Designing Accurate and Low-Cost Stochastic Circuits

by

Te-Hsuan Chen

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2016

Doctoral Committee:

Professor John P. Hayes, Chair
Professor Scott Mahlke
Professor Karem A. Sakallah
Professor Zhengya Zhang

To my family and friends

ACKNOWLEDGEMENTS

I am deeply grateful to all the numerous, wise and caring people who have supported me through my Ph.D. and earlier education. I wish to thank my advisor Professor John P. Hayes for his invaluable advice and encouragement in all aspect of research, including critical thinking, identifying and solving research problems, technical writing, and oral presentation. It has been an honor to be his student. Financial support from the National Science Foundation is also gratefully acknowledged.

I would like to thank the members of my Ph.D. committee, Professors Scott Mahlke, Karem A. Sakallah, and Zhengya Zhang for their helpful feedback. I am also grateful to all the professors I have worked with as a student, including Professors David Blaauw, Edmund H. Durfee, Mingyan Liu, and Valeria Bertacco. Their courses provided a solid foundation to my research. I also wish to thank Christine Feak and Judy Dyer from the English Language Institute at the University of Michigan. I have learned a lot about academic writing from them. I am thankful to Professor Cheng-Wen Wu, my former advisor at National Tsing Hua University, who has been a mentor for research and life.

I am grateful to the members of our research group with whom I have had many wonderful discussions, including Dr. Chien-Chih Yu, Dr. Dae Young Lee, I-Che Chen, Pai-Shun Ting, and William Sullivan. A special thanks to Dr. Armin Alaghi who helped me a lot with the topic of stochastic computing and shared much of his research experience with me. Thank you to all of the valuable friends I met in Ann Arbor for being there for me through the tough journey to my Ph.D. and for lots of great moments we have shared.

Finally, I wish to thank my family who have given me unconditional support for all the decisions I have made. Thanks especially to my parents and my sister for encouraging me to pursue whatever I wanted. Your emotional support has been the strength helping me get through tough times.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADDIE	Adaptive Digital Element
BF	Boolean Function
ITM	Ideal Transfer Matrix
LDPC	Low-Density Parity-Check
LFSR	Linear Feedback Shift Register
MPP	Monotonic Progressive Precision
MSE	Mean Square Error
PP	Progressive Precision
SC	Stochastic Computing
SCC	Stochastic Computing Correlation
SE	Stochastically Equivalent
SEC	Stochastic Equivalence Class
SN	Stochastic Number
SNG	Stochastic Number Generator
PTM	Probabilistic Transfer Matrix
ReSC	Reconfigurable Stochastic Computing Architecture
RV	Random Variable
TT	Truth Table
WBG	Weighted Binary Generator
WT	Weight Table

LIST OF SYMBOLS

Symbol	Meaning
·	The dot sign or juxtaposition denotes multiply and OR in arithmetic and logic expressions, respectively.
+	The plus sign denotes add and AND in arithmetic and logic expressions, respectively.
$\equiv_K$	Stochastic equivalence with respect to a set of constants K. It is simplified to $\equiv$ when $K = (0.5, 0.5, \ldots, 0.5)$.
$\varepsilon_Z^{(i)}$	The stage-i bit-error of an SN Z. It is defined as $2^i \times \left\lvert \hat{p}_Z^{(i)} - p_Z^{(i)} \right\rvert$.
$\mathbb{F}$	An SEC, i.e., a set of stochastically equivalent functions.
K	A set of constant numbers in the unit interval [0,1]. Each number in K corresponds to the numerical value of a stochastic constant in X_C.
n	The size of X, so $n = s + t$.
N	The length of an SN.
p_Z	The probability of seeing a 1 on a wire z. It is also the numerical value of a unipolar SN Z. If Z is bipolar, its numerical value is $2p_Z - 1$.
$\hat{p}_Z$	The measured value when N bits of a unipolar SN Z are collected.
$p_Z^{(i)}$	The numerical value of a unipolar SN Z to i-bit precision.
$\hat{p}_Z^{(i)}$	The measured value of a unipolar SN Z when 2^i bits are sampled.
s	The size of X_V.
t	The size of X_C.
$w(f)$	The weight of a Boolean function $f(x_1, x_2, \ldots, x_n)$ is the number of its minterms i.e., $w(f) = f(0, \ldots, 0,0) + \ \ldots\ + f(1, \ldots, 1,0) + f(1, \ldots, 1,1)$.
X	The set of inputs of a stochastic circuit.
X_V	The subset of the set of inputs X of a stochastic circuit with variable signal probabilities.

X_C	The subset of the set of inputs X of a stochastic circuit with constant signal probabilities.
z	A lowercase letter denotes a wire in a circuit or a literal in a logic expression.
Z	An uppercase letter denotes a bit-stream, i.e. a stochastic number (SN), or the numerical value of the SN Z in an arithmetic expression.

ABSTRACT

Designing Accurate and Low-Cost Stochastic Circuits

by

Te-Hsuan Chen

Chair: John P. Hayes

Stochastic computing (SC) is an unconventional computing approach that processes data represented by pseudo-random bit-streams called stochastic numbers (SNs). It enables arithmetic functions to be implemented by tiny, low-power logic circuits, and is highly error-tolerant. These properties make SC practical for applications that need massive parallelism or operate in noisy environments where conventional binary designs are too costly or too unreliable. SC has recently come to be seen as an attractive choice for tasks such as biomedical image processing and decoding complex error-correcting codes. Despite its desirable properties, SC has features that limit its usefulness, including insufficient accuracy and an inadequate design theory. Accuracy is especially vulnerable to correlation among interacting SNs and to the random fluctuations inherent in SC's data representation. This dissertation examines the major factors affecting accuracy using analytical and experimental approaches based on probability theory and circuit simulation, respectively. We devise methods to quantify the error effects in stochastic circuits by means of probabilistic transfer matrices and Bernouilli processes. These methods make it possible to compare the impact of errors on conventional and stochastic circuits under various conditions. We then analyze correlation in detail and show that correlation-induced

errors can be reduced by the careful insertion of delay elements, a de-correlation technique called isolation. Noting that different logic functions can have the same stochastic behavior when constant SNs are applied to their inputs, we show how to partition logic functions into stochastic equivalence classes (SECs). We derive a procedure for identifying SECs, and apply SEC concepts to the synthesis and optimization of stochastic circuits. While addition, subtraction and multiplication have well-known and simple SC implementations, this is not true for division. We study stochastic division methods and propose a new type of stochastic divider that combines low cost with high accuracy. Finally, we turn to the design of general stochastic circuits and investigate a desirable property of SNs called monotonic progressive precision (MPP) whereby accuracy increases steadily with bit-stream length. We develop an SC design technique which produces results that are accurate and have good MPP. The dissertation concludes with some ideas for future research.

CHAPTER 1
Introduction

Stochastic computing (SC) is a computing paradigm that provides an alternative to computing with conventional binary numbers. Its distinguishing feature is that numbers are represented by random bit-streams that can be interpreted as probabilities. This unusual number representation scheme enables SC to perform low-cost, low-power and error-tolerant computing. SC has recently attracted the attention of researchers interested in applications such as decoding modern error-correcting codes, biomedical image processing, and neuromorphic networks. This chapter introduces SC and its applications. It also discusses the challenges that motivate our research.

1.1 Motivation

The steadily increasing density of integrated circuits (ICs) allows billions of components and complex applications to be packed into small, portable devices. At the same time, it introduces some new design challenges to deal with power and energy consumption. Consequently, a great deal of attention is being paid to low-power design, especially for battery-powered applications [27][54].

A growing applications area with very strict power budgets is the so-called Internet of Things (IoT). IoT refers to large networks of electronic devices with sensors, processors and their associated software embedded in physical objects like buildings and vehicles. Unlike conventional computing platforms that are directly accessible by users, such as desktop computers or smartphones, IoT devices are often deployed in physical environments that are hard to reach, a problem that heightens the importance of low power

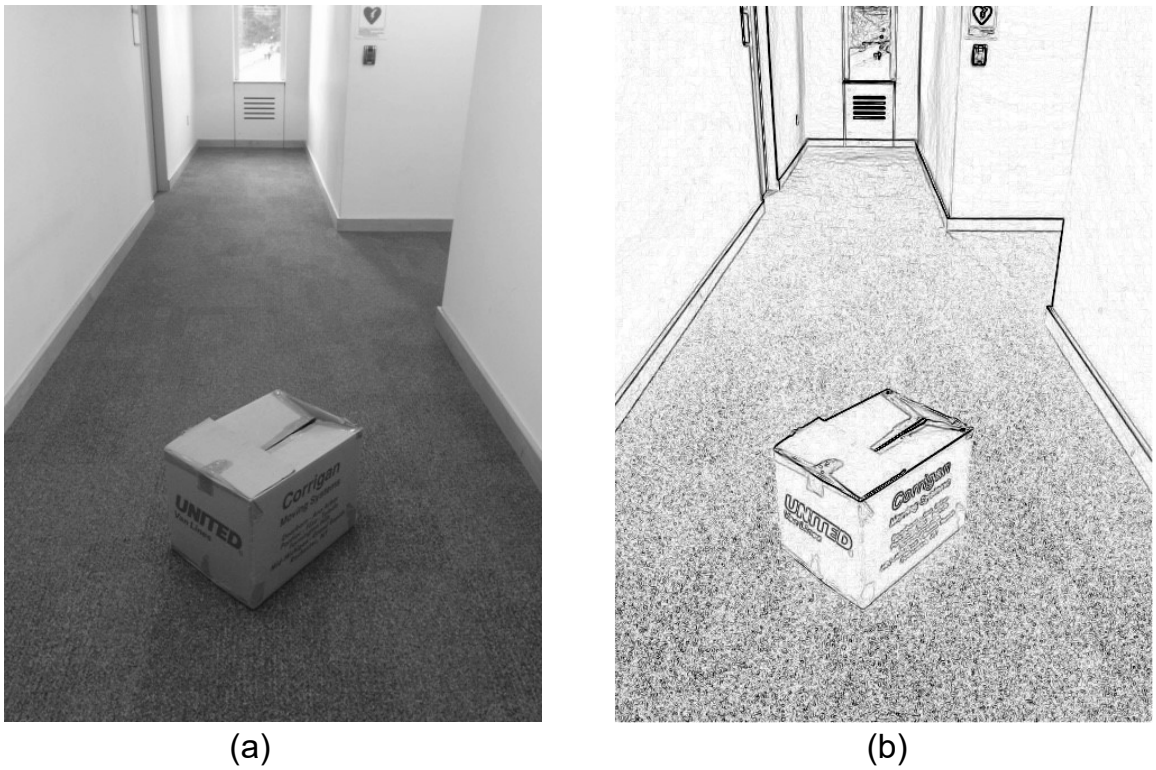

Figure 1.1: An example of edge detection to convert (a) a greyscale image of a corridor with an obstacle into (b) a high-contrast image.

consumption [11] [25]. To illustrate, long lifetime water quality sensors in reservoirs, forest fire detectors, and chemical leakage detectors in rivers all require low-power designs. To be able to deploy them in large and noisy physical environments, these devices must be robust, and the cost per device must be very low [25].

Another class of extremely power-sensitive applications are electronic biomedical implants. Examples here include retinal implants for the visually impaired, heart-rhythm monitors, and nerve implants to help paralyzed people perform basic tasks. For instance, retinal implants acquire and process images to extract information that can be used by the brain. One such image-processing task is edge detection, which generates a high-contrast black-and-white image highlighting the boundaries or edges of objects. For visually impaired people who can only distinguish very bright from very dark features, this

processed image can make obstacle detection, avoidance, and the like possible; see Figure 1.1.

The ability of massive parallelism to enhance performance at low cost is also important in biomedical devices like retinal implants for several reasons. A typical retinal implant chip contains a large array of sensors, one per pixel, and each sensor requires some processing capability, ideally a dedicated pixel processor [65][66]. As there may be thousands of pixels, the pixel processors must be tiny in size and have extremely low power needs. They should also be fast enough to process data or images in real-time. Figure 1.2 illustrates how edge detection can be performed by an array of pixel processors. For a certain standard edge-detection method (the Roberts cross algorithm [37]), a moving average of light intensity $z_{i,j}$ is computed on a window of size 2×2 surrounding each pixel $x_{i,j}$ according to the relatively complex arithmetic formula

$$z_{i,j} = |x_{i,j} - x_{i+1,j+1}| + |x_{i,j+1} - x_{i+1,j}| \tag{1.1}$$

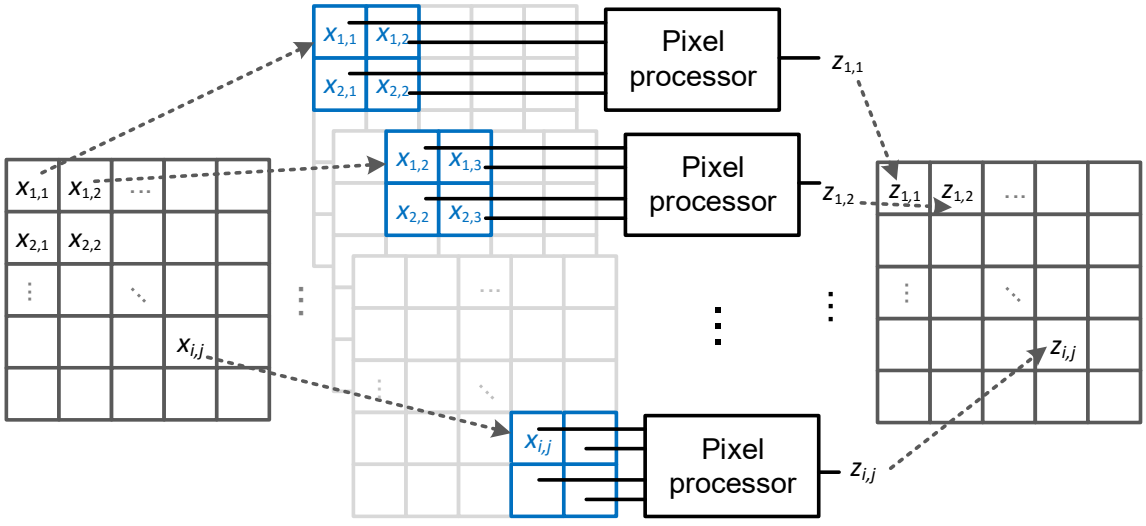

Figure 1.2: Edge detection using an array of pixel processors and a four-pixel window.

A small processor implementing Equation (1.1) and using conventional hardware designs is unlikely to meet the size and power constraints imposed by a retinal implant. However, as we will see later, a design based on stochastic computing can easily do so [3].

Since implantation of biomedical devices requires invasive surgery, they must have long lifetime and very small size [43]. The size requirement also renders high-capacity batteries to ensure long-term operation impractical. Moreover, because human tissues like the retina are very sensitive to temperature, if a biomedical implant dissipates too much power, vital organs can suffer heat damage. To avoid such problems, extremely low-power and small-size circuits are critical. In addition, implanted devices must be robust and insensitive to environmental noise [19].

Many approaches have been proposed to achieve some or all of the foregoing design goals, but each has its own limitations. For example, low-power semiconductor techniques, such as transistors with multiple threshold voltages and oxide thicknesses, reduce power consumption but they require special manufacturing processes that increase IC area and delay [84]. One promising technology that addresses all these issues is stochastic computing (SC), which is the subject of this dissertation. In SC, numbers are represented by random bit-streams which are interpreted as probabilities. For instance, the number 0.25 is represented by the bit-stream $X = 0010100001000010$ which contains four 1s and has length $N = 16$. If bits are randomly sampled in this bit-stream, then the probability of seeing a 1 in any position is 0.25. This probability is easily estimated by counting the number of 1s in X and dividing it by the bit-stream length N. As we will see, within the framework of SC, this unusual number representation scheme has the potential to produce small, low-power, low-cost, and error-tolerant circuits. Furthermore, these circuits can be built using standard digital logic manufacturing methods.

1.2 Stochastic Computing

As noted above, SC operates on random bit-streams using conventional logic circuits; we will refer to such circuits as *stochastic circuits.* The data value associated with

a logic signal (wire) x in a stochastic circuit is the probability p_X of seeing a 1 on x, i.e., x's signal probability. If a stream X of random bits, called a *stochastic number* (SN), is applied to x in some N-bit time-frame (window) W, then X's *numerical value* p_X is defined to be the frequency of 1s in W. Hence, when X contains N_1 1s, p_X is approximated by N_1/N, which we denote by $\hat{p}_X$. In general, $\hat{p}_X \approx p_X$, and the precision of this approximation tends to increase with N. For instance, when N is 4, 8, and 16, the bit-streams 0101, 01011010 and 1010011001010011 are some of the many possible representations of $p_X = 1/2$. Note that $\hat{p}_X$ represents the measured value when N bits are collected, while p_X is the theoretical or "exact" value of interest. To approximate a bit-stream's value we can also sample some part of it. For example, if we reduce N from 16 to 8 bits, we change X from 1010011001010011 to 10100110, but p_X is unchanged. However, reducing Y = 1001001101001010 from 16 to 8 bits introduces an error of $|7/16 - 4/8| = 1/16$.

SC can also process arbitrary (real) numbers if they are suitably approximated and scaled to lie in the unit interval [0, 1]. To handle signed numbers, it is common to interpret the numerical value of a bit-stream X as $2p_X - 1$, in which case the SN format is called

Table 1.1: Numerical values of an N-bit bit-stream X in the unipolar and bipolar formats.

Bit-stream X	**No. of 1s N_1**	**Numerical value**		**No. of bit-streams with the same numerical value**
		Unipolar p_X	**Bipolar $2p_X - 1$**	
0 0 0 … 0 0 0	0	0	−1	1
0 0 0 … 0 0 1 0 0 0 … 0 1 0 ⋮ 1 0 0 … 0 0 0	1	$1/N$	$2/N - 1$	N
⋮	⋮	⋮	⋮	⋮
0 0 0 … 1 1 1 ⋮ 1 1 1 … 0 0 0	$N/2$	0.5	0	$\frac{N(N-1)\cdots(N/2+1)}{N/2\,(N/2-1)\cdots 1}$
⋮	⋮	⋮	⋮	⋮
1 1 1 … 1 1 1	N	1	1	1

bipolar, and the effective number range becomes [−1, 1]. The basic format in which X's numerical value is taken to be p_X is called *unipolar*. Table 1.1 summarizes the numerical values of unipolar and bipolar SNs for different bit-streams of length N. For instance, the bit-stream 000…001, in which there is only one 1, has the value $1/N$ in unipolar format. On the other hand, the same bit-stream is interpreted as $2/N - 1$ if it is bipolar. Table 1.1 also shows that SC has a highly redundant encoding format that allows multiple bit-streams to represent the same p_X value. To illustrate, in addition to 000…001, the bit-streams 000…010, 000…100, …, and 100…000 also have the same numerical values, $1/N$ and $2/N - 1$, in the unipolar and bipolar formats, respectively.

Simple logic operations applied to bit-streams can perform useful arithmetic operations on their probability values. Figure 1.3*a–b* shows stochastic circuits for unipolar multiplication and addition. The two-input AND gate performs the multiplication $p_{X_1} \times p_{X_2}$ on two N-bit bit-streams X_1 and X_2 in N clock cycles because the output z is 1 if and

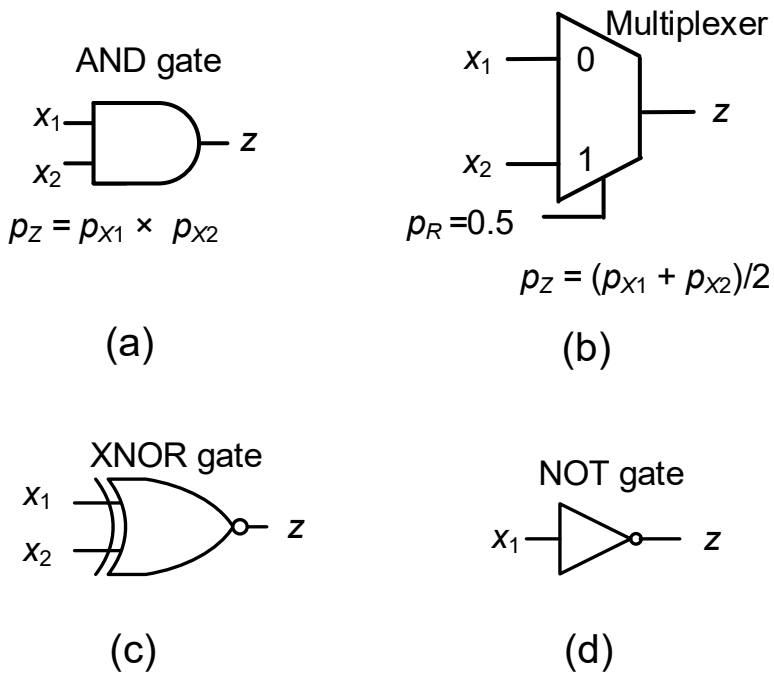

Figure 1.3: A selection of components for stochastic computing: (a) unipolar multiplier, (b) unipolar and bipolar scaled adder, (c) bipolar multiplier, and (d) bipolar negater.

only if both x_1 and x_2 are 1 in the same cycle. Note that for high accuracy, the input bit-streams X_1 and X_2 must be statistically independent (uncorrelated) and sufficiently long to provide acceptable precision, issues we will discuss in detail later.

Addition is implemented by a two-way multiplexer (MUX) in the scaled form $0.5\left(p_{X_1} + p_{X_2}\right)$ which ensures that the sum always lies in the probability interval [0, 1]. The idea behind the MUX-based adder is that the output z randomly receives half its bits from X_1 and half from X_2, so the number of 1s at z is the average number of 1s in X_1 and X_2. The selection of the input bits to be transferred to the output is controlled by applying to the MUX's select input a stochastic number R of constant value $p_R = 0.5$, i.e., a purely random sequence of 0s and 1s. R can also be seen as the source of the scaling factor in the sum.

In the bipolar format, an XNOR gate performs multiplication (Figure 1.3*c*), while a MUX continues to act as a scaled adder. The output z of the MUX is then the scaled sum $2p_Z - 1 = 0.5\left(\left(2p_{X_1} - 1\right) + \left(2p_{X_2} - 1\right)\right)$. Bitwise inversion of an SN X via a NOT gate (Figure 1.3*d*) negates X's numerical value in the bipolar format thus: $(2p_F - 1) = -\left(2p_{X_1} - 1\right)$. Subtraction is easily implemented by combining a multiplexer and a NOT gate. In Figure 1.3 (and in the rest of this dissertation), circuit wires and the signals (Boolean variables) they carry are denoted by small letters x, y, z,…; bit-streams or SNs are denoted by capital letters X, Y, Z,…, and their numerical values are denoted by p_X, p_Y, p_Z, etc.

Number conversion circuits are necessary at any interface between standard binary circuits and stochastic circuits. Figure 1.4*a* shows a typical binary-to-stochastic converter that generates a 2^k-bit SN X with the value $p_X = B/2^k$ from a k-bit binary integer B. This converter is called a *stochastic number generator* (SNG). An SNG contains a comparator that compares B with a random number and generates a 1 when B is larger than the random number. SC typically requires many randomness sources that can produce independent bit-streams with prescribed probability values. A simple binary counter (Figure 1.4*b*) converts

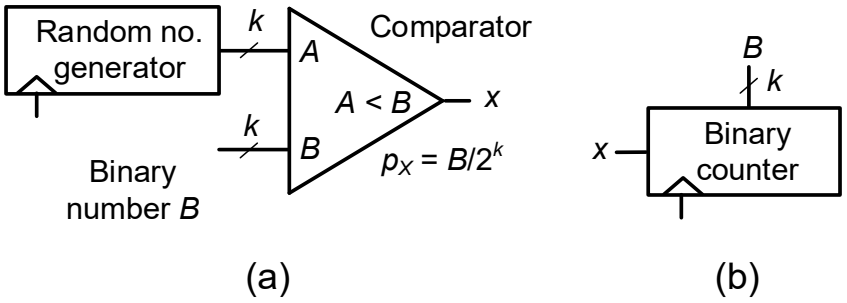

Figure 1.4: Number converters (a) binary-to-stochastic and (b) stochastic-to-binary.

an SN to a conventional binary number as the numerical value of the SN depends solely on the number of its 1s. The bit-stream length N is selected based on accuracy and precision considerations. If N is chosen to be 2^k for some k, as is often the case, then X is considered to have k-bit precision, i.e. the same precision as a k-bit binary number B. For instance, bit-streams 0101, 01011010 and 1010011001010011 are three of the many possible SNs representing $X = 0.5$, with 2-, 3- and 4-bit precision, respectively. This notion of precision means that 2^k bits suffice for X to represent B exactly.

To achieve k-bit precision, bit-streams of length 2^k or more are required, so high precision SC is difficult to achieve and requires long computational times. This fact, as well as the rapidly decreasing cost and increasing speed of conventional binary circuits, caused a loss of interest in SC shortly after it was first proposed back to 1960s [30][69][76]. A few attempts of building stochastic computers were made around that time, and they revealed the aforementioned drawbacks of SC.

However, SC has recently made a come-back as an attractive choice for applications requiring very small, low-power and low-cost hardware [2]. Several studies also have shown the success of applying SC to tasks such as image processing [3][57]. Because of the simplicity of SC computing units, massive parallelism becomes feasible. For example, we can simply use a large number l of AND gates to speed up multiplication l times and still have a very small multiplier. Massive parallelism of this kind makes SC extremely suitable for applications performing the same computation iteratively with different groups of inputs such as the retinal implants discussed earlier. Using SC, the

implant's many pixel processors can be made so small that even when they are replicated thousands of times, they have relatively low power consumption.

Another application that can benefit from SC's massive parallelism is the design of bio-inspired neuromorphic networks [15][16][46][77]. These are special-purpose computing systems that can potentially contain enormous numbers of small, highly interconnected processors called neurons. Messages are passed between the neurons, and a neuron is activated when a combination of its inputs exceeds some threshold [78]. All the neurons have essentially the same structure, so neuromorphic networks also tend to repeat the same computation on many different neurons with different message sets. Since the neurons must be small and low-power, SC is a good candidate for implementing large-scale neuromorphic networks. Another fact that make SC suitable to neuromorphic networks is the similarity between biological neural signals and stochastic numbers. Biological neurons communicate by means of noisy voltage spikes which loosely resemble bit-streams [81]. Furthermore, the frequency or rate of the spikes in a spike train encodes information (although the precise encoding scheme is not understood). Hence, because of the way it encodes information and its ability to realize massive parallelism, SC has great potential for building circuits that connect natural and artificial neural networks.

The first of the recent wave of successful applications of SC was to decode low-density parity-check (LDPC) codes, a class of error-correcting codes that enable data to be transmitted over a noisy transmission channel at rates close to the theoretical maximum (the Shannon limit) [33]. Decoding LDPC codes requires many complex probability computations, which make it difficult to implement in practice. Equation (1.2) shows one such computation called node updating.

$$p_Z = \frac{p_X \times p_Y}{p_X \times p_Y + (1 - p_X) \times (1 - p_Y)} \tag{1.2}$$

Figure 1.5 shows a sequential stochastic circuit that computes the arithmetic function defined by Equation (1.2) [67]. This small component is called an update node and was

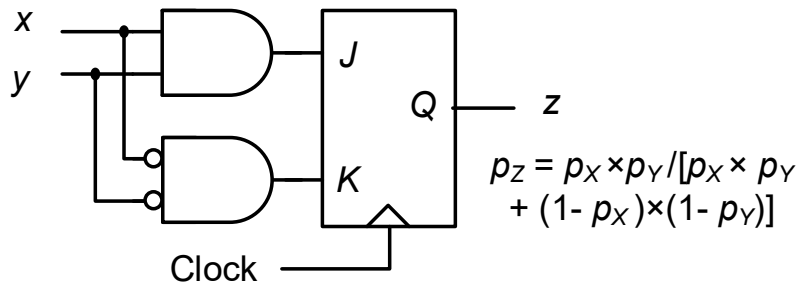

Figure 1.5: Update node for LDPC decoder.

designed for an LDPC decoder employing SC. An equivalent circuit using conventional (weighted-binary) arithmetic circuits contains hundreds of gates, as opposed to just a few gates in the stochastic implementation. Similarly, a stochastic edge detector realizing Robert cross formula of Equation (1.1) is much smaller than an equivalent circuit using conventional arithmetic circuits; it will be investigated in detail in Section 2.6.

SC can also tolerate soft errors caused by environmental noise. Because of its inherently redundant encoding scheme which we discussed earlier, soft errors of the bit-flip type have little effect on the value of an SN. For example, a single bit-flip occurring in an N-bit SN changes the output value by $1/N$, a relatively small error, whose significance diminishes as N increases. Furthermore, if the errors are bidirectional, e.g., if 0-to-1 and 1-to-0 bit-flips are equally likely, then the errors tend to cancel one another. This suggests that stochastic computing can outperform binary in certain applications [21][22].

The main advantages of SC can be summarized as follows. Mainly because of its pseudo-random number representation, SC enables small and low-power computing units using standard logic circuits. It is also extremely suitable for applications that need massive parallelism because the cost of the computing units is so low. Their inherently redundant number encoding format makes SNs less sensitive to environmental noise. On the other hand, SC has some disadvantages which we consider in the next section.

1.3 Research Challenges

Despite its recent successful applications, SC has several features that limit its usefulness. For instance, the need for many number conversion circuits offsets the advantage of SC's small computing units [28][67]. SC's long computational time and precision limitations make SC impractical to use for general-purpose computation [8]. Bit-stream length tends to grow exponentially as precision requirements increase. Inaccuracies caused by random bit fluctuations [31], and awkward scaling requirements [31] also make SC less desirable.

SC requires statistically independent or uncorrelated inputs to generate accurate results [23][31]. In other words, the bit-patterns of SNs that are being processed together should be unrelated. The effect of correlation on accuracy is illustrated by the example in Figure 1.6. The input bit-streams to the AND-gate stochastic multiplier all have the same numerical value $p_{X_i} = 0.5$. In Figure 1.6*a,* the two input streams X_1 and X_2 have very different bit-patterns reflecting a uniform, random-like distribution of their bits; in this case, X_1 and X_2 are intuitively uncorrelated. In Figure 1.6*b*, on the other hand, X_1 and X_2 have identical bit-patterns, implying they are highly correlated. If $x_1 = 0$ (1), then $x_2 = 0$ (1) and the input bit-patterns 01 and 10 never occur; hence, $p_{\bar{X}_1\bar{X}_2} = p_{X_1X_2} = 0.5$ and $p_{\bar{X}_1X_2} = p_{X_1\bar{X}_2} = 0$. The output probability p_Z is 0.25 for Figure 1.6*a* and 0.5 for Figure 1.6*b*. Thus, instead of the accurate result $p_Z = p_{X_1} \times p_{X_2} = 0.25$, Figure 1.6*b* produces a highly inaccurate value $p_Z = 0.5$. SNs that have become unduly correlated are commonly re-randomized or *de-correlated* by converting them back to binary numbers and regenerating

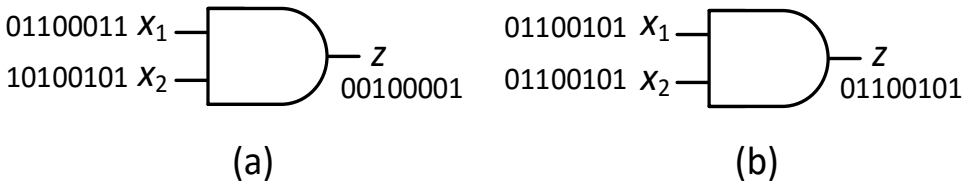

Figure 1.6: Stochastic multiplication with (a) uncorrelated inputs, and (b) highly correlated inputs.

independent versions of the SNs by means of expensive number converters like those in Figure 1.4. The impact of (de) correlation on accuracy has received little attention, and designing efficient stochastic circuits that achieve guaranteed accuracy levels in the presence of correlation is by no means easy.

Error tolerance is another aspect of SC which is not well understood; SC is known for its tolerance to soft errors, but a full and systematic analysis of this property has not been done. Yet another SC's research challenge is formalizing the design requirements of stochastic circuits. For example, the circuit in Figure 1.5 computes a complex and useful arithmetic function, but systematic ways to design sequential stochastic circuits like it are unknown. It is also unclear, for example, whether other stochastic circuits exist that can perform the same arithmetic function as that of Figure 1.5 with greater accuracy or at lower cost. Another problem of SC is that while addition, subtraction and multiplication have simple and well-known SC implementations, this is not true for an operation as basic as division. As a result, a stochastic divider is conspicuously absent from the basic SC component set in Figure 1.3, and division is usually avoided or approximated in SC design. These issues all suggest that finding a general theoretical framework for designing stochastic circuits is essential for achieving low-cost and accurate SC designs.

1.4 Dissertation Outline

When designing computation circuits, many requirements and constraints need to be considered: power consumption, area cost, computation speed or run-time, and the accuracy of the results. Because stochastic circuits are constructed from simple logic circuits, they are usually small and low-power. Computation time is related to the required precision, but very high precision results are not always necessary. Furthermore, stochastic circuits can operate at very high clock speed, which can mitigate the computing time problem. Achieving results of acceptable accuracy is probably the most important and least understood aspect of SC. Accuracy is therefore the main focus of this dissertation, which is divided to two major parts: (1) accuracy analysis and (2) accurate design. Chapters 2 and

3 cover the analysis of factors affecting SC's accuracy, while Chapters 4-6 discuss the design of accurate stochastic circuits.

Chapter 2 introduces an analysis framework to evaluate the impact of soft errors on stochastic circuits. This framework is based on a well-developed probabilistic transfer matrix (PTM) algebra [53] which is well suited to analyzing signal probabilities in SC. A PTM is a matrix in which element E_{ij} is a real number in the interval [0, 1] that denotes the conditional probability of producing output j in response to input i. As discussed earlier, correlation among interacting bit-streams is also a key factor affecting accuracy. We use PTM algebra to analyze the impact of correlation on SC accuracy in Chapter 3.

Next, we move to the design of accurate stochastic circuits. Chapter 4 presents a new way of classifying stochastic circuits, namely the stochastic equivalence class (SEC). Based on the SEC concept, we develop a general stochastic circuit synthesis method and area-cost optimization algorithms. These algorithms can reduce the area cost of stochastic circuits while keeping their accuracy unchanged. Chapter 5 reviews known stochastic division methods and presents a new and more efficient divider design. This design uses standard stochastic number representations and has better accuracy than previous designs. We tackle the issue of inaccuracy due to random fluctuations in Chapter 6. We present a general design method that generate exact results when bit-stream lengths are properly chosen. This method also addresses the reduction of SC's long computation times. Finally, Chapter 7 draws some conclusions and discusses some directions for future work.

CHAPTER 2
Accuracy and Soft Errors

The previous chapter pointed to accuracy as one of the major challenges facing SC. The factors affecting SC negatively including environmental noise, fluctuation of its sources of randomness, and correlation among interacting SNs. This chapter addresses the impact on accuracy of soft errors due to noise and random fluctuations. We first develop a mathematical framework for the analysis in terms Bernoulli random variables (RVs) and probabilistic transfer matrices (PTMs). Bernoulli RVs are widely employed in probability studies [80] and are well-suited to analyzing SN sources. PTMs are specifically intended for analyzing the probabilistic properties of logic circuits such as reliability or soft-error rate [53]. We present a case study in real-time image processing, which shows that stochastic circuits can outperform conventional ones under severe error conditions. The material presented here has been published in [22] and includes contributions by Armin Alaghi.

2.1 Soft Errors

Non-deterministic behavior is becoming common in digital circuits implemented using conventional CMOS transistors or novel nanotechnologies that can potentially replace CMOS [12]. Because of their small physical size, these circuits are easily affected by manufacturing defects and by transient errors due to environmental noise called soft errors, both of which tend to be probabilistic in nature. For example, carbon nanotube field-effect transistors (CNFETs), an emerging alternative to CMOS, exhibit behavioral variations that are difficult to identify and control [79]. Methods of designing circuits that

tolerate errors are also of increasing interest. Von Neumann took the first steps in designing reliable circuits using unreliable switches in the 1950s [82]. Since then, many error-tolerant design techniques have been proposed, ranging from error-correcting codes [59], to replication of hardware, software and/or data [49][71]. Most of these approaches impose high circuit overhead, and tend to be used only in the most cost-insensitive applications.

The ability of stochastic circuits to tolerate soft errors has long been recognized [31], but it has never been thoroughly analyzed. Under extremely noisy conditions, this property may even allow stochastic circuits to generate more accurate results than conventional circuits. In [74], a soft-error analysis of SC circuits is carried out which, however, is limited to bit-flips occurring in the circuit's input data; the stochastic circuits themselves are assumed to be fault- or error-free. In this chapter, we attempt to provide a more general error analysis for stochastic circuits, especially in the presence of high error rates such as are encountered in avionics or spacecraft instrumentation [32].

2.2 Probability Model

To obtain a better understanding of SC, especially with regard to its error behavior, probability theory is useful. A stochastic number X is a bit-stream carrying a probability value p_X where 1 denotes success and 0 denote failure. A stochastic number can therefore be viewed as a set of samples from a real-valued random variable (RV) with a Bernoulli distribution in which the probability of success is p_X [80]. Since probabilistic behavior can be easily modeled and analyzed in terms of Bernoulli RVs, we now use these RVs to give a formal definition of a stochastic number that abstracts away from bit-stream formatting issues: a stochastic number X is a Bernoulli random variable with parameter p_X.

When dealing with RVs, we usually need to sample them in order to estimate their values. This sampling process is, in fact, a very basic form of stochastic computing. For instance, assume that the AND-gate multiplier of Figure 1.3*a* has two input SNs X and Y with known values p_X and p_Y, but the output is an SN Z of unknown value p_Z. Stochastic computation with this circuit involves generating samples for X and Y and measuring the

success rate at z, and thus estimating p_Z. The expected rate of success at z can be calculated by the expected value operator denoted $\mathbb{E}[Z]$. Consequently,

$$p_Z = \mathbb{E}[Z] = \mathbb{E}[X \times Y] = \mathbb{E}[X] \times \mathbb{E}[Y] = p_X \times p_Y$$

assuming X and Y are independent RVs. For example, if $p_X = 0.2$ and $p_Y = 0.3$, then $p_Z = 0.06$, which is the expected rate of success at z. In practice, the success rate is affected by random fluctuations of the data, and usually has a different value $\hat{p}_Z$, which we refer to as the estimated value, in contrast with the exact value p_Z. The estimated value $\hat{p}_Z$ is obtained by sampling the circuit/RV N times and recording the number N_1 of 1s appearing at the output; this yields $\hat{p}_Z = N_1/N$. For example, if the RV Z, with the expected value $p_Z = 0.3$, is sampled 8 times, one possible outcome is 01100000, and the resulting estimate is $\hat{p}_Z = 2/8 = 0.25$.

In general, $\hat{p}_Z$ can be any of the 2^N different bit-streams derived from random sources, which allows p_Z and $\hat{p}_Z$ to differ, sometimes significantly, from one another. This difference between p_Z and $\hat{p}_Z$ is considered to be an error caused by randomness in the bit-stream representation of p_Z. Such random-fluctuation errors are usually measured by the mean square error (MSE) $E_Z = \mathbb{E}[(\hat{p}_Z - p_Z)^2]$. In the case of the Bernoulli RV's of interest here, we have the MSE of the RV Z [80]

$$E_Z = p_Z(1 - p_Z)/N \tag{2.1}$$

This equation implies that the MSE of an SN estimate can be reduced by increasing the number of samples i.e., the bit-stream length N. Also note that E_Z is a function of p_Z and N only, implying that no matter what the circuit is (whether the AND of Figure 1.3*a* or any other circuit), once the expected rate of success p_Z at the output is calculated, we can use Equation (2.1) to calculate its MSE.

Besides the random fluctuations inherent in the selection of a particular bit-stream to represent Z in a stochastic circuit C, various non-deterministic physical phenomena

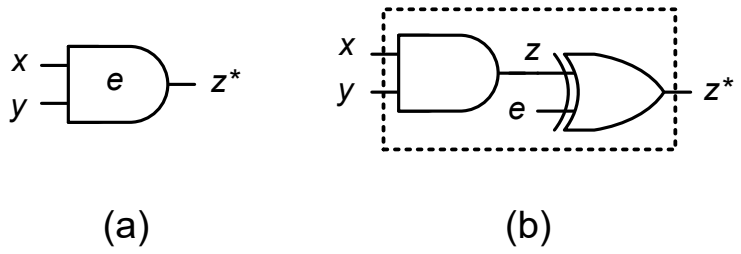

Figure 2.1: Circuit models for a stochastic multiplier with a bit-flip error *e* affecting its output: (a) internal or built-in error, and (b) externally injected error.

associated with *C* itself and its environment affect the sampling process and distort the expected values of *Z*. It is convenient to lump such effects into a bit-flip error *e* that occurs with some probability p_e. For example, it is often assumed in the literature [10] that *e* causes bit-flips in *Z*, which affect 0s and 1s with equal probability p_e. Whatever the error behavior assumed, two basic questions should be addressed: How do we model the impact of *e* on *z*, and how do we introduce *e* into a previously error-free stochastic circuit *C*? First, we assume the error *e* to be a Bernoulli RV with parameter p_e (the bit-flip rate), so it can be treated like another SN associated with *C*. Given this assumption, the circuit's fault-free output *z* then changes to an erroneous function z^*, as illustrated in Figure 2.1*a* for the AND-gate stochastic multiplier. For simulation purposes, it is convenient to have a mechanism for injecting the error in a way that flips the normal signal *z* with probability p_e, resulting in the erroneous output z^*. Figure 2.1*b* shows how to do this by inserting an XOR gate with input e into *C*'s output line. For example, a bit-flip rate of $p_e = 0.05$ with input values $p_X = 0.2$ and $p_Y = 0.3$ changes the expected success rate at the output of the AND multiplier from 0.06 to 0.104. An analytical method of calculating the expected value p_{Z^*} and its MSE will be developed later.

In addition to random fluctuations and soft errors, the accuracy of SC can also be affected by correlated SNs. For instance, the AND gate shown in Figure 1.3*a* will not be an accurate multiplier if its inputs *X* and *Y* are correlated. Since interacting signals are not independent, $\mathbb{E}[X \times Y] \neq \mathbb{E}[X] \times \mathbb{E}[Y]$ and the resulting SN *Z* has the property

$$p_Z = \mathbb{E}[Z] = \mathbb{E}[X \times Y] \neq \mathbb{E}[X] \times \mathbb{E}[Y] = p_X \times p_Y$$

The correlation between X and Y, determines the corresponding MSE. For example, if X and Y are the same bit-stream, they are highly positively correlated and the MSE will be $(p_X^2 - p_X)^2$. Such correlation errors have been investigated in [23] and will be discussed in detail later in Chapter 3.

2.3 Probabilistic Transfer Matrices (PTMs)

A convenient tool for analyzing the probabilistic behavior of logic circuits is the probabilistic transfer matrix (PTM) and its associated algebra [53]. PTMs were introduced at the University of Michigan to analyze the reliability of conventional logic circuits. Although their practical use may be limited by the fact that PTM size grows exponentially with circuit size, this is less of a problem with stochastic circuits, however, which typically consist of just a handful of gates.

In the PTM formulation, the behavior of an n-input m-output combinational circuit is represented by a $2^n \times 2^m$ zero-one matrix whose rows correspond to all input combinations and whose columns correspond to all output combinations. This matrix, which is referred to an *ideal transfer matrix* (ITM), is a slightly modified truth table. For instance, a two-input AND gate has the ITM

$$J_{\text{AND}} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.2}$$

where the rows correspond to xy = 00,01,10,11 and the columns correspond to z = 0,1. A general PTM is obtained from the ITM by allowing the entry in row r and column c to become any real number in the interval [0, 1] that denotes the conditional probability of producing output c in response to input r. In the AND gate's ITM J_{AND} shown in Equation (2.2), the top row tells us that in response to the input xy = 00, the AND produces output z = 0 with probability 1, and output z = 1 with probability 0.

Another basic arithmetic operation, addition, must be done approximately (by an OR gate, for example) or else the result must be scaled to ensure that it lies between 0.0 and 1.0, as required for probabilities. A common solution is the scaled addition $p_Z = 0.5\left(p_{X_1} + p_{X_2}\right)$ performed by the multiplexer of Figure 1.3*b*, whose ITM is

$$M_{\text{MUX}} = \begin{bmatrix} 1 & 0 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0 & 1 \end{bmatrix} \tag{2.3}$$

Observe that the multiplexer's select input carries a constant SN R with probability value 0.5 that affects the entries, but not the size, of M_{MUX}. Because R is an SN, it must be generated by a stochastic number generator (SNG) like that of Figure 1.3*c* and so has a significant impact on hardware cost.

By choosing suitable probability values, PTMs can be constructed to represent a remarkably wide range of error scenarios [53]. For example, the effect of a bit-flip error e with rate p_e on the output of the AND gate model in Figure 2.1*a*, is represented by the PTM

$$M_{\text{AND}} = \begin{bmatrix} 1-p_e & p_e \\ 1-p_e & p_e \\ 1-p_e & p_e \\ p_e & 1-p_e \end{bmatrix} \tag{2.4}$$

$$\begin{bmatrix} p_{e0} & 1-p_{e0} \\ p_{e1} & 1-p_{e1} \\ p_{e2} & 1-p_{e2} \\ 1-p_{e3} & p_{e3} \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) (b) (c) (d)

Figure 2.2: Representative PTMs: (a) NAND gate with four distinct input-dependent bit-flip error rates, (b) NAND gate with its first input stuck-at-1, (c) fanout wiring network with two output branches, and (d) swap or crossover gate that switches the order of two wires.

Input-dependent bit-flips can be modeled by associating a different p_e value with every row. Observe that a PTM must satisfy the stochastic requirement that all entries in each row add up to 1, and that the ITM is just the PTM for the error-free case.

Circuit PTMs can be manipulated by means of a well-defined algebra which loosely resembles linear algebra. Every element of a circuit C is representable by a PTM that describes C's logic function and error status; see Figure 2.2. PTMs can be combined in two basic ways corresponding to the two basic circuit interconnection structures, series and parallel. The PTM of two circuits C_1 and C_2 connected in series is the ordinary matrix product of their PTMs, i.e., $M_1 \times M_2$. The PTM of two circuits connected in parallel is the tensor product of the PTMs, denoted $M_1 \otimes M_2$. In the tensor product, each element of the first matrix M_1 is multiplied by the entire second matrix M_2, which leads to rapid growth in matrix size. A wire corresponds to a 2 × 2 PTM; its ITM case is simply the 2 × 2 identity matrix. A signal is represented by a 1 × 2 row vector $[p_0 \quad p_1]$, where p_0 and p_1 are the probabilities of the signal being 0 and 1, respectively. Signal vectors may be treated as a special kind of PTM, and can be manipulated with the same basic PTM operations.

The PTM of an SN X is processed like a signal vector because we treat it as an RV with a Bernoulli distribution and a parameter p_X denoting the expected probability that a bit of X is 1. For PTM analysis, X is written as the 2-element row vector $M_X = [1 - p_X \quad p_X]$. The joint probability distribution of two uncorrelated SNs X and Y is given by their tensor product $M_X \otimes M_Y$, which evaluates to the 4-element vector

$$\begin{aligned} M_{XY} &= M_X \otimes M_Y \\ &= [(1-p_X)(1-p_Y) \quad (1-p_X)p_Y \quad p_X(1-p_Y) \quad p_X p_Y] \end{aligned} \tag{2.5}$$

The entries of M_{XY} are the probabilities of the input combinations x_1x_2 = 00,01,10,11. For example, consider again the faulty AND-gate multiplier of Figure 2.1*a* with $p_X = 0.2$, $p_Y = 0.3$, and $p_e = 0.05$. To determine the probability of getting a 1 at the gate's output, the input

vectors, namely, M_X = [0.8 0.2] and M_Y = [0.7 0.3] are formed first. These vectors are then combined via the tensor product of Equation (2.5)

$$M_{XY} = M_X \otimes M_Y = [0.56 \quad 0.24 \quad 0.14 \quad 0.06] \tag{2.6}$$

to give the probabilities associated with all four possible input combinations. The resulting input vector is multiplied by the PTM of the error-affected AND gate to obtain the circuit's output vector.

$$M_{Z^*} = M_{XY} \times M_{\text{AND}} = [0.56 \quad 0.24 \quad 0.14 \quad 0.06] \times \begin{bmatrix} 0.95 & 0.05 \\ 0.95 & 0.05 \\ 0.95 & 0.05 \\ 0.05 & 0.95 \end{bmatrix} = [0.896 \quad 0.104]$$

From this, we conclude that p_{Z^*}, i.e., the probability of getting a 1 at z*, is 0.104. Note that the PTM M_{AND} of the AND gate implicitly incorporates the bit-flip error, so there is no need for the XOR gate of Figure 2.1*b*, as is also the case in Equation (2.4).

2.4 Impact on Stochastic Numbers

Consider a stochastic number X with the expected value $\mathbb{E}[X] = p_X$. In a noisy environment, if X is affected by bit-flip error e with expected value p_e, the SN becomes $X^* = X \oplus e$. We therefore have

$$p_{X^*} = \mathbb{E}[X^*] = p_X + p_e(1 - 2p_X) \tag{2.7}$$

Besides the expected value of X^*, we are interested in E_{X^*}, the mean square error of X^*, which denotes the average error occurring in a stochastic circuit, i.e., the average difference between the estimated value $\hat{p}_{X^*}$ and the exact value p_X.

$$E_{X^*} = \mathbb{E}[(\hat{p}_{X^*} - p_X)^2]$$

Note that E_{X^*} reflects both the random fluctuations of the bit-stream representation and the error *e* due to bit-flips. As mentioned earlier, X^* is a Bernoulli RV defined by its expected value, so using only Equation (2.7), we should be able to find p_{X^*} and hence E_{X^*} analytically. Assuming the estimated value $\hat{p}_{X^*} = 1/N \sum_{i=1}^{N} X_i^*$ obtained by summing *N* independent samples of X^*, we obtain

$$\begin{aligned} E_{X^*} = \mathbb{E}[(\hat{p}_{X^*} - p_X)^2] &= \mathbb{E}\left[\hat{p}_{X^*}^2 + p_X^2 - 2p_X\hat{p}_{X^*}\right] \\ &= \mathbb{E}\left[\hat{p}_{X^*}^2\right] + \mathbb{E}[p_X^2] + \mathbb{E}[-2p_X\hat{p}_{X^*}] \\ &= \frac{N^2 p_{X^*}^2 + N p_{X^*}(1 - p_{X^*})}{N^2} + p_X^2 - 2p_X p_{X^*} \\ &= (p_{X^*} - p_X)^2 + \frac{p_{X^*}(1 - p_{X^*})}{N} \end{aligned} \tag{2.8}$$

The first term in Equation (2.8) is the difference between the expected values of *X* and X^*, and its only cause is the bit-flip *e*. The second term is a random-fluctuation error that diminishes with increasing *n*. We can re-write E_{X^*} in terms of p_X and p_e by substituting Equation (2.7) into Equation (2.8) thus:

$$E_{X^*} = p_e^2(1 - 2p_X)^2 + \frac{1}{N}\left[p_X(1 - p_X) + p_e(1 - p_e)\big(1 - 4p_X(1 - p_X)\big)\right]$$

Observe that the MSE error depends on both p_X and p_e. For sufficiently large *N*, E_{X^*} becomes 0 when $p_X = 1/2$, while it becomes p_e for $p_X = 1$.

In a similar way, we can analyze the effect of bit-flip errors on conventional (non-stochastic) binary numbers. An *m*-bit binary number *B* affected by independent and identically distributed bit-flips on each bit becomes B^*, which can potentially be any *m*-bit number with some probability. The error of B^* and its probability of occurrence depend on the number of bit-flips m_{bf}. To find the MSE E_{B^*} in this case, we calculate the weighted average error over all possible B^* values.

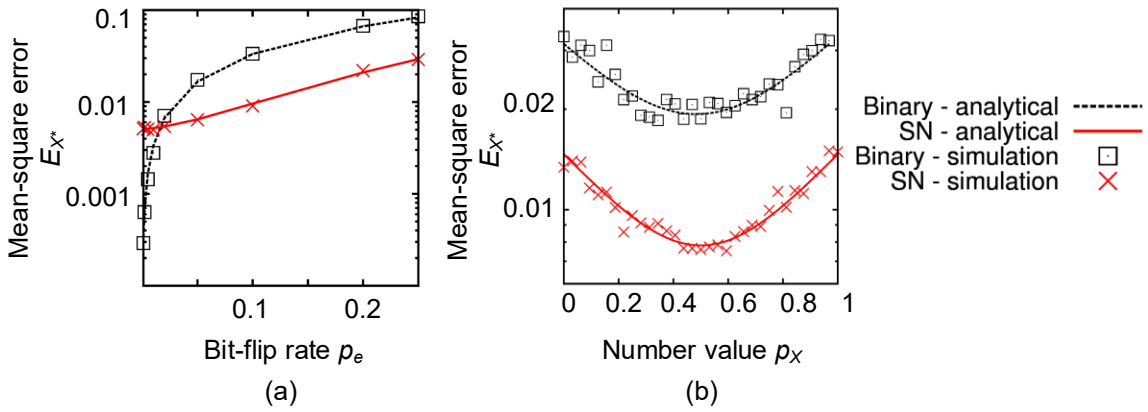

Figure 2.3: MSE of a stochastic and a binary number in the presence of bit-flips calculated using analytical and simulation methods: (a) for different values of p_e, and (b) for different values of p_X.

$$E_{B^*} = \sum_{B_i^*=0}^{2^m} (B_i^* - B)^2 p_e^{m_{bf}} (1 - p_e)^{m - m_{bf}}$$

Using the above equations, we compare the effect of bit-flips on a 5-bit binary number and an SN of length 32 whose precision is also 5 bits. Figure 2.3*a* shows the MSEs E_{X^*} and E_{B^*} at different bit-flips rates. Initially, the SN has a higher error due to its random fluctuations. However, as p_e increases, SN outperforms the binary number with respect to error tolerance. Figure 2.3*b* shows the MSEs E_{X^*} and E_{B^*} at different values of p_X; the MSEs in this case are averaged over several bit-flip rates ranging from $p_e = 0.001$ to 0.25. As can be seen, E_{X^*} is approximately 50% less than E_{B^*}. These analytical results are also confirmed in Figure 2.3 by Monte Carlo simulation.

2.5 Impact on Stochastic Circuits

A key feature of our error analysis is the use of PTMs to estimate the impact of errors on stochastic behavior. PTMs can be used to calculate the probability distribution of all output combinations of a stochastic circuit *C*. Given specific input signal probabilities, the input vector is multiplied by the PTM of the circuit *C* to obtain the output probabilities.

For example, consider again the AND gate Figure 1.3*a* which multiplies two SNs p_X and p_Y, and has output error p_e. Generalizing Equation (2.6) gives the 1 × 4 input vector

$$M_{XY} = [(1 - p_X)(1 - p_Y) \quad (1 - p_X)p_Y \quad p_X(1 - p_Y) \quad p_X p_Y]$$

Now, consider two cases: first, the AND gate is error-free, and second, it contains the error *e* defined by the PTM in Equation (2.4). In the error-free case, the output vector is

$$M_{XY} \times J_{\text{AND}} = [1 - \ p_z \quad p_Z] = [1 - \ p_X p_Y \quad p_X p_Y]$$

indicating that the probability of output 1 is $p_X p_Y$. If error *e* is present, then using M_{AND} from Equation (2.4), the output becomes

$$M_{XY} \times M_{\text{AND}} = \begin{bmatrix} (1 - p_e)\big((1 - p_X)(1 - p_Y) + (1 - p_X)p_Y + p_X(1 - p_Y)\big) + p_e p_X p_Y \\ p_e\big((1 - p_X)(1 - p_Y) + (1 - p_X)p_Y + p_X(1 - p_Y)\big) + (1 - p_e)p_X p_Y \end{bmatrix}^{\text{T}}$$

where T denotes matrix transposition (used to save space). This implies that the expected value of the output is $p_{Z^*} = p_e\big((1 - p_X)(1 - p_Y) + (1 - p_X)p_Y + p_X(1 - p_Y)\big) + (1 - p_e)p_X p_Y$. The MSE E_{Z^*} can now be calculated from Equation (2.8).

We can readily generalize the above technique to arbitrary stochastic circuits to analyze their stochastic behavior under single or multiple errors. First, generate the PTMs and ITMs for each individual logic or wiring gate. Then, apply the ordinary and tensor products repeatedly to calculate the PTM and ITM for the entire circuit [53]. Again, if the circuit has *n* inputs and *m* outputs, its final PTM and ITM will both be $2^n \times 2^m$ matrices.

Besides using the PTM method to analyze the behavior of a stochastic circuit in the presence of errors, we can employ gate-level circuit simulation to achieve the same goal. As in Figure 2.1, we inject the bit-flips into a gate via an XOR gate that flips the output signal *z* of *C* with probability p_e, resulting in a new erroneous signal z^*. For a circuit containing multiple gates, the error is injected into every gate.

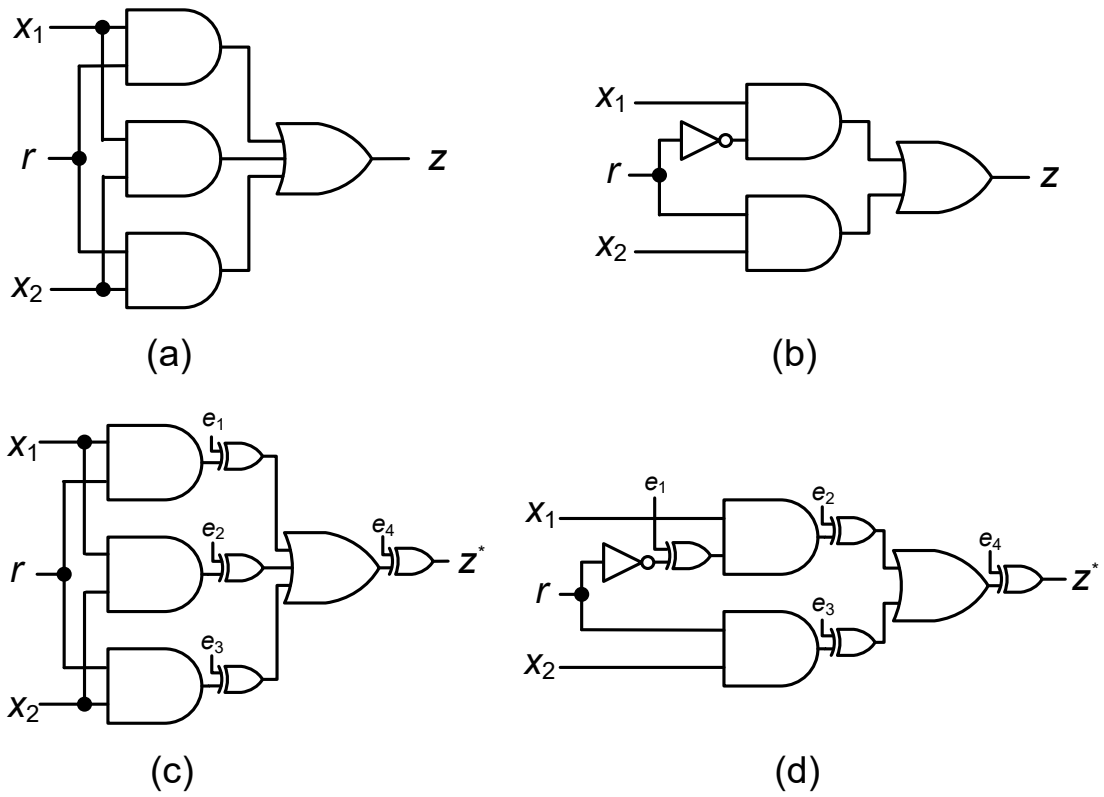

Figure 2.4: Stochastic circuits for the scaled addition $p_Z = 0.5\ (p_{X1} + p_{X2})$: (a) majority-based, (b) multiplexer-based, (c) majority-based with error injection, and (d) multiplexer-based with error injection.

Consider, for example, the stochastic realization of scaled addition. This operation can be implemented either by a majority circuit or a multiplexer [1], as shown in Figure 2.4*a–b*. The special input *r* receives a constant SN of value 0.5. The corresponding circuits with XOR gates added for error injection are shown in Figure 2.4*c–d*. To focus on the behavior of the computational hardware (the logic gates) in the presence of errors, we assume the data sources are not affected by errors.

Figure 2.5 presents error data obtained by PTM analysis and circuit simulation for the three basic gate types AND, OR and NOT, as well as the scaled adder circuits of Figure 2.4. The error rates of all gates are assumed to be the same ($p_{e_1} = p_{e_2} = p_{e_3} = p_{e_4} = p_e$), but they are generated from independent random sources. We simulated the circuit with and without the added XOR gates to get the expected error-free values and the values affected

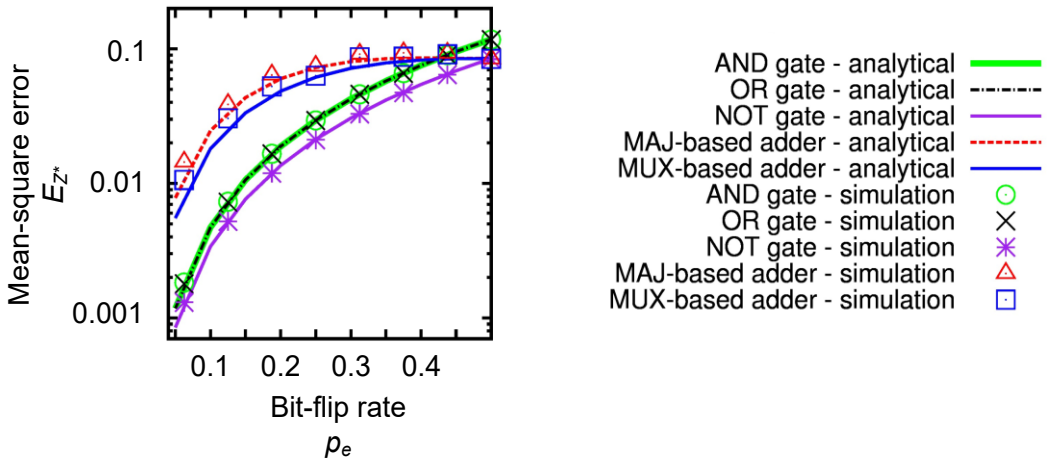

Figure 2.5: MSE at the outputs of representative stochastic circuits in the presence of soft errors calculated using analytical and simulation methods.

by soft errors. The MSE is given by $E_{Z^*} = \mathbb{E}[(\hat{p}_{Z^*} - p_Z)^2]$. As Figure 2.5 shows, the analytical and simulation results are quite consistent.

We also constructed PTMs M_{MAJ} and M_{MUX} for the circuits of Figure 2.4*a*–*b* level by level from the PTMs of their component gates, including wiring gates, according to the method of [53]. In high-level symbolic form, we obtain the PTM expressions

$$M_{\mathrm{MAJ}} = (F_2 \otimes F_2 \otimes F_2)(I \otimes swap \otimes swap \otimes I)\left(AND2_{p_e} \otimes AND2_{p_e} \otimes AND2_{p_e}\right)\left(OR3_{p_e}\right)$$

$$M_{\mathrm{MUX}} = (I \otimes F_2 \otimes I)\left(I \otimes NOT_{p_e} \otimes I \otimes I\right)\left(AND2_{p_e} \otimes AND2_{p_e}\right)\left(OR2_{p_e}\right)$$

Fully expanded, M_{MAJ} and M_{MUX} become 8×2 matrices, which we derived from the above equations with the aid of GNU Octave [35]. The ITMs J_{MAJ} and J_{MUX} for the two circuits, which have $p_e = 0$, take the form

$$J_{\mathrm{MAJ}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}^{\mathrm{T}}$$

$$J_{\mathrm{MUX}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}^{\mathrm{T}}$$

When errors are present, the 0-1 entries of J_{MAJ} and J_{MUX} must be replaced by complex polynomial expressions involving the variable p_e to obtain M_{MAJ} and M_{MUX} in expanded form.

Knowing both the erroneous PTMs and the ITMs, we can calculate the corresponding MSEs; see Figure 2.5. Again, the analytical results confirm the circuit simulations. In other words, both circuit simulation and PTM manipulation are valid methods for estimating soft-error effects in stochastic circuits. These results also show that when multiple errors are present, the errors accumulate. Hence, when the error rate is low, the multi-gate adders have worse MSE than single gates. When the error rate is high, for example, near 0.5, the behavior of all the circuits tends to appear random, so that they all have approximately the same MSE.

2.6 Case Study: Image Edge Detection

As noted in Section 1.2 in connection with retinal implants, edge detection is a fundamental operation in image processing and computer vision. Its goal is to identify significant local changes of intensity in digital images. Stochastic edge detectors have been shown to be significantly smaller, faster, more power-efficient, and more noise-tolerant than conventional ones in real-time image processing; see Figure 2.6 [3]. These designs, which are based on the Roberts cross edge-detection algorithm [37] of Equation (1.1), compute a moving average across a pixel window of size 2 × 2 for each pixel $p_{X_{i,j}}$ at row i and j of the image, and generate the stochastic output value $p_{Z_{i,j}}$

$$p_{Z_{i,j}} = 0.5\left(\left|p_{X_{i,j}} - p_{X_{i+1,j+1}}\right| + \left|p_{X_{i+1,j}} - p_{X_{i,j+1}}\right|\right) \tag{2.9}$$

Note that the stochastic implementation of Equation (2.9) requires the scaling factor 0.5 to perform the addition, and takes advantage of certain correlation properties of stochastic

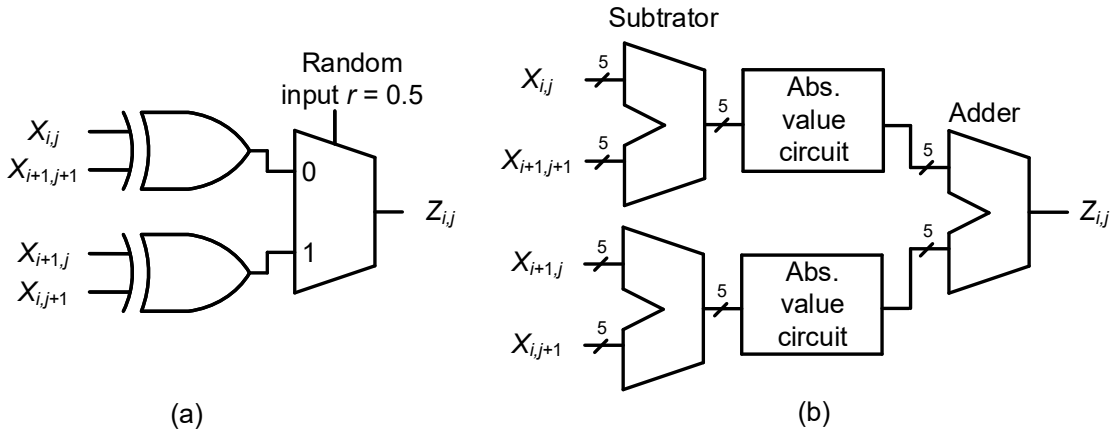

Figure 2.6: Edge detectors: (a) stochastic and (b) conventional.

numbers. An XOR gate $z = x \oplus y$ with uncorrelated (independent) inputs performs the function $p_Z = p_X(1 - p_Y) + (1 - p_X)p_Y$. However, if SNs X and Y are highly correlated with maximum overlap of 1s, the XOR gate's function becomes $p_Z = |p_X - p_Y|$, which allows Equation (2.9) to be realized by two XOR gates and a multiplexer as shown in Figure 2.6*a* [3]. Assuming 5-bit precision, Figure 2.6*b* shows the corresponding binary design which contains several large arithmetic blocks, including addition, subtraction and absolute-value circuits. The stochastic edge detector is about two orders of magnitude smaller than the conventional design.

We now use PTMs to analyze the behavior of these circuits under noisy conditions. The effect of a bit-flip rate of p_e on the output of every gate in the circuits is represented by a suitable PTM. Suppose the PTMs for the stochastic and conventional edge detectors are M_{sc} and M_{conv}, respectively. For each pixel and its 2 × 2 window, we generate the corresponding input vectors M_{in} and M'_{in} for the stochastic and conventional edge detectors, respectively. The result of the edge-detection operation is then calculated as $M_{in} \times M_{sc}$ and $M'_{in} \times M_{conv}$. In this example, we assume that 5-bit precision is required, so the bit-stream length is $2^5 = 32$ for the stochastic design. We do not consider any additional circuits that might be needed for number conversion between the binary and stochastic formats.

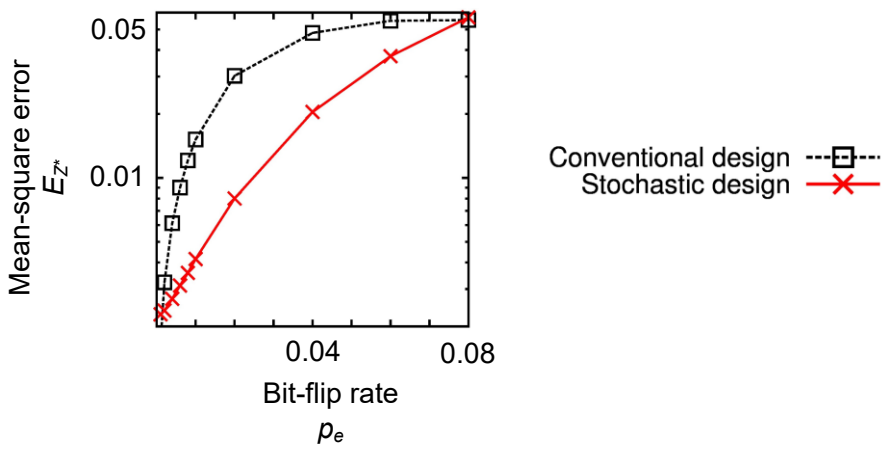

Figure 2.7: MSE of stochastic and conventional edge-detection circuits in the presence of soft-errors.

Figure 2.7 compares the MSEs of the stochastic and conventional designs. As expected, when the error rate is low, the stochastic circuit is more affected by random fluctuation errors and performs worse than the conventional one. However, as the error rate increases, the MSE of the conventional design increases rapidly. When the error rate is very high, all the signal values become essentially random in both designs, so the MSEs coverage to the same value. Note that this result is consistent with the results shown in Figure 2.5.

Figure 2.8 compares the output image quality of the two edge detectors of Figure 2.6 in the presence of errors injected into them to simulate the impact of soft errors on the edge-detection hardware. It shows that when noise causes the output of the conventional circuit to become almost unrecognizable (at around p_e = 2%), the stochastic circuit still produces acceptable results. In this experiment, noise is injected to demonstrate the fault-tolerant behavior of the stochastic circuits. Together, these two experiments show that when the conventional design fails to produce recognizable results, stochastic computing can produce good results in the presence of severe noise that affects both the input image and the edge-detection circuit.

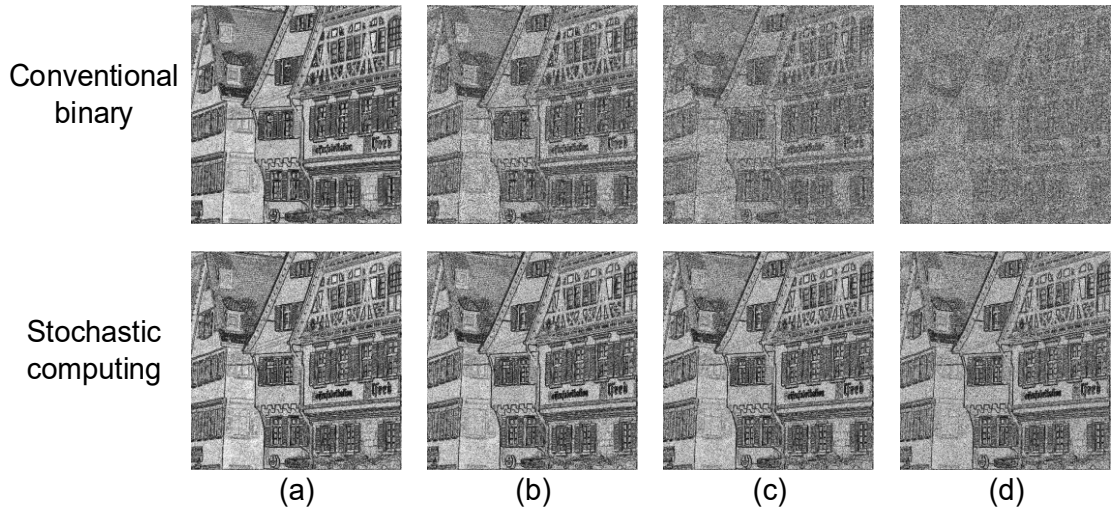

Figure 2.8: Comparison of stochastic and conventional edge detection for various soft-error rates (bit-flips percentages) in the edge-detection circuits: (a) 0.1%, (b) 0.5%, (c) 1% and (d) 2%.

2.7 Summary

We have presented a quantitative study of error tolerance that considers multiple error effects, and accounts for the inherent fluctuations in stochastic data, as well as externally induced bit-flip errors affecting the data-processing circuits. We successfully used two complementary approaches: algebraic analysis with probabilistic transfer matrices (PTMs), and Monte Carlo circuit simulation. The algebraic analysis is more accurate than the simulation approach, but has the disadvantage of being infeasible for large circuits. Since stochastic circuits are very simple by nature, the algebraic approach is generally applicable to them, and is hence preferred. The simulation approach gives reasonably accurate results, and is feasible for circuits of any reasonable size. Our experimental results show that stochastic circuits have far better error tolerance than conventional binary circuits, especially at higher error rates.

CHAPTER 3
Correlation

The effect of soft errors and random sources on the accuracy of SC was examined in the previous chapter. This chapter presents our research on analyzing and controlling the impact of signal dependence or correlation on accuracy. Interacting bit-streams are normally required to be independent or uncorrelated. For example, an AND gate performs multiplication accurately only if it input SNs are highly uncorrelated; see Figure 1.6 As stochastic signals pass through the levels of a circuit and interact with one another, correlations among them tend to increase. Unfortunately, maintaining adequate independence among such signals is costly and not well understood. The goal of this chapter is to quantify the impact of correlation on the accuracy in SC, and to evaluate the major known methods of reducing correlation or, equivalently, maintaining accuracy over multiple computational steps. To this end, we develop a general analytic framework for SC based on PTM algebra. This work has been published previously in [23].

3.1 Analysis Framework

Accuracy-versus-time concerns and the need to generate many uncorrelated randomized inputs have long prevented the widespread use of SC. Our understanding of how correlation affects stochastic circuits is still mainly qualitative, and only a few attempts have been made to quantify it and analyze its properties. Alaghi and Hayes introduced the SC correlation (*SCC*) measure [4], while Ma et al. have analyzed aspects of variance propagation in stochastic circuits when inputs are uncorrelated [62]. It is well-known that correlation can be reduced by the introducing special circuits to re-randomize SNs that have

become correlated, but the impact of re-randomization circuits on accuracy and hardware cost has not been studied.

As Chapter 2 shows, PTMs and their associated algebra constitute a powerful tool for analyzing complex stochastic behavior. This chapter develops a general PTM-based framework for quantitatively analyzing the impact of correlation on the accuracy of stochastic circuits. As discussed in Section 2.2, we can treat an SN X as an RV with a Bernoulli distribution and a parameter p_X denoting the expected probability that a bit of X is 1. We also discussed the PTM analysis of combinational circuits in Section 2.3. For example, if $p_{X_1} = 0.6$ and $p_{X_2} = 0.3$, Equation (2.5) becomes $M_{X_1X_2} = [0.28 \quad 0.12 \quad 0.42 \quad 0.18]$. With input distribution $M_{X_1X_2}$ and J_{AND} of Equation (2.2), an AND gate $z = x_1x_2$ generates the output distribution

$$M_Z = M_{XY} \times J_{\text{AND}} = [0.28 \quad 0.12 \quad 0.42 \quad 0.18] \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = [0.82 \quad 0.18] \qquad (3.1)$$

We now extend that discussion to sequential circuits because in addition to *re-randomizing* SNs, another correlation-reduction approach is to *isolate* correlated SNs by deriving SNs from delayed versions of a single random sequence [31]. This approach inserts delay elements called isolators into parts of a circuit that need their correlation reduced. Isolators are used to shift SNs relative to one another so that correlated bits no longer overlap. A D-type flip-flop suffices to implement an isolator; when inserted into a wire carrying a signal x, it delays x by one clock cycle. This, of course, has the effect of making a combinational circuit sequential. Stochastic circuits containing sequential components have received little attention.

To illustrate the extension of PTM to sequential circuits, consider the JK flip-flop in Figure 3.1. The input combinations $JK = 01$ and 10 set the flip-flop's state Q^+ to 0 and 1, respectively. $JK = 00$ leaves the state unchanged at $Q^+ = Q$, while $JK = 11$ toggles the flip-

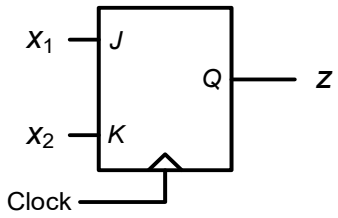

Figure 3.1: JK flip-flop performing the stochastic operation $p_Z = p_{X1} / (p_{X1} + p_{X2})$.

flop, making $Q^+ = \bar{Q}$. Therefore, $p_{Q^+} = p_{00}p_Q + p_{10} + p_{11}(1 - p_Q)$, where $p_{00,}$, p_{01} and p_{11} denote the probability of JK = 00, 10 and 11, respectively. Now $p_z = p_{Q^+} = p_Q$ when the flip-flop is in steady-state, so it is easily seen that $p_Z = (p_{10} + p_{11})/(1 - p_{00} + p_{11})$. With $p_{X_1} = p_J = p_{10} + p_{11}$, $p_{X_2} = p_K = p_{01} + p_{11}$, and $p_{00} + p_{01} + p_{10} + p_{11} = 1$, the output z has the probability $p_Z = p_{X_1}/(p_{X_2} + p_{X_2})$. This approximates the basic division operation $p_Z = p_{X_1}/p_{X_2}$ when the dividend p_{X_1} is small. However it becomes very inaccurate when the divisor p_{X_2} is small. The probabilistic behavior of sequential circuits can also be analyzed by PTMs when pseudo-inputs to represent current state variables are introduced. With the three inputs x_1x_2Q, the (transposed) ITM for the JK flip-flop is

$$J_{\text{JK}} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}^{\text{T}} \tag{3.2}$$

whose entries are next-state values $z = Q^+$.

This chapter presents a PTM-based framework for quantifying the impact of correlation on the accuracy of stochastic circuits. It addresses the two most practical of the known methods for reducing correlation, namely regeneration and isolation. The results provide formulas for calculating or bounding numerical accuracy, and so have the potential to increase the application range of SC. They also demonstrate that the isolation method of reducing correlation offers major advantages in improving accuracy at relatively low cost.

3.2 Representation

Many application-dependent measures of the statistical similarity between bit-streams have been proposed [26]. The standard definition used in communication theory is

$$\rho(X_1, X_2) = \frac{ad - bc}{\sqrt{(a+b)(a+c)(b+d)(c+d)}} \tag{3.3}$$

and is known as Pearson correlation. Here a denotes the number of overlapping 1s, b is the number of overlapping 1s of X_1 and 0s of X_2, c is the number of overlapping 0s of X_1 and 1s of X_2, and d denotes the number of overlapping 0s. Pearson correlation is designed so that $\rho = +1$ or -1 implies that X_1 and X_2 are identical or complementary, respectively. For example, with X_1 = 01100011 and X_2 = 10100101 as in Figure 1.6*a*, we have $a = b = c = d = 2$ and $\rho(X_1, X_2) = 0$. If two bit-streams are highly correlated, such as $X_1 = X_2$ = 0110 0101 in Figure 1.6*b*, then $a = d = 4$, $b = c = 0$ and $\rho(X_1, X_2) = 1$.

Pearson correlation is unsuited to our needs, however. Instead, we use the SC correlation (*SCC*) measure from [4], which was specifically designed for SC. With the notation introduced above for ρ, *SCC* is defined as follows.

$$SCC(X_1, X_2) = \begin{cases} \dfrac{ad - bc}{N \cdot \min(a+b, a+c) - (a+b)(a+c)} & \text{if } ad > \text{bc} \\ \dfrac{ad - bc}{(a+b)(a+c) - N \cdot \max(a-d, 0)} & \text{otherwise} \end{cases} \tag{3.4}$$

where $N = a + b + c + d$ denotes the bit-stream length.

A key property of *SCC* is that the correlation between X_1 and X_2 is not dependent on their probability values. In other words, when the values of two SNs are different, as long as they have maximum/minimum overlap of 1s and 0s, they consistently have the *SCC* value +1/–1. For example, suppose X_1 = 10110010 and X_2 = 10100000, so whenever X_2 has a 1

bit, the corresponding position of X_1 is also 1. Then $SCC(X_1, X_2) = +1$ and we can say that $X_1$and X_2 have the maximum similarity, but $\rho(X_1, X_2) = 0.577$ does not reflect this.

SCC can be further used to model the probabilistic behavior of a circuit whose inputs are correlated. For a two-input stochastic circuit with output $Z(X_1, X_2)$,

$$p_{Z^*}(p_{X_1}, p_{X_2}) = \begin{cases} (1 + SCC) \times Z_0(p_{X_1}, p_{X_2}) - SCC \times Z_{-1}(p_{X_1}, p_{X_2}) & \text{if } SCC < 0 \\ (1 - SCC) \times Z_0(p_{X_1}, p_{X_2}) + SCC \times Z_1(p_{X_1}, p_{X_2}) & \text{otherwise} \end{cases} \quad (3.5)$$

where $Z_0(p_X, p_Y)$, $Z_{-1}(p_X, p_Y)$ and $Z_{+1}(p_X, p_Y)$ are the circuit functions when *SCC* is 0, –1 and +1, respectively. For example, the AND gate performs multiplication when SCC is 0, so $Z_0(p_{X_1}, p_{X_2}) = p_{X_1} \times p_{X_2}$. With *SCC* = –1 and +1, we have $Z_{-1}(p_{X_1}, p_{X_2}) = \max(p_{X_1} + p_{X_2} - 1, 0)$ and $Z_{+1}(p_{X_1}, p_{X_2}) = \min(p_{X_1}, p_{X_2})$ [4]. Thus correlation can be viewed as a function-changing phenomenon.

As demonstrated in Section 3.1, PTMs can represent correlated signals. The 4-element vector *V* giving the joint distribution of two uncorrelated signals X_1 and X_2 can be easily computed via the tensor product. *V* cannot be computed so easily when X_1 and X_2 are correlated. To understand the impact of correlation on accuracy, a new way to compute the joint distribution of two bit-streams is needed, which we now develop.

Consider input bit-streams X_1 and X_2 whose PTMs are $V_{X_1} = [1 - p_{X_1} \quad p_{X_1}]$ and $V_{X_2} = [1 - p_{X_2} \quad p_{X_2}]$, respectively. Let $V_{\text{in}} = [p_{00} \quad p_{01} \quad p_{10} \quad p_{11}]$. When $SCC(X_1, X_2) = 0$, the input vector V_{in} is

$$\begin{aligned} V_0 &= V_{X_1} \otimes V_{X_2} \\ &= [(1 - p_{X_1})(1 - p_{X_2}) \quad (1 - p_{X_1})p_{X_2} \quad p_{X_1}(1 - p_{X_2}) \quad p_{X_1}p_{X_2}] \end{aligned} \quad (3.6)$$

When $SCC(X_1, X_2) = 1$, maximum overlap of the bit-streams is assured. If $p_{X_1} \geq p_{X_2}$, $x_1 x_2 = 01$ can never occur and the probabilities of $x_1 x_2$ being 00, 10 and 11 are $1 - p_{X_1}$, $p_{X_1} - p_{X_2}$ and p_{X_2}, respectively. We can therefore express the input vector as

$$V_{+1} = \begin{cases} [1 - p_{X_1} \quad 0 \quad p_{X_1} - p_{X_2} \quad p_{X_2}] & \text{if } p_{X_1} \geq p_{X_2} \\ [1 - p_{X_2} \quad p_{X_2} - p_{X_1} \quad 0 \quad p_{X_1}] & \text{otherwise} \end{cases} \tag{3.7}$$

or, equivalently,

$$V_{+1} = \begin{bmatrix} 1 - \max(p_{X_1}, p_{X_2}) & \max(0, p_{X_2} - p_{X_1}) \\ \max(0, p_{X_1} - p_{X_2}) & \min(p_{X_1}, p_{X_2}) \end{bmatrix} \tag{3.8}$$

Example 3.1: For example, let $p_{X_1} = 0.9$, $p_{X_2} = 0.2$ with the bit-streams X_1 = 1111111110 and X_2 = 1100000000. We then have $V_{+1} = [1 - 0.9 \quad 0 \quad 0.9 - 0.2 \quad 0.2] = [0.1 \quad 0 \quad 0.7 \quad 0.2]$. □

When $SCC(X_1, X_2) = -1$, minimal overlap occurs, and $x_1 x_2 = 11$ will never appear if $p_{X_1} + p_{X_2} \leq 1$. On the other hand, $x_1 x_2 = 00$ will never appear if $p_{X_1} + p_{X_2} \geq 1$. Therefore, we need to consider these two cases separately.

$$V_{-1} = \begin{cases} [1 - (p_{X_1} + p_{X_2}) \quad p_{X_2} \quad p_{X_1} \quad 0] & \text{if } p_{X_1} + p_{X_2} \leq 1 \\ [0 \quad 1 - p_{X_1} \quad 1 - p_{X_2} \quad p_{X_1} + p_{X_2} - 1] & \text{otherwise} \end{cases} \tag{3.9}$$

Example 3.2: Let $p_{X_1} = 0.6$, $p_{X_2} = 0.2$ with X_1 = 1111110000 and X_2 = 0000000011. Since $p_X + p_Y = 0.8 < 1$,

$$V_{-1} = [1 - (0.6 + 0.2) \quad 0.2 \quad 0.6 \quad 0] = [0.2 \quad 0.2 \quad 0.6 \quad 0]$$

With $p_{X_1} = 0.8$, $p_{X_2} = 0.3$, X_1 = 1111111100 and X_2 = 000000011. Hence, we have $p_{X_1} + p_{X_2} = 1.2 > 1$, therefore $V_{-1} = [0 \quad 0.2 \quad 0.7 \quad 0.1]$. □

Generalizing to arbitrary values of *SCC*, the input vector *V* is a linear combination of V_0, V_{-1}, and V_{+1} given by the following theorem.

Theorem 3.1: Let X_1 and X_2 be SNs with probabilities p_{X_1} and p_{X_2}, respectively, and correlation $SCC(X_1, X_2)$. Their joint distribution $V_{SCC}(X_1, X_2)$ is

$$V_{SCC} = \begin{cases} (1 + SCC) \times V_0 - SCC \times V_{-1} & \text{if } SCC < 0 \\ (1 - SCC) \times V_0 + SCC \times V_{+1} & \text{otherwise} \end{cases} \tag{3.10}$$

where V_0, V_{+1} and V_{-1} are the joint distributions when $SCC = 0$, $+1$ and -1, and given by Equations (3.6), (3.7) and (3.9), respectively.

Note that if the correlation measure is $\rho(X_1, X_2)$ defined by Equation (3.3), the joint distribution of X_1 and X_2 cannot be easily formulated as in Equation (3.10). In fact, we cannot even specify V_{+1} and V_{-1}, because V_{+1} and V_{-1} are defined only when $p_{X_1} = p_{X_2}$ and $p_{X_2} = 1 - p_{X_1}$.

3.3 Impact on Stochastic Circuits

Next we examine the impact of correlation on three basic operations: addition, division and multiplication. We also use PTMs to show that scaled addition and approximate division are correlation-insensitive.

Figure 1.6 shows that uncorrelated input signals are important for accurate stochastic computing. However, not all SC circuits are vulnerable to correlation. For example, the results of the multiplexer-based stochastic adder (Figure 2.4*b*) are accurate even when the inputs are correlated [4]. Note that the same scaled addition can be implemented by a majority gate realizing $z = \text{MAJ}(x_1, x_2, r)$; see Figure 2.4*a*. The majority-based adder must also be correlation-insensitive because its PTM $M_{\text{MAJ}} = M_{\text{MUX}}$; see Equation (2.3) for M_{MUX}.

Addition is sometimes approximated by an OR gate since $p_Z = p_{X_1} + p_{X_2} - p_{X_1}p_{X_2} \approx p_{X_1} + p_{X_2}$ when p_{X_1} and p_{X_2} are small; this is "saturated" addition [31]. It

may provide acceptably accurate results when the input probabilities are very low. However, it is sensitive to correlation, and saturated addition is little used.

Using PTMs, we can show that the accuracy of the approximate divider in Figure 3.1 is unaffected by correlation among its inputs. Again, let $V_{\text{in}} = [p_{00} \quad p_{01} \quad p_{10} \quad p_{11}]$. Since the divider is a sequential circuit whose next state Q^+ depends on the current state Q, we have the effective input vector $V_{\text{in}} \otimes [1 - p_Q \quad p_Q]$. The corresponding output vector is

$$V_{\text{out}} = [1 - p_{Q^+} \quad p_{Q^+}] = (V_{\text{in}} \otimes [1 - p_Q \quad p_Q]) \times J_{\text{JK}}$$

and $p_{Q^+} = p_{00}p_Q + p_{10}(1 - p_Q) + p_{10}p_Q + p_{11}(1 - p_Q)$. Now $p_z = p_{Q^+} = p_Q$ when the flip-flop is in steady-state, so it is easily seen that $p_Z = \frac{p_{10}+p_{11}}{1-p_{00}+p_{11}}$. Again, using $p_{X_1} = p_{10} + p_{11}$, $p_{X_2} = p_{01} + p_{11}$, and $p_{00} + p_{01} + p_{10} + p_{11} = 1$, we conclude that $p_Z = \frac{p_{X_1}}{p_{X_1}+p_{X_2}}$ always holds. The LDPC update node in Figure 1.3*e* is another example showing that the JK flip-flop is correlation insensitive, while its inputs are negatively correlated with *SCC* = –1.

SC multiplication is strongly affected by correlated inputs, as illustrated by Figure 1.6. This error may be analyzed as follows. Let Z and Z^* represent the exact and erroneous multiplication results. Clearly, $p_Z(p_X, p_Y) = p_x \times p_y$, but when correlation is present,

$$p_{Z^*}(p_X, p_Y) = \begin{cases} (1 + SCC) \times p_X p_Y - SCC \times \max(p_X + p_Y - 1, 0) & \text{if } SCC < 0 \\ (1 - SCC) \times p_X p_Y + SCC \times \min(p_X, p_Y) & \text{otherwise} \end{cases} \tag{3.11}$$

The mean-square error (MSE) E_{Z^*} is the average difference between the estimated value $\hat{p}_{Z^*}$ and the exact value p_Z squared, so $E_{Z^*} = \mathbb{E}[(\hat{p}_{Z^*} - p_Z)^2]$, where $\mathbb{E}$ is the expectation operator. Note that $\hat{p}_{Z^*} = 1/N \sum_{i=1}^{n} Z_i^*$ is obtained by summing N independent samples of Z^*, so

$$E_{Z^*} = \mathbb{E}[(\hat{p}_{Z^*} - p_Z)^2] = \mathbb{E}\left[\hat{p}_{Z^*}^2 + p_Z^2 - 2p_Z\hat{p}_{Z^*}\right]$$
$$= \mathbb{E}\left[\hat{p}_{Z^*}^2\right] + \mathbb{E}[p_Z^2] + \mathbb{E}[-2p_Z\hat{p}_{Z^*}]$$
$$= \frac{N^2 p_{Z^*}^2 + N p_{Z^*}(1 - p_{Z^*})}{N^2} + p_Z^2 - 2p_Z p_{Z^*}$$
$$= (p_{Z^*} - p_Z)^2 + \frac{p_{Z^*}(1 - p_{Z^*})}{N} \tag{3.12}$$

Therefore, in the AND multiplier case, the MSE is

$$E_{Z^*}$$
$$= \begin{cases} SCC^2(p_X p_Y - \max(p_X + p_Y - 1, 0))^2 + \dfrac{p_{Z^*}(1 - p_{Z^*})}{N} & \text{if } SCC < 0 \\ SCC^2(p_X p_Y - \min(p_X, p_Y))^2 + \dfrac{p_{Z^*}(1 - p_{Z^*})}{N} & \text{otherwise} \end{cases} \tag{3.13}$$

The second term $p_{Z^*}(1 - p_{Z^*})/N$ of each expression in Equation (3.13) is the random fluctuation error. If the bit-stream length N is very big, as is usually the case, then random fluctuation can be ignored. The first terms in Equation (3.13) are the errors due to

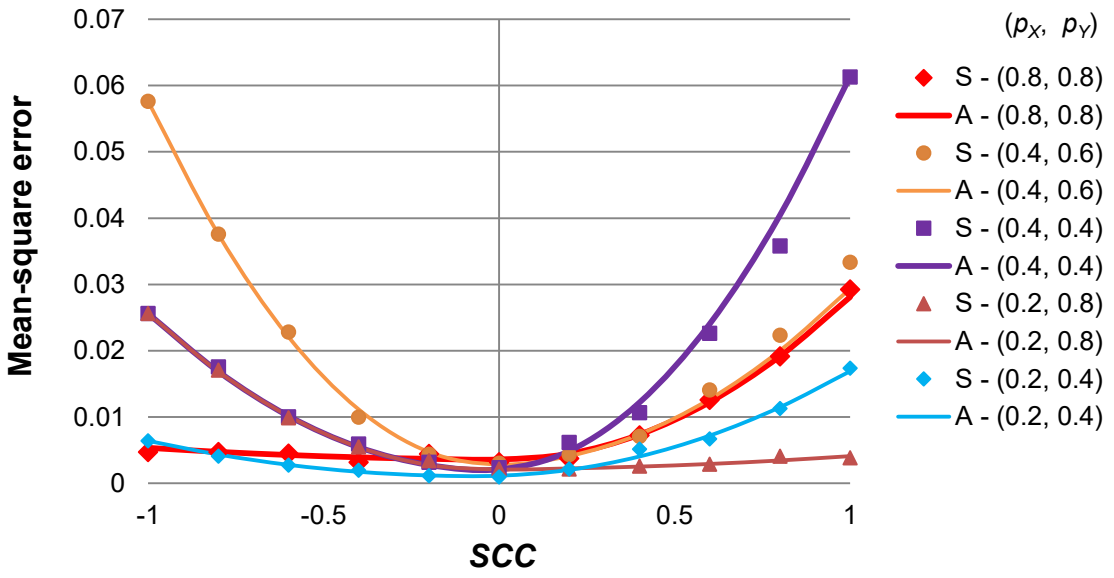

Figure 3.2: MSE of the AND multiplier calculated by analysis (A) and simulation (S) for various combinations of p_X, p_Y and SCC.

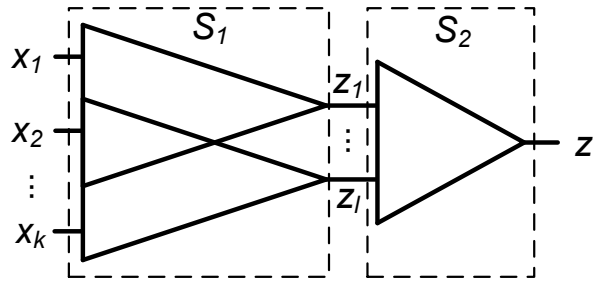

Figure 3.3: Circuit model S for correlation analysis; the triangles are the fan-in cones seen on backtracing from Z_1, …, Z_l and Z.

correlation, and they indicate that the MSE increases quadratically with *SCC*. Figure 3.2 plots the MSE of the AND multiplier assuming $N = 256$. The results are obtained from the analysis "A" in Equation (3.13) and circuit simulation "S". This figure shows that the analytical and simulation results are very close.

Now we generalize the foregoing analysis to the k-input stochastic circuit S of Figure 3.3. It consists of two cascaded sub-circuits S_1 and S_2 with k and l inputs, respectively. Since correlation is defined between two signals we first discuss the case when S_2 is a two-input circuit with inputs z_1, z_2, i.e. $l = 2$, and output z. The internal signals z_1 and z_2 may be correlated because z_1 and z_2 are derived from common inputs. The case when $l > 2$ will be discussed later in this section.

Theorem 3.2: Let Z_1 and Z_2 be SNs with probabilities p_{Z_1} and p_{Z_2}, respectively, and correlation $SCC(Z_1, Z_2)$. If S_2 is a stochastic circuit with inputs Z_1 and Z_2 and output Z as shown in Figure 3.3, the MSE of Z is

$$E_{Z^*} = \begin{cases} SCC^2\left(Z_0(p_{z_1}, p_{z_2}) - Z_{-1}(p_{z_1}, p_{z_2})\right)^2 + \dfrac{p_{Z^*}(1-p_{Z^*})}{N} & \text{if } SCC < 0 \\ SCC^2\left(Z_0(p_{z_1}, p_{z_2}) - Z_1(p_{z_1}, p_{z_2})\right)^2 + \dfrac{p_{Z^*}(1-p_{Z^*})}{N} & \text{otherwise} \end{cases} \quad (3.14)$$

where $SCC = SCC(Z_1, Z_2)$, p_{Z^*} is given by Equation (3.5), and $Z_0(p_{Z_1}, p_{Z_2})$, $Z_{-1}(p_{Z_1}, p_{Z_2})$ and $Z_{+1}(p_{Z_1}, p_{Z_2})$ are the circuit functions of S_2 with SCC = 0, −1 and +1, respectively.

We can use Theorem 3.2 and the PTM algebra to estimate the accuracy of a stochastic circuit like Figure 3.3. The main steps are as follows:

1. Calculate the PTMs of S_1 and S_2, namely M_{S_1} and M_{S_2}.
2. Use Equation (3.10) to obtain the joint distribution V_{in} for the primary inputs x_1, x_2, …, x_k.
3. Calculate $V_{Z_1Z_2} = V_{\text{in}} \times M_{S_1}$.
4. Use $V_{Z_1Z_2}$ and Equation (3.4) to get $SCC(Z_1, Z_2)$.
5. Calculate $Z_0(p_{Z_1}, p_{Z_2})$, $Z_{+1}(p_{Z_1}, p_{Z_2})$ and $Z_{-1}(p_{Z_1}, p_{Z_2})$ by multiplying each of Equations (3.6), (3.8) and (3.9) by M_{S_2}.
6. Determine the MSE E_{Z^*} from Equation (3.14).

Example 3.3: Figure 3.4 illustrates error estimation for a small stochastic circuit. The PTM for sub-circuit S_2 is $M_{S_2} = M_{\text{AND}}$ and the PTM for S_1 is

$$M_{S_1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}^T$$

Assuming the inputs are independent with probability 0.5,

$$V_{\text{in}} = [1/8 \quad 1/8 \quad 1/8 \quad 1/8 \quad 1/8 \quad 1/8 \quad 1/8 \quad 1/8]$$

we get

$$V_{Z_1Z_2} = V_{\text{in}} \times M_{S_1} = [1/8 \quad 1/8 \quad 1/8 \quad 5/8]$$

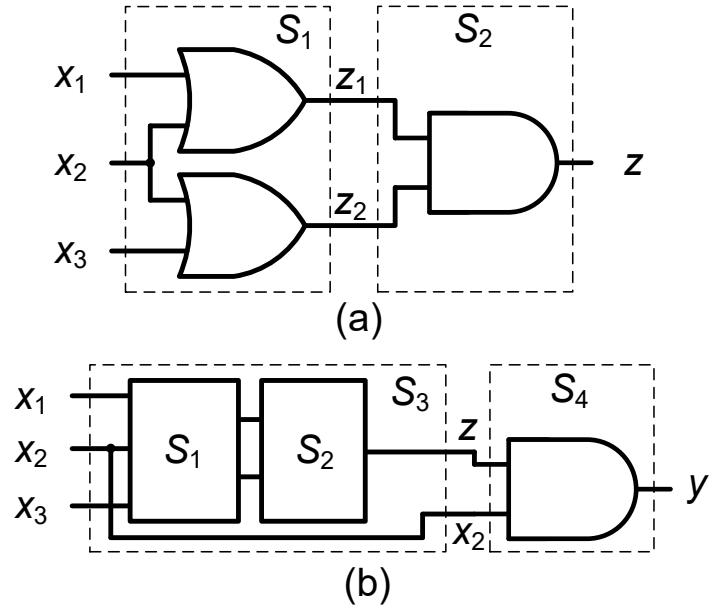

Figure 3.4: Stochastic circuits affected by correlation due to re-convergent signals. The target arithmetic functions are: $p_{Z1} = p_{X1} + p_{X2} - p_{X1} \times p_{X2}$, $p_{Z2} = p_{X2} + p_{X3} - p_{X2} \times p_{X3}$, $p_Z = p_{Z1} \times p_{Z2}$ and $p_Y = p_Z \times p_{X2}$.

Note that $V_{Z_1 Z_2}$ implies $p_{Z_1} = p_{Z_2} = 0.75$ and $SCC(Z_1, Z_2) = 1/3$. Since $M_{S_2} = M_{\text{AND}}$, we have $Z_0(p_{X_1}, p_{X_2}) = p_{X_1} \times p_{X_2}$, $Z_{-1}(p_{X_1}, p_{X_2}) = \max(p_{X_1} + p_{X_2} - 1, 0)$ and $Z_{+1}(p_{X_1}, p_{X_2}) = \min(p_{X_1}, p_{X_2})$. Therefore, we conclude that $p_{Z^*} = 0.625$ and $E_{Z^*} = 3.90625 \times 10^{-3} + \frac{0.234375}{N}$, where N is the bit-stream length. □

This example shows that the accuracy can be increased by increasing the bit-stream length N when N is small. However, as long as N is adequate, increasing N will not necessarily improve accuracy since the MSE is eventually dominated by the correlation error. With $N = 256$, the MSE of Z will be $E_{Z^*} = 4.822 \times 10^{-3}$. Comparing this to the case when $SCC = 0$, the MSE = 9.613×10^{-4} and the accuracy loss is mainly due to random fluctuations. The MSE calculated by circuit simulation is 4.77×10^{-3}, which is consistent with the analysis.

Next, we show how to quickly approximate the MSE of a circuit like Figure 3.3 when its inputs z_1 and z_2 are correlated. For large n, Equation (3.12) is approximated by

$E_{Z^*} \cong (p_{Z^*} - p_Z)^2$. To calculate $p_{Z^*} - p_Z$, assume the joint distribution of z_1 and z_2 is $V^*_{Z_1Z_2} = [p_{00} \quad p_{01} \quad p_{10} \quad p_{11}]$. Therefore, the marginalized probabilities are $M_{Z_1} = [p_{00} + p_{01} \quad p_{10} + p_{11}]$ and $M_{Z_2} = [p_{00} + p_{10} \quad p_{01} + p_{11}]$, i.e., $p_{Z_1} = p_{10} + p_{11}$ and $p_{Z_2} = p_{01} + p_{11}$. If z_1 and z_2 are uncorrelated, their joint probability distribution should be $V_{Z_1Z_2} = M_{Z_1} \otimes M_{Z_2}$, so

$$V_{Z_1Z_2} = [p_{00} + p_{01} \quad p_{10} + p_{11}] \otimes [p_{00} + p_{10} \quad p_{01} + p_{11}] \tag{3.15}$$

Because the expected output distribution is $V_Z = [1 - p_Z \quad p_Z] = V_{Z_1Z_2} \times M_{S_2}$ while the erroneous output distribution is $V^*_Z = [1 - p_{Z^*} \quad p_{Z^*}] = V^*_{Z_1Z_2} \times M_{S_2}$, the error $p_{Z^*} - p_Z$ can be calculated as follows:

$$\begin{aligned} V_{Z^*} - V_Z &= [p_Z - p_{Z^*} \quad p_{Z^*} - p_Z] = V_{Z_1Z_2} \times M_{S_2} - V^*_{Z_1Z_2} \times M_{S_2} \\ &= \left(V^*_{Z_1Z_2} - V_{Z_1Z_2}\right) \times M_{S_2} \end{aligned} \tag{3.16}$$

Since $E_{Z^*} \cong (p_{Z^*} - p_Z)^2$, we can estimate E_{Z^*} by averaging the square of each element in Equation (3.16).

Example 3.4: Continuing the example in Figure 3.4, we have

$$\begin{aligned} V^*_{Z_1Z_2} - V_{Z_1Z_2} &= [1/8 \quad 1/8 \quad 1/8 \quad 5/8] - ([2/8 \quad 6/8] \otimes [2/8 \quad 6/8]) \\ &= [1/16 \quad -1/16 \quad -1/16 \quad 1/16] \end{aligned}$$

Multiplying $V^*_{Z_1Z_2} - V_{Z_1Z_2}$ by the PTM of S_2 yields

$$\left(V^*_{Z_1Z_2} - V_{Z_1Z_2}\right) \times M_{S_2} = [-1/16 \quad 1/16]$$

Therefore, $E_{Z^*} \cong \frac{1}{2}\left(\left(-\frac{1}{16}\right)^2 + \left(\frac{1}{16}\right)^2\right) = 3.90625 \times 10^{-3}$. In other words, when N is big enough, the MSE is dominated by the correlation error, which is not a function of N. □

We can further generalize the foregoing approximation method to the case where S_2 has $l > 2$ inputs.

Theorem 3.3: For the stochastic circuit S in Figure 3.3, let V_{in} be the joint probability distribution for the k primary inputs $X_1, X_2, \ldots, X_k$. Let $V^*_{Z_1 Z_2 \ldots Z_l} = V_{\text{in}} \times M_{S1}$, and let M_{Z_1}, $M_{Z_2}, \ldots, M_{Z_l}$ be the PTMs for $Z_1, Z_2, \ldots, Z_l$ generated by marginalizing their joint distribution $V^*_{Z_1 Z_2 \ldots Z_l}$. Define $V_e = V^*_{Z_1 Z_2 \ldots Z_m} - V_{Z_1 Z_2 \ldots Z_l}$ where $V_{Z_1 Z_2 \ldots Z_l} = M_{Z_1} \otimes M_{Z_2} \otimes \ldots \otimes M_{Z_l}$. The MSE of Z is approximated by

$$E_{Z^*} \cong Avg\left(\left(V_e \times M_{S_2}\right).\text{^}2\right) \tag{3.17}$$

where ". ^2" is the element-wise square operation and *Avg* denotes the average of the matrix elements.

3.4 De-correlation Methods

Throughout the SC literature, the basic way to (re) randomize a pair of correlated SNs is the following two-step technique we call *regeneration*. First, convert at least one of the SNs X to binary form B, and second, use an SNG like that of Figure 1.3*d* to regenerate X from B. Stochastic-to-binary conversion simply requires a counter to sum the 1-bits of X.

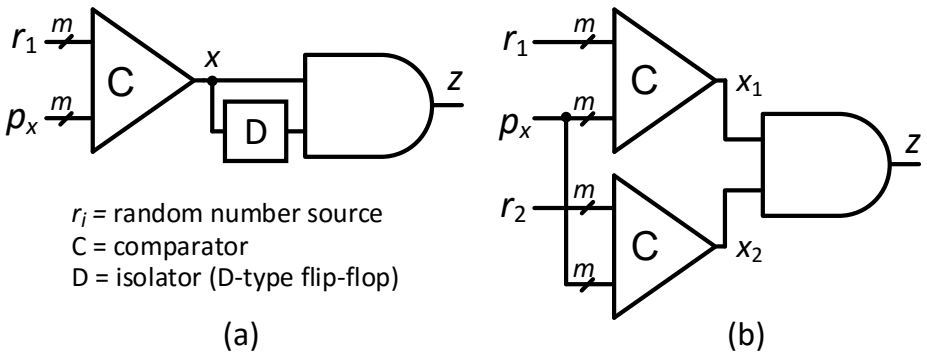

Figure 3.5: Multiplier used as a squarer (a) with one SNG and a stochastic isolator; (b) with two SNGs.

The use of a new random source in the SNG ensures statistical independence and satisfies Equation (3.15). However, since a counter and an SNG are required for each regenerated signal, the hardware cost of regeneration is very high.

Another, less common correlation-reduction approach is to derive SNs from delayed versions of a single random sequence. This is based on an observation of Gaines [31] that a statistically independent version of a Bernoulli sequence may be obtained by delaying it by one or more clock cycles. As mentioned in Section 3.1, the delay is implemented by so-called isolators in the form of D-type flip-flops (DFFs). Figure 3.5 shows stochastic circuits to compute the square function $p_X \times p_X = p_X^2$ using an AND gate combined with (a) a single SNG and an isolator DFF, and (b) a pair of independent SNGs. Note that if the input SNs of the multiplier are not isolated or independently generated, the AND gate will not compute $(p_X)^2$, as demonstrated by Figure 3.5*b*.

Compared to regeneration, a stochastic isolator is much cheaper. However, the relative effect of the two randomization methods on accuracy is unclear, and does not appear to have been studied before. To gain insight into this question, we simulated the two squarer designs of Figure 3.5 using Octave and $N = 256$. The results in Table 3.1 indicate that the MSE of the isolator design (Figure 3.5*a*) is higher by a factor of 1.6 on average than that of the regeneration design (Figure 3.5*b*). This implies that the squarer is affected by auto-correlation in its input bit-stream X, i.e., the cross-correlation of X with itself at different

Table 3.1: Mean square error (MSE) of two stochastic squarer designs.

p_X	(MSE at output z) × 10^{-4}			
	Simulated results		Analytical results	
	Isolator design (Figure 3.5*a*)	Regeneration design (Figure 3.5*b*)	Equation (3.19)	$\frac{(p_X)^2(1-(p_X)^2)}{N}$
0.2	1.97	1.41	1.98	1.50
0.4	8.00	5.50	8.06	5.25
0.5	12.1	7.42	11.84	7.32
0.6	15.7	8.90	15.13	9.00
0.8	16.2	9.05	15.56	9.00

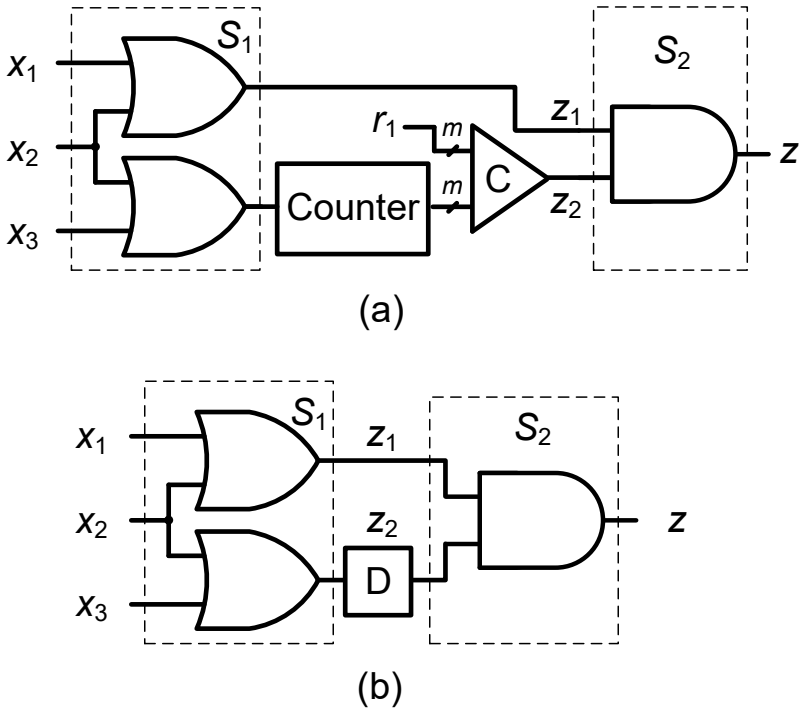

Figure 3.6: Reducing correlation in the circuit of Figure 3.4*a* (a) by regeneration, and (b) by isolation.

clock cycles. In this study, we consider the delayed signals as new SNs, so we allow correlation to include auto-correlation. The simulations also show that regeneration produces results very close to the analytical value for zero correlation.

As seen earlier, the accuracy of the circuit in Figure 3.4*a* is affected by correlation between Z_1 and Z_2 due to their shared input X_2. Figure 3.6 applies the two correlation-reducing methods to Z_1 and Z_2. Figure 3.6*a* uses regeneration with Z_2, while Figure 3.6*b* places an isolator in Z_2. Figure 3.7 shows the simulation results for the MSEs of all the circuits in Figure 3.4 and Figure 3.6. They indicate that stochastic isolation works just as well as regeneration. In fact, the isolator results are a little better, because random fluctuation errors accumulate when SNs are regenerated. In addition, the isolator's area cost is far less than that of regeneration. More importantly, the regenerating circuits require the computation to be paused to count the number of 1s, and the useful progressive precision property [2] will likely be lost.

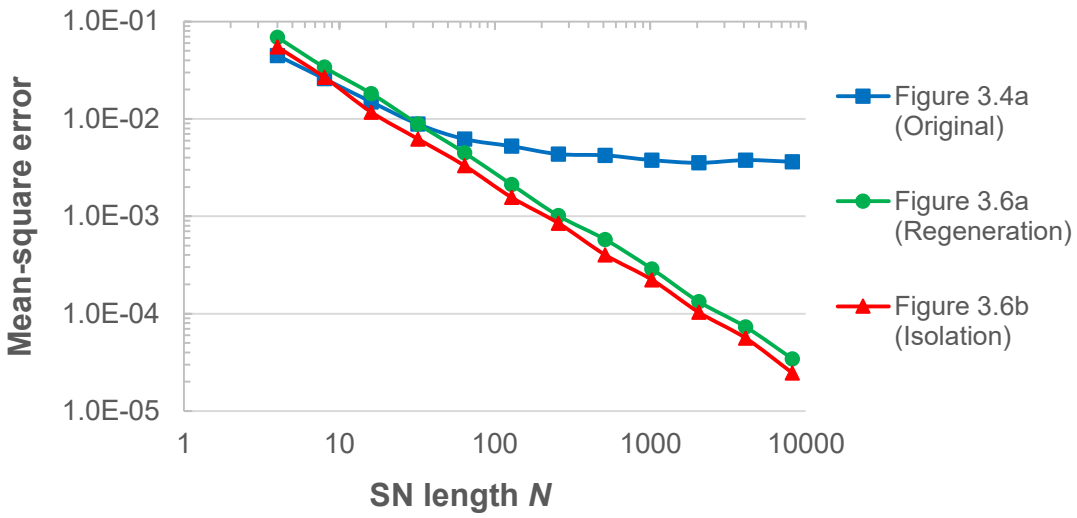

Figure 3.7: MSEs for the circuits of Figure 3.4*a* and Figure 3.6 obtained by simulation.

The squarer in Figure 3.5*a* generates a 1 when two consecutive 1s appear in the input bit-stream. Hence, the probability of seeing a 1 at the output z at clock cycle t depends on the output at $t-1$. The squaring operation can be represented by $z_t = x_{t-1} \times x_t$. This time dependency has a great impact on the MSE at z because our previous analysis assumes that the value of the SN Z has no time dependency. Consider a sequence of Bernoulli RVs X_1, X_2, …, X_N with parameter p_X and let $R = \sum_{t=2}^{N} X_{t-1}X_t$. Klotz [47] showed that the variance of R is

$$Var(R) = (N-1){p_X}^2(1-{p_X}^2) + 2{p_X}^3(1-p_X)\left[(N-2) - \frac{9}{(1-p_X)}\right] \tag{3.18}$$

Since the bit-stream length is $N-1$, the value of the SN Z is $R/(N-1)$. Therefore,

$$E_{Z^*} = \frac{{p_X}^2(1-{p_X}^2)}{N-1} + \frac{2{p_X}^3(1-p_X)}{(N-1)^2}\left[(N-2) - \frac{9}{1-p_X}\right] \tag{3.19}$$

Equation (3.19) is consistent with our simulation results.

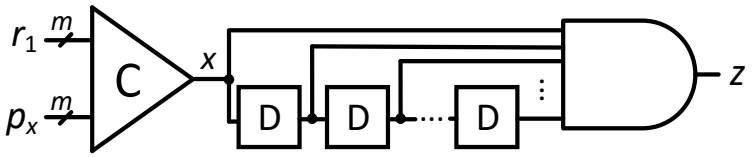

Figure 3.8: Stochastic circuit to generate $z = x^r$ with a single SNG.

When isolators are used to implement more general power functions such as x^3, x^4, etc., the error analysis becomes much more complicated. We therefore propose a simple way to find an upper bound on the MSE.

Theorem 3.4: Suppose a single SNG and $r - 1$ isolators are used to implement $z = x^r$ as shown in Figure 3.8. For large n, the MSE at z satisfies $E_{Z^*} < \sigma^2 \times r$, where σ^2 is the MSE when the function is implemented by r independent SNGs.

This theorem is easy to justify. For example, when $r = 2$, by substituting $SCC = 0$ into Equation (3.13), we can conclude that the MSE is inversely proportional to the SN length N when the inputs are uncorrelated. If we separate the output bit-stream into two sub-streams generated in even and odd clock cycles, we get $z_{2i} = x_{2i-1} \cdot x_{2i}$ and $z_{2i+1} = x_{2i} \cdot x_{2i+1}$ for $i = 1,2 \ldots, N/2$. Now z_{2i} and z_{2i+1} form two bit-streams of length $N/2$ that do not depend on their previous values. Since the output of a squarer is a combination of z_{2i} and z_{2i+1}, the worst-case MSE of the squarer is equivalent to that caused by reducing the SN length in half. Similarly, for the x^r function, the worst-case MSE is the same as reducing the SN length by $1/r$, which means the MSE is increased by a factor of r.

Table 3.2: MSEs of some power functions.

p_X	(MSE at z) $\times 10^{-4}$: simulation / upper bound)		
	x^2	x^3	x^4
0.2	0.520/ 0.75	0.117 / 0.233	0.0234 / 0.0624
0.4	2.17 / 2.62	1.04 / 1.76	0.520 / 0.974
0.5	3.29 / 3.66	2.13 / 3.20	1.35 / 2.29
0.6	4.01 / 4.5	3.67 / 4.96	2.87 / 4.41
0.8	4.22 / 4.5	6.69 / 7.32	8.36 / 9.45

Theorem 3.4 suggests that power functions are a worst case for MSE when isolators are used. In more typical cases like the example of Figure 3.4, correlation is caused by re-convergent signals and isolators can be effective because independent inputs that are not shared in common, such as x_1 and x_3, introduce randomness into later signals.

Table 3.2 summarizes the MSEs for three power functions implemented by a single SNG and one or more stochastic isolators. The results are generated by simulation for various input probabilities. The corresponding MSE upper bounds are computed using the inequality in Theorem 3.4, namely. $E_{Z^*} < \sigma^2 \times r$. The results show that the upper bounds provide good estimates of the corresponding MSEs.

3.5 Summary

This chapter has addressed a key problem of SC, namely, how to manage computational inaccuracy due to correlation. We showed that PTMs can play a very helpful role in the analysis. We developed a systematic method to compute the joint probability distribution of signals given their correlation. We also provided a method based on PTMs and the *SCC* correlation metric to evaluate the accuracy of a stochastic circuit. An approximation depending only on PTMs for quick estimation of accuracy was also derived to cover the complex case of correlation among multiple signals. Finally, the two most common correlation-reducing methods, regeneration and isolation, were evaluated, leading to the conclusion that stochastic isolators are effective for reducing correlation, thereby improving computational accuracy at relatively low cost.

CHAPTER 4
Design of General Stochastic Circuits

Chapters 2 and 3 analyze key factors affecting the accuracy of SC. We now move to another major challenge facing SC: development of a general framework for designing accurate stochastic circuits. Despite significant recent results, important aspects of SC's theoretical foundations remain to be discovered. We begin with the observation that every combinational stochastic circuit realizes a function of the form $f(X) = f(X_V;X_C)$, where X_V and X_C denote inputs with SNs of variable and constant probability, respectively. Two functions $f_1(X_V;X_C)$ and $f_2(X_V;X_C)$ are equivalent if they have the same stochastic behavior; this leads naturally to the notion of stochastic equivalence classes (SECs). We show that while conventional synthesis focuses on finding the best *circuit* to implement a given arithmetic function F, stochastic circuit optimization requires finding the best logic *function* f in its SEC that realizes F. We present an algorithm *ESECS* (Extended SEC-based Synthesis) to solve this problem, along with supporting experimental data. *ESECS* shows the computational richness of SC and leads to significant cost reductions compared to prior design methods. A preliminary version of this chapter's content appears in [24].

4.1 Stochastic Equivalence

Despite the successful application of stochastic circuits in several important domains, most previous SC designs are ad hoc and non-optimal, or else their optimality is undetermined. Recently, some general design methodologies have been proposed, which significantly enrich SC theory. Qian et al. present a general synthesis method employing a design style termed ReSC (Reconfigurable SC Architecture) [74], which is based on the

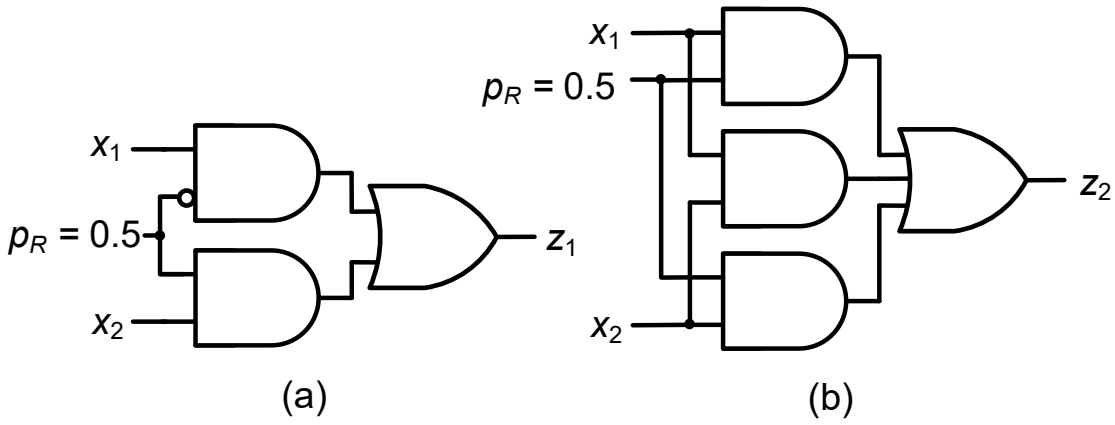

Figure 4.1: Two implementations of scaled addition: (a) a multiplexer (MUX), and (b) an equivalent majority circuit (MAJ).

theory of Bernstein polynomials [74]. Alaghi and Hayes describe a synthesis technique that exploits the relation between stochastic behavior and spectral transforms [1][7]. In [86], Zhao and Qian give another polynomial-based synthesis method. As we demonstrate later, these methods do not always lead to optimal designs.

The following interesting observation is made in [1]: two logic circuits can implement the same stochastic function even though their Boolean functions and design costs are quite different. Figure 4.1 illustrates this for scaled addition. The first circuit (Figure 4.1*a*) is the usual multiplexer design of Figure 1.3*b*, but the second (Figure 4.1*b*) realizes the majority function. However, when one input *r* has the constant SN 0.5 applied to it, both designs implement the same stochastic function $p_Z = 0.5\left(p_{X_1} + p_{X_2}\right)$. This unexpected property provides a new viewpoint on stochastic circuit optimization, which we explore in this chapter. In particular, we generalize the equivalence illustrated by Figure 4.1 to that of a stochastic equivalence class (SEC), investigate the properties of SECs, and apply them to the synthesis and optimization of stochastic circuits. We show that SECs define a rich set of arithmetic functions for building stochastic circuits. Unlike conventional synthesis methods where the target logic functions are fixed a priori, our approach searches the SECs for the best functions to meet the design goals.

We next recall some relevant properties of Boolean functions (BFs) [40]. The *weight* $w(f)$ of a BF $f(x_1, x_2, \dots, x_n)$ is the number of its minterms or, equivalently, $w(f) = f(0, \dots, 0,0) + \dots + f(1, \dots, 1,0) + f(1, \dots, 1,1)$. For example, the AND function $f(x_1, x_2) = x_1 \wedge x_2$ and the OR function $g(x_1, x_2) = x_1 \vee x_2$ have $w(f) = 1$ and $w(g) = 3$, respectively. Using the canonical disjunctive form, we can write

$$f(x_1, x_2, \dots, x_n) = \bigvee_{i=0}^{2^n-1} k_i \wedge m_i \tag{4.1}$$

where the k_i's are 0-1 constants called *discriminants* and the m_i's are *minterms* of the form $\tilde{x}_{i,1} \wedge \tilde{x}_{i,2} \wedge \dots \wedge \tilde{x}_{i,n}$, with $\tilde{x}_{i,j} = x_{i,j}$ or $\bar{x}_{i,j}$ denoting the j^{th} literal of m_i [6][40]. With this notation, $w(f)$ can be expressed as

$$w(f) = \sum_{i=0}^{2^n-1} k_i \tag{4.2}$$

where k_i's are the 0-1 constants in Equation (4.1). The different operator symbolism of Equations (4.1) and (4.2) stresses the fact that they specify Boolean and arithmetic functions, respectively.

A *k-literal cube* is an AND product of $k \leq n$ literals over $X = \{x_1, x_2, \dots, x_n\}$. For instance, $\bar{x}_1 \wedge x_2 \wedge x_3$ is a 3-literal cube. The positive cofactor $f_{x_i}(X)$ of f with respect to x_i is the BF $f(x_1, \dots, x_i, \dots, x_n)$ with x_i set to 1, i.e., $f(x_1, \dots, 1, \dots, x_n)$. Similarly, the negative cofactor $f_{\bar{x}_i}(X)$ is $f(x_1, \dots, 0, \dots, x_n)$. The *cofactor* f_c with respect to a cube c is the result of successive cofactorings of f with respect to all literals in c. For example, the cofactor of $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_1)$ with respect to the cube $c = x_1 \wedge \bar{x}_2$ is $f(1,0, x_3) = (1 \wedge 0) \vee (0 \wedge x_3) \vee (x_3 \wedge 1) = x_3$.

Traditionally, the *probability* p_F of $f(x_1, x_2, \dots, x_n)$ is defined as $w(f)/2^n$, which is the probability of $f = 1$ if the input vector $X = x_1 x_2 \dots x_n$ is randomly chosen from the 2^n possible input vectors [68]. This is often called "signal probability" in digital design. The

definition was originally motivated by random test-pattern selection, so it assumes that the probability of input x_i being 1 is 0.5. In stochastic circuits, inputs can be assigned arbitrary probability values. As introduced in Chapter 1, a stochastic number (SN) Z is a sequence of N bits, each of probability p_z, appearing on some line z. The value of Z is taken to be p_Z in the basic (unipolar) version of SC. Given Z, p_Z can be estimated by counting the 1s in Z. The accuracy of this estimate increases with the length of Z. In stochastic circuits like those of Figs. 1 and 2, each input x_i has probability p_{X_i} denoting the numerical value of some N-bit SN X_i, either a constant or a variable, being applied to that input line in a window of N clock cycles.

As each input x_i can carry an arbitrary probability value p_{X_i}, we can generalize the definition of the probability p_f as follows:

$$\begin{aligned} p_F(X) = {} & f(0,\dots,0,0)\left(1-p_{X_1}\right)\dots\left(1-p_{X_{n-1}}\right)\left(1-p_{X_n}\right) \\ & + f(0,\dots,0,1)\left(1-p_{X_1}\right)\dots\left(1-p_{X_{n-1}}\right)p_{X_n} + \cdots \\ & + f(1,\dots,1,1)p_{X_1}\dots p_{X_{n-1}}p_{X_n} \end{aligned} \tag{4.3}$$

When each $p_{X_i} = 0.5$, $p_F(X)$ is the (signal) probability $w(f)/2^n$.

Equation (4.3) specifies an n-variable *arithmetic* function of the form $F(X_1, X_2, \dots, X_n)$ whose inputs are n unipolar SNs and whose underlying *logic* function is $f(x_1, x_2, \dots, x_n)$. We refer to F as the *stochastic function* or *behavior* of f. For any input combination of SNs, F evaluates to $F\left(p_{x_1}, p_{x_2}, \dots, p_{x_n}\right) = p_F$. For example, when f is AND and $n = 2$, Equation (4.3) implies

$$\begin{aligned} p_F(X) = {} & f(0,0)\left(1-p_{X_1}\right)\left(1-p_{X_2}\right) + f(0,1)\left(1-p_{X_1}\right)p_{X_2} \\ & + f(1,0)p_{X_1}\left(1-p_{X_2}\right) + f(1,1)p_{X_1}p_{X_2} = p_{X_1}p_{X_2} \end{aligned} \tag{4.4}$$

demonstrating that AND implements stochastic multiplication. Equation (4.4) can also be concisely expressed as follows [6].

Theorem 4.1: The stochastic function realized by the Boolean function $f(x_1, x_2, \ldots, x_n)$ is

$$p_f(X) = \sum_{i=0}^{2^n-1} k_i M_i \tag{4.5}$$

where $M_i = \widetilde{M}_{i,1}\widetilde{M}_{i,2} \ldots \widetilde{M}_{i,n}$ and $\widetilde{M}_{i,j} = p_{x_{i,j}}$ if the corresponding minterm m_i of Equation (4.1) has $\tilde{x}_{i,j} = x_{i,j}$; $\widetilde{M}_{i,j}$ is $1 - p_{x_{i,j}}$ if $\tilde{x}_{i,j} = \bar{x}_{i,j}$.

If the SNs are interpreted as bipolar, then $f(x_1, x_2, \ldots, x_n)$ will implement a different arithmetic function $G\left(p_{x_1}, p_{x_2}, \ldots, p_{x_n}\right)$ because the numerical values of interest become $2p_{X_i} - 1$ instead of p_{X_i}. For instance, bipolar multiplication requires $F\left(p_{X_1}, p_{X_2}\right) = \left(2p_{X_1} - 1\right)\left(2p_{X_2} - 1\right)$, which is obtained from Equation (4.4) when *f* is XNOR. We will distinguish between unipolar and bipolar stochastic behavior as the need arises.

Stochastic circuits require randomness sources that can produce independent input bit-streams with prescribed probability values as discussed in Section 1.2. Figure 1.4a shows a typical stochastic number generator (SNG) that converts a *k*-bit binary integer *B* to a 2^k-bit stochastic bit-stream *X* with the value $p_X = B/2^k$. The bit-stream length $N = 2^k$ is selected based on accuracy considerations. The random (actually pseudo-random) number source is typically a linear feedback shift register (LFSR). LFSR sequences have a uniform distribution of 0s and 1s, so an LFSR provides a SN *R* of constant value p_R = 0.5. Constants of other probability values can be derived in various ways from *R* [38][73]. Prior SC work generally assumes that only random sources producing the constant value 0.5 are available as inputs to a stochastic circuit.

This chapter is concerned with the stochastic behavior of combinational logic functions and circuits intended for SC. We partition the *n* inputs *X* into two subsets: X_V denoting *s* variable inputs, and X_C denoting *t* constant inputs, where $s + t = n$. For example, the multiplier of Figure 1.3*a* has $X_V = (x_1, x_2)$ and $X_C = \varnothing$, whereas the adder of Figure

1.3*b* has $X_{\mathrm{V}} = (x_1, x_2)$ and $X_{\mathrm{C}} = r$ with $p_R = 0.5$. With the foregoing assumptions, we can now define the notion of the stochastic equivalence of Boolean functions.

Definition 4.1: Let $f(X) = f(X_{\mathrm{V}}; X_{\mathrm{C}})$ be a BF, where X_{V} and X_{C} partition X into variable and constant inputs, respectively. Let K denote a set of constant probability values assigned to X_{C}. Two BFs f and g defined on X are *stochastically equivalent* (SE) with respect to K, denoted $f \equiv_K g$, if $p_f(X_{\mathrm{V}}; K) = p_g(X_{\mathrm{V}}; K)$. When values in K are all 0.5, we simplify $f \equiv_K g$ to $f \equiv g$. □

Example 4.1: The circuits of Figure 4.1 have $X = (x_1, x_2, x_3)$ with $X_{\mathrm{V}} = (x_1, x_2)$ and $X_{\mathrm{C}} = (x_3) = (r)$, and realize the BFs $z_1 = x_1\bar{x}_3 + x_2x_3$ and $z_2 = x_1x_2 + x_2x_3 + x_3x_1$, (From here on, we adopt the more compact sum-of-products (SOP) notation for BFs.) Equation (4.5) with $p_{X_3} = 0.5$, makes $p_{Z_1} = p_{Z_2} = 0.5(p_{X_1} + p_{X_2})$, and $z_1 \equiv z_2$. In contrast, $p_{Z_1}(x_1, x_2; 0.25) \neq p_{Z_2}(x_1, x_2; 0.25)$, so $z_1 \not\equiv_K z_2$ where $K = (0.25)$. □

Definition 4.1 applies equally to the unipolar and bipolar cases, but the functions that are SE will be different. Zhao and Qian make use of a different type of equivalence among stochastic functions, based on a priori knowledge of the interchangeability of stochastic variables [86]. They note that when two or more stochastic variables have the same probability, they can be switched. For example, logic functions that realize stochastic functions $p_{z_1} = 0.5\left(p_{x_1} + \left(1 - p_{x_2}\right)\right)$ and $p_{z_1} = 0.5\left(\left(1 - p_{x_1}\right) + p_{x_2}\right)$ are equivalent when $p_{x_1} = p_{x_2}$.

Definition 4.1 also allows two Boolean functions to be SE with several sets K of X_{C} values. For example, $f = x_1r_1r_2 + x_2\bar{r}_1\bar{r}_2$ and $g = x_1\bar{r}_1\bar{r}_2 + x_2r_1r_2$ are BFs with $X_{\mathrm{V}} = (x_1, x_2)$ and $X_{\mathrm{C}} = (r_1, r_2)$. If $K = (0.5, 0.5)$, meaning $p_{r_1} = p_{r_2} = 0.5$, then f and g are SE, since $p_f = p_g = 0.25(p_{x_1} + p_{x_2})$. If $K = (0.25, 0.75)$, then f and g are still SE, in this case, with $p_f = p_g = 0.1875(p_{x_1} + p_{x_2})$. It's possible to generalize Definition 4.1 to allow BFs to be SE even if their K's have different sizes. For simplicity, as well as

consistency with both the prior literature and the nature of LFSR-based random sources discussed above, we will assume throughout that SE BFs have *K*'s of the same size and that all *K* values are 0.5.

If $X_V = \varnothing$, then *f* produces an output of constant probability determined by *K*. If $X_C = \varnothing$, then $f \equiv g$ only when $f = g$. To check for stochastic equivalence when $X_C \neq \varnothing$, we make use of the following result.

Theorem 4.2: Two BFs f and g defined on $X = X_V; X_C$ are SE if and only if $w\left(f_{c_i}(X)\right) = w\left(g_{c_i}(X)\right)$ for all *s*-literal cubes c_i on X_V, where $s = |X_V|$.

Proof: Let $X_V = (x_1, x_2, \dots, x_s)$ and $X_C = (x_{s+1}, x_{s+2}, \dots, x_{s+t})$. Let M_f and M_g be the sets of minterms that define f and g, respectively. Let C_V be the set of all *s*-literal cubes on X_V, and let C_C be all *t*-literal cubes on X_C. Let c_i denote an *s*-literal cube on X_V, and let c_j denote an *t*-literal cube on X_C. Let p_{c_i} be the probability of cube c_i and $f_{c_i}(X)$ be the cofactor of f with respect to c_i. Since $p_{x_i} = 0.5$, for all $x_i \in X_C$, $p_{c_j} = \prod_{x_i \in X_C} p_{x_i} = 0.5^t$ for all $c_j \in C_C$. Therefore,

$$p_f(X) = \sum_{c_i \in C_V; c_j \in C_C; c_i \cdot c_j \in M_f} p_{c_i} \times p_{c_j} = \sum_{c_i \in C_V} p_{c_i} \times \sum_{c_j \in C_C; c_i \cdot c_j \in M_f} p_{c_j}$$
$$= \sum_{c_i \in C_V} p_{c_i} \times 0.5^t \times w\left(f_{c_i}(X)\right)$$

Similarly,

$$p_g(X) = \sum_{c_i \in C_V} p_{c_i} \times 0.5^t \times w\left(g_{c_i}(X)\right)$$

(1) *If*: To show that $w\left(f_{c_i}(X)\right) = w\left(g_{c_i}(X)\right)$ implies $f \equiv g$.

$$p_f(X) = \sum_{c_i \in C_V} p_{c_i} \times 0.5^t \times w\left(f_{c_i}(X)\right) = \sum_{c_i \in C_V} p_{c_i} \times 0.5^t \times w\left(g_{c_i}(X)\right) = p_g(X)$$

so by definition, $f \equiv g$.

(2) *Only if*: To show that if $f \equiv g$, then $w\big(f_c(X)\big) = w\big(g_c(X)\big)$. Let $p_f \equiv p_g$. Since

$$p_f(X) = \sum_{c_i \in C_V} p_{c_i} \times 0.5^t \times w\left(f_{c_i}(X)\right)$$

and

$$p_g(X) = \sum_{c_i \in C_V} p_{c_i} \times 0.5^t \times w\left(g_{c_i}(X)\right)$$

we conclude that

$$\sum_{c_i \in C_V} p_{c_i} \times w\left(f_{c_i}(X)\right) = \sum_{c_i \in C_V} p_{c_i} \times w\left(g_{c_i}(X)\right)$$

Each $c_i \in C_V$ is an *s*-literal cube implying it contains exactly *s* literals, so no c_i can be covered by any other cube c in C_V. Furthermore, no c_i can be covered by $\bigcup_{c \in C_V, c \neq c_i} c$, which means c_i is not covered by the union of other cubes. This property implies p_{c_i} cannot be generated by a linear combination of p_c's for $c \neq c_i$. Since each p_{c_i} is unique, $w\left(f_{c_i}(X)\right) = w\left(g_{c_i}(X)\right)$. □

The relation $f\,(X_V; X_C) \equiv g(X_V; X_C)$ for some $X_V; X_C$ defines an equivalence relation that partitions the BFs into classes we call *stochastic equivalence classes* (SECs).

Example 4.2: Continuing Example 4.1, z_2 has $|X_V| = s = 2$, and the corresponding 2-literal cofactors are $z_{2_{\bar{x}_1\bar{x}_2}}(X) = 0$, $z_{2_{\bar{x}_1 x_2}}(X) = x_3$, $z_{2_{x_1\bar{x}_2}}(X) = x_3$ and $z_{2_{x_1 x_2}}(X) = 1$, with

weights 0, 1, 1 and 2, respectively. Note that $z_{2_{x_1\bar{x}_2}}(X) = 1$ has the weight 2 because $z_{2_{x_1\bar{x}_2}}(X)$ has only one input x_3 and both $z_{1_{x_1\bar{x}_2}}(x_3 = 0)$ and $z_{1_{x_1\bar{x}_2}}(x_3 = 1)$ are 1. The cofactors of z_1 are $z_{1_{\bar{x}_1\bar{x}_2}}(X) = 0$, $z_{1_{\bar{x}_1 x_2}}(X) = x_3$, $z_{1_{x_1\bar{x}_2}}(X) = \bar{x}_3$ and $z_{1_{x_1 x_2}}(X) = 1$ with weights 0, 1, 1 and 2, respectively. Hence, $w\left(z_{1_c}(X)\right) = w\left(z_{2_c}(X)\right)$ for all 2-literal cubes on X_V, implying that $z_1 \equiv z_2$. These are two members of an SEC $\mathbb{F}_{ADD}$ that turns out to have a total of four members. □

Clearly, an SEC $\mathbb{F}$ can be characterized by the stochastic function $p_F(X)$ common to all its members. With the input partition $X = X_V; X_C$, $n = s + t$, and $p_{X_i} = 0.5$ for all constant inputs, Equation (4.5) becomes:

$$p_F(X) = \sum_{i=0}^{2^n-1} k_i M_i = \sum_{i=0}^{2^n-1} k_i \widetilde{M}_{i,1}\widetilde{M}_{i,2} \ldots \widetilde{M}_{i,n}$$
$$= \sum_{i=0}^{2^{s+t}-1} \left(k_i \widetilde{M}_{i,1}\widetilde{M}_{i,2} \ldots \widetilde{M}_{i,s}\right)\left(\widetilde{M}_{i,s+1} \ldots \widetilde{M}_{i,s+t}\right)$$

which implies

$$p_{\mathbb{F}}(X) = p_F(X) = 0.5^t \sum_{i=0}^{2^{s+t}-1} k_i \widetilde{M}_{i,1}\widetilde{M}_{i,2} \ldots \widetilde{M}_{i,s} \tag{4.6}$$

because $\widetilde{M}_{i,s+1} = \cdots = \widetilde{M}_{i,s+t} = 0.5$. When multiplied out, Equation (4.6) takes the form of a multilinear polynomial (MLP) in the $p_{X_{i,j}}$'s. (A polynomial is *multilinear* if it is linear in each of its variables).

Example 4.3: To illustrate, consider the scaled add functions of Example 4.2 with MUX z_1 representing their SEC $\mathbb{F}_{ADD}$. Writing z_1 as a sum of minterms (Equation (4.1)), we get

$$z_1 = x_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 x_3 + \bar{x}_1 x_2 x_3$$

With $X_C = x_3$, and $p(x_3) = p(\bar{x}_3) = 0.5$, Equation (4.6) implies

$$p_{\mathbb{F}_{\text{ADD}}}(X) = p_{z_1} = 0.5\left(p_{X_1}p_{X_2} + p_{X_1}(1-p_{X_2}) + p_{X_1}p_{X_2} + (1-p_{X_1})p_{X_2}\right)$$
$$= 0.5p_{X_1} + 0.5p_{X_2}$$

which is scaled addition in its usual MLP form. □

Theorem 4.1 and Theorem 4.2 suggest a way to express an SEC $\mathbb{F}$ in terms of its members' common weight sets.

Theorem 4.3: Let $\mathbb{F}$ be an SEC defined on $X = X_{\text{V}}; X_{\text{C}}$ where $s = |X_{\text{V}}|$ and $t = |X_{\text{C}}|$. Then $p_{\mathbb{F}}(X)$ is given by

$$p_{\mathbb{F}}(X) = 0.5^t \sum_{i=0}^{2^s-1} w_i \widetilde{M}_{i,1}\widetilde{M}_{i,2} \dots \widetilde{M}_{i,s} \tag{4.7}$$

where $w_i = w\left(f_{c_i}(X)\right)$ and the $\widetilde{M}_{i,j}$'s are as defined as in Theorem 4.1, but with respect to X_{V} rather than X.

Proof: Rewriting Equation (4.6), yields

$$\begin{aligned} p_{\mathbb{F}}(X) &= 0.5^t \sum_{i=0}^{2^s-1} \sum_{j=0}^{2^t-1} k_{i^*} \widetilde{M}_{i^*,1}\widetilde{M}_{i^*,2} \dots \widetilde{M}_{i^*,s} \\ &= 0.5^t \sum_{i=0}^{2^s-1} \left(\sum_{j=0}^{2^t-1} k_{i^*} \right) \widetilde{M}_{i^*,1}\widetilde{M}_{i^*,2} \dots \widetilde{M}_{i^*,s} \end{aligned} \tag{4.8}$$

where $i^* = i \cdot 2^t + j$. $\mathbb{F}$'s member f has weights of the form $w_i = w\left(f_{c_i}(X)\right) = \sum_{j=0}^{j=2^t-1} k_{i^*}$ where k_{i^*} is the i^*-th 0-1 coefficient for f defined by Equation (4.1). Also, the $\widetilde{M}$'s in Equation (4.8) are as defined in Theorem 4.1 with respect to X. Therefore, Equation (4.8) reduces to (4.7) where $\widetilde{M}_{i,j}$'s are defined with respect to X_{V} rather than X. □

Theorem 4.3 provides a canonical representation of an SEC $\mathbb{F}$. We refer to $p_{\mathbb{F}}(X)$ as $\mathbb{F}$'s *characteristic function* and Equation (4.7) as its *characteristic equation.* Continuing Examples 4.2 and 4.3, Equation (4.7) specifies $\mathbb{F}_{\text{ADD}}$ as follows:

$$p_{\mathbb{F}_{ADD}}(X) = 0.5\big(0 \cdot \big(1 - p_{x_1}\big)\big(1 - p_{x_2}\big) + 1 \cdot p_{x_1}\big(1 - p_{x_2}\big) + 1 \cdot \big(1 - p_{x_1}\big)p_{x_2} + 2 \cdot p_{x_1}p_{x_2}\big) = 0.5p_{x_1} + 0.5p_{x_2}$$

Observe that Equations (4.6) and (4.7) express a stochastic function in two distinct ways: one derived from the discriminants, i.e., the k_i's, and minterms of X, and the other derived from the weights and s-literal cubes of X, which are minterms of X_{V}.

Because different Boolean functions in the same SEC $\mathbb{F}$ have different discriminants, Equation (4.6) which uses discriminants does not provide a canonical representation for $\mathbb{F}$. Equation (4.7) is canonical, however, and enables efficient stochastic equivalency checking.

Theorem 4.4: The Boolean functions defined on $X = X_{\text{V}}; X_{\text{C}}$ where $|X_{\text{V}}| = s$ and $|X_{\text{C}}| = t$, have $(2^t + 1)^{2^s}$ SECs.

Proof: Since $|X_{\text{V}}| = s$, there are 2^s s-literal cubes on $|X_{\text{V}}|$, each corresponding to a cofactor $f_c(X)$. From Theorem 2, we know that two functions f and g are stochastically equivalent if and only if $w\big(f_c(X)\big) = w\big(g_c(X)\big)$ for all c. Now consider the possible values of $w\big(f_c(X)\big)$. Since $|X_{\text{c}}| = t$, we have $w\big(f_c(X)\big) \in \{0,1, \dots, 2^t\}$. Therefore, $w\big(f_c(X)\big)$ has $2^t + 1$ possible values. Since there are 2^s s-literal cubes, their weights can be any integers between 0 to 2^t. There are $(2^t + 1)^{2^s}$ possible combinations of weights, implying $(2^t + 1)^{2^s}$ SECs. □

Theorem 4.5: The size of an SEC is $\prod_{i=0}^{2^s-1} \binom{2^t}{w_i}$.

Proof: Since $|X_V| = s$, $|X_V|$has 2^s *s*-literal cubes c_i, each corresponding to a cofactor $f_{c_i}(X)$. The number of combinations of the possible cofactors with respect to c_i is $\binom{2^t}{w_i}$ because c_i is a *t*-input Boolean function with weight w_i. Therefore, the total number of combinations of stochastically equivalent functions, i.e., the size of the SEC $\mathbb{F}$, is $\prod_{i=0}^{2^s-1}\binom{2^t}{w_i}$. □

Clearly, the maximum size of an SEC is $\binom{2^t}{2^{t-1}}^{2^s}$. If $w(f_c(X)) = k$ for some *s*-literal cube *c*, the number of possible f_c's is $\binom{2^t}{k}$. Since the maximum value of $\binom{2^t}{k}$ is $\binom{2^t}{2^{t-1}}$, the largest SEC is the class of all *s*-literal cubes *c* on $|X_V|$ with weight $w(f_c(X))$ is 2^{t-1}. Hence, there will be $\binom{2^t}{2^{t-1}}$ combinations for all *s*-literal cubes, and $\binom{2^t}{2^{t-1}}^{2^s}$ SE functions.

Example 4.4: Consider the 16 two-variable functions $f(X) = f(x_1; r_1)$ with $X_V = x_1$ and $X_c = r_1$. They form nine SECs and the size of the largest class is $\binom{2^1}{2^{1-1}}^{2^1} = 4$. All the SECs and their stochastic behavior for both the unipolar and bipolar formats are listed in Table 4.1. The stochastic behaviors can be interpreted as arithmetic functions in various ways, some of which are potentially useful. For example, the class-2 functions implement $F = 0.5p_{X_1}$, which can be seen as multiplication by 0.5 or division by 2. This *F* also performs the scaling operation seen throughout Table 4.1, as well as in stochastic addition (Figure 4.1). □

The preceding example illustrates the fact that *every* logic function, including the simplest kind, implements one or more non-trivial arithmetic operations that may be exploited in SC design. A class-8 function in Table 4.1, for instance, performs three elementary arithmetic operations on bipolar SNs: decrement (subtract 1), divide by 2, and

change the sign. Remarkably, the SC hardware for this consists of just an OR gate and an inverter.

Table 4.1 reveals some other interesting properties of SECs. First, functions in the same class can be generated by replacing some or all variables in X_C by their complements, or by other variables in X_C. This is to be expected because all their probabilities will be the same, namely 0.5. Second, a new SEC is generated by complementing all the functions in a given SEC. For example, functions in class 2 are the complements of those in class 8. We also see that the largest SEC must contain the degenerate one-variable functions $f(X) = r_i$, where $r_i \in X_C$.

Table 4.1: All SECs for the 2-variable logic functions $f(x_1; r_1)$ with $X_V = x_1$ and $X_C = r_1$.

Class	Size	Logic functions *f*	Stochastic behavior *F*		Arithmetic interpretation *A*	
			Unipolar *p*	Bipolar $2p - 1$	Unipolar	Bipolar
1	1	0	0	−1	Constant 0	Constant −1
2	2	x_1r_1, x_1r_1'	$0.5p_{x1}$	$0.5[(2p_{x1} - 1) - 1]$	Scale = Multiply by 0.5	Decrement and scale
3	1	x_1	p_{x1}	$2p_{x1} - 1$	Identity	Identity
4	2	$x_1'r_1$, $x_1'r_1'$	$0.5(1 - p_{x1})$	$-0.5[(2p_{x1} - 1) + 1]$	Complement and scale	Increment, scale and negate
5	4	r_1, $x_1'r_1 + x_1r_1'$, $x_1'r_1' + x_1r_1$, r_1'	0.5	0	Constant 0.5	Constant 0
6	2	$x_1 + r_1$, $x_1 + r_1'$	$0.5(p_{x1} + 1)$	$0.5[(2p_{x1} - 1) + 1]$	Increment and scale	Increment and scale
7	1	x_1'	$1 - p_{x1}$	$-(2p_{x1} - 1)$	Complement (with respect to 1)	Negate
8	2	$x_1' + r_1$, $x_1' + r_1'$	$1 - 0.5p_{x1}$	$-0.5[(2p_{x1} - 1) - 1]$	Scale and complement	Decrement, scale and negate
9	1	1	1	1	Constant 1	Constant 1

4.2 SEC-based Synthesis

As Table 4.1 illustrates, an SEC identifies a set of logic functions $\mathbb{F}$ that have the same stochastic behavior or, equivalently, implement the same arithmetic function A. A basic question in stochastic circuit synthesis is therefore: Given a desired arithmetic function A, what is the corresponding SEC $\mathbb{F}$ whose stochastic behavior is A? Prior design methods like those of [1][7][74] try to determine just one member f of $\mathbb{F}$, and then focus on f's implementation and optimization in some preferred design style.

We now address a more fundamental question: Which of the many functions $\{f_i\}$ in $\mathbb{F}$ are most likely to lead to optimal designs in terms of area cost? We make the following general assumptions:

1. A cell library L is available that includes components needed to implement A. Any functionally complete set of logic gates will do, but other well-defined arithmetic components such as MUX are helpful. For bipolar SC, a basic L might include NOT (for negation), OR (for logical completeness with NOT), XNOR (for multiplication), and MUX (for scaled addition).

2. Area cost is measured *literal count* + *term count* in an (optimal) SOP expression for each f_i. This cost measure is widely used in practice, for example by the two-level optimization program *espresso* [14]. While area-optimal stochastic circuits are not necessarily two-level, they have few levels, an important consideration when processing long bit-streams. Fewer levels are also desirable because of the signal correlation build-up occurring as bit-streams propagate through multiple levels.

Our preliminary method *SECS* (SEC-based Synthesis) for stochastic circuit design was presented in [24]. It requires an initial Boolean function f to identify the SEC for a given A. Several approaches to finding f are considered by SECS, including *direct mapping,* which tries to map an expression E for A directly to a stochastic circuit C using components from L. This process is analogous to technology mapping in conventional logic

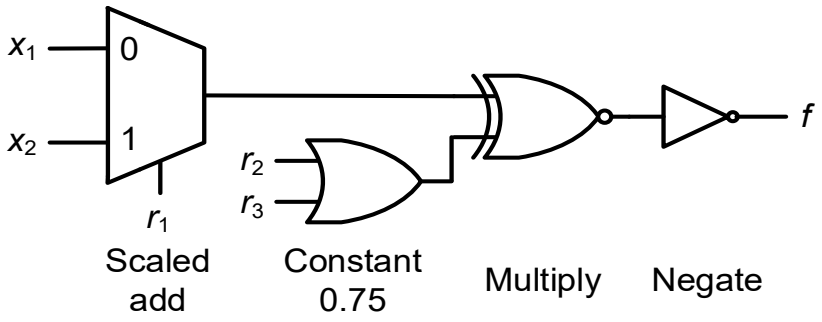

Figure 4.2: Direct-mapped implementation of Example 4.5.

design. Since it is composed of standard logic elements, it can be analyzed by Boolean algebra to determine its output function *f*. Note, however, that not every arithmetic function has a corresponding *f* [7].

Example 4.5: Suppose we are given the target arithmetic expression $E = -0.25(X_1 + X_2)$, the cell library L = {NOT, OR, XNOR, MUX}, and employ bipolar SNs. E maps directly into the circuit of Figure 4.2, which specifies the initial Boolean function of the form $f(x_1, x_2; r_1, r_2, r_3)$. The MUX computes $0.5(X_1 + X_2)$, which the XNOR then multiplies by 0.5. The bipolar form of 0.5 requires a p_x value satisfying $2p_x - 1 = 0.5$, i.e., $p_x = 0.75$. This is provided by the OR gate, whose stochastic behavior is $p_{r_2} + p_{r_3} - p_{r_2}p_{r_3}$. Finally, the NOT gate changes the sign. From Figure 4.2 we get

$$f = x_1\bar{r}_1\bar{r}_2\bar{r}_3 + x_2r_1\bar{r}_2\bar{r}_3 + \bar{x}_1\bar{r}_1r_2 + \bar{x}_2r_1r_2 + \bar{x}_1\bar{r}_1r_3 + \bar{x}_2r_1r_3 \qquad (4.9)$$

which has cost 26. Later we will see that *f*'s SEC contains functions of much lower cost than *f*. □

The *SECS* algorithm [24] maps a given arithmetic function *A* to a logic function *f** that has a low-cost implementation of *A*. It can use essentially any SC synthesis method, such as the direct mapping approach of Example 4, to find an initial or "base" *f* with the desired stochastic behavior *A*. After *f* is found, its weights are calculated to identify its SEC $\mathbb{F}$. Then *SECS* systematically explores $\mathbb{F}$ for equivalent functions of lower cost using a search-based procedure *SECO* (SEC-based Optimization). If the SEC is small enough,

1. Given an arithmetic function $A(X_1,X_2,...,X_s)$, identify some SEC $\mathbb{F}$ by either of the following methods:
   (A) Generate a base stochastic circuit implementing an SEC-member f by direct mapping, spectral transformation (STRAUSS), ReSC, etc. Then compute the $w_i = w(f_{ci})$ for all s-literal cubes c_i of f.
   (B) *SECI*: Use the first step of STRAUSS to format the target function A as an MLP $F(X_1,X_2,...,X_s)$ and then compute the weights w_i's directly.
   $\mathbb{F}$ is defined by these $w(f_{ci})$'s.

2. Determine an optimal member of $\mathbb{F}$ by either of the following two procedures:
   (A) *SECO*: Using Theorem 4.2, search for a set of (possibly all) representative functions g that are stochastically equivalent to f. Evaluate the cost of each g via a logic optimization tool such as *espresso*. Retain (and eventually return) f^*, a lowest-cost g.
   (B) *SECM*: Divide the stochastic function optimization problem into two steps. First, reduce the literal count in X_V by mapping the problem to finding an exact cover with minimal cost. Then use a vertex-coloring algorithm to find and assign stochastic constants.

Figure 4.3: Overview of the extended SEC-based algorithm *ESECS* to determine a low-cost stochastic circuit.

SECO searches the entire SEC. The implementation cost of each function examined by *SECO* can be measured by any convenient logic optimization tool; we chose *espresso* [14].

We now present an enhanced version of *SECS* called *ESECS* (Extended SECS), which is summarized in Figure 4.3. *ESECS* is the same as SECS except for the addition of the new procedures *SECI* and *SECM* in Steps 1(B) and 2(B), respectively. *SECI* (Stochastic Equivalence Class Identification) allows *ESECS* to find an SEC directly without generating a base design f; see Figure 4.4. The target arithmetic function $A(X_V)$ is first approximated by an MLP $F(X_V)$. This is done by using a Taylor series expansion for $A(X_V)$ and replacing non-linear terms by linear terms [7]. Next, using Theorem 4.3, *SECI* calculates weights $w_i^* = w^*(f_{c_i}) = a \cdot F(K_i)$ where a is the smallest positive integer such that all coefficients in $F^*(X_1, X_2, \dots, X_s) = a \cdot F(X_1, X_2, \dots, X_s)$ are integers. K_i is a set of 0-1 constants $(k_{i,1}, k_{i,2}, \dots, k_{i,t})$, where $k_{i,j} = 1$ if the corresponding j-th literal in cube c_i is

1. Given an MLP $F(X_V)$ where $X_V = x_1, x_2, ..., x_s$, find a smallest positive integer a such that all coefficients in $F^*(X_V) = a \cdot F(X_V)$ are integers.
2. Calculate $w_i^* = w^*(f_{ci}) = F^*(K_i)$ where K_i is a set of 0-1 constants $K_i = (k_{i,1}, k_{i,2}, \ldots, k_{i,t})$. $k_{i,j} = 1$ if the corresponding j-th literal in cube c_i is $x_{i,j}$; $k_{i,j}$ is 0 when the literal is $x'_{i,j}$.
3. Repeat step 2 for all w_i^*'s. If the minimum value of w_i^*'s is negative, shift the values by $w_i = w_i^* - \min(w_i^*)$ to make sure all weights are positive or zero. Otherwise, $w_i = w_i^*$.
4. The SEC $\mathbb{F}$ is defined by w_i's, $|X_V| = s$ and $|X_C| = t$ where s is the numbers of A's inputs and t is the smallest integer such that $2^t \geq \max(w_i)$.

Figure 4.4: Overview of *SECI* used by *ESECS* to identify an SEC for a given MLP.

$x_{i,j}$; $k_{i,j}$ is 0 when the literal is $\bar{x}_{i,j}$. We also need to make sure all the weights are nonnegative integers. This is done by shifting the values of w_i^*'s by adding $b = |\min(w_i)|$ when $\min(w_i) < 0$ and $b = 0$, otherwise. The resulting SEC realizes $0.5^t \cdot (a \cdot F(X_1, X_2, \ldots, X_s) + b)$. Shifting and scaling are necessary for SC to process numbers that are negative or outside the unit interval, respectively.

Example 4.6: Consider the arithmetic function $A(X_1, X_2)$ with MLP $F_A(X_1, X_2) = 0.3X_1 - 0.4X_2$. With $a = 10$, we have $F^*(X_1, X_2) = 3X_1 - 4X_2$. In addition, $w_0^* = 0$, $w_1^* = -4$, $w_2^* = 3$, and $w_3^* = -1$. Since the minimum value of w_i is –4, we shift the weights by $b = |-4|$ to get $w_0 = 4$, $w_1 = 0$, $w_2 = 7$, and $w_4 = 3$. Equation (4.7) then implies that

$$p_{F^*} = 0.5^3\left(4\left(1 - p_{X_1}\right)\left(1 - p_{X_2}\right)\right) + 7\left(p_{X_1}\left(1 - p_{X_2}\right) + 3p_{X_1}p_{X_2}\right)$$
$$= 0.5^3 \cdot \left(3p_{X_1} - 4p_{X_2} + 4\right)$$ □

As summarized in Figure 4.3, *ESECS* takes an arithmetic function as its input. The minimization methods used in conventional CAD tools such as *espresso* cannot find the optimal design directly because the inputs of these CAD tools are Boolean functions, not arithmetic functions or stochastic equivalence classes. However, as indicated in Figure 4.3,

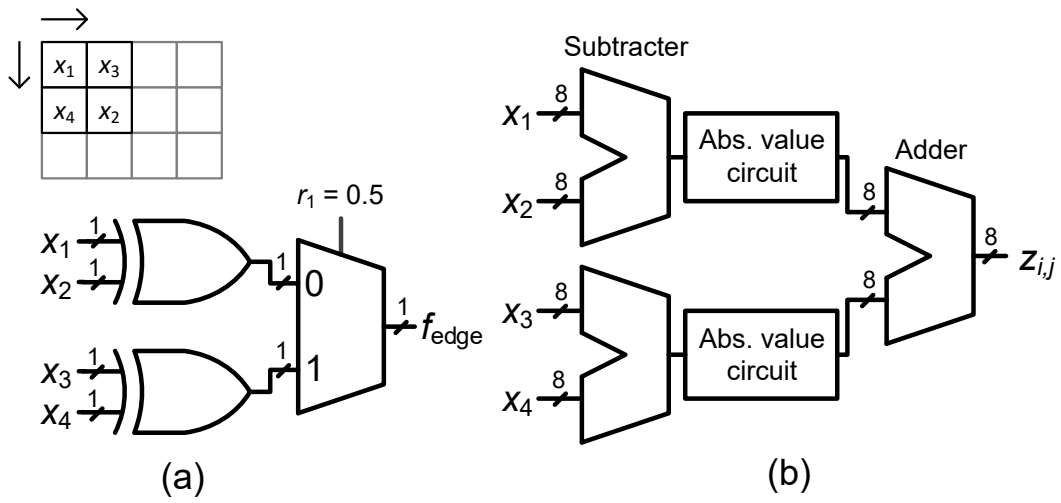

Figure 4.5: Edge detectors: (a) stochastic, (b) conventional [3].

the conventional tools can be used in Step 2A to evaluate and weed out larger designs in the SEC.

Before we turn to the general task of optimizing a new stochastic function, we consider the problem of checking the optimality of a known design implementing some function $f(X_V; X_C)$. The idea is to examine all BFs stochastically equivalent to f and compare their costs. This is feasible when $n = s + t$ is small, or when f is amenable to proof by induction.

Example 4.7: The stochastic circuit in Figure 4.5*a* is taken from [3], and implements the Roberts Cross function for edge detection in images using a four-pixel window. It computes the function

$$Z_{i,j} = 0.5 \times (|X_1 - X_2| + |X_3 - X_4|) \tag{4.10}$$

provided the input SNs meet certain correlation conditions; these conditions do not affect SEC membership. As noted in Section 2.6, this stochastic edge detector requires x_1 and x_2 to have maximum overlapping of 1s so that the XOR gate serves as an absolute-subtracter $|X_1 - X_2|$. Similarly, x_3 and x_4 need to have maximum overlapping of 1s. *ESECS* makes no

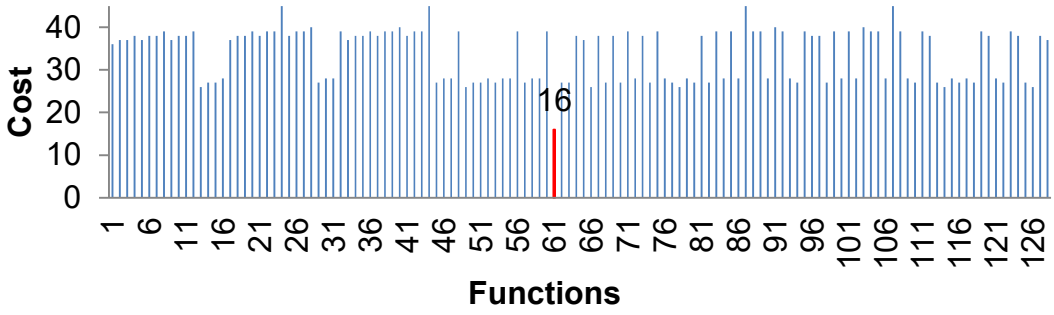

Figure 4.6: Cost of the equivalent edge-detector functions.

assumptions about input correlation, although in most SC circuits, primary inputs are required to be uncorrelated. *ESECS* works on both correlated and uncorrelated inputs. The stochastic edge detector's area cost is less by a factor of about 100 than that of the conventional, non- stochastic implementation of Equation (4.10) in Figure 4.5*b.* Note that the total number of logic inputs is only five, far less than the 32 or so in Figure 4.5*b*. Also, the stochastic design has far fewer logic levels. The BF realized by the stochastic design has the optimal SOP form,

$$f_{\text{edge}} = \bar{x}_1 x_2 \bar{r}_1 + x_1 \bar{x}_2 \bar{r}_1 + \bar{x}_3 x_4 r_1 + x_3 \bar{x}_4 r_1$$

and has cost 16. The variable inputs x_1, x_2, x_3 and x_4 denote light intensity and define X_V. The remaining input r_1 is a constant of probability 0.5 and defines X_C. The SEC for f_{edge} contains 256 functions. However, we only need to evaluate half of them because the cost of $g \equiv f$ is the same as that of f when g is formed by complementing inputs in X_C. For example, $f_1 = x_1\bar{r}_1 + x_2 r_2$ and $f_2 = x_1 r_1 + x_2 \bar{r}_1$ with $X_C = (r_1)$ are SE, and f_2 is formed by complementing r_1 in f_1. Since f_1 and f_2 have the same structure, their literal cost is the same, and we only need to evaluate one of them. Figure 4.6 shows the cost of these 128 functions as computed by *ESECS*. Since the lowest cost is 16, this experiment confirms the optimality of the original edge-detector design. □

We performed a similar optimality check on the special SNG called a weighted binary generator (WBG) presented in [38]. Like the conventional SNG of Figure 1.4*a*, the

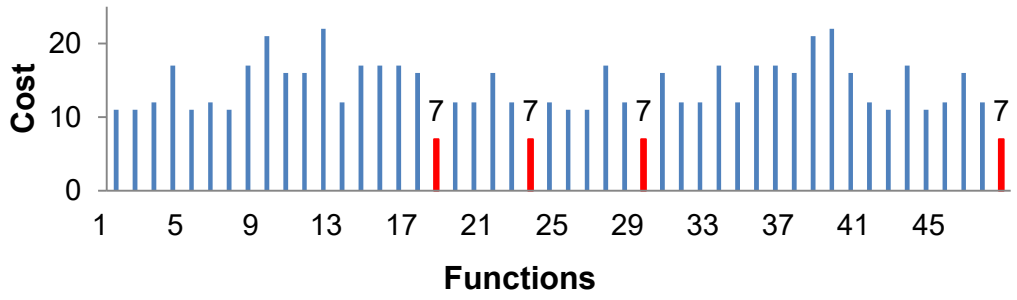

Figure 4.7: Cost of the equivalent 2-bit WBG functions.

WBG in Figure 5.2 maps a k-bit binary number $x_1x_2 \ldots x_k$ to a 2^k-bit SN $F_{\mathrm{WBG}k}$. Unlike the conventional SNG, however, which directly compares X with a k-bit pseudorandom number R, a WBG composes the output SN from k non-overlapping SNs $W_1, W_2, \ldots, W_k$ derived from R that have probability $p_{W_i} = 0.5^i$, and ORs them to form the sum $p_{F_{\mathrm{WBG}k}} = \sum_{i=1}^{k} 0.5^i p_{X_i}$. The WBG has the advantage of generating an SN whose value is exactly that of X. A detailed discussion of the WBG design will be provided later in Sections 5.1 and 6.1.

Example 4.8: A 2-bit WBG implements $f_{\mathrm{WBG2}} = x_1r_1 + x_2\bar{r}_1r_2$, where $x_1, x_2 \in X_V$ and $r_1, r_2 \in X_C$. The cost of f_{WBG2} is 7. Figure 4.7 shows the results of searching (half) the corresponding SEC. It confirms the optimality of the original design, but it also finds alternative functions that have the same cost, such as $f^*_{WBG2} = x_1r_1 + x_2\bar{r}_1\bar{r}_2$. These functions are obtained by permuting or complementing variables in X_C. □

In the examples so far, we have been able to exhaustively search the SEC defined by a small base function. We now describe two ways to handle larger functions: induction when the target function has a well-defined recursive form, and guided search for less structured cases. Suppose a BF can be expressed as

$$f_s(x_1, x_2, \ldots, x_s; X_C) = f_{s-1}(x_1, x_2, \ldots, x_{s-1}; X_C) + h_s(x_s; X_C)$$

We can try to prove the optimality of small cases and then generalize them to larger ones. *ESECS* may be used to check the optimality of $f_{s\text{-}1}$ and h_s. If both are optimal, f_s must be optimal as the cost of f_s is the sum of the costs of $f_{s\text{-}1}$ and h_s.

To illustrate, consider the class of *k*-bit WBGs. We showed by exhaustive SEC search that $f_{\mathrm{WBG}2}$ is an optimal function. Assume $f_{\mathrm{WBG}n}$ is also optimal when $k = n$. Then

$$f_{\mathrm{WBG}n} = x_1 r_1 + x_2 \bar{r}_1 r_2 + x_3 \bar{r}_1 \bar{r}_2 r_3 + \cdots + x_n \bar{r}_1 \bar{r}_2 \ldots \bar{r}_{n-1} r_n$$

and let its cost be $cost_n$. For $k = n + 1$, we have

$$\begin{aligned} f_{\mathrm{WBG}n+1} &= x_1 r_1 + x_2 \bar{r}_1 r_2 + x_3 \bar{r}_1 \bar{r}_2 r_3 + \cdots + x_n \bar{r}_1 \bar{r}_2 \ldots \bar{r}_{n-1} r_n + x_n \bar{r}_1 \bar{r}_2 \ldots \bar{r}_n \\ &= f_{\mathrm{WBG}n} + x_{n+1} \bar{r}_1 \bar{r}_2 \ldots \bar{r}_n r_{n+1} \end{aligned}$$

Therefore, the cost of $f_{\mathrm{WBG}n+1}$ is $cost_{n+1} = cost_n + (n + 2 + 1)$ as we have an additional term, and the literal count of $x_{n+1} \bar{r}_1 \bar{r}_2 \ldots \bar{r}_n r_{n+1}$ is $n + 2$. Suppose there exists some other function $f^*_{\mathrm{WBG}n+1} \equiv f_{\mathrm{WBG}n+1}$.For the input combination $x_1 x_2 \ldots x_n = 00 \ldots 0$, we have $p_{F^*_{\mathrm{WBG}n+1}} = 0.5^{n+1} p_{x_{n+1}}$, which needs at least $n + 1$ stochastic constants and one variable x_{n+1} to generate it. Hence, the cost of $f^*_{\mathrm{WBG}n+1}$ must be at least $cost_n + n + 2 + 1$, implying $f_{\mathrm{WBG}n+1}$ is one of the optimal functions. Since $f_{\mathrm{WBG}2}$ is optimal, we conclude that all $f_{\mathrm{WBG}n}$'s are optimal.

4.3 Search-based Optimization

To speed up the search process when dealing with functions or circuits with many variables, *SECO* incorporates a guided search heuristic; see Figure 4.8. Again, assume the initial stochastic function *f* is defined on $X = X_V; X_C$, and note that $f = \sum_{c_i \in C_V} c_i f_{c_i}$ where C_V is the set of all *s*-literal cubes. *SECO* first calculates the weight of *f*'s cofactor with respect to c_i, and then generates and sorts by cost all functions that have the same weight, as lines 16-18 show. This process is repeated for all c_i's. In lines 15-19, F_{ci} denotes the function space that is searched by the *for* loop starting in line 14. For small cases like

```
1  Input: f(X_V;X_C) or          // |X_V| = s and |X_C| = t
2         W = (w_1, w_2, ...)   // w_i = w(f_ci), weights of f's SEC 𝔽
3  Output: The lowest cost f* in f's SEC 𝔽
4  Initialization:   //Initially make f the result
5     If input is f
6        f* = f;
7        cost_f* = espresso(f*);
8        For all s-literal cubes c_i on X_V
9           Compute w_i = w(f_ci);   // f_ci is cofactor of f wrt c_i
10    else
11       f* = ϕ;
12       cost_f* = ∞;
13 Search SEC 𝔽 for the lowest-cost member
14    For all s-literal cubes c_i on X_V
15       Generate set of functions F_ci on X_C with weight w_i;
16       For all g in F_ci
17          cost_g = espresso(f);
18       Sort functions in F_ci by cost;     // Give higher priority to
19    Construct  E by combining F_ci's;   //    locally optimal functions
20    While E is not empty and search time limit is not reached
21       Select a function g in E;
22       If cost_f* > cost_g
23          f* = g;
24          cost_f* = cost_g;   // Keep the lower-cost function
25       E = E – g;
26    Return f* ;
```

Figure 4.8: Procedure *SECO* used by *ESECS* to find a lowest-cost member of an SEC.

Examples 4.6 and 4.6 this *for* loop searches F_{c_i} completely and usually finishes in a reasonable time, given that stochastic circuits are often quite small.

For large circuits, F_{c_i} denotes a sampled subset of the entire function space. Next the SEC is constructed using the sorted sets of functions equivalent to f_{c_i}. In other words, members in the SEC with lower-cost cofactors are evaluated first, as they are locally optimal. The costs of the SEC members are then calculated and the lowest-cost member is kept. *SECO* repeats the cost evaluation until the entire SEC has been examined, or else a search time limit is reached.

Example 4.9: Consider the polynomial function

$$\hat{F}(X_1, X_2) = 0.4375 - 0.125(X_1 + X_2) - 0.5625X_1X_2$$

defined in [1]. The corresponding BF generated by the spectral transform method (later called STRAUSS [7]) is

$$\begin{aligned} f(x_1, x_2; r_1, r_2, r_3, r_4) \\ &= \bar{r}_1\bar{x}_1\bar{x}_2 + \bar{r}_2\bar{r}_3\bar{x}_1\bar{x}_2 + \bar{r}_2\bar{r}_4\bar{x}_1\bar{x}_2 + r_1r_2x_1x_2 + r_1r_3x_1x_2 \\ &+ r_1r_4x_1x_2 \end{aligned} \quad (4.11)$$

where $X_{\mathrm{V}} = (x_1, x_2)$ and $X_{\mathrm{C}} = (r_1, r_2, r_3, r_4)$ and *f*'s cost is 29. It is stated in [1] that the cost of Equation (4.11) is about 60% that of the ReSC-style [74]. Nevertheless, *SECO* finds a function

$$f^* = \bar{r}_1\bar{x}_1\bar{x}_2 + \bar{r}_2\bar{r}_3\bar{x}_1\bar{x}_2 + \bar{r}_2\bar{r}_4\bar{x}_1\bar{x}_2 + r_1r_2x_1x_2 + r_3r_4x_1x_2$$

with $f^* \equiv f$ and cost 24, a further 17% reduction. □

As Figure 4.3 suggests, *ESECS* can serve as a complete, self-contained technique for stochastic circuit synthesis. The input is a desired stochastic (arithmetic) function *A* and a cell library *L* that can implement *A*. The target SEC is identified by either *SECI* or an initial Boolean function *f* that realizes *A* found by direct mapping or some other stochastic circuit synthesis method. *ESECS* then searches the target SEC for a function *f** that has minimal cost.

Example 4.10: Returning to Example 4.5, the target function was $A = -0.25(X_1 + X_2)$. Equation (4.9) defines a cost-26 function $f(x_1, x_2; r_1, r_2, r_3)$ obtained by direct mapping from *A*. Application of *SECO* to *f*'s SEC produces an equivalent function $f^* = \bar{x}_1r_1 + \bar{x}_2r_2 + r_1r_2$ whose cost is just 9, a 65% reduction. The number of stochastic constants is also decreased, which may reduce the logic needed to generate stochastic constants. □

SECO's computational complexity depends on the size of the SEC as indicated by Theorem 4.5. We used a symmetry property of SECs to reduce the search space in half (see Example 4.7), and this defines the worst-case scenario of *SECO*. If the SEC is searched exhaustively, $\prod_{i=0}^{2^s-1}\binom{2^t}{w_i}/2$ functions need to be evaluated. This complexity is still high, which usually makes finding a strict optimum infeasible—as in most CAD problems. *SECO* uses a time limit parameter to prevent the explosion of search complexity and deliver a best-effort result under a given time budget. Later in this section, we provide experimental results showing that, even with a limited search, *SECO* can reduce area cost significantly.

To evaluate *SECO* more broadly, we performed experiments on randomly generated functions $f(X_V; X_C)$. For various values of $s = |X_V|$ and $t = |X_C|$, a hundred functions were randomly generated and their average minimum cost was calculated. Figure 4.9 summarizes the results for two representative *s,t* configurations; we tried many other

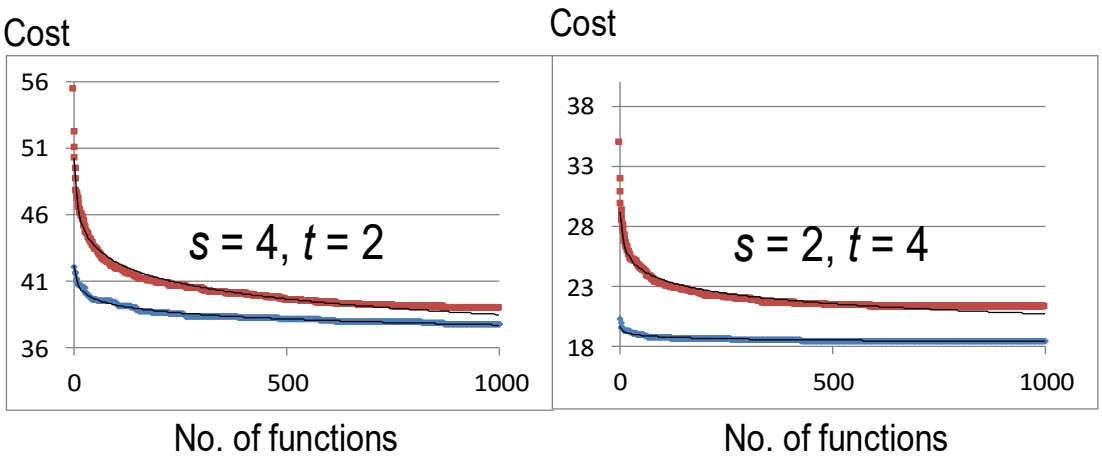

Figure 4.9: Average minimum cost of functions found by *SECO* (blue) and random sampling (red).

s,t pairs, and all gave similar results. The X-axis is the number of equivalent functions evaluated, while the Y-axis is the average minimum cost. These results suggest that design cost can be reduced substantially even when a relatively small part of the SEC is evaluated. Figure 4.9 shows that an average cost reduction of 40% is achieved when a thousand stochastically equivalent functions are sampled. The run-time mainly depends on the search time limit set in line 20 of Figure 4.8, which was two minutes in our experiments.

Figure 4.9 also compares *SECO* with random sampling of the SEC functions. As the set of weights is a canonical representation of an SEC (Theorem 4.3), we randomly generate a set of integer weights to obtain each sample SEC. *SECO* gives higher priority to SEC members that are locally optimal. This heuristic can reduce the cost significantly, and tends to outperform the random sampling approach.

Again, we performed experiments on randomly generated functions $f(X_V; X_C)$. For various values of $s = |X_V|$ and $t = |X_C|$, a hundred functions were randomly generated and then optimized by *SECO* to evaluate the average area cost reduction. Our experimental results show that *SECO* scales well to relatively large input sizes and achieves higher cost reduction for functions with larger $|X_C|$; see Figure 4.10. Because of this, *SECO* is suitable for high-accuracy stochastic circuit optimization. The accuracy of a stochastic circuit is

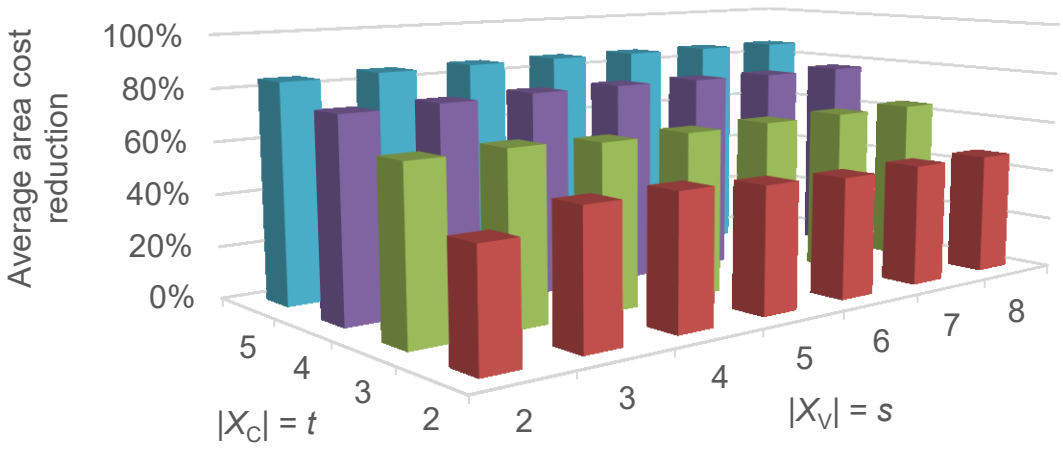

Figure 4.10: Average area cost reduction achieved by *SECO*.

affected by various factors including $|X_C|$, which is determined when a base stochastic circuit or an SEC is selected in Step 1 of *ESECS*. A large $|X_C|$ is necessary when a more accurate real number is synthesized. Due to the nature of the LFSR-based random sources discussed earlier, real numbers are approximated by expressions of the form $\sum 1/2^i$. For example, if the arithmetic function $p_F = 0.3p_{X_1}$ is needed, the coefficient 0.3 must be approximated as it cannot be generated directly. If only one stochastic constant r_i is present, we can only have the weak approximation $\hat{p}_F = 0.5p_{X_1}$ implying the error $|\hat{p}_F - p_F| = 0.2p_{X_1}$. More stochastic constants are needed to achieve a better approximation of the target arithmetic function. Two stochastic constants reduce the error to $0.05p_{X_1}$, and so on. STRAUSS or other known SC synthesis methods may be used to handle this constant number approximation. In general, higher-accuracy synthesis needs more stochastic constants, implying that *SECO* can improve the synthesis process significantly.

4.4 Cover-based Optimization

As an alternative to search procedures like *SECO*, we propose a synthesis approach for stochastic circuits that is roughly analogous to two-level minimization in conventional logic design [14].

Theorem 4.3 states that a function *f* from an SEC with $X = X_V; X_C$ is defined by the weights of the function for all *s*-literal cubes on X_V. This suggests representing *f* by its weights in tabular form—a *weight table*—similar to a truth table or a Karnaugh map. Concepts such as implicant can then be extended from Boolean to stochastic functions in a natural way, so that stochastic logic design becomes an implicant covering problem resembling classical two-level minimization.

Example 4.11: Figure 4.11*a* is a truth table (TT) for all four members of the SEC that realizes the scaled addition function $0.5(p_{X_1} + p_{X_2})$. If we abstract away the X_C constant r_1, we can represent each function, and therefore the entire SEC, by the weight table (WT) defined on $X_V = (x_1, x_2)$ appearing in Figure 4.11*b*. Each entry of the WT is the weight of

	x_1	x_2	r_1	f	g	h	i
0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
2	0	1	0	0	0	1	1
3	0	1	1	1	1	0	0
4	1	0	0	0	1	1	0
5	1	0	1	1	0	0	1
6	1	1	0	1	1	1	1
7	1	1	1	1	1	1	1

(a)

x_1	x_2	$\mathbb{F}$
0	0	0
0	1	1
1	0	1
1	1	2

(b)

Figure 4.11: Truth table for the SEC of the BFs realizing stochastic addition; (b) weight-table representation.

the corresponding 2-literal cube on X_V in the original TT. For example, $x_1\bar{x}_2$ is defined by TT rows 2-3, and each function in the SEC has both a 0 and a 1 in the corresponding two output columns, making $x_1\bar{x}_2$'s weight 0 + 1 = 1. Similarly, rows 6-7 give the weight of x_1x_2 as 2. In this way, the TT of Figure 4.11*a* reduces to the simpler form of Figure 4.11*b*.□

In conventional design, an implicant *i* is a cube containing one or more minterms of a BF *f*. We say *i* implies f if f takes the value 1 whenever *i* is 1. Each implicant has a cost equal to its literal count, and two-level synthesis tries to find a set of implicants with minimal cost that cover each minterm of *f*. We define a *stochastic implicant I* as a set of one or more *s*-literal cubes on *X*v of a stochastic function $F(X_V; X_C)$. The stochastic implicant *I* thus covers one or more *s*-literal cubes in *F*'s WT. It stochastically implies *F* if *F* also takes the value 1 with a non-zero probability whenever *I* is 1. This non-zero probability is 0.5^t times the number of *s*-literal cubes covered by *I*. A *stochastic prime implicant* is a stochastic implicant that cannot be covered by a larger (one with fewer literals) stochastic implicant. Figure 4.12*a* shows the WT for the stochastic edge detector of Figure 4.5*a*. In this example, $\bar{x}_1x_2$, $x_1\bar{x}_2$, $\bar{x}_3x_4$ and $x_3\bar{x}_4$ are the stochastic prime implicants.

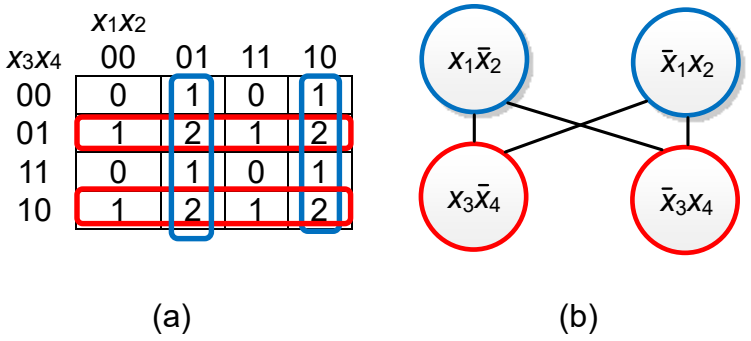

Figure 4.12: (a) Weight table in K-map format, and (b) implicant covering graph for the edge detector in Figure 4.5*a*.

The stochastic synthesis procedure *SECM* (SEC-based Mapping) is based on these ideas and Theorem 4.3, which indicates how the characteristic function of an SEC is formulated in terms of sub-cube weights. *SECM* is described in Figure 4.13. It starts by identifying all stochastic implicants and prime implicants. This is done by changing all non-zero cell values in the WT to 1, a process called "projecting" the WT. All cells in the *projected weight table* (PWT) have 0-1 values, as in a K-map. Implicants in the PWT are stochastic implicants for the WT. PWTs allow us to use conventional synthesis methods to find stochastic (prime) implicants. Next, *SECM* solves a covering problem for the WT whose goal is to find a set of stochastic implicants that cover every s-literal cube c_i exactly w_i times; we call this the *WT covering problem* and its solution the *WT cover*. A similar covering problem called the λ-cube intersection problem, can be found in [75] but its goal is different from ours.

For example, the stochastic edge detector requires four stochastic prime implicants $\{\bar{x}_1x_2, x_1\bar{x}_2, \bar{x}_3x_4, x_3\bar{x}_4\}$ to ensure each 4-literal cube c_i of weight w_i is covered exactly w_i times; see Figure 4.12*a*. If the weight of cube $\bar{x}_1x_2x_3x_4$ were 2 instead of 1, we would need to add an extra stochastic implicant $\bar{x}_1x_2x_3x_4$ to make sure the cube is covered by exactly two stochastic implicants. The WT cover would then become $\{\bar{x}_1x_2, x_1\bar{x}_2, \bar{x}_3x_4, x_3\bar{x}_4, \bar{x}_1x_2x_3x_4\}$.

```
1  Input: W = (w1, w2, ...)   // wi = w(fci), weights of an SEC 𝔽
2  Output: A low cost f* in SEC 𝔽
3  Initialization:
4     f* = ϕ;
5  Generate a member f* of the SEC 𝔽 by finding 𝔽's lowest-
6  cost cover
7     For all wi in W   // Project SEC 𝔽 to a Boolean function fp
8       If wi == 0
9          ki = 0;
10      else ki = 1;
11    fp (x1, x2, …, xs) = ∨(ki ∧ mi);   // fp is a Boolean function
12                                       //    represented by Equation (4.1)
13    I = implicants(fp);   // I is a set of implicants of fp
14                          //    found by tools such as espresso
15    C = wt-cover(I, W);   // Use methods such as mixed-integer
16                          //    linear programming to find the lowest-
17                          //    cost WT cover C = (c1, c2, ...)
18    V = C;   // Construct the vertex V and edge E for  the coloring
19             //    problem; WT cover C is the vertices V = (v1, v2, ...)
20    For all pairs of vi and vj in V
21       If vi and vj cover the same s-literal cube
22          E = E + (i, j);   // (i, j) in E indicates an edge in vi and vj
23    A = vertex-coloring(V, E);   // A = (a1, a2, ...) is the assignment
24                                 //    of colors using binary encoding
25    f = circuit-mapping(C, A);   // Map the stochastic circuit to a
26                                 //    two-level SOP design by
27                                 //    f = ∨(ai∧ci) where ∨ is OR and
28                                 //    ∧ is AND
29    f* = two-level-optimization(f);   // Optimize the circuit again
30    Return f*;                        //    by conventional optimizer
```

Figure 4.13: Procedure *SECM* used by *ESECS* to generate a low-cost stochastic circuit.

We use literal count to measure the cost of stochastic implicants. The cost of a WT cover is the sum of costs of its stochastic implicants. Note that the WT cover and its associated cost are not always unique. The problem of interest then is to find a WT cover (if any) with lowest cost.

A solution to the minimum-cost WT covering problem reduces the literal costs of the stochastic variables X_V. The next step of *SECM* targets minimizing the cost associated with the stochastic constants X_C. These constants need to be assigned to the stochastic implicants to ensure the characteristic function values fall within the unit interval. The SNs

generated by stochastic implicants are uniformly scaled. To map the characteristic function to a two-level SOP design, we must ensure the bit-streams generated by scaled stochastic implicants have non-overlapping 1s. This requirement enables the OR gate at the output stage of an AND-OR circuit to perform exact addition of the type seen in the WBG [38]. When two stochastic implicants only cover different *s*-literal cubes, they produce bit-streams with non-overlapping 1s; otherwise, we need to take special measures.

The method we propose to meet the foregoing non-overlapping 1s requirement is to formulate and solve it as a graph coloring problem. A *valid coloring* is an assignment of colors to each vertex of a graph such that no edge connects vertices of the same color. Each stochastic implicant in a WT cover is a vertex of the proposed graph. An edge connects two vertices if the corresponding stochastic implicants intersect. For example, in the stochastic edge detector with WT cover $\{\bar{x}_1x_2, x_1\bar{x}_2, \bar{x}_3x_4, x_3\bar{x}_4\}$, both stochastic implicants $\bar{x}_1x_2$ and $x_3\bar{x}_4$ cover the cube $\bar{x}_1x_2x_3\bar{x}_4$. We therefore connect vertices $\bar{x}_1x_2$ and $x_3\bar{x}_4$, as shown in Figure 4.12*b*, to form an *implicant covering graph G*. To distinguish bit-streams with non-overlapping 1s, we use different colors. The problem of interest now becomes the following coloring problem: Find a valid coloring of *G* with the minimum number of colors. This ensures that the number of bit-streams used is minimal. Clearly, for the stochastic edge detector graph in Figure 4.12*b*, two colors suffice.

The bit-streams with non-overlapping 1s serve as scale factors, and all have the same probability. One simple way to generate them using a minimum number of stochastic constants r_i is binary encoding. This technique assigns a unique pattern of 1s and 0s in normal binary sequence to each color of a valid coloring. AND gates are then used to generate the required bit-streams. The AND gates' *i*-th input is $\bar{r}_i$ or r_i if the corresponding *i*-th bits of their patterns are 0 or 1, respectively. In general, *t* stochastic constants can generate 2^t such bit-streams corresponding to $\bar{r}_1 \ldots \bar{r}_{t-1}\bar{r}_t$, $\bar{r}_1 \ldots \bar{r}_{t-1}r_t$, $\bar{r}_1 \ldots r_{t-1}\bar{r}_t$, …, $r_1 \ldots r_{t-1}r_t$. For example, to obtain four bit-streams, we can use two stochastic constants r_1 and r_2 and generate the bit-streams by r_1r_2, $r_1\bar{r}_2$, $\bar{r}_1r_2$ and $\bar{r}_1\bar{r}_2$. When *k* bit-streams (colors) are needed, we must use at least $|X_C| = t$ stochastic constants, where $2^t \geq k$. The final BF

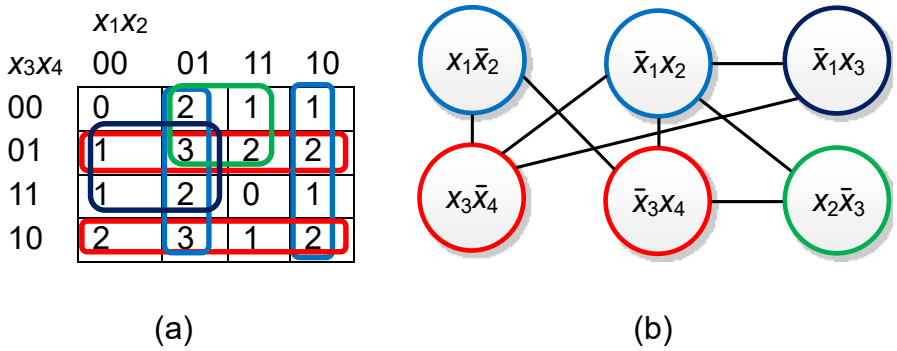

Figure 4.14: (a) Weight table in K-map format for Example 12, and (b) its implicant covering graph.

is mapped to an SOP expression in which the OR gate performs exact addition and the scaling factors are multiplied by stochastic implicants using AND operations. For example, the stochastic edge detector needs only two colors, so we can use one stochastic constant r_1, and assign r_1 and $\bar{r}_1$ to produce the desired BF $z_{i,j} = r_1 x_1 \bar{x}_2 + r_1 \bar{x}_1 x_2 + \bar{r}_1 x_3 \bar{x}_4 + \bar{r}_1 \bar{x}_3 x_4$ or $z_{i,j} = \bar{r}_1 x_1 \bar{x}_2 + \bar{r}_1 \bar{x}_1 x_2 + r_1 x_3 \bar{x}_4 + r_1 \bar{x}_3 x_4$. Each can be implemented with two XOR gates and one 2-to-1 multiplexer, as in Figure 4.5*a*.

Example 4.12: Figure 4.14*a* shows the WT for an SEC whose WT cover found by *SECM* is $\{\bar{x}_1 x_2, x_1 \bar{x}_2, \bar{x}_3 x_4, x_3 \bar{x}_4, \bar{x}_1 x_3, x_2 \bar{x}_3\}$. The corresponding implicant covering graph is shown in Figure 4.14*b*, and requires at least four colors for a valid coloring. Therefore, we need two stochastic constants r_1 and r_2 to generate four bit-streams with non-overlapping 1s using $\bar{r}_1 \bar{r}_2$, $r_1 \bar{r}_2$, $r_1 \bar{r}_2$, and $r_1 r_2$. The final BF obtained is $z_{i,j} = \bar{r}_1 \bar{r}_2 x_1 \bar{x}_2 + \bar{r}_1 \bar{r}_2 \bar{x}_1 x_2 + r_1 \bar{r}_2 x_3 \bar{x}_4 + r_1 \bar{r}_2 \bar{x}_3 x_4 + \bar{r}_1 r_2 \bar{x}_1 x_3 + r_1 r_2 x_2 \bar{x}_3$. □

SECM divides stochastic circuit optimization into two sub-problems, so its complexity reduces from handling a single (s + t)-input circuit to handling two smaller circuits with s and t inputs, respectively. It maps the synthesis problem to the vertex coloring and the covering problems, which can be solved by standard algorithms. The complexity of *SECM* is therefore ultimately determined by the complexity of the solvers used. As the run-time and the computation resources of circuit optimization grow exponentially with the circuit input size, *SECM* scales better than *SECO*. For example, for

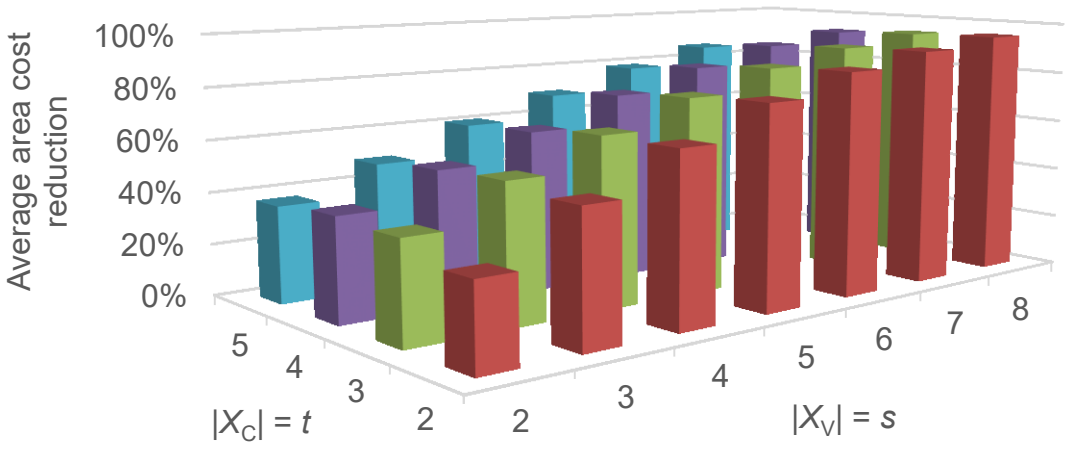

Figure 4.15: Average area cost reduction achieved by *SECM*.

the stochastic edge detector, *SECO* needs to optimize up to 128 5-input BFs to obtain a result, whereas *SECM* only has to optimize one 4-input and one 1-input BF.

Using *SECM, w*e repeated the experiment performed with *SECO* and summarized in Figure 4.15. The corresponding average literal cost reduction achieved by *SECM* is shown in Figure 4.15. Unlike *SECO* whose area cost reduction is mainly determined by $|X_C|$, *SECM* performs well as $|X_V|$ increases. This is because by using linear programming, *SECM* ensures that the cost of the WT cover on X_V is minimal. The second step of *SECM* minimizes the number of stochastic constants $|X_C|$, not literal counts over X_C. Therefore, *SECM* can achieve greater literal cost reduction as the number of stochastic variables increases. *SECO* performs better as $|X_C|$ increases because when $|X_V|$ is fixed, a larger $|X_C|$ provides more stochastically equivalent functions to be sampled and searched, and therefore increases the chance of finding a lower-cost design. For example, in the extreme case where $|X_C| = 0$, there is no stochastically equivalent function, so *SECO* cannot achieve any area cost reduction.

Figure 4.16 compares *SECO* and *SECM* by the difference of average area cost reduction when *SECO* is replaced by *SECM*. The Y-axis shows (*average area cost reduction by SECM − average area cost reduction by SECO*), so positive values indicate

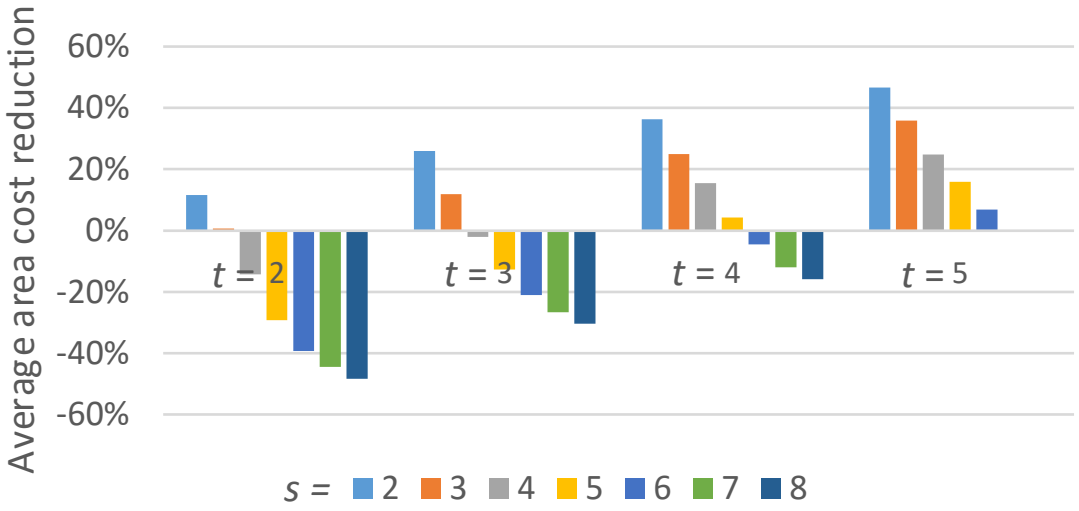

Figure 4.16: Average area cost reduction when *SECO* is replaced by *SECM*.

that *SECM* outperforms *SECO*. It can be seen that *SECO* outperforms *SECM* only when the input sizes are small, as *SECO* can then search the SEC entirely to get a global optimum. However, as explained earlier, as $|X_V|$ increases, *SECM* outperforms *SECO*. Therefore, *SECO* seems preferable for smaller designs, while *SECM* works better for larger ones.

4.5 Summary

The usual starting point of conventional logic synthesis is a Boolean function $f(X)$ to be realized. This is not necessarily the case in stochastic computing, however, where many equivalent Boolean functions of varying area cost can satisfy the problem specification. We have shown that partitioning the inputs X between variables X_V and constants X_C leads to a useful concept of stochastic equivalence among Boolean functions. We derived some basic properties of stochastic equivalence classes (SECs) and showed them to be helpful for understanding and implementing stochastic circuit synthesis and optimization. We proposed *ESECS*, an SEC-based synthesis algorithm for stochastic circuits. *ESECS* contains three main procedures: *SECI* identifies an SEC for the target arithmetic function; *SECO* optimizes a known stochastic circuit by searching its SEC; and *SECM* generates a low-cost stochastic circuit in a fashion similar to classical two-level

design. We have presented experimental data which show the *ESECS* approach can check the optimality of existing SC designs, and produce new SC designs of relatively low cost.

CHAPTER 5
Design of Dividers

The preceding chapter examined the design of combinational stochastic circuits that realize polynomials, which only require addition, subtraction and multiplication. While these operations have extremely simple stochastic implementations, this is not true for division. Most currently known stochastic dividers employ sequential circuits whose accuracy, convergence properties, etc., are unsatisfactory or not well understood. As a result, division is usually avoided or approximated in SC design. This chapter first reviews and analyzes in depth the existing approaches to stochastic division. It then proposes a novel division technique called CORDIV (correlated division) that exploits correlation between the input parameters. CORDIV is based on two key observations: (1) the conditional probability $p_{X_1|X_2}$ of X_1 given X_2 naturally defines the stochastic division operation $p_{X_1X_2}/p_{X_2}$,and (2) X_1 and X_2 can be correlated to efficiently transform this operation to p_{X_1}/p_{X_2}. We then present a CORDIV-based divider which has area cost far lower than that of conventional stochastic dividers, while achieving better accuracy. CORDIV is compatible with conventional unipolar and bipolar SN formats, unlike some stochastic dividers that achieve similar area cost by using alternative SN representations.

5.1 Stochastic Dividers

Stochastic division was first discussed by Gaines, who pioneered SC back in the1960s [31]. He noted that an approximate form of division is achieved by a simple JK flip-flop circuit (Figure 3.1). As shown in Section 3.1, the JK flip-flop's output z has the probability $p_Z = p_{X_1}/(p_{X_1} + p_{X_2})$. This approximates $p_Z = p_{X_1}/p_{X_2}$ when the dividend

p_{X_1} is small. However, the approximation becomes inaccurate when the divisor p_{X_2} is small. Nevertheless, because of its simplicity, the JK flip-flop divider has been successfully applied to the design of the update node used by LDPC decoding; see Figure 1.5 [67].

To implement $p_Z = p_{X_1}/p_{X_2}$ more accurately, Gaines proposed using an opamp-like sequential component called an ADDIE (ADaptive DIgital Element), one form of which is an up-down counter with feedback [31]. The ADDIE makes an estimate $\hat{p}_Z$ of p_{X_1}/p_{X_2} in binary form, and uses an SNG to generate the stochastic form of $\hat{p}_Z$. Since $\hat{p}_Z \cong p_Z = p_{X_1}/p_{X_2}$, the product $p_{X_2}\hat{p}_Z$ should equal p_{X_1}.

Figure 5.1*a* shows the unipolar version of the ADDIE-based divider, which calculates $\hat{p}_Z$ dynamically [31]. When $p_{X_1} > p_{X_2}\hat{p}_Z$, $\hat{p}_Z$ is less than p_Z and the counter is incremented. When $p_{X_1} < p_{X_2}\hat{p}_Z$, $\hat{p}_Z > p_{X_1}/p_{X_2}$ and the counter is decremented. Its output *z* is fed back to an AND gate which performs the multiplication $p_{X_2}\hat{p}_Z$. The dividend SN X_1 with numerical value p_{X_1} is connected to the Up control of the counter, and the product $p_{X_2}\hat{p}_Z$ is sent to the Down line. The counter thus accumulates the value $p_{X_1} - p_{X_2}\hat{p}_Z$, and the required division $\hat{p}_Z = p_{X_1}/p_{X_2}$ is completed when the overall system is in equilibrium. As the counter's value is estimated by SNs that fluctuate randomly, the binary form of $\hat{p}_Z$ is also a random number. Gaines noted that the variance of $\hat{p}_Z$ is inversely proportional to the number of states 2^k in the counter, since the binary form of $\hat{p}_Z$ changes value by $1/2^k$ when the Up and Down signals are different. This $1/2^k$ factor defines an error bound for Gaines' ADDIE-based design. An SN *Z* generated by combinational SC circuits has a variance $p_Z(1 - p_Z)/N$, where *N* is the bit-stream length, so *Z* can have arbitrarily small variance given an arbitrarily long bit-stream. However, this is not the case for the ADDIE-based designs, as we will show in Section 5.3.

Figure 5.1*b* gives the bipolar version of Gaines' ADDIE-based divider. It requires some non-trivial modifications to compute $2p_Z - 1 = (2p_{X_1} - 1)/(2p_{X_2} - 1)$. Since bipolar SNs may be negative, $X_1 > X_2\hat{Z}$ does not always imply $\hat{Z} < X_1/X_2$; the sign of X_2

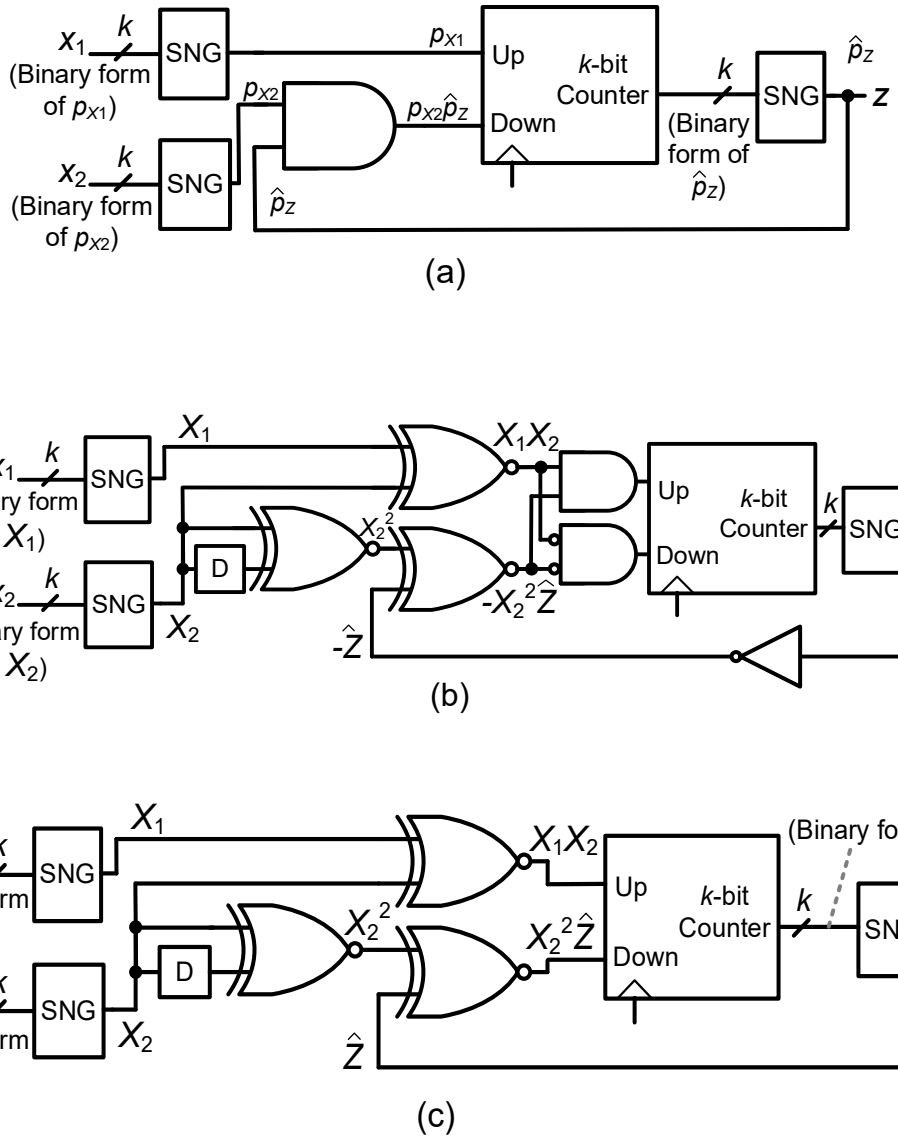

Figure 5.1: Gaines' ADDIE-based (a) unipolar and (b) bipolar stochastic dividers [31]; (c) equivalent circuit for Figure 5.1*b*.

affects the result of comparison between $\hat{Z}$ and X_1/X_2. Similarly, $X_1 < X_2\hat{Z}$ does not guarantee $\hat{Z} > X_1/X_2$. To reduce the uncertainty caused by signed numbers, the divider of Figure 5.1*b* compares X_1X_2 and $X_2^2\hat{Z}$, instead of X_1 and $X_2\hat{Z}$. Assuming the divisor X_2 will not be zero, X_2^2 is always positive, so $X_1X_2 > X_2^2\hat{Z}$ implies $X_1/X_2 > \hat{Z}$. On the other hand,

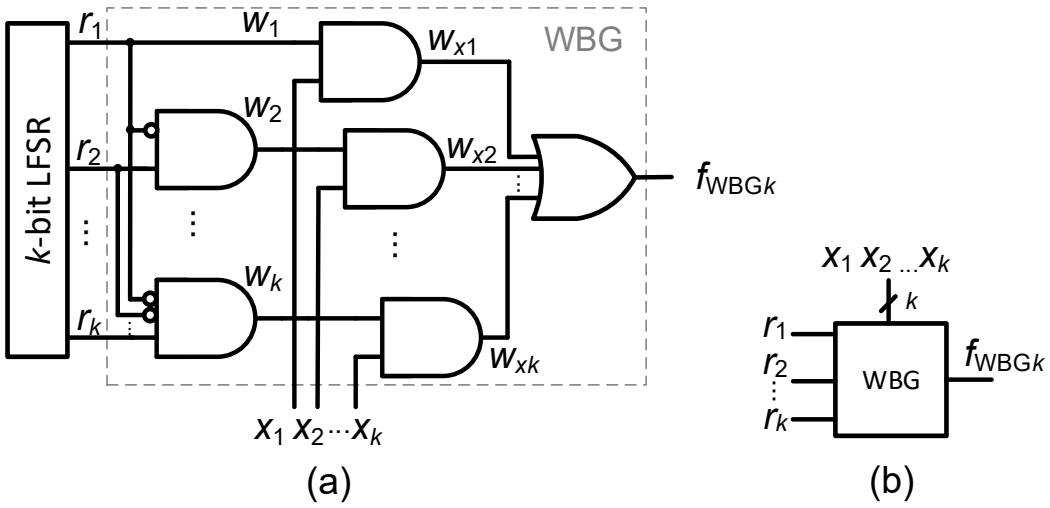

Figure 5.2: (a) The k-bit weighted binary generator (WBG) of SNs [38] and (b) its symbol.

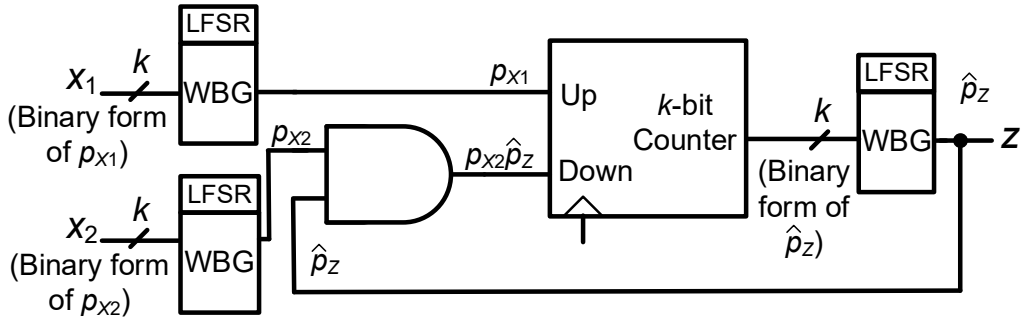

Figure 5.3: Ananth's ADDIE-based unipolar stochastic divider.

$X_1X_2 < X_2^2\hat{Z}$ indicates $\hat{Z} > X_1/X_2$. Note that Gaines' original bipolar divider in Figure 5.1*b* can be simplified to the equivalent circuit shown in Figure 5.1*c*.

In a little-noticed patent on stochastic processor design, Ananth [9] describes another ADDIE-based divider (Figure 5.3) in which the SNGs of Figure 1.4*a* are replaced by special SNGs called weighted binary generators (WBGs), originally due to Gupta and Kumaresan [38]. As depicted in Figure 5.2, a WBG converts a k-bit binary number $x_1x_2 \ldots x_k$ to a (unipolar) SN $F_{\text{WBG}k}$ of length 2^k with essentially no error. $F_{\text{WBG}k}$ is guaranteed to have the value $p_{F_{\text{WBG}k}} = \sum_{i=1}^{k} 0.5^i p_{X_i}$ after 2^k bits have been generated.

Unlike a conventional SNG, which depends on number comparison, the WBG decomposes the output probability $p_{F_{\mathrm{WBG}k}}$ into k SNs W_1, W_2,…,W_k with probabilities $p_{W_i} = 0.5^i$, for i = 1,2,…,k. The output $F_{\mathrm{WBG}k}$ is formed by combining these k SNs via an OR gate, as shown in Figure 5.2. For example, a 3-bit LFSR generates three bit-streams $r_1 = 01110010$, $r_2 = 10111000$, and $r_3 = 01011100$, which give $w_1 = 01110010$, $w_2 = 10001000$, and $w_3 = 00000100$. A binary number $x_1x_2x_3$ = 011 representing 0.375 generates an SN $F_{\mathrm{WBG}k} = 1000\ 1100$. Note that the 1s in the W_i's are non-overlapping, so they are effectively added by the OR gate in Figure 5.2.

Example 5.1: Figure 5.4 illustrates the behavior of the unipolar ADDIE-based divider in Figure 5.1*a*. The size k of the counter is 4, so the maximum number the counter can hold is 15. The initial value of the counter is set to 8, which is half the range of the counter. To demonstrate the worst-case convergence of the counter, we choose $p_{X_1} = 0$ and $p_{X_2} = 1$, so ideally the output $p_Z = p_{X_1}/p_{X_2}$ and the value stored in the counter are both 0. The X-axis in Figure 5.4 represents clock cycles; the blue and red lines are the binary values stored in the counter and the probability of the output SN Z. This example shows that the counter converges to the desired value 0 at the 53rd clock cycle, while the output bit-stream has probability 0.151. □

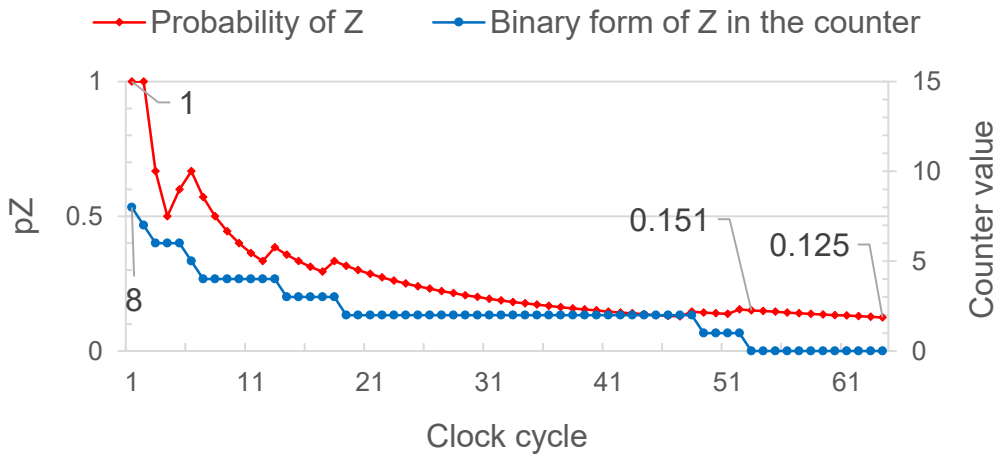

Figure 5.4: Convergence behavior of Gaines' unipolar ADDIE-based divider in Figure 5.1*a* with k = 4, p_{X1} = 0 and p_{X2} = 1.

Several alternative SN formats have been proposed that aim to simplify division. Gaines describes a single-line format with infinite range, where an SN Z on line z has the numerical value p_Z/q_Z with q_Z denoting $1 - p_Z$ [31]; the same representation was rediscovered later by Min et al. [64]. This *ratio* format allows the reciprocal of a number to be calculated by a NOT gate. If the input x_1 and output z of the NOT gate have values p_{X_1}/q_{X_1} and p_Z/q_Z, respectively, then $p_Z/q_Z = q_{X_1}/p_{X_1}$. The ratio format enables a relatively simple divider design based on a JK flip-flop, as depicted in Figure 5.5*a*. Here $p_J = p_{X_1}q_{X_2}$ and $p_K = q_{X_1}p_{X_2}$, so $p_Z = p_{X_1}q_{X_2}/(p_{X_1}q_{X_2} + q_{X_1}p_{X_2})$, leading to $p_Z/q_Z = (p_{X_1}/q_{X_1})/(p_{X_2}/q_{X_2})$.

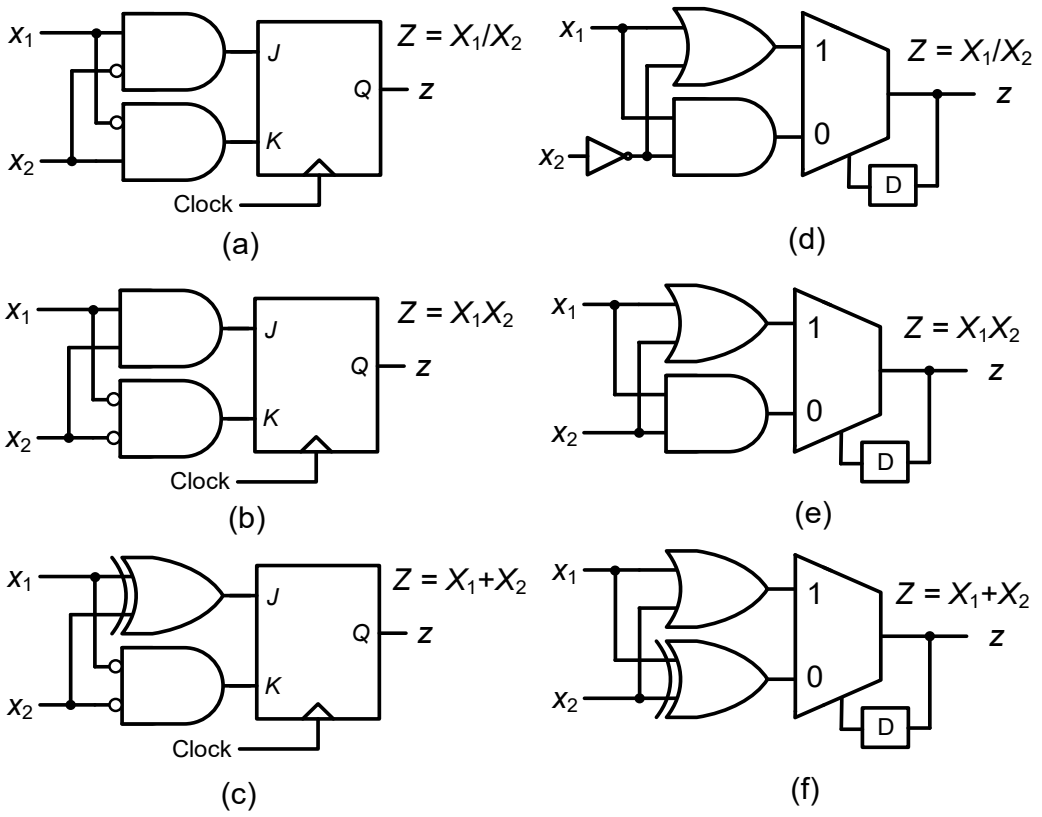

Figure 5.5: Gaines' basic components for the ratio format: (a) divider, (b) multiplier and (c) adder [31]; Min et al.'s (d) divider, (e) multiplier and (f) adder [64].

Example 5.2: : The SNs X_1 = 00110000 and X_2 = 10011100 with $p_{X_1} = 2/8$ and $p_{X_1} = 4/8$ have numerical values 1/3 and 1, respectively. For the divider in Figure 5.5*a*, we have J = 00100000 and K = 10001100, which yield Z = 00110000 with $p_Z = 2/8$. Z's value is 1/3, which is the result of the division (1/3)/1. □

While the ratio format simplifies division, it complicates other SC operations, notably multiplication and addition. Figure 5.5*b* shows a ratio multiplier where $p_J = p_{X_1}p_{X_2}$ and $p_K = q_{X_1}q_{X_2}$. Since $p_Z = p_J/(p_J + p_K)$, we have the output probability $p_Z = p_{X_1}p_{X_2}/(p_{X_1}p_{X_2} + q_{X_1}q_{X_2})$ and $p_Z/q_Z = (p_{X_1}/q_{X_1}) \times (p_{X_2}/q_{X_2})$, which implements multiplication. Similarly, when $p_J = p_{X_1}q_{X_2} + q_{X_1}p_{X_2}$ and $p_K = q_{X_1}q_{X_2}$, the output z has $p_Z = (p_{X_1}q_{X_2} + q_{X_1}p_{X_2})/\big((p_{X_1}q_{X_2} + q_{X_1}p_{X_2}) + q_{X_1}q_{X_2}\big)$, yielding the add operation $p_Z/q_Z = (p_{X_1}/q_{X_1}) + (p_{X_2}/q_{X_2})$ illustrated in Figure 5.5*c*. A similar set of SC components for the ratio format is found in Min et al. [64]; see Figure 5.5*d–f*. They present components based on a MUX and a D-type flip-flop and generalize the adder and multiplier to m SNs. For m = 2, however, their multiplier, adder and divider are logically equivalent to Gaines' JK-based designs in Figure 5.5*a–c*.

Canals et al. recently presented a two-line SN format that employs the ratio of two bipolar SNs X_N and X_D [18]. In this encoding scheme, an SN's numerical value is $X = X_N/X_D = (2p_{X_N} - 1)/(2p_{X_D} - 1)$, and the bipolar SC devices in Figure 1.3 serve as computational building blocks. Multiplication takes the form $Z = X_1X_2 = (X_{1_N}X_{2_N})/(X_{1_D}X_{2_D})$. Since X_{i_N} and X_{i_D} are bipolar SNs, the result Z represented by Z_N and Z_D is calculated by two bipolar multiplications, $Z_N = X_{1_N}X_{2_N}$ and $Z_D = X_{1_D}X_{2_D}$. Division is implemented by a relatively simple multiplication step $Z = X_1/X_2 = (X_{1_N}X_{2_D})/(X_{1_D}X_{2_N})$ where $Z_N = X_{1_N}X_{2_D}$ and $Z_D = X_{1_D}X_{2_N}$. On the other hand, addition becomes significantly more complicated: $Z = X_1 + X_2 = (X_{1_N}/X_{1_D}) + (X_{2_N}/X_{2_D}) = (X_{1_N}X_{2_D} + X_{1_D}X_{2_N})/(X_{1_D}X_{2_D})$. Note that while bipolar multiplications can be performed without scaling, the standard bipolar adder (Figure 1.3*b*) can only realize

the scaled addition $Z = 0.5(X_1 + X_2)$. This makes the SN format of Canals et al. [18] less attractive than Gaines' ratio format, where the adder can be realized without scaling as in Figure 5.5*c*.

Ratio formats are not only incompatible with the simple conventional SN formats, but have other, less obvious, limitations. Conventional unipolar and bipolar SNs are very tolerant of errors. Each bit of X has the same weight, and flipping a few bits of X has little effect on p_X or $2p_X - 1$. However, if X denotes a ratio p/q, then a bit-flip affecting q can have a very large impact on X's value. The ratio format also increases the inherent redundancy of SNs, since the value represented by p/q is the same as that of kp/kq for any k.

Summarizing the foregoing discussion, the JK flip-flop-based design performs approximate division at low area cost using the standard, and relatively simple, unipolar SN format. (A bipolar version of this division approach is not discussed in [31].) ADDIE-based designs implement the desired division operation X_1/X_2 directly, but they have large area overhead due to the presence of an SNG and a counter, whose size limits the achievable accuracy. Gaines' ratio format provides a simple divider but the corresponding adder and multiplier are much bigger than the corresponding components of Figure 1.3. Compared to Gaines' ratio-format divider, the divider proposed by Canals et al. has slightly less area because basic stochastic components can be used, but like Gaines' ratio format, it too is incompatible with the standard SN formats.

The previously proposed dividers all have at least one of the following shortcomings: they replace X_1/X_2 by a related but approximate form of division; they employ a nonstandard SN format; or they have relatively high cost in terms of area, accuracy or error tolerance. The CORDIV scheme introduced next attempts to avoid these disadvantages by providing a true divider of low cost and high efficiency.

5.2 CORDIV Method

The proposed new division method takes advantage of the fact that the conditional probability of X_1 given X_2 is the quotient of the probability of the joint event X_1X_2 and the probability of X_2 [80]; in other words,

$$p_{X_1|X_2} = p_{X_1X_2} / p_{X_2} \tag{5.1}$$

If X_1 and X_2 are SNs that are correlated by having the maximum number of overlapping 1s for the values they denote, and $p_{X_1} < p_{X_2}$, then $p_{X_1X_2} = p_{X_1}$. In that case, Equation (5.1) reduces to

$$p_{X_1|X_2} = p_{X_1} / p_{X_2} \tag{5.2}$$

which is the target division operation. Hence, to compute p_{X_1} / p_{X_2}, we only need evaluate the probability of $X_1|X_2$, which is the probability of $X_1 = 1$ given that $X_2 = 1$ with the specified correlation property. This $X_1|X_2$ is another SN Z constructed by taking a bit from X_1 whenever the corresponding X_2 bit is 1. This construction is the basis of CORDIV. Note that the correlation-based operations take place entirely inside the divider so that no special correlation requirements are imposed on the input-output parameters. Moreover, the correlation-based operations only need one random number generator, so the area cost is lower than that of a typical divider, which requires two SNGs to provide independent SNs representing X_1 and X_2.

The correlation property of interest is best understood in terms of the *SCC* (SC correlation) metric introduced in Section 3.2 [4]. The *SCC* of two SNs X_1 and X_2 is defined in Equation (3.4). $SCC(X_1, X_2) = 1$ implies X_1 and X_2 have maximum overlap of 1s and 0s, as required for Equation (5.2) to hold.

Example 5.3: The SNs X_1 = 1001010000000000 and X_2 = 1001110000010000 have $SCC(X_1, X_2) = 1$. Hence, $Z = X_1|X_2$=11010, yielding $p_{X_1|X_2} = 3/5$, which is equivalent to $p_{X_1}/p_{X_2} = (3/16)/(5/16)$. □

X_1 and X_2 have a consistent bit-stream length N, but that is not the case for the SN $Z = X_1|X_2$, as Example 5.3 illustrates. The length of Z varies with the number of 1s in X_2. CORDIV therefore includes a "padding" method to extend the length of Z to N. It employs an l-bit padding memory to store $l \geq 1$ bits of $Z = X_1|X_2$, which it subsequently inserts into Z to make the latter's final length equal to N. The padded N-bit SN is denoted by $\hat{Z}$ and the bits in $\hat{Z}$ generated when x_2 is 1 and 0 are called *effective* and *padded* bits, respectively. A similar number length mismatch can also be found in conventional binary arithmetic. For example, k-bit precision binary division using a sequential shift-and-add/subtract circuit may need to pad the result to k bits with trailing 0s when the remainder reaches zero within k clock cycles. On the other hand, if the divider needs more than k clock cycles to generate an exact result, the output is truncated to k bits. While binary numbers are padded by leading or trailing 0s, the CORDIV quotient Z is padded with random bits that have the same probability as Z.

Figure 5.6 shows the basic CORDIV design when l = 1. The select signal of the MUX is controlled by x_2, generating the conditional SN $Z = X_1|X_2$, so that $p_{\hat{Z}} = p_{X_1}/p_{X_2}$ when x_2 = 1. When x_2 is 0, the output bit $\hat{z}$ is the previous result bit stored in the D-type flip-flop (DFF), which has the probability $p_{\text{DFF}} = p_{X_1}/p_{X_2}$. Therefore, the MUX outputs $p_{\hat{Z}} = p_{X_2} \times p_{X_1}/p_{X_2} + (1 - p_{X_2}) \times p_{\text{DFF}} = p_{X_1}/p_{X_2}$. The CORDIV divider also contains two comparators and a random number generator. To ensure that $SCC(X_1, X_2) = 1$, the SNGs generating X_1 and X_2 share a common random number source [4]. This configuration guarantees that whenever x_1 is 1, x_2 is also 1, given the binary form of $p_{X_1} \leq p_{X_2}$. Note that like the ADDIE-based designs, the results of CORDIV saturate to the largest representable number, namely 1, when $p_{X_1} > p_{X_2}$.

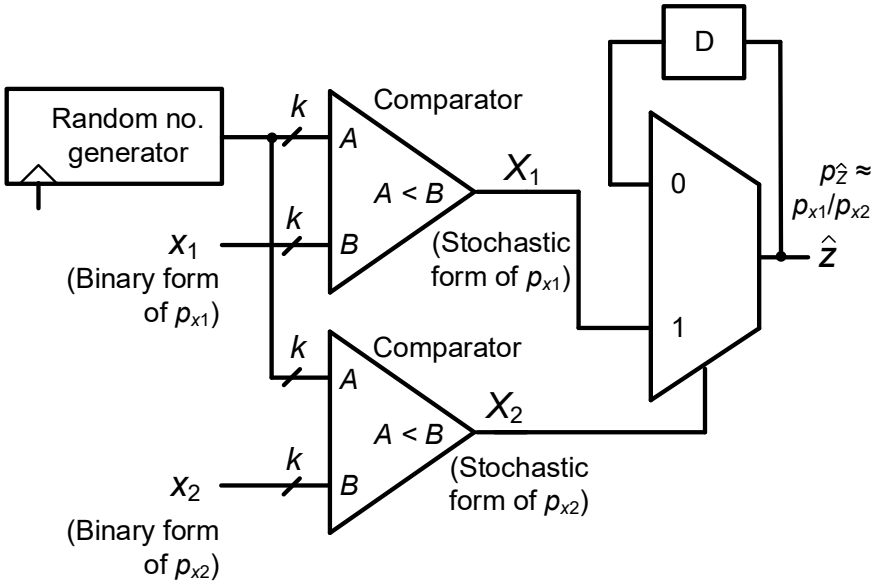

Figure 5.6: Basic design for a CORDIV stochastic divider.

Example 5.4: Continuing with Example 5.3, if X_1 = 1001010000000000 and X_2 = 1001110000010000, the CORDIV divider of Figure 5.6 computes $\hat{Z} =$ 1111011111100000 with $p_{\hat{Z}} = 10/16 = 0.625$, which is a good approximation to $p_Z =$ 0.6. □

A padding memory of size $l > 1$ can be realized in several ways. It can simply be an l-bit shift register that stores l consecutive bits in $\hat{Z}$. The MUX in Figure 5.6 selects the signal from the D-type flip-flop when a bit in the SN X_2 is 0. If X_2 has long runs of consecutive 0s, the padded SN $\hat{Z}$ will repeatedly copy the bit-stream stored in the shift register. To avoid such repetition, the divider should have a larger l-bit padding memory when X_2 has many consecutive 0s. However, a large padding memory may add significantly to cost as l increases. Later, we will see that for short SNs, a larger l is not usually preferable, as l clock cycles are needed to "warm up" and fill the padding memory with effective bits.

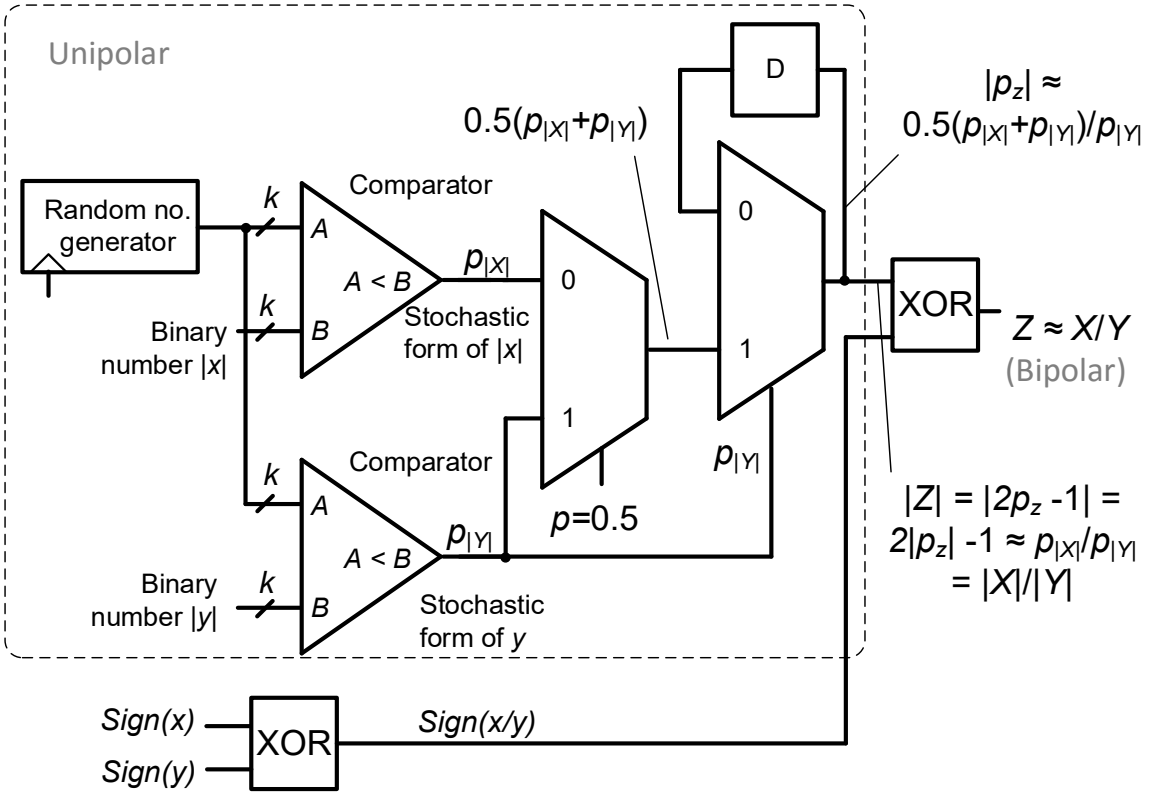

Figure 5.7: Design for a bipolar CORDIV stochastic divider.

The CORDIV design can also be extended to bipolar division at the cost of additional logic to handle signed numbers. Gaines' ADDIE-based bipolar divider in Figure 5.5*b* uses a squaring operation to ensure the two SNs being compared have the same sign. A bipolar CORDIV divider only needs an additional MUX to perform scaling and two XORs to process the sign of the output SN; see Figure 5.7. This bipolar design takes numbers using sign-magnitude binary representation and generates the results of division in bipolar representation.

The padding memory can also be realized by an l-bit register R whose bits can be randomly read. This requires extra random number sources to uniformly select a bit from R and send it to the MUX. This improves the randomness of the output SN Z as new random sources are introduced. However, the output accuracy remains the same since every bit in the padding memory has the same probability. The new random sources also significantly

increase the area cost. We therefore use the simpler shift-register approach in our experiments.

5.3 Experimental Results

Next, we compare the accuracy of the ADDIE-based stochastic dividers covered in Section 5.2 with that of the proposed CORDIV approach. Stochastic dividers using alternative SN formats are not considered because they are incompatible with standard SN formats and are rarely used. We also compare the results generated by the various dividers with the corresponding exact analytic values. To quantify accuracy, we use mean-square error (MSE), so high MSE indicates an inaccurate computation.

We first compare the padded n-bit SN $\hat{Z}$ to the conditional SN $Z = X_1|X_2$, which only includes effective bits. The padded N-bit SN $\hat{Z}$ is generated by the design of Figure 5.6. We randomly sample 1,000 pairs of p_{X_1} and p_{X_2} values in the unit interval [0,1] to

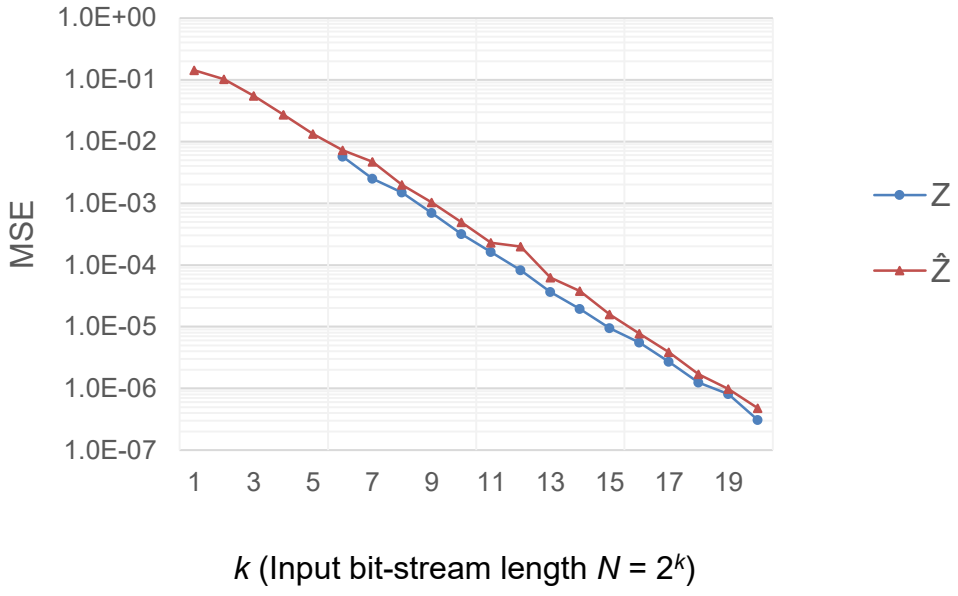

Figure 5.8: Accuracy comparison between the unpadded conditional SN $Z = X_1|X_2$ (blue) and the padded SN $\hat{Z}$ (red).

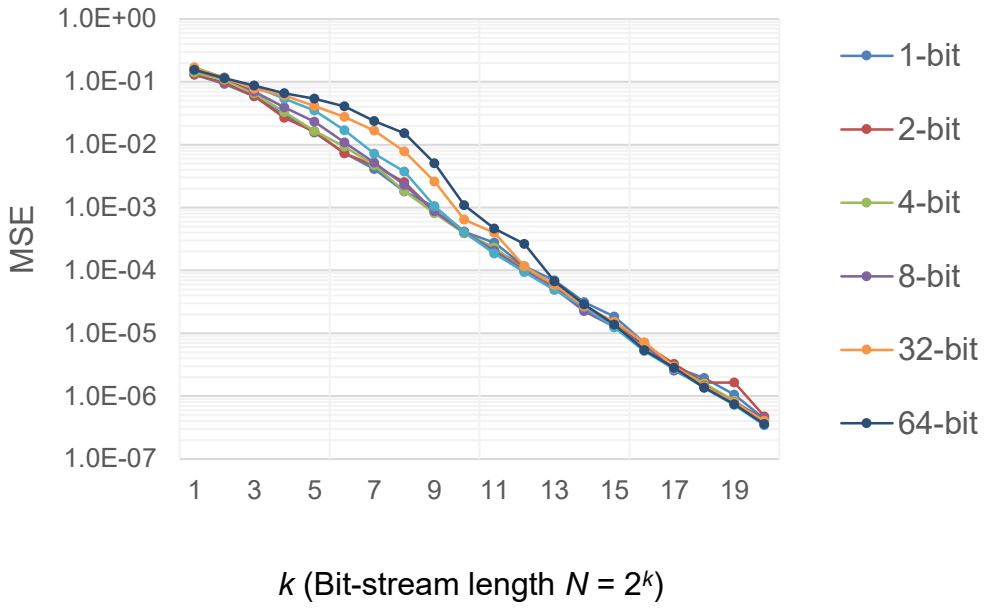

Figure 5.9: Accuracy of the CORDIV divider for different sizes of the padding memory.

generate the SNs X_1 and X_2. This experiment is repeated for different bit-stream lengths N. Figure 5.8 shows the result of this comparison. The X-axis represents the input precision k corresponding to the bit-stream length $N = 2^k$. The Y-axis is our accuracy measure MSE. Since $\hat{Z}$ only consists of effective bits, its accuracy is better than the padded Z, as expected. However, the results indicate that the difference between Z and $\hat{Z}$ is very small, implying that padding has very little impact on accuracy.

Although the unpadded SN $Z = X_1|X_2$ is more accurate, its length changes with X_2. This property makes the unpadded Z less attractive as it cannot interact directly with other n-bit SNs. Figure 5.9 illustrates the effect on MSE of varying the size of the l-bit padding memory. Again, 1,000 pairs of p_{X_1} and p_{X_2} values are randomly sampled to generate X_1 and X_2. The output SN $\hat{Z}$ is obtained by the proposed padding method, and the experiment is repeated for different bit-stream lengths $N = 2^k$. The results show that accuracy is essentially unaffected by the size l of the padding memory when N is large. However, for shorter bit-streams, larger padding memories perform worse than smaller ones due to the fact that the padding memory needs at least l clock cycles to fill with effective bits.

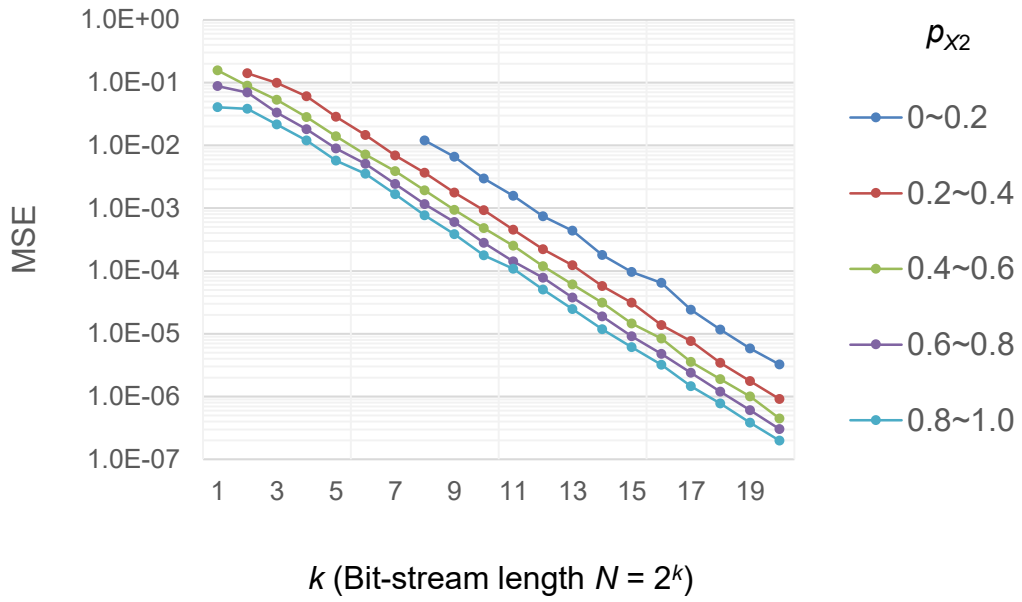

Figure 5.10: Accuracy of the CORDIV divider for different values of p_{X_2}.

For all values of l, the accuracy of $\hat{Z}$ converges to the same value, so the baseline design ($l = 1$), which has lowest cost, is preferred.

As noted in Section 5.2, the padded SN $\hat{Z}$ has fewer effective bits when the divisor X_2 has fewer 1s. We evaluate the accuracy of the proposed CORDIV stochastic divider with different values of the divisor p_{X_2}. Figure 5.9 shows that a 1-bit shift-register is sufficient, so we only consider the design in Figure 5.6. The simulation results in Figure 5.10 demonstrate that, as expected, smaller p_{X_2} has worse accuracy. Lower MSE can be achieved by increasing the bit-stream length N.

The accuracy of the ADDIE-based dividers in Figures 5.1*a* and 5.3 is evaluated in a similar fashion. The data in Figures 5.11 and 5.12 show the accuracy of these designs is bounded by the counter size, as discussed in Section 5.2. Unlike CORDIV, the MSE of the ADDIE-based dividers cannot be made arbitrarily small by increasing bit-stream length. Compared to Gaines' approach, replacing the SNGs by WBGs improves the accuracy, but

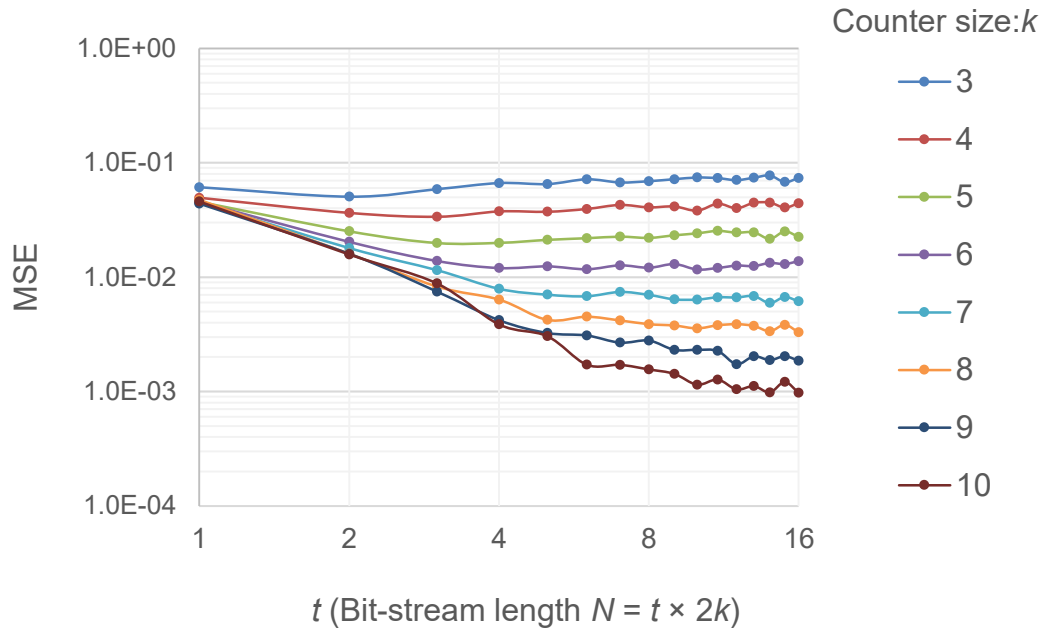

Figure 5.11: Accuracy of Gaines' ADDIE-based divider as counter size k varies.

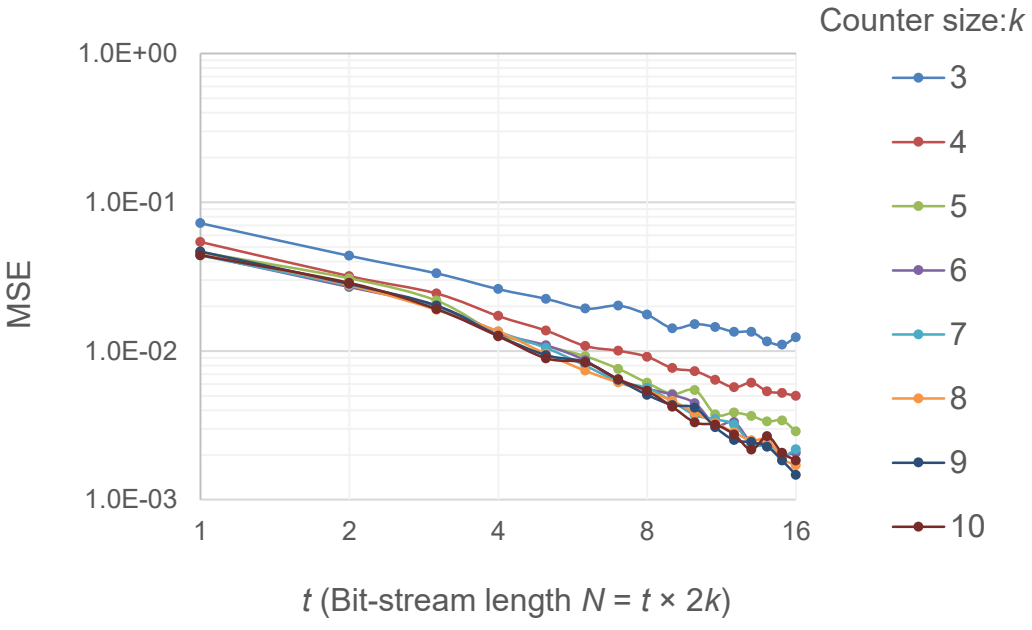

Figure 5.12: Accuracy of Ananth's ADDIE-based divider as counter size k varies.

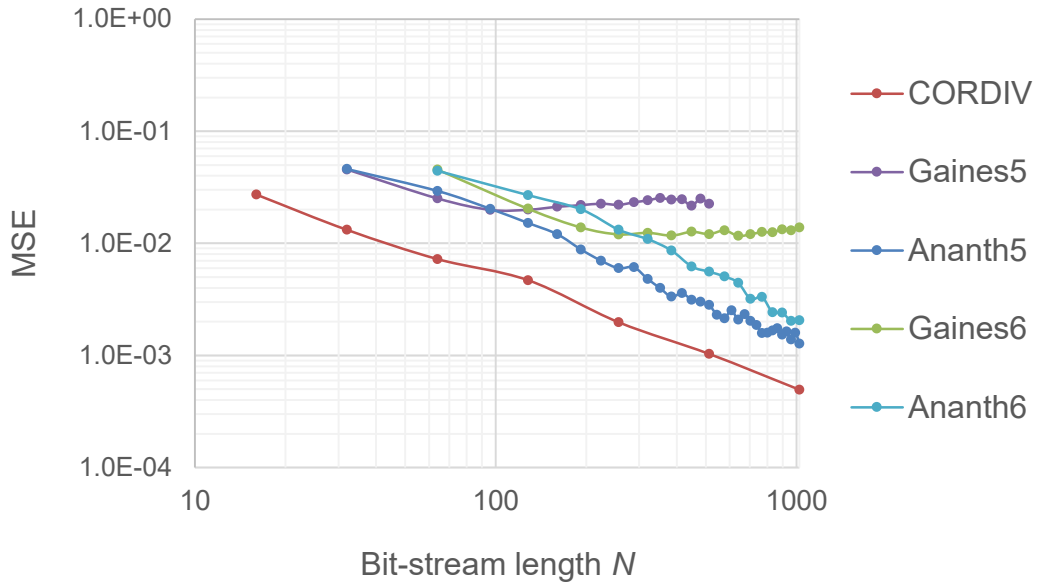

Figure 5.13: Accuracy comparison between CORDIV, and Ananth's and Gaines' dividers with 5- and 6-bit counters.

the results for $k = 3$ and 4 indicate that the accuracy improvement slows down as bit-stream length increases.

Figure 5.13 is a summary comparison of the CORDIV stochastic divider (Figure 5.6) and Ananth's and Gaines' dividers with 5- and 6- bit counters. CORDIV clearly has the best accuracy. Its average MSE is 3.39×10^{-4}, a more than 10x improvement over the other designs' average MSEs 3.3×10^{-3}, 6.296×10^{-3}, 2.3847×10^{-2}, and 1.4069×10^{-2}. Moreover, CORDIV needs only 1.5 SNGs (one random number source and two comparators), one D-type flip-flop, and one MUX, while the ADDIE-based dividers have 3 SNGs (or 3 WBGs), an AND gate, and a k-bit counter. Table 5.1 shows the approximate gate counts of the various dividers with and without including input SNGs. It can been seen that CORDIV is significantly smaller than the others.

Table 5.1: Area comparison between CORDIV, and Ananth's and Gaines' dividers with 5- and 6-bit counters.

Gate count	CORDIV	Gaines5	Gaines6	Ananth5	Ananth6
Including input SNGs	201	629	751	512	625
Excluding input SNGs	9	435	520	396	478

5.4 Summary

Division has long been the "missing operation" in SC applications, reflecting the lack of accurate, low-cost implementations handling standard (unipolar or bipolar) formats. We have addressed this problem with a new approach, CORDIV, which is built on the relation between conditional probability and bit-stream correlation. CORDIV also incorporates a novel padding technique to make the input and output bit-streams the same length. Compared to earlier SC dividers, CORDIV uses less area and is significantly more accurate. It shows that correlation can be a design parameter for division whose role, especially in larger arithmetic systems, deserves further study.

CHAPTER 6
Monotonic Progressive Precision

Chapters 1-5 mostly discussed accuracy and design aspects of SC separately. We now consider them together and present a novel SC design technique called *ASCoMPP* (Accurate Stochastic Computing with Monotonic Progressive Precision). This general-purpose design technique implements any stochastic arithmetic function with high accuracy. We first prove that very accurate results are obtained when the random numbers used to generate interacting SNs are carefully sampled. The sampling process ensures that accuracy increases steadily with bit-stream length, a very desirable property we term monotonic progressive precision (MPP). We further show how simple counting sequences can achieve good MPP at relatively low cost. Finally, we present analytical and experimental data which demonstrate that, with appropriate bit-stream types and lengths, *ASCoMPP* produces results that are both highly accurate and have good MPP.

6.1 Exact Stochastic Computing

The inaccuracy of SN-based computation has long been considered a major factor limiting SC to low-precision applications. However, it is known that for multiplication and certain bit-stream formats, accurate or even exact SNs can be produced. Gupta and Kumaresan noticed that the inaccuracy due to random fluctuations can be essentially eliminated by deriving SNs from pseudonoise (PN) sequences [38]. A k^{th}-order PN sequence (also called an m-sequence) is a bit-stream of maximum period $2^k - 1$ generated by certain types of k-bit LFSRs. PN sequences pass many randomness tests and their theory is well understood [36]. A free-running LFSR goes through all its $2^k - 1$ states (the all-0

state is missing) in a deterministic but random-like order. Note the similarity to a free-running k-bit binary counter, which is also deterministic but goes through all 2^k states.

As introduced in Section 5.1, the SNG proposed in [38] is a *weighted binary generator* (WBG). It converts a k-bit binary number $x_1 x_2 \ldots x_k$ to a (unipolar) SN X of length $2^k - 1$ with an extremely small error due the missing state which makes the numbers of 0s and 1s slightly different. An LSFR is easily modified to include the all-0 state and eliminate this error. Although Gupta and Kumaresan do not include the all-0 state in their design, in the rest of the chapter, we assume that LFSRs have been modified in this way.

The SN X from a WBG is guaranteed to have the exact value of $\hat{p}_X = p_X = \sum_{i=1}^{k} 0.5^i x_i$ after 2^k bits have been generated. As discussed above, the (modified) LFSR goes through every possible state. Therefore, each stage or flip-flop r_i of the LFSR produces a bit-stream R_i with equal numbers of 0s and 1s. Hence R_i is the source of an SN of length 2^k with the exact value $p_{R_i} = 0.5$. Unlike the standard SNG of Figure 1.4*a*, which relies on number comparison, the WBG decomposes the SN generated by the LFSR into k SNs W_1, $W_2,\ldots,W_k$ with probabilities $p_{W_i} = 0.5^i$, for $i = 1,2,\ldots,k$. The WBG's output SN X is formed by combining these k SNs via an OR gate, as shown in Figure 5.2. Suppose, for example, the WBG contains a 3-bit LFSR that generates the three bit-streams $R_1 = 01110010$, $R_2 = 10111000$, and $R_3 = 01011100$, which make $W_1 = 01110010$, $W_2 = 10001000$, and $W_3 = 00000100$. If a binary number $x_1x_2x_3 = 011$ representing 0.375 is applied to the WBG, it outputs $X = 10001100$, an exact SN representation of the binary input. Since the 1s in the W_i's are always non-overlapping, they are added exactly by the OR gate in Figure 5.2. Note that if a unmodified LFSR is used, so the all-0 state is excluded, the LFSR generates $R_1 = 0111001$, $R_2 = 1011100$, and $R_3 = 0101110$. The output SN then becomes $X = 1000110$, which has the small error $|3/7 - 3/8| = 3/56$.

Using WBGs, Gupta and Kumaresan also demonstrated that not only can SNs be generated accurately, but they can be accurately multiplied in SC fashion [38]. The idea is to use two WBGs to generate two exact SNs X and Y, and then use a stochastic multiplier

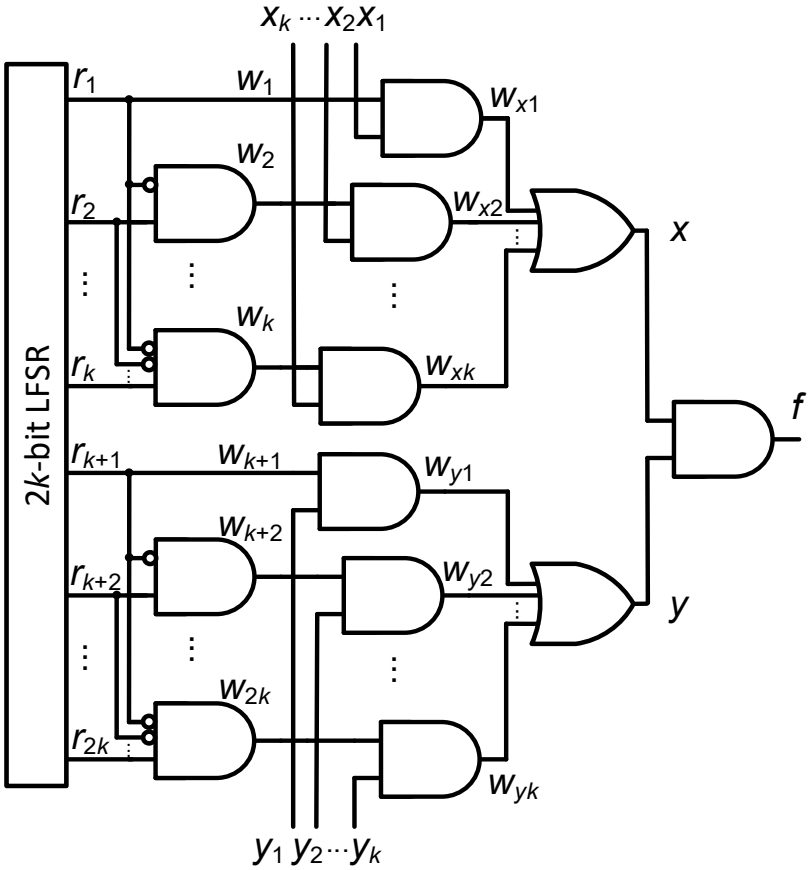

Figure 6.1: The k-bit exact stochastic multiplier [38].

(an AND gate) to compute $p_F = p_X \times p_Y$. As Figure 6.1 illustrates, instead of two individual k-bit LFSRs, the two WBGs share a single $2k$-bit LFSR. This configuration ensures that X and Y have exact values when their bit-stream length is 2^{2k}. As X and Y are generated by two disjoint sets of independent SNs (R_1, R_2, …, R_k) and (R_{k+1}, R_{k+2}, …, R_{2k}), they are statistically independent or uncorrelated. From the definition of independence of two SNs [44], we see that the two 2^{2k}-bit binary sequences $X = (X(1), X(2), \ldots, X(2^{2k}))$ and $Y = (Y(1), Y(2), \ldots, Y(2^{2k}))$ satisfy

$$\sum_{i=1}^{2^{2k}} X(i)Y(i) = \frac{\sum_{i=1}^{2^{2k}} X(i) \times \sum_{i=1}^{2^{2k}} Y(i)}{2^{2k}}$$

As both $p_X = \hat{p}_X = \sum_{i=1}^{2^{2k}} X(i)/2^{2k}$ and $p_Y = \hat{p}_Y = \sum_{i=1}^{2^{2k}} Y(i)/2^{2k}$ are exact, the result of ANDing X and Y has $\hat{p}_F = \sum_{i=1}^{2^{2k}} X(i)Y(i)/2^{2k} = \sum_{i=1}^{2^{2k}} X(i) \times \sum_{i=1}^{2^{2k}} Y(i)/2^{2k} \times 2^{2k} = \hat{p}_X \times \hat{p}_Y = p_X \times p_Y = p_F$. Since $E_F = \mathbb{E}[(\hat{p}_F - p_F)^2] = 0$ always holds, the multiplier gives exact results.

Example 6.1: A 2-bit version of the multiplier of Figure 6.1 has a 4-bit LFSR and generates the bit-streams shown in Table 6.1 when the binary forms of X and Y are both 11, representing 0.75. We then have $\sum_{i=1}^{2^{2k}} X(i) = \sum_{i=1}^{2^{2k}} Y(i) = 12$ and $\sum_{i=1}^{2^{2k}} X(i)Y(i) = 9$. As $k = 2$ in this example, we find $9 = (12 \times 12)/2^4$, implying the estimated value of F, $\hat{p}_F = 9/16$, which is exactly the result of 0.75×0.75. □

Despite its successful use to produce an accurate stochastic multiplier, this design approach has not been extended to other arithmetic operations. We propose such an extension here to any stochastic arithmetic function that is realizable by a Boolean logic circuit. It is well-known [1][7] that every Boolean function $f(x_1, x_2, \ldots, x_n)$ maps to a unique stochastic function $p_F(p_{X_1}, p_{X_2}, \ldots, p_{X_n})$, as specified by Equation (4.3). For example, if f is the two-input AND function, then Equation (4.3) implies

Table 6.1: Bit-streams for the exact 2-bit multiplication of Example 6.1.

Line *l*	Bit-stream on *l*	Numerical value
r_1	1010 1111 0000 1100	8/16
r_2	0101 1110 0001 1001	8/16
r_3	1011 1100 0011 0010	8/16
r_4	0111 1000 0110 0101	8/16
w_{x1}	1010 1111 0000 1100	8/16
w_{x2}	0101 0000 0001 0001	4/16
w_{y1}	1011 1100 0011 0010	8/16
w_{y2}	0100 0000 0100 0101	4/16
x	1111 1111 0001 1101	12/16
y	1111 1100 0111 0111	12/16
f	1111 1100 0001 0101	9/16

$$p_F(p_{X_1}, p_{X_2}) = f(0,0)(1-p_{X_1})(1-p_{X_2}) + f(0,1)(1-p_{X_1})p_{X_2} + f(1,0)p_{X_1}(1-p_{X_2}) + f(1,1)p_{X_1}p_{X_2} = p_{X_1}p_{X_2}$$

It is not true, however, that every arithmetic function is a stochastic function directly implementable by a logic circuit. For example, the unscaled sum $F = X + Y$ cannot be realized because the value of F can fall outside the unit interval [0,1] which is the range of (unipolar) SNs. Several recent studies of stochastic circuit synthesis [1][7][74] have addressed this issue. They show that by introducing new stochastic constants, un-synthesizable arithmetic function can be approximated or scaled to make them realizable. The scaling of addition by 0.5 in a multiplexer is an obvious example. To simplify our discussion, we only consider arithmetic functions that can be realized directly by logic circuits.

Let B be an m-variable Boolean function $f(x,y,\ldots,z)$ that realizes the stochastic function $F(p_X, p_Y, \ldots, p_Z)$. For example, f could be AND and F could be multiply.

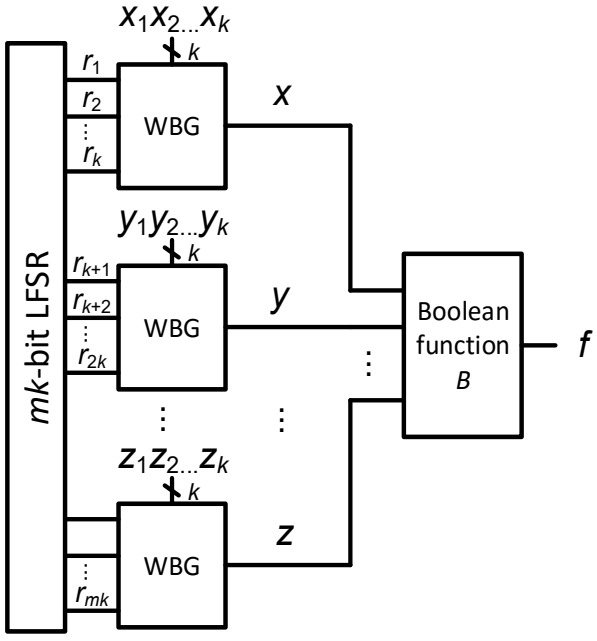

Figure 6.2: *ASC* design for exact implementation of $F(p_X, p_Y, \ldots, p_Z)$.

Figure 6.2 shows the proposed method called *ASC* (Accurate Stochastic Computing), a preliminary version of the aforementioned *ASCoMPP* method for exact computation of *F*. The SNs representing p_X, p_Y, ..., p_Z are 2^{mk}-bit sequences generated by m separate k-bit WBGs fed by an mk-bit (modified) LFSR. The stochastic behavior F of B is $p_F = F(p_X, p_Y, \ldots, p_Z)$. The following theorem asserts that, when the full 2^{mk}-bit input sequences are applied, the output bit-stream representing p_F is such that $\hat{p}_F = p_F$, i.e., the estimated value of F is exact.

Theorem 6.1: Let X, Y, …, Z be m SNs generated by m WBGs with m k-bit independent inputs, $x_1x_2...x_k$, $y_1y_2...y_k$, …, $z_1z_2...z_k$ using the *ASC* design of Figure 6.2. Let B be the Boolean function $f(x,y,...,z)$ that realizes the stochastic function $F(p_X, p_Y, \ldots, p_Z)$. The estimated value $\hat{p}_F$ of the output SN F when the bit-stream length $N = 2^{mk}$ is exact, and is given by

$$\hat{p}_F = p_F = F(p_X, p_Y, \ldots, p_Z)$$

where $p_X = \sum_{i=1}^{k} 0.5^i x_i$, $p_Y = \sum_{i=1}^{k} 0.5^i y_i$, …, $p_Z = \sum_{i=1}^{k} 0.5^i z_i$.

Proof: Let SN X = ($X(1)$, $X(2)$, …, $X(2^{mk})$), so that $\hat{p}_X = \sum_{i=1}^{2^{mk}} X(i)/2^{mk}$, $\hat{p}_Y = \sum_{i=1}^{2^{mk}} Y(i)/2^{mk}$, and $\hat{p}_Z = \sum_{i=1}^{2^{mk}} Z(i)/2^{mk}$. The output bit-stream F = ($F(1)$, $F(2)$, ... , $F(2^{mk})$) has the numerical value $\hat{p}_F = \sum_{i=1}^{2^{mk}} F(i)/2^{mk}$. Now $F(i) = f\big(X(i), Y(i), \ldots, Z(i)\big)$. As X, Y, ..., and Z do not share common random sources, all pairs of them are statistically independent or uncorrelated. The stochastic behavior of the Boolean function B with independent inputs can be characterized as some function of the input signal probabilities, p_X, p_Y,…, and p_Z, i.e. $p_F = F(p_X, p_Y, \ldots, p_Z)$. □

Without loss of generality, we single out X for discussion, and show that when the bit-stream length is 2^{mk}, the estimated value $\hat{p}_X = \sum_{i=1}^{2^{mk}} X(i)/2^{mk}$ will be $p_X = \sum_{i=1}^{k} 0.5^i x_i$. Let SN X^* = ($X^*(1)$, $X^*(2)$, …, $X^*(2^k)$) be a 2^k bit-stream generated by the WBG design of Figure 5.2, so that $\hat{p}_{X^*} = \sum_{i=1}^{2^k} X^*(i)/2^k$. The WBG ensures the estimated value

is exact, i.e. $\hat{p}_{X^*} = \sum_{i=1}^{k} 0.5^i x_i = p_X$. This is not obvious as the pseudo-random inputs of X's WBG do not come from a k-bit (modified) LFSR; instead, they come from an mk-bit (modified) LFSR. Since $r_1 r_2 \ldots r_k$ repeat every pattern 2^{mk-k} times, the number of 1s in X will be 2^{mk-k} times of the number of 1s in X^*. Therefore, we have

$$\hat{p}_X = \frac{1}{2^{mk}} \cdot \sum_{i=1}^{2^{mk}} X(i) = \frac{1}{2^{mk-k} \cdot 2^k} \cdot \left(2^{mk-k} \cdot \sum_{i=1}^{2^k} X^*(i) \right) = \frac{1}{2^k} \cdot \sum_{i=1}^{2^k} X^*(i) = \sum_{i=1}^{k} 0.5^i x_i$$
$$= p_X$$

Similarly, we have $\hat{p}_Y = p_Y$, …, $\hat{p}_Z = p_Z$. In other words, the SNs X, Y, …, Z are exact when bit-stream length is 2^{mk}. Since

$$\hat{p}_F = \frac{1}{2^{mk}} \cdot \sum_{i=1}^{2^{mk}} F(i) = \frac{1}{2^{mk}} \cdot \sum_{i=1}^{2^{mk}} f\big(X(i), Y(i), \ldots, Z(i)\big)$$
$$= F\left(\sum_{i=1}^{2^{mk}} X(i) \Big/ 2^{mk}, \sum_{i=1}^{2^{mk}} X(i) \Big/ 2^{mk}, \ldots, \sum_{i=1}^{2^{mk}} X(i) \Big/ 2^{mk} \right)$$
$$= F(\hat{p}_X, \hat{p}_Y, \ldots, \hat{p}_Z)$$

we have $\hat{p}_F = F(\hat{p}_X, \hat{p}_Y, \ldots, \hat{p}_Z) = F(p_X, p_Y, \ldots, p_Z) = p_F$, which indicates that the output SN F is exact, $E_F = \mathbb{E}[(\hat{p}_F - p_F)^2] = 0$ when the bit-stream length is 2^{mk}. □

The preceding proof is illustrated by Table 6.1 of Example 6.1. When the binary input x_1x_2 = 11 representing 0.75 is sent to a 2-bit WBG to generate the SN X^*, the bit-stream length is 2^2 = 4 and the number of 1s in X^* is 3. This gives the exact result $\hat{p}_{X^*} = 3/4$. Using an *ASC* design, a 2-bit multiplication operation requires a 4-bit LFSR so the length of X is 2^4. As Table 6.1 shows, r_1r_2 is used to generate X^*, and all possible combinations of r_1r_2 repeat $2^{(4-2)} = 4$ times, so the number of 1s in X is four times that in X^*. Therefore, we get the exact result $\hat{p}_X = 12/16 = \hat{p}_{X^*} = 3/4$. Similarly, $\hat{p}_Y$ is also exact when the bit-stream length is 16. Since the interacting SNs X and Y do not share a

random source, i.e. no r_i is used more than once, X and Y are independent and the AND gate realizes multiplication. Hence, $\hat{p}_F = \hat{p}_X \times \hat{p}_Y = 3/4 \times 3/4 = 9/16$.

Example 6.2: Consider the function $f = a_1 b_1 \bar{r} + a_2 b_2 r$, which implements the scaled inner-product operation $F = 0.5(A_1 B_1 + A_2 B_2)$ when $p_R = 0.5$. The corresponding *ASC* design has five SNs, A_1, A_2, B_1, B_2 and R. Note that the stochastic constant R is a special case whose WBG is simply a pass-through wire. Let the precision k be 2. The size of the LFSR is $4 \times 2 + 1 = 9$, indicating that the bit-stream length needed for an exact result is $2^9 = 512$. □

The operation of a conventional SNG can be seen as a Monte Carlo sampling process [5]. It is also helpful to view *ASC* in terms of Monte Carlo sampling, where the samples are mk-bit vectors from an m-dimensional space S whose axes are the input variables of B in Figure 6.2. Each axis is divided in two by the value of its variable, so that S can be partitioned into 2^m blocks. A block represents B's minterms or maxterms in a fashion similar to a Karnaugh map, but the blocks need not be the same size. Let M_B denotes the blocks corresponding to the minterms of B, so M_B represents the cases when B's output f is 1. It is not hard to see that the stochastic value of f is given by

$$p_F = (Volume\ of\ M_B)/(Volume\ of\ S)$$

The estimated $\hat{p}_F$ can be obtained by sampling points in S.

$$\hat{p}_F = (No.of\ samples\ in\ M_B)/(Total\ no.of\ samples)$$

which is a typical Monte Carlo approach. With *ASC*, the samples are generated by an mk-bit (modified) LFSR. When the LFSR traverses a full cycle of its states, the entire space S is uniformly sampled, and $\hat{p}_F$ is accurate to mk-bit precision, making $\hat{p}_F = p_F$.

Example 6.3: Consider again the exact stochastic multiplier of Figure 6.1 with the AND function $f = xy$. Its sample space S has two dimensions x and y, as depicted in Figure 6.3.

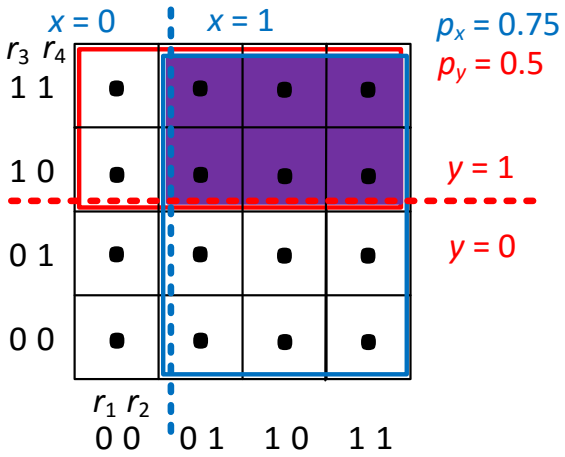

Figure 6.3: The sample space S of the 2-bit exact stochastic multiplier in Example 6.3.

Each axis is partitioned into two sub-spaces according to the corresponding probability values p_X and p_Y. Figure 6.3 shows the case when $p_X = 0.75$ and $p_Y = 0.5$, represented by blue and red rectangles, respectively. The red rectangle comprises the cases where $y = 1$ and blue rectangle indicates where $x = 1$. These partitions ensure that whenever a point is sampled, the probability of falling into the red (or blue) rectangle is $p_X = 0.75$ (or $p_Y = 0.5$) as its area is 0.75x (or 0.5x) the overall area. Since the LFSR $r_1r_2r_3r_4$ is viewed as the sample source, the dots in Figure 6.3 indicates the sample points. The correspondence between these sample points and x values is given explicitly for $p_X = 0.75$ by Table 6.1; the correspondence between the sample points and y values is obtained similarly for $p_Y = 0.5$. The AND operation of stochastic multiplication gives an M_B which is the intersection of the blue and red rectangles, i.e., the purple region. When the bit-stream length is 16, the LFSR cycles through all its states, and all the uniformly distributed points shown in Figure 6.3 are sampled. We then have p_F = (*Area of* M_B) / (*Area of the whole 2-dimensional space* S) = 6/16, which is the exact result for 0.75×0.5 = 0.375. □

Example 6.4: Figure 6.4 gives another instance of *ASC* design, this time to implement the 2-bit scaled addition $F = 0.5(X + Y)$ exactly. Figure 6.5 illustrates the corresponding sample space S in the manner of Figure 6.3. With $x = 11$ and $y = 10$, denoting 0.75 and 0.5, respectively, and a stochastic constant $p_{R1} = 0.5$, S is partitioned into eight sub-spaces

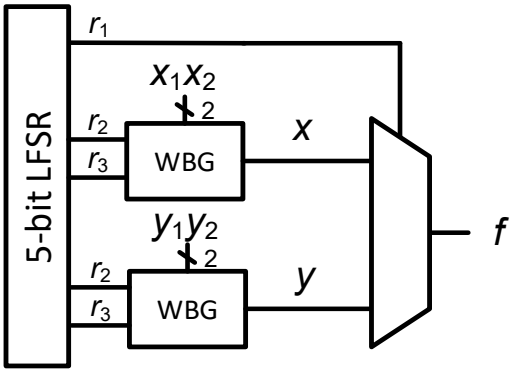

Figure 6.4: Two-bit exact stochastic scaled adder.

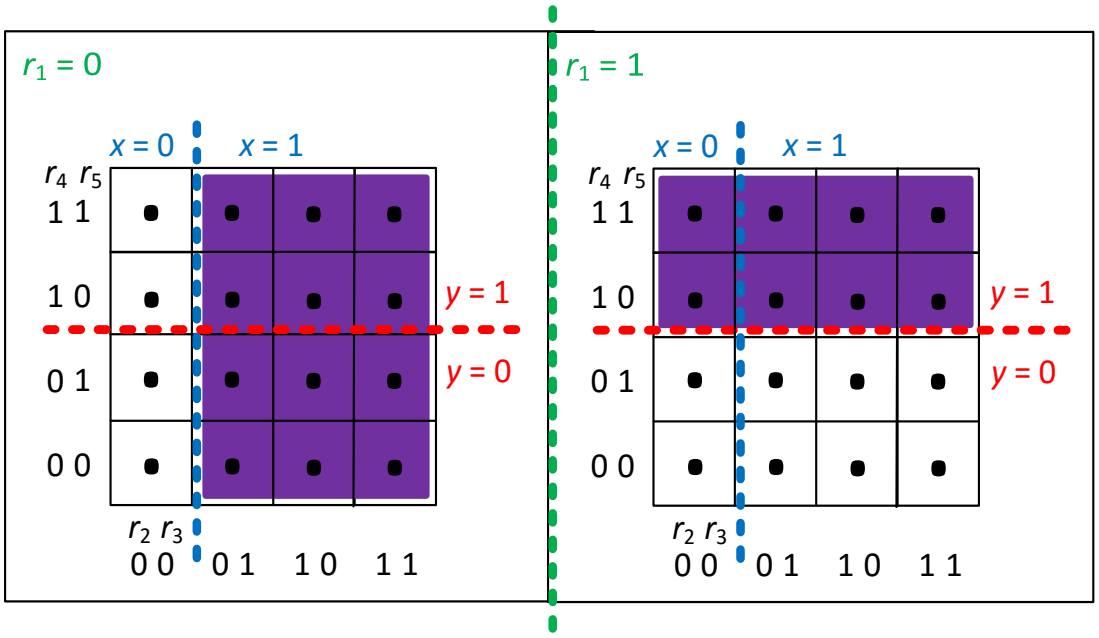

Figure 6.5: Sample space of the 2-bit stochastic scaled adder in Example 6.4.

suggested by the blue and red dashed lines. The Boolean function for scaled add is $f = x\overline{r_1} + yr_1$, so its minterms are r_1xy = 010, 011, 101, 111, and form the shaded part M_B of Figure 6.5. If the bit-stream length is $2^5 = 32$, then p_z = (*Volume of* M_B) / (*Volume of the whole 3-dimensional space)* is estimated by 20/32, which is exactly 0.5 × (0.5 + 0.75). □

6.2 Progressive Precision

The previous section presented a way to achieve exact results for any stochastic arithmetic function. An exact output SN F is only obtained after the mk-bit LFSR driving the computation goes through a maximum-length sequence of 2^{mk} clock cycles. F's accuracy is not guaranteed if the output SN is sampled for fewer clock cycles, e.g., to generate a faster, but less precise result. In general, the value of F can fluctuate up and down over a wide range as the computation proceeds. This section presents a method (*ASCoMPP*), which ensures that *F's* accuracy increases steadily and predictably over time.

In general, progressive precision (PP) refers to a computation whose results improve with more computation time [2]. It implies that the first 2^{k^*} bits of a 2^k-bit result X provide a good approximation to X's final value. For example the bit-stream X = 01011100 with p_X = 0.5 is approximated well by sampling its first four bits 0101; this illustrates "good" progressive behavior since $\hat{p}_{X^*} = p_X$. On the other hand, Y = 00011101, also with p_Y = 0.5, has "bad" progressive behavior because its first four bits 0001 have value 0.25, which is a poor approximation to p_Y.

Alaghi and Hayes show that SNs with good progressive behavior can be obtained by using low-discrepancy sequences as the SNGs' pseudo-random number sources [5]. To measure the quality of PP, they define the *bit-error* of an n-bit SN X as $\varepsilon_X = n \times |\hat{p}_{X^*} - p_X|$. An SN is called l-PP if the bit-error of its initial sub-sequence of length 2^i is at most l for all i [5]. For example, X = 1011111100001111 has the exact value p_X = 11/16. X's initial subsequences of length 2, 4, 8, and 16 are 10, 1011, 10111111, and 1011111100001111, respectively. The corresponding bit-errors are 0.375, 0.25, 1.5, and 0. The SN X has 1.5-PP since its maximum bit-error is 1.5. Note that the WBG ensures that the bit-error of the SN *w*ith length 2^k is zero.

The l-PP metric only considers the PP of an isolated SN. We now propose a different PP measure $\varepsilon_X^{(i)}$ which links the precision values of the input binary number and

the output stochastic bit-stream. Intuitively, $\varepsilon_X^{(i)}$ indicates how far the first 2^i -bit subsequence of SN is from having the expected number of 1s needed for *i*-bit precision.

Definition 6.1: Let the 2^k-bit stream $X = (X(1), X(2),\ldots, X(2^k))$ be an SN generated from a *k*-bit binary number $x_1x_2\ldots x_k$. Then, for any $i \leq k$, *X*'s *stage-i bit-error* is defined as

$$\varepsilon_X^{(i)} = 2^i \times \left|\hat{p}_X^{(i)} - p_X^{(i)}\right| \tag{6.1}$$

where $\hat{p}_X^{(i)} = \sum_{j=1}^{2^i} X(j)/2^i$ and $p_X^{(i)} = \sum_{j=1}^{i} 0.5^j x_j$. □

In this definition, $\hat{p}_X^{(i)}$ and $p_X^{(i)}$ are the estimated and expected values of *X* to *i*-bit precision, respectively. Equation (6.1) can be rewritten in the following equivalent form:

$$\varepsilon_X^{(i)} = \left|\sum_{j=1}^{2^i} X(j) - \sum\nolimits_{j=1}^{i} 2^{i-j} x_j\right| \tag{6.2}$$

Equation (6.2) indicates that $\varepsilon_X^{(i)}$ represents the difference between the number of 1s in the first 2^i -bit subsequence of *X* and the expected number of 1s needed to specify *X* with *i*-bit precision. This equation gives a measure of the number of erroneous bits in an SN.

Example 6.5: Let X = 1110110011011011 be a 16-bit SN derived from the 4-bit binary number $x_1x_2x_3x_4$ = 1011 denoting the decimal number 0.6875. *X*'s initial 2-, 4-, 8- and 16-bit sub-sequences are 11, 1110, 11101100 and 1110110011011011, respectively. Its stage-*i* expected estimated, expected and error values derived from Definition 6.1 are as follows:

i	$\hat{p}_X^{(i)}$	$p_X^{(i)}$	$\varepsilon_X^{(i)}$
1	1.0	0.5	1
2	0.75	0.5	1
3	0.625	0.625	0
4	0.6875	0.6875	0

The calculations for the case $i = 2$ are:

$$\hat{p}_X^{(2)} = \sum_{j=1}^{4} X(j)/4 = 1/4+1/4+1/4+0/4 = 0.75$$

$$p_X^{(2)} = \sum_{j=1}^{2} 0.5^j x_j = 0.5 \times 1 + 0.5^2 \times 0 = 0.5$$

$$\varepsilon_X^{(2)} = 2^2 \times |0.75 - 0.5| = 1$$

This implies that if generation of the SN terminates after $2^i = 4$ bits with $X = 1110\ldots$, its estimated value $\hat{p}_X^{(2)}$, which is 0.11 in binary, has 1 bit less precision than the first $i = 2$ bits of x_1x_2, whose value is 0.10. □

We now formalize progressive precision in terms of the stage-i bit-error concept.

Definition 6.2: An 2^k-bit SN $X = (X(1), X(2),\ldots, X(2^k))$ generated from a k-bit binary number $x_1x_2\ldots x_k$ has *monotonic progressive precision* (MPP) if the stage-i bit-errors decrease monotonically with i, i.e. $\varepsilon_X^{(i+1)} \leq \varepsilon_X^{(i)}$ for all i. We say that X has *strict* MPP if $\varepsilon_X^{(i)} = 0$ for all i. □

The SN X in Example 6.5 has MPP, but not strict MPP. To achieve accurate SC with very good progressive precision, we want SNs that have strict MPP because MPP only ensures the error bits decrease monotonically while strict MPP ensures exact results to i-bit precision. To this end, we re-examine the WBG design of Figure 5.2. A k-bit WBG ensures that the stage-k bit-error is always zero, but it does not guarantee strict MPP, or even non-strict MPP. For instance, if X is replaced by Y^* = 1111011011101010 in Example 6.5, the expected values are unchanged, but the estimated values become 1, 1, 0.75, and 0.6875. The corresponding stage-i bit-errors $\varepsilon_X^{(i)}$ are 1, 2, 1 and 0, so Y does not have MPP. This example shows that SNs generated by the WBG design do not always has MPP.

The preceding discussion indicates that WBG-based SNs do not necessarily have MPP. To achieve MPP, the pseudo-random sequences driving the WPG must be carefully chosen. Because the WBG decomposes the output SN generated by the LFSR into k SNs W_1, W_2,…,W_k with probabilities $p_{W_i} = 0.5^i$, for $i = 1,2,\ldots,k$, the output SN has good MPP if the W_i's also have good MPP. A simple method to achieve this is to use ordinary counting sequences, as they are low-discrepancy sequences, which have been shown to achieve good PP [5]. Although good PP does not always lead to good MPP, ordinary counting sequences have patterns of regularly repeating 1s that gives good MPP. This observation suggests replacing the LFSR in *ASC* by a counter, as shown in Figure 6.6, whose accuracy naturally tends to increase over time.

Theorem 6.2: The SN $X = (X(1), X(2), \ldots, X(2^k))$ generated by the circuit of Figure 6.6, which substitutes a k-bit counter for the LFSR in the WBG of Figure 5.2, has strict MPP.

Proof: Since the bit-streams on the c_i's in Figure 6.6 are counting sequences, the resulting SNs W_i and W_{Xi} are both exact if their length is 2^i, implying that $\sum_{j=1}^{2^i} X(j)/2^i = \sum_{j=1}^{i} 0.5^j x_j$, i.e., $\hat{p}_X^{(i)} = p_X^{(i)}$. Therefore, $\varepsilon_X^{(i)} = 2^i \times \left|\hat{p}_X^{(i)} - p_X^{(i)}\right| = 0$ for all i. □

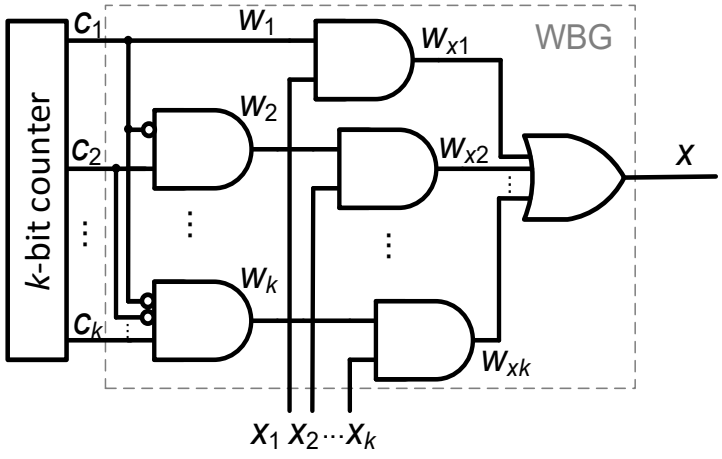

Figure 6.6: A k-bit weighted binary SNG with strict MPP.

We conclude that replacing a k^{th}-order PN sequence by a k-bit counting sequence endows the result with strict MPP. We now generalize this feature to match that of the exact stochastic function generator in *ASC* (Figure 6.2). As m SNs, X, Y, …, Z, are interacting to produce $F = (F(1), F(2),\ldots, F(2^{mk}))$, the stage-$i$ bit-error of F becomes

$$\varepsilon_F^{(i)} = 2^{mi} \times \left| \hat{p}_F^{(mi)} - p_F^{(mi)} \right|$$

where $p_F^{(mi)} = F\left(p_X^{(i)}, p_Y^{(i)}, \ldots, p_Z^{(i)}\right)$, which is the result of computing m separate i-bit precision SNs. As the m SNs interact, F's initial sub-sequence length must be 2^{mi} to give exact results. For example, Gupta and Kumaresan's k-bit multiplier generates exact results when the output bit-stream length is 2^{2k}.

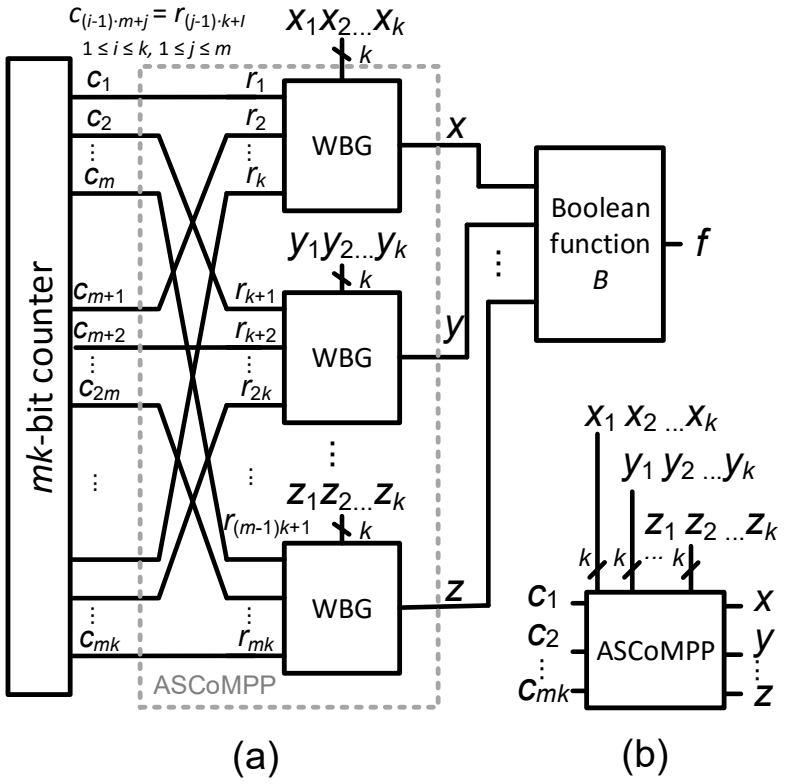

Figure 6.7: (a) *ASCoMPP* design for accurate stochastic computing with strict MPP, and (b) its symbol.

Theorem 6.3: The 2^{mk}-bit SN $F = (F(1), F(2),\ldots, F(2^{mk}))$ generated by the *ASCoMPP* circuit in Figure 6.7 has strict MPP.

Proof: To determine whether strict MPP is present, we need to calculate the numerical values of F's initial sub-sequences of length 2^{mi} for all $1 \le i \le k$. For $i = 1$, the first 2^m sequences are generated when c_{m+1}-c_{km} are all 0. The circuit is then equivalent to one with $k = 1$, so $\varepsilon_F^{(1)} = 2^m \times \left|\sum_{j=1}^{2^m} F(j)/2^m - p_F^{(1)}\right| = 0$. Similarly, when $i = 2$, the initial 2^{2m} sequences are generated when c_{2m+1}-c_{km} are zero, and the circuit is equivalent to one with $k = 2$, so $\varepsilon_Z^{(2)} = 0$. Eventually, we get $\varepsilon_F^{(1)} = \varepsilon_F^{(2)} = \cdots = \varepsilon_F^{(k)} = 0$, indicating that F has strict MPP. □

Example 6.6: Continuing with Example 6.3 and replacing the LFSR in Figure 6.1 by a counter in the *ASCoMPP* fashion, yields the sample sequence shown by the boldface numbers inside the cells of Figure 6.8. These numbers indicate the generation order of the samples, and Table 6.2 gives the corresponding bit-streams. The output f is 1 if and only if the sample points fall in the purple area at the intersection of red and blue rectangles. The binary inputs are $x_1x_2 = 11$ and $y_1y_2 = 10$, so the first and second expected values are 0.25

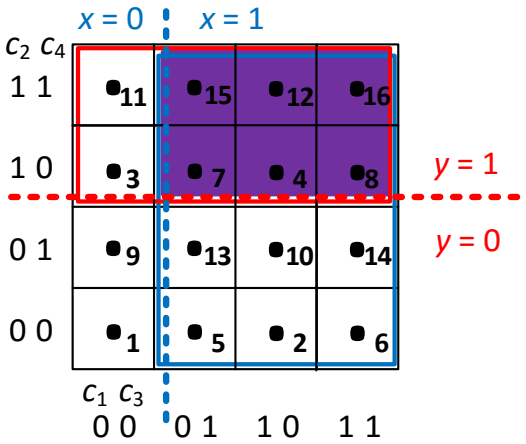

Figure 6.8: Counting sequence of the 2-bit exact stochastic multiplier in Example 6.3.

Table 6.2: Bit-streams for the 2-bit multiplication using *ASCoMPP*.

Line *l*	Bit-stream on *l*	Numerical value	
		Initial 16 bits	Initial 4 bits
c_1	0101 0101 0101 0101	8/16	2/4
c_2	0011 0011 0011 0011	8/16	2/4
c_3	0000 1111 0000 1111	8/16	2/4
c_4	0000 0000 1111 1111	8/16	2/4
r_1	0101 0101 0101 0101	8/16	2/4
r_2	0000 1111 0000 1111	8/16	2/4
r_3	0011 0011 0011 0011	8/16	2/4
r_4	0000 0000 1111 1111	8/16	2/4
w_1	0101 0101 0101 0101	8/16	2/4
w_2	0000 1010 0000 1010	4/16	0/4
w_3	0011 0011 0011 0011	8/16	2/4
w_4	0000 0000 1100 1100	4/16	0/4
w_{x1}	0101 0101 0101 0101	8/16	2/4
w_{x2}	0000 1010 0000 1010	4/16	0/4
w_{y1}	0011 0011 0011 0011	8/16	2/4
w_{y2}	0000 0000 0000 0000	0/16	0/4
x	0101 1111 0101 1111	12/16	2/4
y	0011 0011 0011 0011	8/16	2/4
f	0001 0011 0001 0011	6/16	1/4

and 0.375, respectively. F's initial 4 and 16-bit sequences are 0001 and 0001001100010011, respectively. The corresponding stage-i bit-errors are both zero, so F has strict MPP. □

6.3 Case Study

The inner (dot) product is a useful operation in applications such as image processing, digital filter design, and neural networks. The inner product of two vectors $A = [A_1\ A_2\ \dots\ A_n]$ and $B = [B_1\ B_2\ \dots\ B_n]$ is $A \cdot B = A_1B_1 + A_2B_2 + \dots + A_nB_n = \sum_{i=1}^{n} A_iB_i$. In SC, addition must be scaled, so the stochastic inner product becomes

$$F = 1/n\ (\sum_{i=1}^{n} A_iB_i).$$

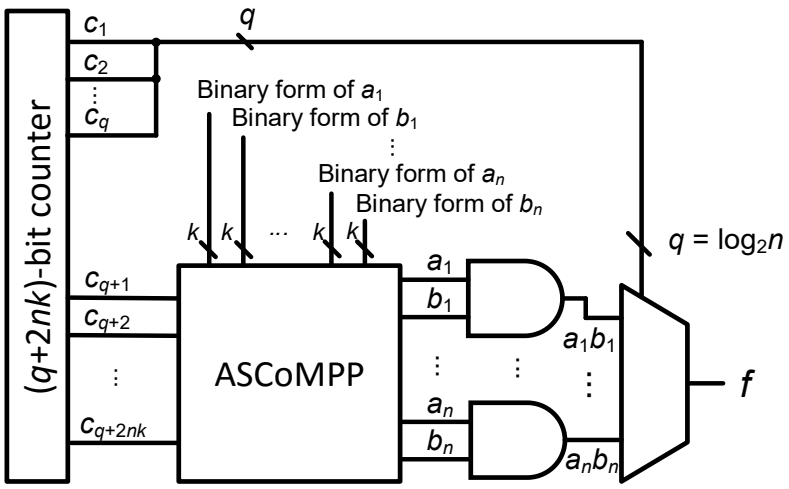

Figure 6.9: Stochastic inner product circuit designed using *ASCoMPP*.

This operation can be realized by *ASCoMPP* to generate SNs with strict MPP; see Figure 6.9. Note that the n should be a positive power of 2 to ensure $q = \log_2 n$ is a positive integer. These q signals are considered q SNs with a constant probability 0.5 generated by 1-bit WBGs. Each of the 1-bit WBGs reduces to a wire. As there are q stochastic constants and $2n$ separate k-bit precision stochastic variables, we need a $(q + 2nk)$-bit counter. The first q bits of the counter are assigned to the MUX's select signal and the rest of bits are sent to the *ASCoMPP* block in Figure 6.9.

Consider the stochastic inner product $F = 1/2\ (\sum_{i=1}^{2} A_i B_i)$ where A_i and B_i are SNs derived from 4-bit binary numbers. The output bit-stream must have at least $2^{4\times4+1}$ bits to generate an exact result with 4-bit precision. The smallest value of the result can be $0.5(2^{-4} \times 2^{-4} + 0) = 1/2^{17}$. In this case, when the output bit-stream length is 2^{17}, the stage-4 error is $\varepsilon_F^{(4)} = 0$. For sequences shorter than 2^{17}, the results will still be exact with respect to the corresponding expected values, as Theorem 6.3 asserts.

We used Matlab to simulate the stochastic inner product circuit. The input binary numbers A_1, B_2, B_1, B_2 were randomly sampled from 0 to $2^4 - 1$. The values of $\hat{p}_F^{(i)}$, $p_F^{(i)}$ and $\varepsilon_F^{(i)}$ were then computed at the output f for $i = 1, 2, 3$ and 4. The experiment was

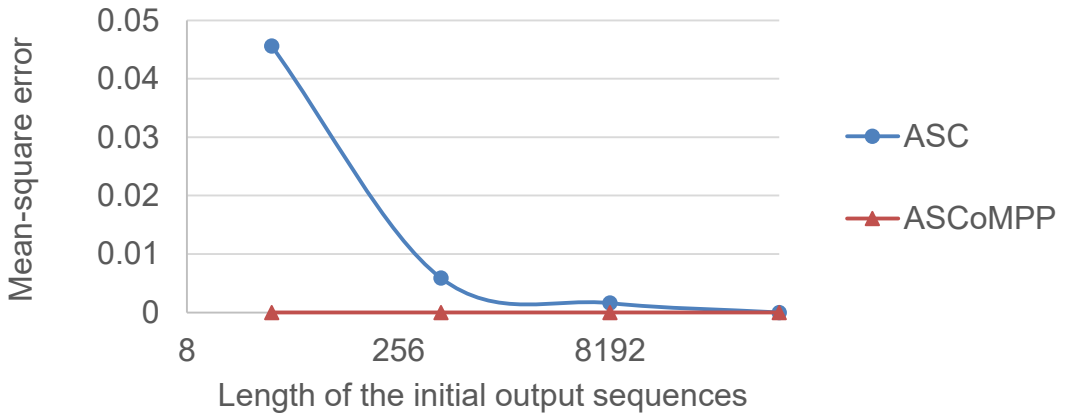

Figure 6.10: MSE of the stochastic inner products for different bit-streams of length using *ASC* and *ASCoMPP* designs.

repeated 1,000 times to compute the MSE. Figure 6.10 shows the simulation results. It compares the counter-based approach used by *ASCoMPP* with the corresponding LFSR-based *ASC* design. As expected, all the MSEs in the *ASCoMPP* design are zero, indicating the output SN has strict MPP. However, when the counter and LFSR cycle through all possible states, the *ASC* and *ASCoMPP* designs both give exact results.

In the course of this experiment, we observed that the MSEs of the LFSR-based *ASC* approach are strongly affected by the LFSR's initial state. In other words, the quality of *ASC*'s progressive precision varies with its initial state. To illustrate this, we use the simpler Example 6.1.The initial state of $r_1r_2r_3r_4$ = 1010 in Table 6.1 gives the f's initial 4-bit sub-sequence 1111, which has larger error compared to $r_1r_2r_3r_4$ = 1111. When $r_1r_2r_3r_4$ = 1111, f's 4-bit sub-sequence 1100 gives better accuracy. In Figure 6.10, the initial state is assumed to be the all-1s state.

Because the stochastic inner product circuit designed using *ASCoMPP* ensures the accuracy of the values of initial 2^{4i+1} sequences to i-bit precision, i.e. $\hat{p}_F^{(i)} = p_F^{(i)}$, these values may be smaller than the expected results to 4-bit precision $p_F^{(4)}$. The corresponding error can be formulated as $e = p_F^{(4)} - \hat{p}_F^{(i)} = p_F^{(4)} - p_F^{(i)}$. From this, we can find the lower

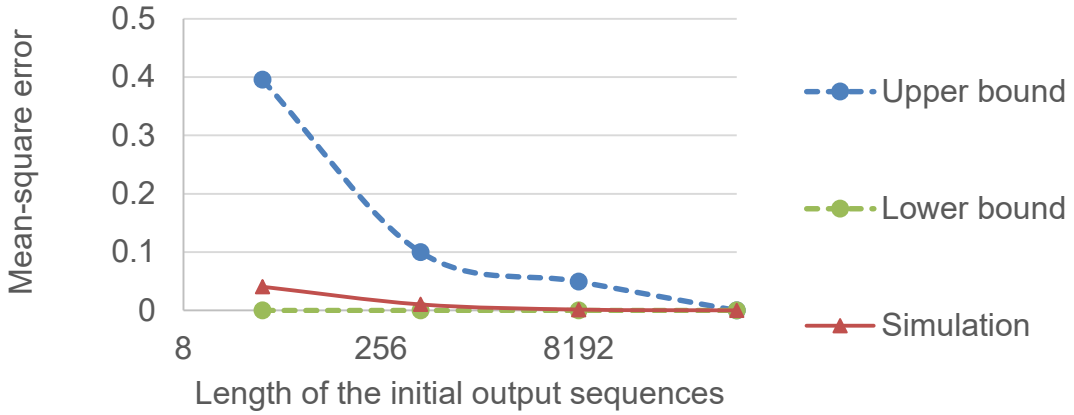

Figure 6.11: Inner-product MSE for various bit-stream lengths, compared to the expected values for 4-bit precision.

and upper bounds of the mean-square error. The experiment was repeated and the binary values of A_1, A_2, B_1, B_2 are randomly sampled 1,000 times to compute the MSE. Figure 6.11 shows the experimental results and the analytical upper and lower bounds derived from $e = p_F^{(4)} - p_F^{(i)}$. As expected, the MSE values are not zero, but decrease monotonically. This experiment shows the error caused by insufficient bit-stream length of the output SN.

As another case study, we examined a relatively complex stochastic circuit generated by the STRAUSS synthesizer [7] to implement the stochastic function $p_F = 0.6875 - 0.6875(p_{X_1} + p_{X_2}) + 1.125p_{X_1}p_{X_2}$. Figure 6.12 illustrates this circuit when it is placed in the *ASCoMPP* framework. Again, we repeat the experiment of comparing MSEs for different bit-stream lengths using *ASC* and *ASCoMPP*. The results, shown in Figure 6.13, are very similar to the inner-product case (Figure 6.11). This example shows that *ASCoMPP* applies not only to ad-hoc designs, but also to stochastic circuits synthesized by systematic methods. The combination of stochastic synthesis and *ASCoMPP* points to a way to achieving accurate results with good progressive precision for any stochastically realizable arithmetic function.

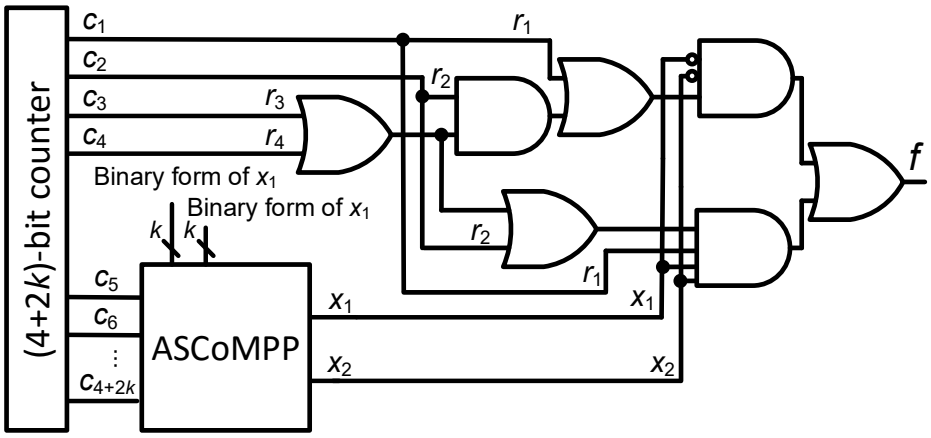

Figure 6.12: *ASCoMPP* implementation of $p_F = 0.6875 - 0.6875 \times (p_{X1} + p_{X2}) + 1.125 \times p_{X1} \times p_{X2}$.

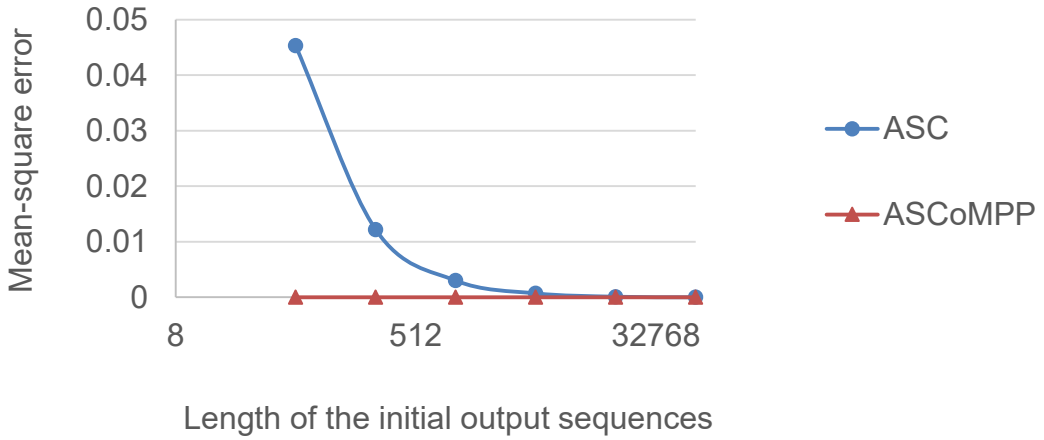

Figure 6.13: MSE of the stochastic circuit of $p_F = 0.6875 - 0.6875 \times (p_{X1} + p_{X2}) + 1.125 \times p_{X1} \times p_{X2}$ products for different bit-streams of length using *ASC* and *ASCoMPP* designs.

6.4 Summary

In this chapter, we showed how to obtain both exact results and monotonic progressive precision (MPP) in stochastic implementations of general Boolean functions. While *ASC* (Accurate Stochastic Computing) implements any stochastic arithmetic function with exact results given a specific bit-stream length, MPP is a desirable property whereby accuracy increases steadily with bit-stream length. These ideas, *ASC* and MPP,

are combined in the *ASCoMPP* (Accurate Stochastic Computing with Monotonic Progressive Precision) design approach. With a suitably chosen source of pseudo-random input vectors (samples), *ASCoMPP*-based circuits can perform stochastic computing with guaranteed accuracy. We showed how to use an ordinary binary counter to provide the sample sequence, which allows *ASCoMPP* to achieve strict MPP. We presented analytical results and experimental results to demonstrate that *ASCoMPP* provides a novel way to achieve accurate results with very good MPP.

CHAPTER 7
Conclusions

This dissertation has addressed a few of the major challenges posed by stochastic computing (SC), especially concerning its accuracy and design methodology. We highlight our main contributions in this chapter, and then point to some promising directions for future research.

7.1 Summary of Contributions

Our earlier research on designing a stochastic decoder for convolutional codes [21] convinced us of the need for a deeper understanding of the factors influencing the accuracy of SC. Accuracy and related issues therefore became the main focus our Ph.D. research. We began by investigating the behavior of stochastic circuits under various error conditions [22], as reported in Chapter 2. A systematic method based on probabilistic transfer matrices (PTMs) was developed for the algebraic analysis of stochastic behavior, complemented by circuit simulation. The PTM-based analysis provided some theoretical insights, such as the fact that PTM-based analysis is accurate and provides theoretical results for all possible input combinations. Circuit simulation, on the other hand, is based on Monte Carlo methods whose accuracy depends on its sample size, but simulation is able to handle larger circuits. A case study comparing edge-detection circuits implemented by SC and conventional approaches was also presented. Our results indicate that, under similar error conditions, stochastic circuits provide significantly better error tolerance.

Chapter 3 addressed another major factor affecting the accuracy of SC, namely correlation. SC allows arithmetic operations to be implemented at very low cost, but

interacting SNs must usually be statistically independent or uncorrelated to achieve acceptable accuracy. We again successfully applied PTMs and circuit simulation to quantify correlation-induced errors for the first time. In particular, we used these two methods to analyze correlation effects in the basic SC components. We also investigated and compared the two most common correlation-reducing methods, regeneration and isolation. Regeneration converts SNs back to binary form and introduces new random sources to regenerate SNs. Isolation uses delays (D-type flip-flops) to derive multiple statistically independent SNs without additional random sources. We derived bounds on the accuracy loss due to isolator insertion and compared its hardware cost to that of regeneration. We concluded that the isolation method offers significant cost advantages in reducing correlation errors.

After investigating the accuracy analysis aspect of SC, we moved to the design requirements of stochastic circuits. Chapter 4 introduced the concept of stochastic equivalence classes (SECs), and investigated their properties and applications. We observed that the set of inputs X of a Boolean function used in SC can be partitioned into two groups X_V and X_C to which variable and constant SNs, respectively, are applied. This implies that many equivalent Boolean functions with different implementation costs have the same stochastic behavior. Building on this insight, we constructed a general stochastic circuit synthesis method called *SECS* (SEC-based Synthesis), and an associated search-based optimization procedure called *SECO* (SEC-based Optimization) for stochastic circuit design [24]. *SECO* searches the SEC that contains Boolean functions with the desired stochastic behavior for a low-cost, preferably minimum-cost, implementation. We were also able to use this procedure to verify the optimality several important known stochastic circuits. Experimental data obtained via *SECO* showed that, in many cases, it can effectively reduce the cost of a stochastic operation without searching the entire SEC.

In Chapter 4, we further enhanced *SECS* to obtain *ESECS* (Extended *SECS*) by introducing two new SEC-related procedures that provide more flexible synthesis methods. The first procedure *SECI* (Stochastic Equivalence Class Identification) allows *ESECS* to

find an SEC directly, without generating the base design *f* required by *SECS*. The same chapter presented an alternative to *SECO* called *SECM* (SEC-based Mapping). *SECM*'s optimization technique is roughly analogous to the classical two-level minimization method in conventional logic design in its use of a weight table similar to a truth table or a Karnaugh map. Building on this similarity, we introduced stochastic prime implicants, which allow stochastic circuit optimization to be formulated in a fashion similar to the covering problem of two-level minimization. Our experimental results show that the two optimization methods *SECO* and *SECM* complement each other. While *SECO* works better for stochastic circuits with many stochastic constants, *SECM* achieves higher area cost reduction for circuits with more stochastic variables.

Chapter 5 tackled the difficult problem of designing accurate stochastic dividers. Although multiplication and addition have very simple logic circuit implementations, that is not the case for division. After reviewing and comparing the known stochastic division methods, including a stochastic divider in a long-overlooked patent, we presented a novel division technique called CORDIV (correlated division). It is based on the observation that the conditional probability $p_{X_1|X_2}$ of X_1 given X_2 leads naturally the basic division operation $p_{X_1X_2}/p_{X_2}$. CORDIV is unique in that it deliberately introduces correlation between the input parameters X_1 and X_2 to efficiently transform $p_{X_1|X_2}$ to p_{X_1}/p_{X_2}. We designed CORDIV-based dividers for both the unipolar and bipolar SN formats. Their area cost is lower than that of previous stochastic dividers, and they achieve much better accuracy.

Finally, in Chapter 6, we attempted to find ways to satisfy the design requirements of high accuracy and low area at the same time. We first demonstrated that any stochastic arithmetic function can be implemented with exact results by our *ASC* (Accurate Stochastic Computing) technique. We then showed how very accurate results are obtainable when the random numbers used for SN generation are sampled by a process which ensures that accuracy increases steadily with bit-stream length. We named this very desirable property monotonic progressive precision (MPP). We incorporated this concept into a general-purpose SC design technique *ASCoMPP* (Accurate Stochastic Computing with Monotonic

Progressive Precision). Finally, we presented analytical and experimental data which demonstrate that *ASCoMPP* is able to produce results that are both highly accurate and have good MPP.

7.2 Directions for Future Work

In this section, we discuss some open research problems in the SC field and several potential extensions of our work.

Despite the progress that has been made on designing general-purpose combinational stochastic circuits, design methods for sequential circuits are either still ad hoc or restricted to very specific structures [56]. A preliminary study of the stochastic functions realized by sequential circuits can be found in [6]. Although this study sheds some light on design requirements, it is mainly limited to a few examples illustrating the behavior of sequential stochastic circuits. However, it also suggests to us a possible extension of our SEC ideas, since it shows that sequential circuits with different state-machine structures can implement the same stochastic function. For instance, Figure 7.1, taken from [6], illustrates two sequential circuits that implement the same stochastic function $p_Z = (2p_X - 2)/(p_X - 2)$. We observe that such cases can be analyzed by a technique called time-frame expansion, which effectively models a sequential circuits by a combinational one, and is used for simulating the behavior of a sequential circuit cycle by

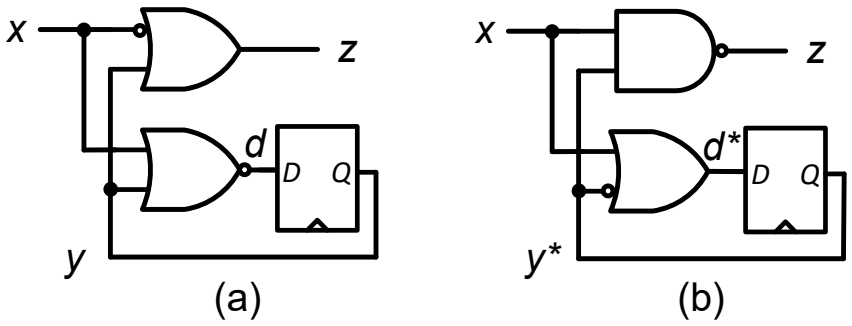

Figure 7.1: Sequential stochastic circuits implementing $p_Z = (2p_X - 2) / (p_X - 2)$ [6].

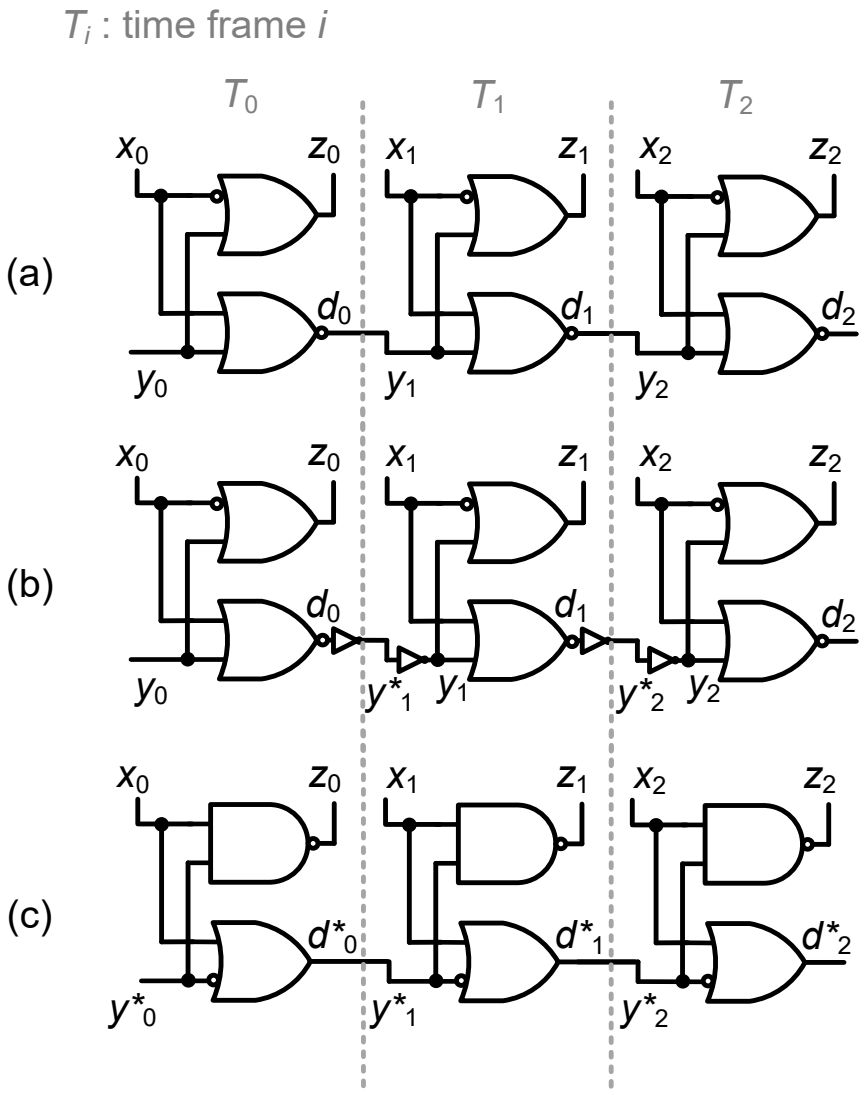

Figure 7.2: Expanded circuits with three time frames for the sequential circuits in Figure 7.1.

cycle [17]. Figures 7.2*a* and 7.2*c* illustrate the expansion of Figures 7.1*a* and 7.1*b,* respectively, with three clock cycles. The dashed lines mark the boundaries of different time frames. If we add a pair of inverters before and after the time frame boundaries as shown in Figure 7.2*b*, the Boolean function of the expanded circuit is unchanged but, the sub-circuit for time frame T_2 in Figure 7.2*b*, can be simplified to that of time frame T_2 in

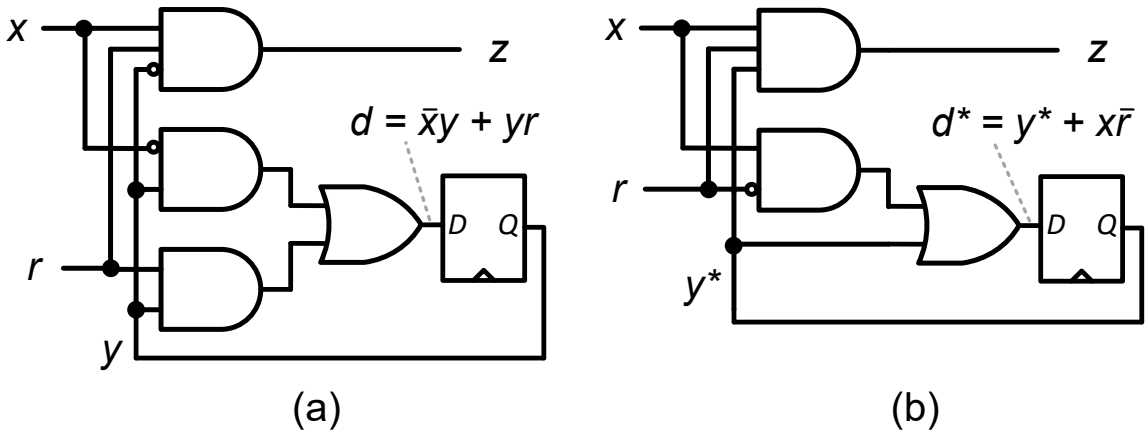

Figure 7.3: Stochastically equivalent sequential circuits with different costs.

Figure 7.2*c*. This observation suggests a sufficient condition for checking equivalence among sequential stochastic circuit, and is a path to further investigation.

The foregoing observations also bring up the possibility of optimizing sequential stochastic circuits by searching their equivalence classes for lower-cost implementations. Figure 7.3 shows another example of stochastically equivalent sequential circuits. Their equivalence can be explained by time-frame expansion. By adding a pair of inverters before and after the time frame boundaries, we have $\bar{d}^* = \bar{x}\bar{y}^* + \bar{y}^* r$. This function is further simplified to $d^* = \overline{\bar{x}\bar{y}^* + \bar{y}^* r} = (x + y^*)(y^* + \bar{r}) = y^* + x\bar{r}$, which gives the lower-cost design of Figure 7.3*b*. In addition to optimizing combinational and sequential stochastic circuits separately, it may even be possible to combine their optimization methods to create a more comprehensive technique.

Another possible direction of future work is integrating CORDIV-based dividers with our stochastic circuit synthesis method *ESECS*. As indicated in Chapter 4, *ESECS* approximates non-linear terms such as p_X/p_Y in target arithmetic functions by means of multi-linear polynomials; there was no good linear approximation for division. Since CORDIV-based dividers uses number representations that are compatible with *ESECS*, such dividers could provide a new synthesis resource to reduce the errors in SC due to poor approximation of non-linear terms.

Chapter 1 highlighted several applications that require massive parallelism, such as image-processing circuits for retinal implants and other biomedical devices. These systems typically contains a pixel sensor array, analog or digital pre-processing circuits for noise filtering, and high-level image processing circuits [65]. Although a preliminary study on stochastic circuits for real-time image processing has been made [3], that work only addressed the pixel processing part of a retinal implant. It also seems worthwhile to investigate the application of SC to the design of noise filters [60][61] and high-level image processing to further increase the performance of retinal implants. Our *ESECS* and *ASCoMPP* algorithms provide potential approaches to these applications.

The Internet of Things (IoT) is another promising candidate for SC. As discussed in Chapter 1, the IOT is characterized by large networks of sensors and processors that have very low cost requirements, strict power budgets, and often incorporate massive parallelism. The sensors perform tasks like temperature sensing, liquid flow-rate calculation, image and sound recording, pressure measurement, or chemical detection. The vast amounts of data they generate are often pre-processed or compressed to extract useful information. To illustrate, consider a long-lifetime quality sensor for a water distribution system intended to detect pollution via an array of electronic chemical detectors and a pattern processor to recognize pollution sources [39]. The sensors are clear candidates for SC, as we pointed out in the case of image processing. The pattern-recognition system can be expected to involve many linear regression models like

$$y(t) = \sum_{i=1}^{n} x_i(t)\alpha_i + \beta$$

where $y(t)$ is an indicator of water pollution persistence, and the x_i's are prediction variables [45]. Since $y(t)$ is defined by a multi-linear polynomial, it can be readily realized by stochastic circuits generated by *ESECS* or similar SC design algorithms.

In conclusion, stochastic computing has demonstrated great potential to deliver low-cost, low-power and fault-tolerant circuit designs. However it continues to have a

relatively small range of applications such as image processing, error-correction decoders, and neural networks. We hope that the theory and techniques for SC that are presented in this dissertation prove helpful in designing future stochastic circuits, and stimulate the development of useful new applications of this important technology.

BIBLIOGRAPHY

[1] Alaghi, A. and Hayes, J. P., "A spectral transform approach to stochastic circuits," *Proc. Int. Conf. Computer Design*, pp. 315-321, 2012.

[2] Alaghi, A. and Hayes, J. P., "Survey of stochastic computing," *ACM Trans. Embedded Computing Systems*, 12, 2s, pp. 92:11-92:19, 2013.

[3] Alaghi, A., Li, C. and Hayes, J. P., "Stochastic circuits for real-time image-processing applications," *Proc. Design Automation Conf.,* pp. 1-6, 2013.

[4] Alaghi, A. and Hayes, J. P., "Exploiting correlation in stochastic circuit design," *Proc. Int. Conf. Computer Design,* pp. 39-46, 2013.

[5] Alaghi, A. and Hayes, J. P., "Fast and accurate computation using stochastic circuits," *Proc. Conf. on Design, Automation & Test in Europe*, pp. 76:1-76:4, 2014.

[6] Alaghi, A. and Hayes, J. P., "On the functions realized by stochastic computing circuits," *Proc. Great Lakes Symp. on VLSI*, pp. 331-336, 2015.

[7] Alaghi, A. and Hayes, J. P., "STRAUSS: spectral transform use in stochastic circuit synthesis," *IEEE Trans. CAD,* 34, 1770-1783, 2015.

[8] Alaghi, A. and Hayes, J. P., "Dimension reduction in statistical simulation of digital circuits," *Proc. on Theory of Modeling and Simulation*, pp. 1-8, 2015.

[9] Ananth, R. S., "Programmable supervisory circuit and applications thereof," US Patent no. 6,618,711, 2003.

[10] Baumann, R. C., "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Trans. Device & Materials Reliability*, 5, pp. 305-316, 2005.

[11] Blaauw, D., Sylvester, D., Dutta, P., Lee, Y., Lee, I., Bang, S., Kim, Y., Kim, G., Pannuto, P., Kuo, Y.-S., Yoon, D., Jung, W., Foo, Z., Chen, Y.-P., Oh, S., Jeong, S. and Choi, M., "IoT design space challenges: Circuits and systems," *Proc. Symp. on VLSI Technology (VLSI-Technology): Digest of Technical Papers*, pp. 1-2, 2014.

[12] Borkar, S., "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, 25, 6, pp. 10-16, 2005.

[13] Braendler, D., Hendtlass, T. and O'Donoghue, P., "Deterministic bit-stream digital neurons," *IEEE Trans. Neural Networks*, 13, 6, pp. 1514-1525, 2002.

[14] Brayton, R. K., Hachtel, G. D., McMullen, C. and Sangiovanni-Vincentelli, A., *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[15] Brown, B. D. and Card, H. C., "Stochastic neural computation I: computational elements," *IEEE Trans. Computers*, 50, pp. 891-905, 2001.

[16] Brown, B. D. and Card, H. C., "Stochastic neural computation II: soft competitive learning," *IEEE Trans. Computers*, 50, pp. 906-920, 2001.

[17] Bushnell, M. L. and Agrawal, V. D., *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, Kluwer, 2000.

[18] Canals, V., Morro, A., Oliver, A., Alomar, M. L. and Rosselló, J. L., "A new stochastic computing methodology for efficient neural network implementation," *IEEE Trans. Neural Networks and Learning Syst.* 27, pp. 551-564, 2015.

[19] Chandrakasan, A. P., Verma, N. and Daly, D. C., "Ultralow-power electronics for biomedical applications," *Annual Review of Biomedical Engineering*, 10, 1, pp. 247-274, 2008.

[20] Chen, H. and Han, J., "Stochastic computational models for accurate reliability evaluation of logic circuits," *Proc. Great Lakes Symp. VLSI*, pp. 61-66, 2010.

[21] Chen, T.-H. and Hayes, J. P., "Design of stochastic Viterbi decoders for convolutional codes," *Proc. Int. Symp. on Design and Diagnostics of Electronic Circuits Syst.,* pp. 66-71, 2013.

[22] Chen, T.-H., Alaghi, A. and Hayes, J. P., "Behavior of stochastic circuits under severe error conditions," *Info. Tech.*, 56, 4, pp. 182-191, 2014.

[23] Chen, T.-H. and Hayes, J. P., "Analyzing and controlling accuracy in stochastic circuits," *Proc. Int. Conf. on Computer Design,* 367-373, 2014.

[24] Chen, T.-H. and Hayes, J. P., "Equivalence among stochastic logic circuits and its application," *Proc. Design Automation Conf.*, pp. 131:1-131:6, 2015.

[25] Chen, Y. K., "Challenges and opportunities of internet of things," *Proc. Asia and South Pacific Design Automation Conf.,* pp. 383-388, 2012.

[26] Choi, S. S., Chia, S. H. and Tappert, C., "A survey of binary similarity and distance measures," *Journ. Systemics, Cybernetics and Informatics,* 8, pp. 43-48, 2010.

[27] De, V. and Borkar, S., "Technology and design challenges for low power and high performance," *Proc. Int. Symp. on Low Power Electronics and Design*, pp. 163-168, 1999.

[28] de Aguiar, J. M. and Khatri, S. P,. "Exploring the viability of stochastic computing," *Proc. Int. Conf. on Computer Design*, pp. 391-394, 2015.

[29] Dong, Q. T., Arzel, M., Jego, C. and Gross, W. J., "Stochastic decoding of Turbo codes," *IEEE Trans. Signal Processing*, 58, 12, pp. 6421-6425, 2010.

[30] Gaines, B. R., "Stochastic computing," *Proc. AFIPS Spring Joint Computer Conf.*, pp. 149-156, 1967.

[31] Gaines, B. R., "Stochastic computing systems," *Advances in Inform. Systems Science,* 2, pp. 37-172, 1969.

[32] Gal-Edd. J. and Fatig, C. C., "L2-James Webb Space Telescope operationally friendly environment?" *Proc. Aerospace Conf.*, pp. 105-110, 2004.

[33] Gallager, R. G., "Low-density parity-check codes," *IRE Trans. Inform. Theory*, 8, pp. 21-28, 1962.

[34] Gaudet, V. C. and Rapley, A. C., "Iterative decoding using stochastic computation," *Electronics Letters*, 39, pp. 299-301, 2003.

[35] GNU, *Octave*, https://www.gnu.org/software/octave/, accessed April 2016.

[36] Golomb, S. W., "On the classification of balanced binary sequences of period $2^n - 1$," *IEEE Trans. Info. Theory.* 26, 6, pp. 730-732, 1980.

[37] Gonzalez, R. C. and Woods, R. E., *Digital Image Processing*, 2nd ed., Prentice Hall, 2002.

[38] Gupta, P. K. and Kumaresan, R., "Binary multiplication with PN sequences," *IEEE Trans. Acoustics, Speech and Signal Processing*, 36, pp. 603-606, 1988.

[39] Gutierrez-Osuna, R. and Nagle, H. T., "A method for evaluating data-preprocessing techniques for odor classification with an array of gas sensors," *IEEE Trans. on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29, 5, pp. 626-632, 1999.

[40] Hachtel, G. G. and Somenzi, F., *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, 1996.

[41] Hikawa, H, "Digital pulse mode neuron with robust nonlinear activation function," *Proc. Int. Joint Conf. on Neural Networks*, pp. 2665-2670, 2004.

[42] Hu, J., Deng, Y., Chen, J. and Ling, X., "High speed Turbo decoder design based on stochastic computation," *Int. Conf. on Communications, Circuits and Systems*, 1, pp. 235-239, 2013.

[43] Joung, Y.-H., "Development of implantable medical devices: from an engineering perspective," *Int. Neurourology Journ.*, 17, 3, pp. 98-106, 2013.

[44] Jeavons, P., Cohen, D. A. and Shawe-Taylor, J., "Generating binary sequences for stochastic computing," *IEEE Trans. Information Theory*, 40, pp. 716-720, 1994.

[45] Kehoe, M. J., Chun, K. P. and Baulch, H. M. "Who smells? Forecasting taste and odor in a drinking water reservoir," *ACS Environ. Sci. Tech.*, 49, 18, pp. 10984-10992, 2015.

[46] Kim, Y-C. and Shanblatt, M. A., "Architecture and statistical model of a pulse-mode digital multilayer neural network," *IEEE Trans. Neural Networks*, 6, pp. 1109-1118, 1995.

[47] Klotz, J., "Statistical inference in Bernoulli trials with dependence," *Annals of Statistics*, 1, 2, pp. 373-379, 1973.

[48] Koren, I., *Computer Arithmetic Algorithms*, 2nd ed., A. K. Peters, 2002.

[49] Koren, I. and Krishna, C. M., *Fault-Tolerant Systems*, Morgan Kaufmann, 2007.

[50] Krim, H. and Viberg, M., "Two decades of array signal processing research: the parametric approach," *IEEE Signal Processing Magazine*, 13, 4, pp. 67-94, 1996.

[51] Krishnaswamy, S., Viamontes, G. F., Markov, I. L. and Hayes, J. P., "Accurate reliability evaluation and enhancement via probabilistic transfer matrices," *Proc. Design, Automation and Test in Europe*, pp. 282-287, 2005.

[52] Krishnaswamy, S., Markov, I. L. and Hayes, J. P., "Tracking uncertainty with probabilistic logic circuit testing," *IEEE Design & Test of Computers,* 24, pp. 312-321, 2007.

[53] Krishnaswamy, S., Viamontes, G. F., Markov, I. L. and Hayes, J. P., "Probabilistic transfer matrices in symbolic reliability analysis of logic circuits," *ACM Trans. Design Automation of Electronic Systems.*, 13, 1, pp. 8:1-8:35, 2008.

[54] Lahiri, K., Dey, S., Panigrahi, D. and Raghunathan, A., "Battery-driven system design: a new frontier in low power design," *Proc. Asia and South Pacific Design Automation Conf.*, p. 261-267, 2002.

[55] Li, P. and Lilja D. J., "Using stochastic computing to implement digital image processing algorithms," *Proc. Int. Conf. Computer Design*, pp. 154-161, 2011.

[56] Li, P., Lilja, D. J., Qian, W., Bazargan, K. and Riedel, M. D., "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," *Proc. Int. Conf. Computer-Aided Design*, pp. 480-487, 2012.

[57] Li, P., Lilja, D. J., Qian, W., Bazargan, K.and Riedel, M. D., "Computation on stochastic bit streams digital image processing case studies," *IEEE Trans. VLSI Syst.*, 22, 3, pp. 449-462, 2014.

[58] Li, X., Qian, W., Riedel, M. D., Bazargan, K. and Lilja, D. J., "A reconfigurable stochastic architecture for highly reliable computing," *Proc. Great Lakes Symp. VLSI*, pp. 315-320, 2009.

[59] Lin, S. and Costello, D. J., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 2004.

[60] Liu, Y. and Parhi, K. K., "Lattice FIR digital filter architectures using stochastic computing," *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 1027-1031, 2015.

[61] Liu, Y. and Parhi, K. K., "Architectures for stochastic normalized and modified lattice IIR filters," *Proc. Asilomar Conf. on Signals, Systems and Computers*, pp. 1351-1358, 2015.

[62] Ma, C., Zhong, S. and Dang, H., "Understanding variance propagation in stochastic computing systems," *Proc. Int. Conf. Computer Design*, pp. 213-218, 2013.

[63] Marin, S. L. T., Reboul, J. M. Q. and Franquelo, L. G., "Digital stochastic realization of complex analog controllers," *IEEE Trans. Industrial Electronics*, 49, pp. 1101-1109, 2002.

[64] Min, S.-J., Lee, E.-W. and Chae, S.-I., "A study on the stochastic computation using the ratio of one pulses and zero pulses," *Proc. IEEE Int. Symp. on Circuits and Systems*, 6, pp. 471-474.

[65] Moini, A., *Vision Chips*, Kluwer, 2000.

[66] Mokwa, W., "Retinal implants to restore vision in blind people," *Proc. Solid-State Sensors, Actuators and Microsystems Conf.*, 2011, pp. 2825-2830, 2011.

[67] Naderi, A., Mannor, S., Sawan, M. and Gross, W. J., "Delayed stochastic decoding of LDPC codes," *IEEE Trans. Signal Processing,* 59, pp. 5617-5626, 2011.

[68] Parker, K. P. and McCluskey, E. J., "Analysis of logic circuits with faults using input signal probabilities," *IEEE Trans. Computers*, 24, 5, pp. 573-578, 1975.

[69] Poppelbaum, W. J., Afuso, C. and Esch, J. W., "Stochastic computing elements and systems," *Proc. AFIPS Fall Joint Computer Conf.*, pp. 635-644, 1967.

[70] Poppelbaum, W. J., "Statistical processors," *Advances in Computers*, 14, pp. 187-230, 1976.

[71] Pratt, B., Caffrey, M., Carroll, J. F., Graham, P., Morgan, K. and Wirthlin, M., "Fine-grain SEU mitigation for FPGAs using partial TMR," *IEEE Trans. on Nuclear Science*, 55, 4, pp. 2274-2280, 2008.

[72] Qian, W. and Riedel, M. D., "The synthesis of robust polynomial arithmetic with stochastic logic," *Proc. Design Automation Conf.*, pp. 648-653, 2008.

[73] Qian, W., Riedel, M. D., Bazargan, K. and Lilja, D. J., "The synthesis of combinational logic to generate probabilities," *Proc. Int. Conf. on Computer Aided Design*, pp. 367-374, 2009.

[74] Qian, W., Li, X., Riedel, M. D., Bazargan, K. and Lilja, D. J., "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Computers*, 60, pp. 93-105, 2011.

[75] Qian, W., Riedel, M. D. and Rosenberg, I., "Synthesizing cubes to satisfy a given intersection pattern," *Discrete Applied Mathematics*, 193, 1, pp. 11-38, 2015.

[76] Ribeiro, S. T., "Random-pulse machines," *IEEE Trans. Electronic Computers*, EC-16, pp. 261-276, 1967.

[77] Rossello, J. L., Canals, V. and Morro, A., Probabilistic-based neural network implementation," *Int. Joint Conf. on Neural Networks*, pp. 1-7, 2012.

[78] Russell, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, 3rd ed., Pearson, 2009.

[79] Shulaker, M. M., Hills, G., Patil, N., Wei, H., Chen, H.-Y., Wong, H.-S. P. and Mitra, S., "Carbon nanotube computer," *Nature*, 501, 7468, pp. 526-530, 2013.

[80] Stark, H. and Woods, J. W., *Probability and Random Processes with Applications to Image Processing*, 3rd ed., Prentice Hall, 2002.

[81] Strong, S. P., Koberle, R., de Ruyter van Steveninck, R. R. and Bialek, W., "Entropy and Information in Neural Spike Trains," *Phys. Rev. Lett.*, 80, 1, pp. 197-200, 1998.

[82] von Neumann, J., "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata Studies*, Princeton Univ. Press, pp. 43-98, 1956.

[83] Wang, Z., Saraf, N., Bazargan, K. and Scheel, A., "Randomness meets feedback: stochastic implementation of logistic map dynamical system," *Proc. Design Automation Conf.*, pp. 132:1-132:7, 2015.

[84] Weste, N. and Harris, D., *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed., Pearson, 2010.

[85] Zhang, Z., Anantharam, V., Wainwright, M. J. and Nikolic, B., "An efficient 10GBASE-T Ethernet LDPC decoder design with low error floors," *IEEE Journ. Solid-State Circuits*, 45, pp. 843-855, 2010.

[86] Zhao, Z. and Qian, W., "A general design of stochastic circuit and its synthesis," *Proc. Design, Automation and Test in Europe*, pp. 1467-1472, 2015.