicator variables denoted $X_{i,j}$ and $Y_{i,j}$ where i denotes the row index and j denotes the SN offset (Figure 6.3a). These two symbolic matrices will encode the number sequences for the X and Y SNGs and are constrained such that:

$$\forall i \in [0, N], j \in [0, N) : \; X_{i,j} \in \{0, 1\}, \; Y_{i,j} \in \{0, 1\}$$

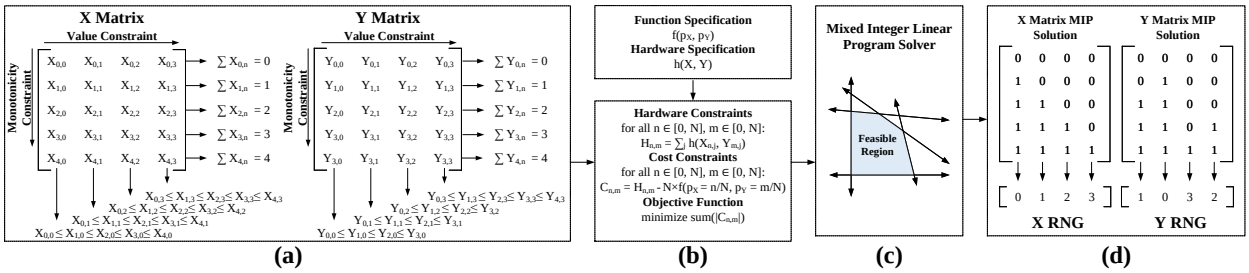


Figure 6.3: Mixed integer linear program synthesis formulation and flow. (a) Value and monotonicity constraints. (b) Hardware functionality constraints. (c) Apply mixed integer linear program solver. (d) Decode synthesized solutions.

The ith row of each matrix effectively encodes the SN encoding for the value i/N. Under this encoding, the sum of each row must equal i since, under unipolar SC representations, each position that is 1 in the SN has a weight of +1. Since an SN can take $N + 1$ possible values, the matrix of indicator variables has $N + 1$ rows where each row is N variable wide. I refer to this set of constraint as the *value constraints* which are:

$$\forall i \in [0, N] : \sum_{j=0}^{N-1} X_{i,j} = i, \sum_{j=0}^{N-1} Y_{i,j} = i$$

I also introduce *monotonicity constraints* which require the values in each column of S_X and S_Y to increase monotonically. Suppose two SNs X_i and X_{i+1} encoding the values i/N and $(i + 1)/N$ respectively and are generated from the same number sequence S_X. The key insight is that if a bit at position n in X_i is 1, then the bit at position n must also be 1 in X_{i+1}. This is because if $S_X[n] < i/N$ for a given position n, then it must be the case than $S_X[n] < (i + 1)/N$. Therefore, I have the constraints:

$$\forall i \in [0, N), j \in [0, N) : \; X_{i,j} \leq X_{i+1,j}, \; Y_{i,j} \leq Y_{i+1,j}$$

The monotonicity constraint combined with the value constraints enforces uniqueness in that (1) no two SNs encode the same value, and (2) each encoded number within each sequence is unique.

Table 6.5: Integer linear program constraint encodings for basic logic gates.

Gate	Constraint Encoding
Z = AND(X, Y)	$Z \geq X + Y - 1, Z \leq X, Z \leq Y, 0 \leq Z \leq 1$
Z = OR(X, Y)	$Z \leq X + Y, Z \geq X, Z \geq Y, 0 \leq Z \leq 1$
Z = XOR(X, Y)	$Z \leq X + Y, Z \geq X - Y, Z \geq Y - X$ $Z \leq 2 - X - Y, 0 \leq Z \leq 1$
Z = NOT(X)	$Z = 1 - X, 0 \leq Z \leq 1$

To express the desired circuit functionality, I convert the underlying hardware function $h(X, Y)$ into its equivalent MILP formulation and set the objective to minimize absolute error. I assume a set of constraints $H_{X,Y}$ represents the set of *hardware constraints* that enforces the hardware functionality of $h(X, Y)$. MILP formulations of Boolean logic gates such as AND, OR, NOT, and XOR are shown in Table 6.5. I implement multiplexors (MUX) using compositions of basic logic gates. State elements like D-flip-flops (DFFs) are implemented by passing the previous cycle value in the SN. New indicator variables are introduced as necessary to express each constraint. Given the circuit outputs, capturing error is expressed by:

$$\forall n, m \in [0, N] : \; H_{n,m} = \sum_{j=0}^{N-1} h(X_{n,j}, Y_{m,j})$$

$$\forall n, m \in [0, N] : \; C_{n,m} = H_{n,m} - \text{enc}(f(p_X = \text{dec}(n), p_Y = \text{dec}(m)))$$

Where $C_{n,m}$ captures the error between the target functionality and the resulting SNs of the synthesis formulation. An encoding function $\text{enc}(p_Z)$ converts the function result $p_Z = f(p_X, p_Y)$ to the number of 1-bits expected in the output SN Z. Similarly, an inverse function $\text{dec}(p_Z)$ converts the number of expected 1-bits in an SN Z back to a value p_Z. For unipolar circuits, $\text{enc}(p) = N \cdot p$,

because $p \in [0, 1]$ whereas $H_{n,m}$ is in the range $[0, N]$. For bipolar circuits, $\text{enc}(p) = N \cdot (p + 1)/2$ because $p \in [-1, 1]$. I then minimize the cost over the absolute error as the objective function[2]:

$$\text{minimize} \sum_{n=0}^{N} \sum_{m=0}^{N} |C_{n,m}|$$

The absolute value function is implemented using two auxiliary variables per term. Given a cost term $C_{n,m}$, I define two auxiliary variables $t_{n,m+}$ and $t_{n,m-}$ and impose the constraints:

$$\begin{aligned} \forall n \in [0, N], m \in [0, N] : t_{n,m+} - t_{n,m-} &= C_{n,m} \\ |C_{n,m}| &= t_{n,m+} + t_{n,m-} \\ t_{n,m+} \geq 0,\ t_{n,m-} \geq 0,\ C_{n,m} &\geq 0 \end{aligned}$$

If $C_{n,m}$ is positive than $t_{n,m+} = C_{n,m}$ and $t_{n,m-} = 0$, otherwise $t_{n,m+} = 0$ and $t_{n,m-} = -C_{n,m}$. The absolute value of this cost component is then expressed as $t_{n,m+} + t_{n,m-}$. Since I am minimizing over the cost terms, the solver should force either $t_{n,m+}$ or $t_{n,m-}$ to be zero since other assignments to these variables would be suboptimal.

The resulting number sequences S_X and S_Y can be recovered from the variables $X_{i,j}$ and $Y_{i,j}$ by summing over the columns of the vector respectively and subtracting the sum from N. Recall that SNs are generated by taking the number sequence value s and checking if it is less than the target value x. If $s < x$, the D/S converter emits a 1 otherwise it emits a 0 which means the number of zeros is proportional to the number of values where $s < x$. I can decode the number sequence values by summing the number of zeros in each column and subtracting from N (Figure 6.3d). More precisely:

$$\forall i \in [0, N] :\ S_X[i] = N - \sum_{j=0}^{N} X_{i,j},\ S_Y[i] = N - \sum_{j=0}^{N} Y_{i,j}$$

The constraint encoding and solver flow for a SNG of length $N = 4$ is shown in Figure 6.3.

---

[2]Mean squared error (MSE) formulations can be done using quadratic programming but are significantly slower. I find that average absolute error is a good approximation for MSE.

6.2.3 Optimization Constraints

I now introduce two constraint optimizations which I use to improve the performance of the synthesis formulation.

Initial and Final Sequences

I can introduce constraints for the vectors of the rows corresponding to 0 and N. Recall that the sum of each row is equal to the row index. Thus, the sum of row 0 must also be zero and the sum of row N must be N. The only way to enforce the constraints $\sum_{j=0}^{N-1} X_{0,j}$ and $\sum_{j=0}^{N-1} X_{N,j}$ is $\forall j \in [0, N) : X_{0,j} = 0, X_{N,j} = 1$ since variables are either 0 or 1, and there are N positions in the row. While this optimization appears trivial, it provides a modest improvement to solver time since the solver does not need to deduce this itself.

Relative Ordering Invariance

In Subsection 2.1.3, I showed that for combinational circuits, the numbers in two number sequences can be rotated or swapped as long as the relative pairing of numbers is preserved since this preserves correlation (relative ordering invariance). This observation is important because it means there are many solutions that are equivalent and have the same objective function value, and hence the same accuracy. This can be problematic for the solver since it must expend time exploring each equivalent solution to determine that they all have the same accuracy when verifying optimality.

Fortunately, I can exploit relative ordering invariance to eliminate these equivalent solutions by initializing one of the number sequences to any solution that satisfies the constraints. For instance, I can initialize constraints such that S_X is the ramp function {0, 1, 2, 3, ..., N-1} (or any other valid number sequence). This eliminates the task of synthesizing one of the number sequences and reduces the number of symbolic variables by half since it is no longer necessary to solve for $\forall i, j : X_{i,j}$. Once a solution S_Y is synthesized, I can rotate the number sequences or swap the number positions to transform them into other iso-accurate solutions with the same correlation. In fact, the

only important information synthesized by the solver is the relative pairing of the numbers between number sequences since this governs correlation.

6.2.4 Evaluation Methodology

I evaluate synthesis problems for known arithmetic stochastic circuits to verify that the synthesis formulation is correct. I also show that the synthesis formulation can synthesize more accurate number sequences for existing arithmetic operations. A full list of synthesis target specifications is shown in Table 6.6. I then show how the proposed synthesis formulation can be extended to support larger circuits with more inputs in Subsection 6.2.7.

The proposed synthesis formulation is implemented on top of IBM CPLEX version 12.8.0 [1]. Each benchmark was run on Microsoft Azure F72 v2 virtual machines running Ubuntu 16.04, which have 72 2.7 GHz Intel Xeon Platinum 8168 processors and 144GB of RAM. In some cases, CPLEX is not able to fully use all cores since certain synthesis problem do not contain sufficient parallelism for CPLEX to exploit when exploring the solution space.

I can also relax the optimality constraint on the problem or ask the CPLEX solver to return the best solution given a computation budget. To do this, I can relax the optimality gap g and instead synthesize solutions that are within some percentage of optimal. The optimality gap is a CPLEX parameter which is the distance between the minimal solution and the lower bound for all solutions. For instance, I can set $g = 0.05$ to express that I want a solution that is within 5% of optimal[3]. This allows the solver to trade off optimality for speed. I can also bound the solver time and configure it to return the best solution found in that given computation time budget. This does not provide a bound of the objective function optimality but again trades optimality for speed. We generally find that for difficult problems, the solutions are still within 5% of the globally optimal objective function value.

To evaluate correctness, I use the synthesized number sequences to evaluate the average absolute error across all possible input value combinations. I compare the average absolute error against

[3]A gap of 0% means the solver has returned the optimal solution

those produced by using well-known number sequences for S_X and S_Y for each functional unit as the baseline accuracies. Most prior work uses a combination of linear feedback shift register (LFSR), Van der Corput (VDC), Halton sequences, and ramp sequences which are generated by known mathematical functions. When evaluating error for an arithmetic circuit, I report the baseline average error using number sequence combinations identified in previous work.

6.2.5 Evaluation Results

I now present evaluation results for the synthesis targets defined in Table 6.6. For benchmarks like maximum, division, and minimum the synthesis formulation generates optimally positively correlated results which results in no errors and match the known optimal solutions in the literature [6, 27]. For saturating addition, the synthesizer correctly identifies maximally negatively correlated number sequences which also results in no accuracy errors. For benchmarks like multiplication, the synthesis formulation finds number sequences which results in more accurate results across all unipolar and bipolar encodings. While the proposed formulation optimizes average absolute error, the results are still comparable or more accurate than baseline number sequences in terms of mean squared error (MSE). Finally, I find that synthesis times vary significantly between formulations but generally increase exponentially with search space size.

Examples of synthesized sequences for multipliers with SN length of $N = 16$ are shown in Table 6.7. I find that the synthesized results for multiplication achieve better overall accuracy by 2.5× over previously solutions using a ramp, Van der Corput, or Halton sequences (Figure 6.4). In particular, I find that the average SCC over all bitstreams generated by the synthesized sequences is zero which is better than the average SCC of prior work. The evaluation results also show I can generate more accurate bitstreams for the squaring function by up to 20× for N = 128. Unlike multiplication, the accuracy of the squaring circuits depends on the autocorrelation within the single input SN and existing sequences do not produce as accurate baselines.

The key strength of the proposed formulation is that provided sufficient computation resources it can *automatically identify optimally correlated deterministic number sequences*. Table 6.7 compares the SCC for unipolar and bipolar multiplication and shows that the *average* SCC across all SNs

Table 6.6: Number sequence synthesis benchmarks, specifications, and average absolute error (lower is better) results compared against baseline solutions. Synthesized solutions are as accurate or more accurate than baseline solutions. Solutions marked with † are not optimal but are still more accurate than baseline sequences.

Function	Encoding	Function $f(p_X, p_Y)$	Hardware Z=h(X, Y)	Synthesized Error (This Work) N=16	N=32	N=64	N=128	Baseline Error N=16	N=32	N=64	N=128
Multiplier	Unipolar	$p_X p_Y$	Z=AND(X, Y)	0.016	0.009	0.005†	0.003†	0.032	0.020	0.012	0.007
	Bipolar		Z=XNOR(X, Y)	0.061	0.035	0.020†	0.010†	0.067	0.043	0.023	0.014
Adder	Unipolar	$\frac{(p_X+p_Y)}{2}$	Z=MUX(X, Y, R)	0.016	0.008	0.004	0.002	0.016	0.008	0.004	0.002
	Bipolar			0.031	0.016	0.008	0.004	0.031	0.016	0.008	0.004
Squarer	Unipolar	p_X^2	W=DFF(X) Z=AND(X, W)	0.015	0.008	0.004	0.002†	0.030	0.030	0.034	0.040
Saturating Adder	Unipolar	`max` $(1, p_X + p_Y)$	Z=OR(X, Y)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Divider	Unipolar	$\lfloor p_X/p_Y \rfloor$	Z=MUX(X, W, Y) W=DFF(Z)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Maximum	Unipolar	`max` (p_X, p_Y)	Z=OR(X, Y)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Bipolar			0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Minimum	Unipolar	`min` (p_X, p_Y)	Z=AND(X, Y)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Bipolar			0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 6.7: Synthesized number sequences compared to existing solutions (N=16). Multiplier sequences on average are more optimally uncorrelated.

Functionality	Synthesized Sequences (This Work)	Baseline Sequences	Synthesized SCC	Baseline SCC
Unipolar	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]	0.0	0.45
Multiply	[6, 13, 1, 10, 8, 3, 15, 4, 11, 0, 12, 7, 5, 14, 2, 9]	[8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15, 0]		
Bipolar	[0, 1, 2, 3, 4 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]	[0, 1, 2, 3, 4 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]	0.0	0.23
Multiply	[6, 13, 1, 10, 8, 3, 15, 4, 11, 0, 12, 7, 5, 14, 2, 9]	[0 ,1 ,3 ,7 ,15 ,14 ,13 ,10 ,5 ,11 ,6 ,12 ,9 ,2 ,4 ,8]		
Square	[2, 0, 8, 12, 11, 7, 6, 1, 4, 13, 14, 5, 9, 10, 3, 15]	[0 ,5 ,11 ,2 ,7 ,12 ,4 ,9 ,14 ,1 ,6, 11, 2 ,8 ,13 ,4]	–	–

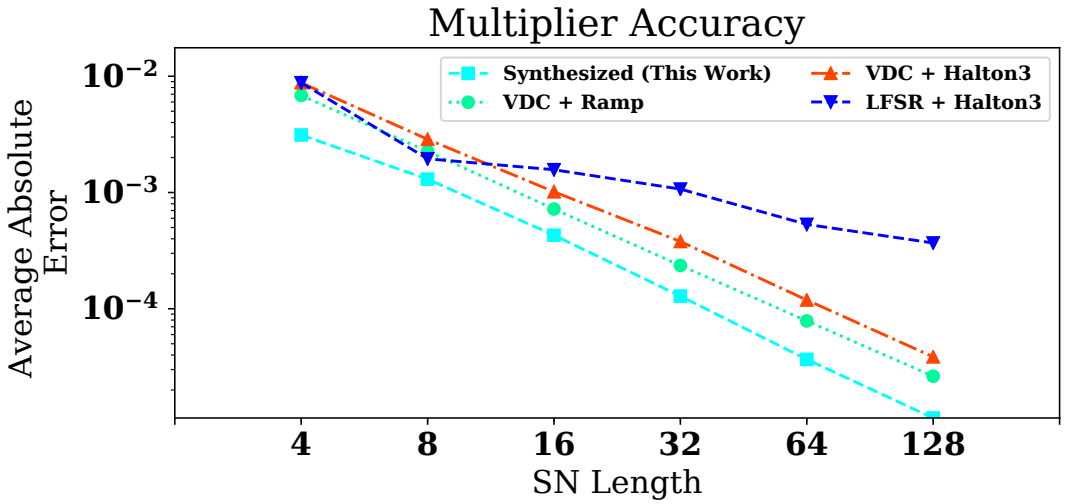


Figure 6.4: Multiplication accuracy using synthesized number sequences and prior work. Synthesized number sequences are optimally accurate.

generated by synthesized number sequences is better than using the ramp and Halton sequences proposed in [54]. Because the synthesized sequences are optimal, the resulting accuracy of the underlying circuit is also optimal. Recall for combinational circuits, I can exploit relative ordering invariance to yield number sequences that achieve the same correlation and accuracy. As a result, number sequences with the same correlation and accuracy can be recovered from the results.

6.2.6 Power, Area, and Energy Evaluation

Number sequence generators are significantly larger and consume more power than individual stochastic arithmetic operations. For instance, a 4-bit LFSR is still 41.5× larger and consumes 173.8× more power than a single stochastic multiply. As a result, it is common practice to exploit judicious application data reuse to amortize the cost of number sequence generators and conversion circuits across many arithmetic operations [42].

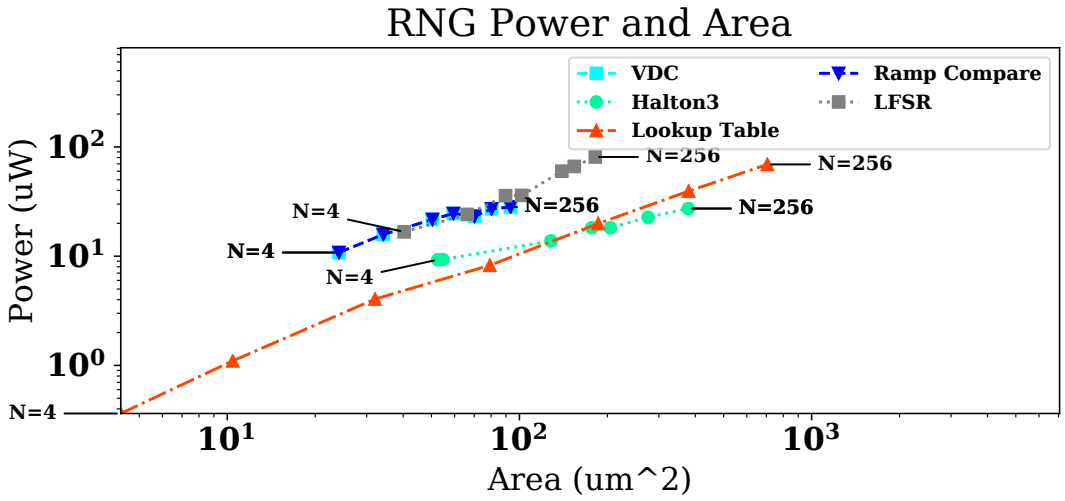


Figure 6.5: Power and area of individual number sequence generators for $N = 4, 8, 16, 32, 64, 128$.

I evaluate the power, area, and energy cost of the synthesized random number sequences by using Synopsys Design Compiler, IC Compiler, and PrimeTime Power using a 65 nm TSMC library to measure power, area, and energy. I compare VDC, Halton3, and LFSR sequences with a lookup table architecture for synthesized number sequences since synthesized sequences have no obvious efficient hardware implementation. For a two-input function, I only need to use a lookup table to generate S_Y since I can initialize S_X to a ramp function which in turn can be used to drive the lookup table selection. The architecture for this pair of number sequence generators is shown in Figure 6.6.

To compare scalability, I evaluate number sequence generators for $N = 4, 8, 16, 32, 64, 128$. Figure 6.5 shows the power and area comparison of several known number generators. Compared to existing number sequence generators, individual synthesized number sequence generators consume more power and area for N=128 length SNs by up to 4.7× and 2.5× respectively; but for shorter bitstream lengths, this gap quickly closes. While these relative gaps may appear large, in the context of an end-to-end accelerator, this power and area differential has limited impact.

To measure the overall impact on power and area of the number sequence generators, I evaluate a convolution and matrix-vector multiplication kernel, and compare the estimated power and area

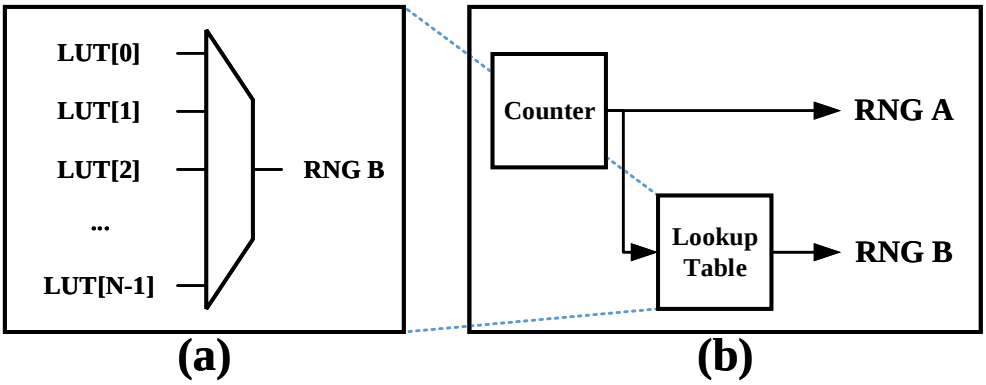


Figure 6.6: (a) Lookup table number generator for synthesized number sequences. (b) A counter serves as both a number sequence source and lookup table driver.

overheads. For the convolution, I assume an 8×8 input tile and a 5×5 kernel window. For the matrix-vector multiplication, I assume a 32×32 matrix and 32-dimensional weight vector. I measure power and area using post-placement and route results using random simulation data inputs to make the results data agnostic. I find that number sequence generators consume less than 1.6% and 1.3% of total power and area of the end-to-end designs respectively using the synthesized number sequence generators. Compared to designs using a ramp and VDC number sequence generator, overall a design using the synthesized number sequence generators increase the overall accelerator area by a mere 4.5%. Furthermore, not all of the additional area overhead is attributed to the number sequence generators. This is to be expected since end-to-end accelerators are dominated by compute, S/D, and D/S conversion overheads. As a result, a marginal increase in SNG size has minimal impact on overall power and area while offering improved accuracy. Depending on the application context, these energy and accuracy tradeoffs can be justified.

A key limitation of the synthesis formulation is its scalability. Unfortunately, synthesis times scale poorly with SN length since each additional degree of freedom doubles the search space size. In practice, most solver synthesis times are faster than worst case exponential times because solvers are able to prune away large portions of the space. For difficult MILP formulations, experimental

evidence shows that the CPLEX solver is limited to SN lengths between 64 and 256. But this drawback is not fatal to the technique since prior work shows that SC is only viable at low operating precisions [53, 71]. In addition, results from synthesis runs can be cached and reused so the speed of the synthesis for difficult problems can be amortized over many uses.

6.2.7 Scaling to Multiple Input Circuits

Solver-aided techniques are notorious for having poor scalability since the solution space grows exponentially with the number of variables in the synthesis formulation. As a result, I find that directly synthesizing multiple number sequences for an entire circuit can exceed reasonable MILP solver times for problems with many constraints, or constraints that make it difficult to prune the search space for the solver. To scale to additional inputs or larger circuits, I decompose the synthesis problem into smaller subproblems which each can be solved individually. The key insight is that in many cases an N-input circuit, can be decomposed into $N - 1$ smaller two-input circuits, each which have their own function specification $f_n(p_X, p_Y)$ and hardware specification $h_n(X, Y)$.

Figure 6.7(a) shows an example of how a fused-multiply add can be decomposed into two subproblems as shown in Figure 6.7(b). Notice that each subproblem encapsulates its own component of the circuit functionality and hardware component. For the fused-multiply add circuit, I have two subproblems: one for the multiplication (subproblem 0) and one for the saturating addition (subproblem 1). I would first synthesize the number sequences for subproblem 0 (the multiplication) since it occurs first in the topological ordering of the circuit. This synthesis problem will generate the optimal number sequences for the SNGs of X_0 and X_1. Using the synthesized number sequences for X_0 and X_1, I exhaustively generate all possible output SNs from the multiplier and record them in a $(N + 1) * (N + 1) \times N$ dimensional matrix Y. Each row in matrix Y corresponds to a possible SN output from subproblem 0.

To synthesize the optimal number sequence for X_2, I construct a second synthesis problem. Unlike subproblem 0, I use the output SNs in Y from subproblem 0 as one of the inputs instead of a symbolic matrix corresponding to a number sequence. I still assign a matrix of symbolic

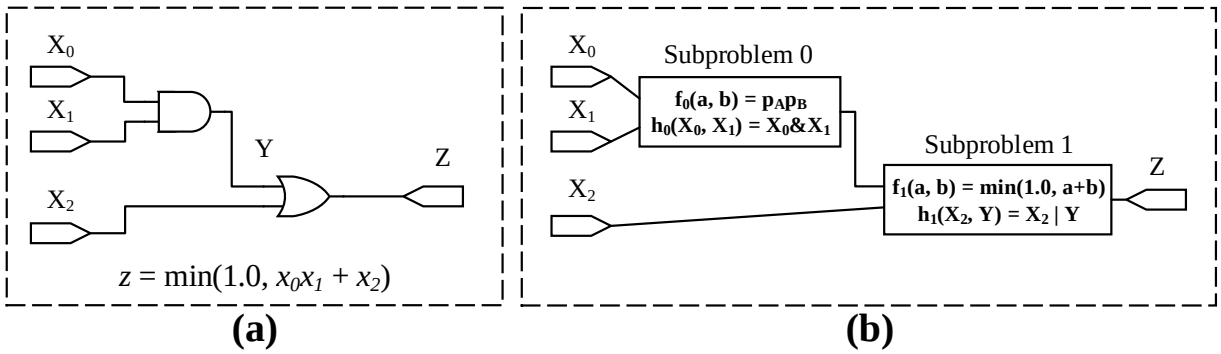


Figure 6.7: Circuits with more inputs can be decomposed into smaller subproblems. (a) Fused multiply-add with three inputs decomposed into (b) two subproblems.

variables for X_2 since I still need to identify the optimal number sequence. I then exhaustively encode hardware and function constraints, and solve for the optimal number sequence for X_2.

One potential problem with this approach is that the number of resulting SNs generated by subproblem 0 increases quadratically with SN length. To mitigate this, I apply deduplication of the rows in Y before attempting the second synthesis problem. The key insight is that the first subproblem may generate redundant SNs (identical SNs) which can be deduplicated. The degree of redundancy depends on the encoded computation and hardware specification.

Decompositions present their own unique tradeoffs. For instance, using decompositions trades global optimality guarantees for scalability; solutions for each individual subproblem are only locally optimal. By partially calculating the resulting values after subproblems 0, I eliminate the need to solve for all input number sequences at the same time which dramatically improves scalability by reducing the search space size. In addition, I am able to synthesize locally optimal number sequences for intermediary portions of the computation which is a non-trivial design task. Unfortunately, the synthesized number sequence results are only optimal for that subproblem or component of the circuit and does not guarantee that the synthesized number sequences are globally optimal for the composed circuit.

Table 6.8: Average discrepancy comparison of synthesized and existing sequences with subsequence length M = 4.

Sequence Type	N=8	N=16	N=32	N=64	N=128	N=256
VDC	0.063	0.073	0.078	0.081	0.082	0.083
Halton3	0.094	0.096	0.106	0.108	0.108	0.110
LFSR	0.156	0.176	0.183	0.188	0.191	0.194
Synthesized	0.063	0.073	0.078	0.081	0.082	0.083

6.2.8 Constraint Extensions for Enforcing Low Discrepancy

I can also introduce constraints to minimize the discrepancy of number sequences. The discrepancy of a SN X with value p_X is defined as the deviation of the value of a contiguous subsequence from p_X. Intuitively, this is a measure of how evenly distributed 1s and 0s are within a SN since sequences with long runs of 1s and 0s will have subsequences which deviate markedly from the encoded value. Low discrepancy sequences can be desirable for several reasons: (1) they are often uncorrelated with SNs generated by other sequences and (2) to facilitate progressive precision [8]. For a SN X with value p_X, using a subsequence length M, I can define the average discrepancy of a sequence as:

$$D_{avg} = \frac{1}{N-M} \cdot \sum_{n=0}^{N-M-1} \left| \frac{\sum_m^M X[n+m]}{M} - p_X \right|$$

I can synthesize low discrepancy sequences by setting the cost function to minimize the average discrepancy of each row of the matrix. I still use a matrix of indicator variables to symbolically encode the number sequence values, and impose value and monotonicity constraints. Note that the relative ordering invariance optimization does not apply here since I am only synthesizing a single number sequence. The results of the synthesis formulation are shown in Table 6.8; compared to sequences like VDC or Halton the synthesized sequences have comparable average discrepancy.

6.3 Future Work: Cosynthesis of Stochastic Circuits

Stochastic synthesis and MILP-based number sequences synthesis each has their own unique limitations. Stochastic synthesis (Section 6.1) requires the user to specify the functional specification and correlation specification through test cases to synthesize the hardware specification. The ILP-based number sequence synthesis (Section 6.2) requires the user to specify the functional specification and underlying hardware specification to synthesize the optimal correlation specification. As a result, neither solution completely solves the problem where the user only has to provide a functional specification. The ideal design framework for designing stochastic circuits would only require the user to specify the functional specification and synthesize the best hardware and correlation specification. In this section, I propose a framework which combines multiple synthesis formulations to automatically generate stochastic circuits from only functional specifications. I then discuss the limitations and future work required to make this technique feasible.

A potential synthesis framework for automatically synthesizing stochastic circuits and their number sequences is shown in Figure 6.8. The key novelty of this framework is that it unifies existing automated techniques to provide a general solution to the stochastic circuit synthesis problem. More precisely, this framework takes an input specification $f(...)$ and automatically generates the hardware specification $h(...)$ and number sequences $\{S_X, ...\}$ that best approximates the specified function. The proposed technique would improve upon the prior synthesis formulations proposed in this chapter by leveraging a combination of mixed integer programs, stochastic synthesis, and enumerative search.

The potential framework first attempts to generate candidate hardware circuit specifications using a combination of stochastic and enumerative search [55,84,93], and polynomial approximation. Enumerative search is used to quickly evaluate all possible small circuits. The stochastic synthesis formulation is the same as proposed in Section 6.1 and provides a more scalable solution for iterating over the larger space of potential circuits. Polynomial approximations supplement the stochastic and enumerative search using known polynomial SC circuit synthesis techniques like STRAUSS [10]. Once a hardware specification is generated, the ILP formulation proposed in Section 6.2 can be

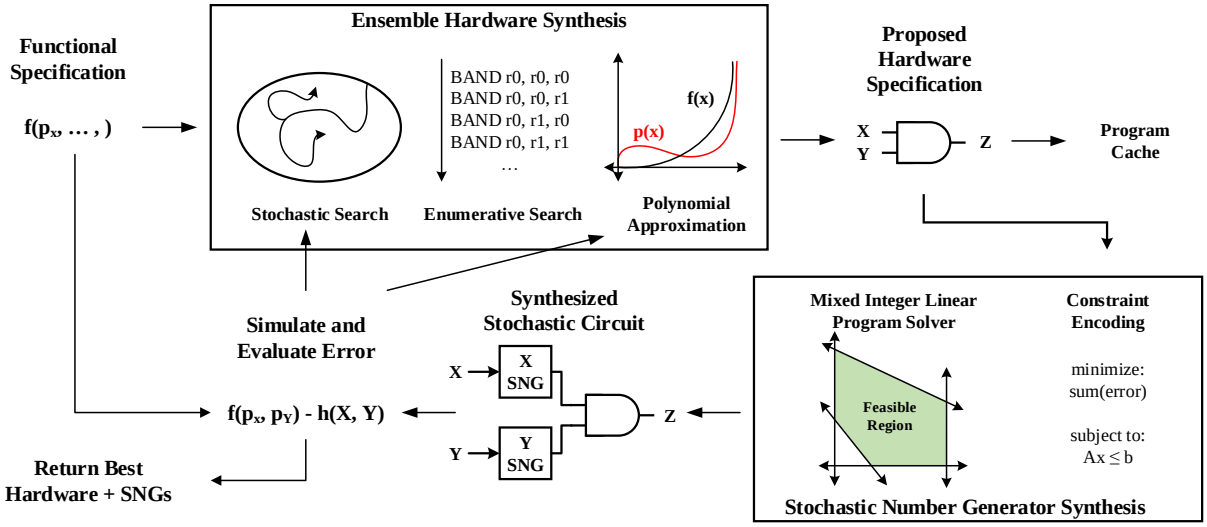


Figure 6.8: Design flow of proposed synthesis formulation. (a) Input functional specification. (b) Ensemble synthesis techniques generate candidate hardware specifications. (c) Mixed integer program solver synthesizes SNG number sequences. (d) Generated hardware and number sequences evaluated for quality.

used to identify the optimal RNG combinations. The search process can then iterate between hardware candidate generation and number sequence synthesis. Like stochastic synthesis, the search terminates when a sufficiently optimal circuit and set of number sequences is found or computation resources are exhausted.

While the cosynthesis formulation in this section provides a fully automated method for generating stochastic circuits, it does not scale well to larger circuits due to ILP solver speed which bottlenecks the number sequence synthesis step. Furthermore, the ILP formulation required to synthesize number sequences from Section 6.2 still suffers from the same scalability issues for longer bitstream length. In order to make this synthesis formulation achieve practical run time performance, future work improving the heuristics and solver run times is needed. Solver run time can be improved by adding constraints to the formulation which make it easier to solve or eliminating redundant variables. Nevertheless, the synthesis formulation outlined in this section is still valuable for future work as a potential method for fully automating most of the design burden of generating new stochastic circuits.

6.4 Related Work

This section highlights prior work in synthesizing stochastic circuits and number sequences. I also briefly compare my proposed work against prior techniques.

6.4.1 Stochastic Circuit Synthesis

A key strength of stochastic synthesis over previous techniques is that it is not limited to any particular class of functions or circuit properties. Tools such as STRAUSS [10] and ReSC [87] are limited to synthesizing feed-forward stochastic circuits without sequential elements for polynomial evaluation. To use polynomial evaluation to approximate functions, the user must identify and tune the parameters of the desired polynomial beforehand. In many cases, the desired polynomial is not obvious when trying to approximate functions. Similarly, SC synthesis techniques for sequential stochastic circuits are limited to rational functions and cannot identify ways to manage correlation.

Stochastic synthesis does not have these limitations and can synthesize both polynomial target functions and correlation manipulating sequential circuits without any prior information.

Stochastic synthesis is also able to correctly synthesize circuits for scaling by constants and reproduce the results originally shown by Ting et al. [111]. The results in my work show that stochastic synthesis is able to both identify the modulo counter-based solutions as well as the correlation insensitive solutions. I also showed that stochastic synthesis can also discover when it is appropriate to insert isolators (ex. uncorrelated multiplier) or identify ways to break correlation (ex. correlated multiplier). Furthermore, stochastic synthesis is able to find approximate implementations for a target function when an exact solution may not exist which makes it more powerful than existing SC synthesis methods.

6.4.2 Number Generator Synthesis

Number sequence selection for SC in the past has been predominantly a manual design task that relies on designer insight to identify appropriate number sequences. Prior works concentrate on improving the implementation cost or randomness of number generators. Ichihara et al. [42] propose sharing rotated versions of LFSRs to amortize implementation cost over two SNGs. Neugebauer et al. [76] propose a new number sequence generator SBoNG which improves autocorrelation and cross correlation of generated SNs. Zhakatayev et al. [119] improve SNG implementation cost by using even distribution coding. Kim et al. [48] propose a SNG that uses an auxiliary RNG to shuffle bits of an existing SN to generate a new SN similar to the decorrelator proposed in [53]. Yang et al. [118], Wang et al. [116], and Venkatesan et al. [113] all propose exploiting nanoscale devices to construct better RNGs. However, there is little prior work that focuses on exploring the remaining space of number sequences for SNGs. My work is the first to automate the design task of generating optimally correlated, deterministic number sequences for stochastic circuits.

6.5 Summary

In this section, I presented two new design techniques based on program synthesis techniques for automating the design burden associated with designing new stochastic circuits. I first proposed using stochastic synthesis to automatically identify the hardware implementation given a target functional specification and testcases to express the correlation conditions under which the circuit would operate. The results show that the stochastic synthesis formulation works well for identifying new approximations for functionalities where existing solutions do not exist. I also proposed a new mixed integer linear program formulation for automatically synthesizing the optimally correlated number sequences given a target functionality and hardware implementation. The key benefit of this technique is that it eliminates the design burden of identifying optimally correlated number sequences which can be an arcane task. Together, the proposed automatic techniques directly improves the practicality of SC by reducing the design burden required to identifying circuits for new target functionality.

Chapter 7

FUTURE WORK

In this chapter, I explore future directions to enhance the immediate practicality of SC. I also provide longer term future research directions for SC at the intersection of other disciplines.

7.1 Reconfigurability

A key challenge facing existing SC accelerator designs is their general lack of reusability. Most prior work, including the architectures explored in this thesis, assume that accelerators are implemented as fixed function ASICs using standard cell libraries. While these design evaluations are critical for proving the viability of stochastic computation, they do not yet reflect a practical accelerator design due to their lack of flexibility. Many commercial application specific processors, even if tailored to a specific application, still support some degree of reconfigurability that tolerate changes in application parameters.

In addition, typical commercial deployments of application specific circuits require high volumes to achieve compelling cost benefits. To achieve such volume, an application specific accelerator can either target a sufficiently ubiquitous application instance or improve generality to more broadly support many application instances. Identifying a sufficiently ubiquitous application instance is difficult as it requires a relatively static application landscape and application parameters. Emerging applications today are rapidly evolving and have proven to be difficult to provision application specific deployments for.

Alternatively improving the flexibility and generality of the accelerator architecture enhances the range of functionalities and also reduces vulnerability to sudden changes in application demands. Improving generality is a delicate design challenge which requires balancing the minimum amount of reconfigurability to maximize hardware reuse. Overprovisioning reconfigurability can waste

resources if they are never used. On the other hand, underprovisioning resources diminishes the utility and functionality of the reconfigurable accelerator which restricts its possible applications.

There are several key open questions concerning the design of a reconfigurable stochastic computation accelerator. The first principle design challenge is identifying sufficiently general computational elements to embed in the reconfigurable accelerator architecture. It is an open design question whether traditional general-purpose reconfigurable units such as FPGA look up tables are sufficient or whether different computational logic primitives are needed. Since stochastic units are smaller and often do not have complex datapaths like BE units, a smaller general-purpose logic unit may be more efficient for a reconfigurable SC unit. Furthermore, certain hardware units in SC are multifunctional and can be used as different operations by simply changing the correlation between operands.

Reconfigurable fabrics such as FPGAs typically contain hardened functional units for commonly used functionalities. Hardened logic units are desirable because they typically consume less power, use less area, and are more energy efficient than implementing equivalent logic using general-purpose reconfigurable elements. For instance, traditional FPGA architectures embed hardened digital signal processing (DSP) and shift registers into their architectures because they are commonly used across a large range of applications in BE computation. The types of functional units which should be hardened into a reconfigurable SC fabric is an open question. In addition, it is not clear what the density (i.e. ratio of hardened functional units to general logic units) of each type of hardened functional unit should be in such a reconfigurable fabric.

Since elements in SC are typically much smaller than BE ones they can be implemented with minimal numbers of configurable units. This can potentially improve routing pressure since logic units are no longer composed of large numbers of general-purpose logic elements as they would for BE functional units (ex. multiple lookup tables are used to implement a BE adder). As a result, a reconfigurable SC accelerator could be made exceedingly compact. In addition, since arithmetic units in stochastic are fundamentally smaller and have less complex datapaths, the maximum clock frequency of a reconfigurable fabric can be increased.

The second key design challenge is identifying the optimal routing interconnect architecture to support the accelerator logic elements. For instance, FPGAs typically use an island style routing fabric and have many redundant wires of different lengths connecting computational logic blocks throughout the fabric [20]. Since FPGAs are typically designed to support the wide datapaths for BE computation, FPGA designs typically provision a significant fraction of the accelerator fabric for wires, switch boxes, and router nodes. On the other hand, stochastic computation only requires 1-bit datapaths between logic elements and the number of logic elements per arithmetic unit is small. This highlights one of the key opportunities for SC reconfigurable accelerators: significantly reducing the routing overhead and routing complexity between and within compute units.

A third key design challenge is balancing the overhead of reconfigurable units with the overhead of logic units. This is particularly challenging because the cost of reconfigurability is high relative to the size of many SC elements. For instance, a single two-input multiplexor which may be used to marshal data signals is the same cost as some stochastic circuits (ex. scaled addition). In contrast, for BE arithmetic a n-bit wide multiplexor is small compared to the datapath of a BE multiplier. As a result, reconfigurable units will likely dominate the design cost and impose an Amdahl's limitation on the potential power, area, and energy improvements that a reconfigurable SC accelerator could yield.

In order to improve the reusability and flexibility of SC accelerators, work on reconfiguration will eventually become necessary to improve the functional range of the technology.

7.2 Alternative Computing Technologies

Stochastic computation is not bound to CMOS implementations and is implementable on top of other computing technologies such as biological, quantum, and other computing substrates. Considering alternative computing technologies is a promising direction of future work because it alters the implementation cost of logic versus state elements. A key challenge with the practicality of SC on CMOS is the high cost of state elements with respect to compute logic elements. As a result, stochastic arithmetic circuits with even a modest number of state elements can consume significantly more energy than those without. For instance, the stochastic correlation insensitive adder presented

in Section 3.1 consumes 10.7× more power and 5.6× more area than the multiplexor-based stochastic adder. In this case, the difference in microarchitecture is a single TFF and a handful of wires, and illustrates the high overheads of state elements in CMOS.

This exposes a painful tradeoff in SC: highly accurate stochastic circuits require many state elements but as a result become significantly less energy efficient. In order to change the calculus, the implementation cost between state elements and logic must be reduced to alleviate this bottleneck. This highlights a potential opportunity for stochastic computation in non-CMOS technologies where the ratio between state elements and logic is fundamentally different. However, the actual implementation costs in and tradeoffs imposed by many of these emerging technologies is not obvious and is the subject of future work. I briefly explore some of these potential opportunities and related work here.

7.2.1 Biological Computation

Bio-compatibility has long been postulated as an advantage of SC [11, 12]. However, there are few prior works that actually reconciles the two computing technologies. Historically, a key challenge has been the maturity of biological computing technologies and the cost of synthesizing biological computing components. Recent developments in the last decade however have shown promising improvements and practicality of biological computing solutions. For instance, biological computing technologies such as DNA-strand displacement reactions [120] and biological pathways [115] are promising new technologies where stochastic encodings may prove useful. In addition, there have been significant improvements in the cost of constructing synthetic DNA sequences which has enabled practical DNA displacement-based computation [2, 101].

Recent work has also proposed computer-aided design flows which can translate elementary logic into these technologies [18, 79]. Prior work has also shown the feasibility of implementing traditional analog components and digital logic components such as elementary logic gates [21]. Thus, it would not be intractable to leverage these abstractions and tools to translate stochastic circuit implementations in CMOS to biological computing solutions. Determining how to leverage SC encodings within biological computation techniques is a promising area of research. To properly

capitalize on this computing substrate, immediate future work should focus on (1) implementation and empirical measurement of the performance for elementary stochastic arithmetic operations, (2) proof-of-concept end-to-end circuit implementation including overheads such as conversion units and RNGs, and (3) end-to-end application implementations to establish the practicality of combining the two technologies.

7.2.2 Quantum Computation

Quantum computation is a re-emerging computation paradigm which has recently gained traction as an alternative to CMOS computation with new tooling and emerging implementations [100, 102]. Like SC, value representations in quantum computation are inherently probabilistic. This suggests a degree of compatibility between the two technologies but little work has been done at the intersection between these two fields. For instance, values in quantum computing are encoded in probabilistic wave functions while values in stochastic are represented as the probability of a bitstream.

Future work exploring the relationship between the two technologies could yield insights in both theory and design methodologies. For instance, it is not clear whether computational machinery from one technology can be applied to or reused (even with modifications) in the other. Quantum computing has also rapidly matured towards practical implementations in recent years enabling new explorations and applications of the technology. Reconciling these two similar technologies would be an interesting direction of future work.

7.2.3 Nanotechnology Devices

An emerging body of work has also explored the viability of stochastic circuits on top of nanotechnology devices. Like biological and quantum computing, these devices also offer a different state and logic implementation costs. Recent work has proposed stochastic computation on top of memristor devices [36], and spintronic devices [113]. Additional work focuses on improving the cost of random number generators using nanotechnology devices [116, 118]. While these works are valuable, they are still proof-of-concept and are not yet practical for actual deployment. Future

work should focus on showing prototype implementations and empirically measuring the tradeoffs for these technologies as they begin to mature.

7.3 Beyond Stochastic Computing Encodings

Stochastic computing remains one of many potential alternatives to binary-encodings. To explore the large space of potential computation encodings, some form of design automation is necessary to alleviate the design burden of searching through a potentially large number of unintuitive encodings. In this section, I describe a set of specifications which can present a useful abstraction to search for and identify new computation encodings.

To leverage automated frameworks such as constraint solvers or integer linear program solvers, the encoding problems must be decomposed into specifications that can be interpretable by these automated tools. At a high level, a computation can be described by a set of four specifications:

- A *functional specification* describing its mathematical functionality.
- A *hardware specification* describing the behavior of the circuit that operates on the bit-level representation of values.
- A *correlation specification* describing how bits in the representation are correlated.
- An *encoding specification* describing how values are mapped to and from their bit representation.

Together, these specifications present a well-formed problem for automated computer-aided design and synthesis techniques.

7.3.1 Basic Definitions

All computation operates on *values* which I denote with lowercase variables x, y, z. A value is a numeric quantity that is to be encoded and can be real, integer, or imaginary. I define a *representation* as the concrete bit values that represent the encoded value. More specifically, a representation X lies

in the space $\{\{0,1\}^{W_t}\}^T$ where W_t denotes the width of the representation at time t and T denotes the temporal length of the representation. For instance, a value x has representation X if X encodes the value x. I use $X_{i,t}$ to denote the bit at index i at time t of representation X.

Representations have two primary characteristics: (1) width and (2) temporal length. A representation can be wide in that it uses many bits over a single cycle to encode a value. For instance, binary encodings are an example of a wide representation since each value is encoded in a single cycle using multiple bits. A representation can also be temporal in that multiple cycles are required to encode a value. For instance, stochastic encodings are an example of an encoding with a long temporal representation since it encodes values as a probability over a unary bitstream. Computation takes one or more input values and transforms them into one or more output values.

7.3.2 Functional Specification

The first specification I define is the *functional specification*. A functional specification $f(x_1, ..., x_n)$ is a mathematical description of the circuit behavior given a set of input values $\{x_1, ..., x_N\}$. For instance, the functional specification of a two-input multiplier is $f(x_1, x_2) = x_1 x_2$. The functional specification conveys user intent and is the easiest specification for a designer to identify. While it is possible to generate a functional specification based on the other three specifications, the resulting specification can be difficult to interpret and of little utility.

7.3.3 Hardware Implementation Specification

The *hardware specification* $H(X_1, ..., X_N)$ describes the underlying circuit implementation where $X_1, ..., X_N$ is the representation of the input values $x_1, ..., x_N$ respectively. The hardware specification consists of an instruction set specification and dataflow graph specification. The instruction set defines available operations in the circuit. Instruction sets may consist of low-level operations such as elementary logic gates or coarser-grained instructions like fused-multiply add. Instruction sets are often defined by the designer to ground them in available hardware circuit primitives.

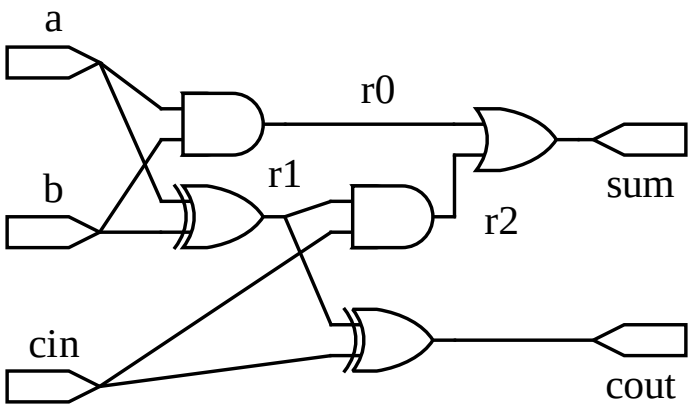


Figure 7.1: Example hardware specification for a full adder using an instruction set consisting of elementary logic gates.

The dataflow graph specifies the actual circuit implementation using the available operations in the instruction set. Edges in the dataflow graph correspond to wires, while vertices in the graph correspond to instructions and the intermediary values they produce. Each vertex has an associated instruction defining its hardware implementation. Since each circuit element can be viewed as an instruction, a circuit can be directly translated into a dataflow program in terms of the instruction set and vice versa. The dataflow graph program is important as it provides the abstraction appropriate for leveraging program synthesis techniques. An example hardware specification is shown in Figure 7.1.

7.3.4 Correlation Specification

The correlation specification defines how the bits within each cycle of a value or across cycles are correlated. Correlation is important because it also encodes information which may be manipulated in the computation. The correlation specification can concretely manifest in several different ways depending on the compute encoding and technique. For example, in SC the correlation specification manifests as the SCC or more concretely as the number sequences used to generate bitstreams. Note that the correlation specification is trivial for certain computation encodings. For example, BE

representations do not require a correlation specification since they are wide representations that are single cycle.

7.3.5 Encoding Specification

The encoding specification defines how values are mapped to the representation and how the bits of a representation are interpreted to convert them back into values. I denote the *i*th bit of the *j*th offset of a bit representation X as $X_{i,j}$. For instance, the third bit of the fifth cycle of the representation X is denoted $X_{3,5}$. More specifically, the encoding specification manifests as an encoding function $enc(x)$ and decoding function $dec(X)$. The encoding and decoding specifications must also be provided the width of the representation $W(t)$ at each cycle in the representation t. The encoding function $enc(x)$ takes a value x and produces its bit representation X.

The decoding specification $dec(X)$ interprets the bit representation X and produces its numeric value x. Note that bit representations may be distributed over multiple cycles and have fixed or variable numbers of bits per cycle. Encoding and decoding specifications for many existing computation encodings are symmetric in that $enc(dec(X)) = X$ and $dec(enc(x)) = x$. However, it is not necessary that this property holds for arbitrary computation.

7.3.6 Case Studies of Existing Encodings

I now show how each of these specifications manifest in existing computation paradigms. For simplicity, I use addition as a driving example to explore how each specification manifests. Without loss of generality, a different arithmetic operation or functionality can be used as the functional specification.

Binary-Encoded Computation

Binary-encoded computation is a wide representation with a fixed width, single cycle bit representation. The functional specification for binary-encodings is $f(x, y) = x + y$. The hardware specification for an addition is simply one of the known Boolean formulations for an adder chain. Since values

in binary-encodings do not require multiple cycles to encode, there is no correlation specification. To encode a value, the encoding specification of an n-bit value x is $X_{i,0} = \{(\text{x} >> \text{i}) \ \& \ 0\text{x}1\}$ with $W(t) = b$ if $t = 0$ and 0 otherwise. Similarly, to decode a b-bit representation, the decoding function $dec(X) = \sum_{i=0}^{b-1} 2^i * X_{i,0}$.

Stochastic Computing

Stochastic computing is an example of a time-multiplexed representation. Since stochastic computation cannot realize true addition, the functional specification for addition is either the scaled addition $f(x, y) = \lfloor N \times \frac{x+y}{2} \rfloor / N$ or the saturating addition $f(x, y) = min(1.0, x + y)$. The hardware specifications for the scaled addition is either the correlation insensitive adder $h(X, Y) = \{\{\text{XOR } X, Y, r0\}, \{\text{TFF } r0, r1\}, \{\text{MUX } r1, Y, r0, Z\}\}$ or the standard scaled adder $h(X, Y, R) = \{\text{MUX } X, Y, R, Z\}$ where Z represents the output operand and R serves as the auxiliary input with value 0.5. Stochastic computation is one instance where a functional specification may have multiple equivalent hardware specifications. The correlation specification for stochastic computation is implicitly encoded in the choice of random number generator. Finally, the encoding specification for a value x to a bitstream of length N is $X_{0,j} = x < RNG(j)$ where $RNG(j)$ gives the output of a random number generators at cycle j and $W(t) = 1$ for $0 \leq t < N$. The decoding specification would be $x = \frac{1}{N} \sum_{i=0}^{N-1} X_{0,j}$.

Multirail Computation

One generalization of stochastic computation is multirail computation which is lies between single cycle, wide, fixed-point encodings and narrow 1-bit stochastic encodings. An instance of multirail computation is Extended Stochastic Logic (ESL) proposed by Canals et al. [24]. ESL is a dual rail encoding, in other words a value is encoded as a time series using two wires per cycle and can be interpreted as a set of bitstreams. This enables additional information to be encoded per cycle (with the additional bit) but also introduces additional challenges with managing correlation between bitstreams.

The ESL encoding proposed in [24] encodes values such that the ratio of the first rail over the second rail encodes the value. More precisely, first rail encodes the numerator of the encoded value and the second rail encodes the denominator. For instance, the set of bitstreams $X = \{\{1,1\},\{1,1\},\{0,1\},\{0,1\}\}$ encodes the value 0.5 because the first rail has total weight 2 and the second rail has a total weight of 4. As a result, one encoding specification for a value x for a set of bitstreams of length N is $X_{0,n} = x < RNG(j), X_{1,n} = 1$ if $x <= 1.0$, otherwise $X_{0,n} = 1, X_{1,n} = 1/x < RNG(j)$ with a datapath width parameter $W(t) = 2$ for $0 <= t < N$. Note that there can be multiple equivalent encodings that map to the same value in ESL because some fractional values can be represented multiple ways. The decoding specification for a set of bitstreams of length N is $x = \frac{\sum_{n=0}^{N-1} X_{0,n}}{\sum_{n=0}^{N-1} X_{1,n}}$.

The hardware specification for an addition in ESL is realized using multipliers to generate a representation with least common denominators. For instance, to add two values $X = x_n/x_d$ and $Y = y_n/y_d$ together, the term $X + Y$ is generated by multiplying X by y_d/y_d and Y by x_d/x_d [24]. This yields the term $\frac{x_n y_d}{x_d y_d} + \frac{y_n x_d}{x_d y_d} = \frac{x_n y_d + y_n x_d}{x_d y_d}$. To compute the sum in ESL, a stochastic scaled addition unit (multiplexor) is used which yields a scale factor; each product is computed using an ESL multiplier (XNOR). As a result, the resulting encoded value is $\frac{x_n y_d + y_n x_d}{2 x_d y_d}$. The hardware specification is therefore the circuit given by: {{XNOR x_n, y_d, r_0}, {XNOR x_d, y_n, r_1}, {XNOR x_d, y_d, z_d}, {MUX r_0, r_1, a_0, z_n}} [24].

The ESL computation presented here is only one instance of a broader class of computation encodings. More generally, multirail encodings is not a well-understood design space because of its vast size. Whether there are more efficient approximations of existing functional units is a subject of future work which requires a degree of automation to make the design burden tractable.

7.3.7 Future Work: Encoding Synthesis

The specifications provided in this section provide a blueprint for automating the search of more efficient computation encodings. The key idea is that more efficient encodings of computation may potentially exist in the vast design space that could improve the area, power, throughput, or energy efficiency of computation. The specifications presented in this section provides one potential

encoding for automated techniques and solvers to aid the search for more efficient computation encodings. Almost all encodings for computation have been manually designed which leaves a large space of encodings which may hold more promising and efficient encodings of computation.

While these specifications form a useful abstraction to enable automated design space exploration for alternative computation encodings, they are limited by the speed of the synthesis formulation in modern solvers. As I explored in Section 6.2, the solver speed can be prohibitive to the practicality of automation techniques. To make this technique practical, significant work towards improving the formulations, search heuristics, and solver performance must be addressed.

7.4 Summary

In this chapter, I proposed several near-term and long-term avenues of future work. Short-term areas of future work include evaluating how SC can harness reconfigurable accelerator architectures to improve flexibility. Introducing reconfigurability to SC accelerators allow for more general purpose applicability across applications.

Longer term areas of future work include the exploration of alternative technologies and computing encodings. The work in this thesis focused exclusively on CMOS implementations of SC. The advent of new computing paradigms such as quantum, biological, and memristor-based substrates has opened new potential opportunities for SC. Determining how compatible SC is with each of these areas if a subject of future work. There is also an opportunity to generalize beyond SC and fixed-point encodings and explore alternative computation encodings. Alternative computation encodings may allow more energy efficient, more compact, or lower power implementations of existing arithmetic units.

Chapter 8

CONCLUSIONS

Stochastic computing is a re-emerging computing paradigm which still faces many challenges before it can become a practical alternative to binary-encoded arithmetic. In this dissertation, I proposed new circuits, architectural design guidelines, application codesign case studies, and automated techniques in the effort towards enhancing the practicality of stochastic computing.

I first proposed a set of new stochastic circuits to improve the accuracy, energy efficiency, power consumption, or area footprint of existing stochastic circuits. These included a new correlation insensitive adder and a new multiplier configuration to improve the accuracy of these ubiquitous computation primitives. I also introduced a new set of correlation manipulating circuits which were smaller and lower power than existing correlation manipulating circuits. I showed how these units could be used to improve the accuracy and implementation cost of arithmetic units such as maximum, minimum, and saturating addition. By improving the implementation efficiency and accuracy of stochastic circuits, my contributions improve the practicality of stochastic computing by making its implementation tradeoffs more competitive compared to binary-encoded arithmetic.

The second key component of this dissertation is an architectural accelerator design space exploration to understand when and why stochastic computing could yield compelling energy efficiency gains. In this study, I found that individual arithmetic units in stochastic computing were often not necessarily more energy efficient than binary-encoded operations. However, I found that after accounting for architectural overheads such as conversion units, pipelining, and control buffers, stochastic computing accelerators could achieve energy efficiency gains for a limited range of precision. I then synthesized these results into several architectural design consideration guidelines which could be used as a blueprint to guide future stochastic computing accelerator designs.

Third, I explored the energy efficiency, power, and area improvements of stochastic computing accelerators in the context of several emerging applications for stochastic computing. In each case study, I explored codesign opportunities and measured the energy efficiency and accuracy tradeoffs yielded by a stochastic computing accelerator. I found that for applications such as neural networks, stochastic computing can achieve reasonable energy efficiency gains and accuracy degradations. However, for other applications such as similarity search and support vector machines, stochastic computing encounters high overheads and achieves poor improvements.

Finally, I proposed two new automated synthesis formulations for automatically generating stochastic circuits and their associated number generators. I first proposed a stochastic synthesis formulation which I showed could synthesize existing as well as new stochastic circuits. I then proposed an integer linear programming formulation for synthesizing the random number generators for stochastic circuits. I showed that this formulation can identify number sequences that improve the accuracy of existing stochastic circuits. Unlike prior work, these formulations generalize to arbitrary circuits.

While these contributions do not make stochastic computing an immediately practical replacement for binary-encoded computation, they provide significant progress towards improving the viability of stochastic computing. There are many opportunities for future work such as exploring reconfigurable stochastic computing architectures and additional application codesign. I expect that my work combined with future work will eventually move the field of stochastic computing towards a state where it can become a practical technology.

BIBLIOGRAPHY

[1] Cplex optimizer. Accessed: 04-17-2018.

[2] Dna sequencing costs: Data. https://www.genome.gov/27541954/dna-sequencing-costs-data/. Accessed: 02-05-2019.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] A. Alaghi, W. T. J. Chan, J. P. Hayes, A. B. Kahng, and J. Li. Optimizing stochastic circuits for accuracy-energy tradeoffs. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 178–185, Nov 2015.

[5] A. Alaghi and J. P. Hayes. A spectral transform approach to stochastic circuits. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*, pages 315–321, Sept 2012.

[6] A. Alaghi and J. P. Hayes. Exploiting correlation in stochastic circuit design. In *2013 IEEE 31st International Conference on Computer Design (ICCD)* [7], pages 39–46.

[7] A. Alaghi and J. P. Hayes. Exploiting correlation in stochastic circuit design. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 39–46, Oct 2013.

[8] A. Alaghi and J. P. Hayes. Fast and accurate computation using stochastic circuits. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)* [9], pages 1–4.

[9] A. Alaghi and J. P. Hayes. Fast and accurate computation using stochastic circuits. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–4, March 2014.

[10] A. Alaghi and J. P. Hayes. Strauss: Spectral transform use in stochastic circuit synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1770–1783, Nov 2015.

[11] Armin Alaghi. *The Logic of Random Pulses*. PhD thesis, Ann Arbor, MI, USA, 2015.

[12] Armin Alaghi and John P. Hayes. Survey of stochastic computing. *ACM Transactions on Embedded Computer Systems*, 12(2s):92:1–92:19, May 2013.

[13] Armin Alaghi and John P. Hayes. On the functions realized by stochastic computing circuits. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pages 331–336, New York, NY, USA, 2015. ACM.

[14] Armin Alaghi, Cheng Li, and John P. Hayes. Stochastic circuits for real-time image-processing applications. In *Proceedings of the 50th Annual Design Automation Conference* [15], pages 136:1–136:6.

[15] Armin Alaghi, Cheng Li, and John P. Hayes. Stochastic circuits for real-time image-processing applications. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 136:1–136:6, New York, NY, USA, 2013. ACM.

[16] M. Alawad and M. Lin. Stochastic-based deep convolutional networks with reconfigurable logic fabric. *IEEE Transactions on Multi-Scale Computing Systems*, 2(4):242–256, Oct 2016.

[17] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 1–13, June 2016.

[18] Jennifer A.N. Brophy and Christopher A Voigt. Principles of genetic circuit design. *Nature methods*, 11:508–520, 04 2014.

[19] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross. Vlsi implementation of deep neural networks using integral stochastic computing. In *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pages 216–220, Sept 2016.

[20] Vaughn Betz and Jonathan Rose. Fpga routing architecture: Segmentation and buffering to optimize speed and density. In *FPGA*, 1999.

[21] Swapnil P. Bhatia, Michael J. Smanski, Christopher A. Voigt, and Douglas M. Densmore. Genetic design via combinatorial constraint specification. *ACS Synthetic Biology*, 6(11):2130–2135, 2017. PMID: 28874044.

[22] B. D. Brown and H. C. Card. Stochastic neural computation. i. computational elements. *IEEE Transactions on Computers*, 50(9):891–905, Sep 2001.

[23] B. D. Brown and H. C. Card. Stochastic neural computation. ii. soft competitive learning. *IEEE Transactions on Computers*, 50(9):906–920, Sep 2001.

[24] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rossello. A new stochastic computing methodology for efficient neural network implementation. *IEEE Transactions on Neural Networks and Learning Systems*, 27(3):551–564, March 2016.

[25] Y. N. Chang and K. K. Parhi. Architectures for digital filters using stochastic computing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2697–2701, May 2013.

[26] Y. N. Chang and K. K. Parhi. Architectures for digital filters using stochastic computing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2697–2701, May 2013.

[27] T. H. Chen and J. P. Hayes. Design of division circuits for stochastic computing. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 116–121, July 2016.

[28] T. H. Chen and J. P. Hayes. Design of division circuits for stochastic computing. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 116–121, July 2016.

[29] T. H. Chen and J. P. Hayes. Equivalence among stochastic logic circuits and its application to synthesis. *IEEE Transactions on Emerging Topics in Computing*, PP(99):1–1, 2017.

[30] Vinay K. Chippa, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. Storm: A stochastic recognition and mining processor. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, ISLPED '14, pages 39–44, New York, NY, USA, 2014. ACM.

[31] François Chollet et al. Keras. `https://github.com/keras-team/keras`, 2015.

[32] J. M. de Aguiar and S. P. Khatri. Exploring the viability of stochastic computing. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*, pages 391–394, Oct 2015.

[33] Q. T. Dong, M. Arzel, C. Jego, and W. J. Gross. Stochastic decoding of turbo codes. *IEEE Transactions on Signal Processing*, 58(12):6421–6425, Dec 2010.

[34] Matthijs Douze, Hervé Jégou, Harsimrat Sandhawalia, Laurent Amsaleg, and Cordelia Schmid. Evaluation of GIST Descriptors for Web-scale Image Search. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, CIVR '09, pages 19:1–19:8, New York, NY, USA, 2009. ACM.

[35] D. Fick, G. Kim, A. Wang, D. Blaauw, and D. Sylvester. Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2d edge detection and noise filtering. In *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, pages 1–4, Sept 2014.

[36] Siddharth Gaba, Phil Knag, Zhengya Zhang, and Wei Lu. Memristive devices for stochastic computing. *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2592–2595, 2014.

[37] B. R. Gaines. Stochastic computing. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 149–156, New York, NY, USA, 1967. ACM.

[38] B. R. Gaines. *Stochastic Computing Systems*, pages 37–172. Springer US, Boston, MA, 1969.

[39] V. C. Gaudet and A. C. Rapley. Iterative decoding using stochastic computation. *Electronics Letters*, 39(3):299–301, Feb 2003.

[40] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 431–437, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[41] James Hegarty, John Brunhaver, Zachary DeVito, Jonathan Ragan-Kelley, Noy Cohen, Steven Bell, Artem Vasilyev, Mark Horowitz, and Pat Hanrahan. Darkroom: Compiling high-level image processing code into hardware pipelines. *ACM Trans. Graph.*, 33(4):144:1–144:11, July 2014.

[42] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue. Compact and accurate stochastic circuits with shared random number sources. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pages 361–366, Oct 2014.

[43] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, January 2011.

[44] Herve Jegou, Florent Perronnin, Matthijs Douze, Jorge Sanchez, Patrick Perez, and Cordelia Schmid. Aggregating local image descriptors into compact codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1704–1716, September 2012.

[45] H. Jiang, C. Shen, P. Jonker, F. Lombardi, and J. Han. Adaptive filter design using stochastic circuits. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 122–127, July 2016.

[46] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[47] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori. Slack redistribution for graceful degradation under voltage overscaling. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 825–831, Jan 2010.

[48] K. Kim, J. Lee, and K. Choi. An energy-efficient random number generator for stochastic circuits. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 256–261, Jan 2016.

[49] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 124:1–124:6, New York, NY, USA, 2016. ACM.

[50] Young-Chul Kim and M. A. Shanblatt. Architecture and statistical model of a pulse-mode digital multilayer neural network. *IEEE Transactions on Neural Networks*, 6(5):1109–1118, Sep 1995.

[51] Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[52] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[53] V. T. Lee, A. Alaghi, and L. Ceze. Correlation manipulating circuits for stochastic computing. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2018*, March 2018.

[54] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 13–18, March 2017.

[55] Vincent T. Lee, Armin Alaghi, Luis Ceze, and Mark Oskin. Stochastic synthesis for stochastic computing. *CoRR*, abs/1810.04756, 2018.

[56] Vincent T. Lee, Samuel Archibald Elliot, Armin Alaghi, and Luis Ceze. Synthesizing number generators for stochastic computing using mixed integer programming. *CoRR*, abs/1902.05971, 2019.

[57] X. R. Lee, C. L. Chen, H. C. Chang, and C. Y. Lee. A 7.92 gb/s 437.2 mw stochastic ldpc decoder chip for ieee 802.15.3c applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(2):507–516, Feb 2015.

[58] Bingzhe Li, M. Hassan Najafi, and David J. Lilja. Using stochastic computing to reduce the hardware requirements for a restricted boltzmann machine classifier. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 36–41, New York, NY, USA, 2016. ACM.

[59] Ji Li, Zihao Yuan, Zhe Li, Caiwen Ding, Ao Ren, Qinru Qiu, Jeffrey T. Draper, and Yanzhi Wang. Hardware-driven nonlinear activation for stochastic computing based deep convolutional neural networks. *CoRR*, abs/1703.04135, 2017.

[60] P. Li and D. J. Lilja. Using stochastic computing to implement digital image processing algorithms. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*, pages 154–161, Oct 2011.

[61] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel. The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic. In *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 480–487, Nov 2012.

[62] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. 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, March 2014.

[63] Peng Li, W. Qian, M. D. Riedel, K. Bazargan, and D. J. Lilja. The synthesis of linear finite state machine-based stochastic computational elements. In *17th Asia and South Pacific Design Automation Conference*, pages 757–762, Jan 2012.

[64] M. Lichman. UCI machine learning repository, 2013.

[65] S. Liu and J. Han. Energy efficient stochastic computing with sobol sequences. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 650–653, March 2017.

[66] Y. Liu and K. K. Parhi. Architectures for recursive digital filters using stochastic computing. *IEEE Transactions on Signal Processing*, 64(14):3705–3718, July 2016.

[67] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[68] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. Intermittent computing: Challenges and opportunities. In *2nd Summit on Advances in Programming Languages, SNAPL 2017, May 7-10, 2017, Asilomar, CA, USA*, pages 8:1–8:14, 2017.

[69] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 950–961. VLDB Endowment, 2007.

[70] R. Manohar. Comparing stochastic and deterministic computing. *IEEE Computer Architecture Letters*, 14(2):119–122, July 2015.

[71] B. Moons and M. Verhelst. Energy and accuracy in multi-stage stochastic computing. In *2014 IEEE 12th International New Circuits and Systems Conference (NEWCAS)*, pages 197–200, June 2014.

[72] Thierry Moreau, Felipe Augusto, Patrick Howe, Armin Alaghi, and Luis Ceze. Exploiting quality-energy tradeoffs with arbitrary quantization: Special session paper. In *Proceedings of the Twelfth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion*, CODES '17, pages 30:1–30:2, New York, NY, USA, 2017. ACM.

[73] Thierry Moreau, Felipe Augusto, Patrick Howe, Armin Alaghi, and Luis Ceze. Qappa: A framework for navigating quality-energy tradeoffs with arbitrary quantization. Technical report, Seattle, WA, USA, 2017.

[74] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[75] M. H. Najafi and D. J. Lilja. High-speed stochastic circuits using synchronous analog pulses. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 481–487, 2017.

[76] F. Neugebauer, I. Polian, and J. P. Hayes. Building a better random number generator for stochastic computing. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 1–8, Aug 2017.

[77] F. Neugebauer, I. Polian, and J. P. Hayes. Framework for quantifying and managing accuracy in stochastic circuit design. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1–6, March 2017.

[78] A. E. Nielsen. *Neural Networks and Deep Learning*. Determination Press.

[79] Alec A. K. Nielsen, Bryan S. Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A. Strychalski, David Ross, Douglas Densmore, and Christopher A. Voigt. Genetic circuit design automation. *Science*, 352(6281), 2016.

[80] N. Onizawa, V. C. Gaudet, T. Hanyu, and W. J. Gross. Asynchronous stochastic decoding of low-density parity-check codes. In *2012 IEEE 42nd International Symposium on Multiple-Valued Logic*, pages 92–97, May 2012.

[81] M. Parhi, M. D. Riedel, and K. K. Parhi. Effect of bit-level correlation in stochastic computing. In *2015 IEEE International Conference on Digital Signal Processing (DSP)*, pages 463–467, July 2015.

[82] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543, 2014.

[83] I. Perez-Andrade, X. Zuo, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo. Analysis of voltage- and clock-scaling-induced timing errors in stochastic ldpc decoders. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 4293–4298, April 2013.

[84] Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodik, and Dinakar Dhurjati. Scaling up superoptimization. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 297–310, New York, NY, USA, 2016. ACM.

[85] W. J. Poppelbaum, C. Afuso, and J. W. Esch. Stochastic computing elements and systems. In *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*, AFIPS '67 (Fall), pages 635–644, New York, NY, USA, 1967. ACM.

[86] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 41–50. Eurographics Association, 2003.

[87] Weikang Qian and M. D. Riedel. The synthesis of robust polynomial arithmetic with stochastic logic. In *2008 45th ACM/IEEE Design Automation Conference*, pages 648–653, June 2008.

[88] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, pages 405–418, New York, NY, USA, 2017. ACM.

[89] N. Saraf, K. Bazargan, D. J. Lilja, and M. D. Riedel. Stochastic functions using sequential logic. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 507–510, Oct 2013.

[90] N. Saraf, K. Bazargan, D. J. Lilja, and M. D. Riedel. Iir filters using stochastic arithmetic. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.

[91] G. Sarkis and W. J. Gross. Efficient stochastic decoding of non-binary ldpc codes with degree-two variable nodes. *IEEE Communications Letters*, 16(3):389–391, March 2012.

[92] G. Sarkis, S. Mannor, and W. J. Gross. Stochastic decoding of ldpc codes over gf(q). In *2009 IEEE International Conference on Communications*, pages 1–5, June 2009.

[93] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 305–316, New York, NY, USA, 2013. ACM.

[94] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic optimization of floating-point programs with tunable precision. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 53–64, New York, NY, USA, 2014. ACM.

[95] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 807–814, New York, NY, USA, 2007. ACM.

[96] Chanop Silpa-Anan and Richard I. Hartley. Optimised kd-trees for fast image descriptor matching. In *CVPR*. IEEE Computer Society, 2008.

[97] H. Sim, D. Nguyen, J. Lee, and K. Choi. Scalable stochastic-computing accelerator for convolutional neural networks. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 696–701, Jan 2017.

[98] Hyeonuk Sim and Jongeun Lee. A new stochastic computing multiplier with application to deep convolutional neural networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, pages 29:1–29:6, New York, NY, USA, 2017. ACM.

[99] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1470–, Washington, DC, USA, 2003. IEEE Computer Society.

[100] M. Soeken, T. Haener, and M. Roetteler. Programming quantum computers using design automation. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 137–146, March 2018.

[101] Robert Carlson Stephen C. Aldrich, James Newcomb. Genome synthesis and design future: Implications for the us economy. 2007.

[102] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, RWDSL2018, pages 7:1–7:10, New York, NY, USA, 2018. ACM.

[103] S. S. Tehrani, A. Naderi, G. A. Kamendje, S. Mannor, and W. J. Gross. Tracking forecast memories in stochastic decoders. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing* [104], pages 561–564.

[104] S. S. Tehrani, A. Naderi, G. A. Kamendje, S. Mannor, and W. J. Gross. Tracking forecast memories in stochastic decoders. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 561–564, April 2009.

[105] S. Sharifi Tehrani, W. J. Gross, and S. Mannor. Stochastic decoding of ldpc codes. *IEEE Communications Letters*, 10(10):716–718, Oct 2006.

[106] S. Sharifi Tehrani, S. Mannor, and W. J. Gross. Fully parallel stochastic ldpc decoders. *IEEE Transactions on Signal Processing*, 56(11):5692–5703, Nov 2008.

[107] S. Sharifi Tehrani, A. Naderi, G. Kamendje, S. Hemati, S. Mannor, and W. J. Gross. Majority-based tracking forecast memories for stochastic ldpc decoding. *IEEE Transactions on Signal Processing*, 58(9):4883–4896, Sep. 2010.

[108] P. S. Ting and J. P. Hayes. Stochastic logic realization of matrix operations. In *2014 17th Euromicro Conference on Digital System Design* [109], pages 356–364.

[109] P. S. Ting and J. P. Hayes. Stochastic logic realization of matrix operations. In *2014 17th Euromicro Conference on Digital System Design*, pages 356–364, Aug 2014.

[110] P. S. Ting and J. P. Hayes. Isolation-based decorrelation of stochastic circuits. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 88–95, Oct 2016.

[111] Paishun Ting and John P. Hayes. Eliminating a hidden error source in stochastic circuits. In *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, July 2017.

[112] Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, November 2008.

[113] R. Venkatesan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan. Spintastic: Spin-based stochastic logic for energy-efficient computing. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1575–1578, March 2015.

[114] N. Verma and A. P. Chandrakasan. An ultra low energy 12-bit rate-resolution scalable sar adc for wireless sensor nodes. *IEEE Journal of Solid-State Circuits*, 42(6):1196–1205, June 2007.

[115] Christopher A. Voigt. Genetic parts to program basteria. *Current Opinion in Biotechnology*, 17:548–557, 2006.

[116] Yandan Wang, Wei Wen, Hai Li, and Miao Hu. A novel true random number generator design leveraging emerging memristor technology. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pages 271–276, New York, NY, USA, 2015. ACM.

[117] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, December 2007.

[118] M. Yang, J. P. Hayes, D. Fan, and W. Qian. Design of accurate stochastic number generators with noisy emerging devices for stochastic computing. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 638–644, Nov 2017.

[119] A. Zhakatayev, K. Kim, K. Choi, and J. Lee. An efficient and accurate stochastic number generator using even-distribution coding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, PP(99):1–1, 2018.

[120] David K Y Zhang and Georg Seelig. Dynamic dna nanotechnology using strand-displacement reactions. *Nature chemistry*, 3 2:103–13, 2011.

Appendix A

FULL TAXONOMY OF STOCHASTIC COMPUTING CIRCUITS

Functional Unit	Function $f(p_X, p_Y)$	Design	Correlation Affinity	Quantization Direction	Progressive Precision	Correlation Manipulating	Exact	Pseudostochastic	Discrepancy Sensitive
Scaled Adder	$\frac{1}{2}(p_X + p_Y)$	[38]	0	MSB	✓	✗	✗	✗	✓
Scaled Adder	$\frac{1}{2}(p_X + p_Y)$	[54]	$*$	MSB	✓	✗	✓	✗	✗
Saturating Adder	$\texttt{min}(1, p_X + p_Y)$	[6]	$-$	LSB	✓	✗	✓	✗	✗
Subtractor	$\lvert p_X - p_Y \rvert$	[6]	+	LSB	✓	✗	✓	✗	✗
Multiplier	$p_X p_Y$	[38]	0	MSB	✓	✗	✗	✗	✓
Divider	p_X / p_Y	[27]	+	MSB	✓	✗	✗	✗	✗
Maximum	$\texttt{max}(p_X, p_Y)$	[6]	+	None	✓	✗	✓	✗	✓
Maximum	$\texttt{max}(p_X, p_Y)$	[88]	$*$	None	✗	✗	✓	✓	✗
Minimum	$\texttt{min}(p_X, p_Y)$	[6]	+	None	✓	✗	✓	✗	✗
Synchronizer	(p_X, p_Y)	[53]	$*$	None	✓	✓	N/A	✗	✓
Desynchronizer	(p_X, p_Y)	[53]	$*$	None	✓	✓	N/A	✗	✓
Decorrelator	(p_X, p_Y)	[53]	$*$	None	✓	✓	N/A	✗	✓
Tracking Forecast Memory	p_X	[104]	$*$	None	✓	✓	N/A	✗	✓
Rectified Linear Unit	$\texttt{max}(0, p_X)$	[59]	$*$	None	✗	✗	✓	✓	✗

Hyperbolic Tangent	$\texttt{tanh}(p_X)$	[22]	0†	MSB	✗	✗	✗	✓	✓
Accumulative Parallel Counter	$sum(p_X, ...)$	[108]	$*$	None	✗	✗	✓	✓	✗
Square	p_X^2	[38]	0†	MSB	✓	✗	✗	✗	✓
Square Root	$\sqrt{p_X}$	[38]	+†	MSB	✗	✗	✗	✓	✓
Square Root	$\sqrt{p_X}$		0†	MSB	✓	✗	✗	✗	✓
Exponentiation	e^{-2*G*p_X}	[22]	0†	MSB	✗	✗	✗	✓	✓
Exponentiation	$p_X^{p_Y}$		0	MSB	✓	✗	✗	✗	✓
S/D Converter	p_X	[38]	$*$	MSB	✗	✗	✓	✓	✗
D/S Converter	$\texttt{sum}(p_X)$	[38]	N/A	None	✗	✗	✓	✓	N/A

Table A.1: Stochastic circuit taxonomy.