The Logic of Random Pulses: Stochastic Computing

by

Armin Alaghi

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
2015

Doctoral Committee:

Professor John P. Hayes, Chair
Professor Igor L. Markov
Professor Karem A. Sakallah
Professor David D. Wentzloff

0101010001101000011001010010000001110111011011110111001001101100011001000
0100000011100110110000101110111001000000110111001101111001000000111000001
1100100110111101100110011010010111010000100000011001100111001001101111011
0110100100000011011010111100100100000011000100110010101100011011011110110
1101011010010110111001100111000011010000101001001110011011111011100100010
0000110100101110100011100110010000001110011011100000110110001100101011011
1001100100011011110111001000100000011000010110111001100100001000000110111
0011011110110001001101001011011000110100101110100011110010010000001100001
0111010101100111011011010110010101101110011101000110010101100100001000000
1100010011110010010000001101101011110010010000001101100011001010110000101
1101100110100101101110011001110000110100001010010101000110100001100101011
1001001100101001000000110100101110011001000000110111001101111001000000110
1111011011100110010100100000011001100111001001101111011011010010000001110
1110110100001101111011011010010000001100010011011110111010001101000001000
0001101101011010010110111001100101001000000110010101100001011100100111001
1001000000011010000110000101110110011001010010000001101000011001010110000
1011100100110010000100000000011010000101001010111011010000110000101110100
0100000011100000111010101110010011100000110111101110011011001010010000001
1001000110100101100100000100000011100110110010101110010011101100110010100
1001000000110110101111001001000000110001101101111011011010110100101101110
0111001000000110000101101110011001000010000001100111011011110110100101101
11001100111

از آمـــدنـــم نـــبـــود گـــردون را ســـود وز رفــتــن مــن جــاه و جــلالــش نــفــزود

وز هــیــچ کــســی نــیــز دو گــوشــم نــشــنــود کــایــن آمــدن و رفــتــنــم از بــهــر چــه بــود

2015

Dedicated to my Elly

A C K N O W L E D G M E N T S

Many people have helped me throughout my PhD work and I am deeply grateful to all of them.

First and foremost, I wish to thank my advisor Professor John P. Hayes who has guided me and encouraged me all the way. He is an excellent mentor, a world-class researcher, and a wonderful person with a great sense of humor. I have learned a lot from his invaluable advice, patience, kindness, and experience. It has been an honor to be his student.

I am grateful to the members of my PhD committee, Professors Igor L. Markov, Karem A. Sakallah, and David D. Wentzloff. Their suggestions and helpful discussions have greatly improved this work. A special thanks to Professor Markov who introduced me to the topic of stochastic computing and shared lots of interesting ideas with me.

I wish to thank Professors Ilia Polian, Andrew Kahng, Scott Mahlke, Thomas Wenisch, Marios Papaefthymiou, Todd Austin, Jason Mars, and Lingjia Tang for their support and helpful advice.

Thank you to Professor Zainalabedin Navabi, my advisor at the University of Tehran, who trained me as a researcher and prepared me for the tough journey of my PhD.

I am thankful to the members of our research group with whom I have had a lot of great discussions. Dr. Smita Krishnaswamy, Dr. Kenneth Zick, Dr. Dae Young Lee, Dr. Chien-Chih Yu, (soon to be) Dr. Te-Hsuan (Sean) Chen, I-Che (Jonathan) Chen, Ahsen Tahir, Pai-Shun Ting, and William Sullivan. I also wish to thank Dr. Alexandru Paler from the University of Passau, and Wei-Ting (Jonas) Chan and Jiajia Li from the University of California-San Diego.

I have been lucky to be an honorary member of the wireless integrated circuits and systems group at the University of Michigan. I have had many memorable moments with them, and I wish to acknowledge them here. Special thanks to Kuo-Ken Huang, Muhammad Faisal, Osama Khan, Nathan Roberts, Jonathan Brown, Ryan Roger, David Moore, Mike Kines, and Avish Kosari.

I would like to express my gratitude to CSE staff members who have helped me in many ways. Thank you Dawn Freysinger, Karen Liska, Lauri Johnson-Rafalski, Stephen Roger, Steve Crang, Denise DuPrie, and Kyle Banas. I would also like to thank all the CSE graduate students with whom I have taken courses, done projects, played games, and had a lot of fun memories.

My valuable friends in Ann Arbor have helped me remember that life is all about love and friendship. I wish to acknowledge them all for being there for me. A special thanks to the members of the music bands with whom I have shared lots of great moments. My deepest gratitude to Dr. Mehrzad Samadi whose advice and unconditional help has been the most precious.

I wish to thank my family Farkhondeh Kokabi, Soleiman Alaghi, Robab Parvizi, and Dariush Ansari for everything they have given to me. Thank you to my brother Lachen Alaghi, and my sisters Shamin Alaghi, Sahar Ansari, and Samin Ansari. A unique thanks to my cousin, Sepehr Attarchi who taught me image processing, and much more.

Above all, I want to express my most sincere gratitude to my dear wife Elnaz Ansari who has been with me every single moment for the past ten years. Her technical and emotional support has been the driving force of my career. Thank you for all the late nights and early mornings, and most of all, thank you for being my best friend. I am in your debt, forever.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADC Analog-to-Digital Converter

ADDIE ADaptive DIgital Element

APS Asymmetric Polynomial Selection

BD Boolean Difference

BF Boolean Function

BRV Bernoulli Random Variable

CCC Correlated Combinational Circuit (algorithm)

CI Correlation Insensitive

CMOS Complementary Metal-Oxide Semiconductor

CNTFET Carbon Nanotube Field-Effect Transistor

DAC Digital-to-Analog Converter

ECC Error-Correcting Code

FDSOI Fully Depleted Silicon On Insulator

FPGA Field-Programmable Gate Array

FSM Finite State Machine

IC Integrated Circuit

ITM Ideal Transfer Matrix

LD Low Discrepancy

LDPC Low-Density Parity-Check

LFSR Linear Feedback Shift Register

MC Monte Carlo

PE Processing Element

PFM Pulse Frequency Modulation

pmf Probability Mass Function

PP Progressive Precision

PTM Probabilistic Transfer Matrix

QMC Quasi-Monte Carlo

ReSC Reconfigurable Stochastic Computing Architecture

RNG Random Number Generator

RV Random Variable

SC Stochastic Computing

SCC Stochastic Computing Correlation

SCG Single Constant Generation

SF Stochastic Function

SN Stochastic Number

SNR Signal to Noise Ratio

SNG Stochastic Number Generator

SPA Sum-Product Algorithm

SPE Stochastic Processing Element

SS Statistical Simulation

STRAUSS Spectral TRAnsform Use in Stochastic Circuit Synthesis

STT Symmetric Truth Table

TT Truth Table

WCI Weakly Correlation Insensitive

ABSTRACT

The Logic of Random Pulses: Stochastic Computing

by

Armin Alaghi

Chair: John P. Hayes

Recent developments in the field of electronics have produced nano-scale devices whose operation can only be described in probabilistic terms. In contrast with the conventional deterministic computing that has dominated the digital world for decades, we investigate a fundamentally different technique that is probabilistic by nature, namely, stochastic computing (SC). In SC, numbers are represented by bit-streams of 0's and 1's, in which the probability of seeing a 1 denotes the value of the number. The main benefit of SC is that complicated arithmetic computation can be performed by simple logic circuits. For example, a single (logic) AND gate performs multiplication. The dissertation begins with a comprehensive survey of SC and its applications. We highlight its main challenges, which include long computation time and low accuracy, as well as the lack of general design methods. We then address some of the more important challenges. We introduce a new SC design method, called STRAUSS, that generates efficient SC circuits for arbitrary target functions. We then address the problems arising from correlation among stochastic numbers (SNs). In particular, we show that, contrary to general belief, correlation can sometimes serve as a resource in SC design. We also show that unlike conventional circuits, SC circuits can tolerate high error rates and are hence useful in some new applications that involve nondeterministic behavior in the underlying circuitry. Finally, we show how SC's properties can be exploited in the design of an efficient vision chip that is suitable for retinal implants. In particular, we show that SC circuits can directly operate on signals with neural encoding, which eliminates the need for data conversion.

Chapter 1

Introduction

Stochastic computing (SC) is a (re-)emerging computing technique that was invented in the 1960s as a low-cost alternative to conventional binary computing. It is unique in that it represents and processes information in the form of probabilities. More recently, it has been shown to be useful in important applications such as image-processing and communications. In this dissertation, we address some of the challenges of SC, and show how it can be used to implement efficient image-processing circuits intended for retinal implants. This chapter provides the motivation behind this work, as well as a brief introduction to SC and a summary of our contributions.

1.1 Motivation

Advances in semiconductor technology have enabled many exciting new applications of embedded computers. They have also exposed problems and opportunities that cannot be easily addressed using conventional design approaches. An example that motivates our work is the provision of retinal implants for the visually impaired [88]. This involves designing an integrated circuit (IC) chip that can be surgically placed on a dysfunctional retina to sense images (or process images sent wirelessly from an external camera) and convert an array of pixel streams to streams of neural-style electrical signals that stimulate useful visual sensations. The implanted IC is linked to an external power supply and must not dissipate more than a few mW/mm^2 to avoid heat damage to the eye [98] [30].

Most of the state-of-the-art retinal implants only have the stimulation circuit implanted inside the eye. Image acquisition and processing are performed in an external device (usually a camera and a general-purpose processor) and only the stimulation signals are transmitted to the implant [46] [64]. The images perceived through retinal implants have a low

resolution, which makes proper image processing tasks necessary. One such task is edge detection, which is the process of finding the edges of an image. Figure 1.1 shows how edge detection converts the raw image of a corridor to a high-contrast image that can be easily perceived. Due to limitations of conventional image-processing circuits (energy/power among others), unconventional computing techniques have been proposed to enable integration of image sensors and processors into the implant. Ohta et al. [96] propose using "pulse-mode" image-processing techniques to avoid the cost of data conversion, because the signals required to stimulate the retina are in fact pulse frequency modulated (PFM) signals.

Similar energy and power constraints have become major challenges to IC design of mobile embedded systems such as wearable devices. These devices have strict power and energy requirements due to battery capacity and physical limitations. Various approaches have been proposed to overcome such energy/power problems. Notably, embedded systems are usually designed for specific applications, which allows designers to use dedicated hardware with more desirable physical and/or logical characteristics than conventional general-purpose designs.

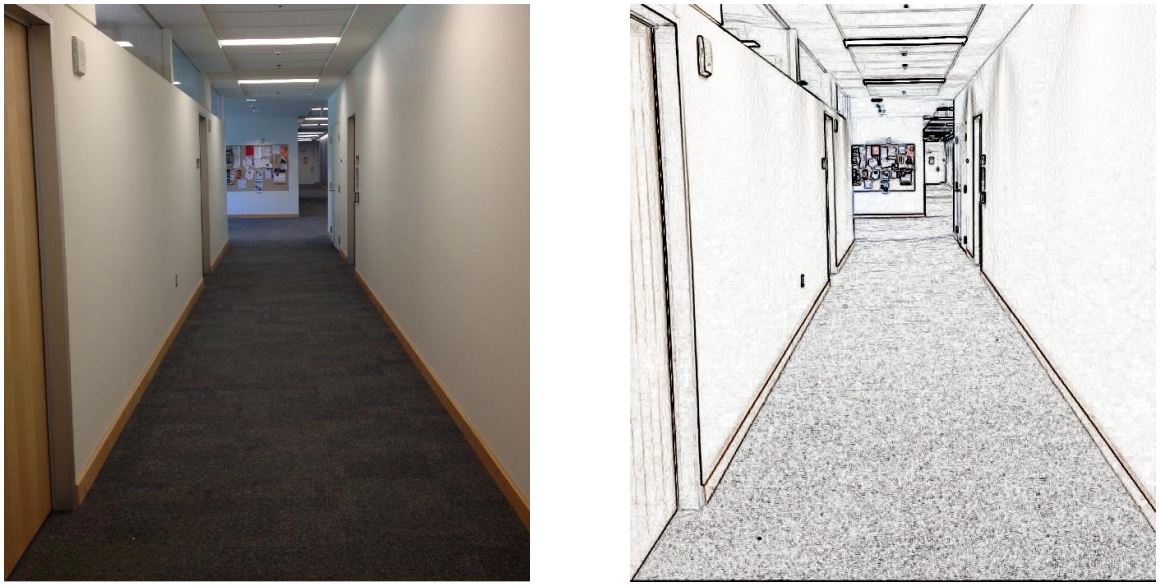


Figure 1.1: Example of an image-processing task for retinal implants: edge detection converts the raw image of a corridor (a) into a high-contrast image (b).

In addition to the stringent application requirements like extremely small size, low power consumption, and high reliability, modern computing hardware is also subject to physical phenomena like manufacturing process variations and soft errors, which give rise to error-prone behavior that can best be described in probabilistic terms. Nondeterministic behavior is ubiquitous in digital circuits implemented using conventional complementary metal-oxide semiconductor (CMOS) transistors or various novel nanotechnologies [14]. For instance, because of their small size, these circuits are easily affected by small manufacturing defects and by transient errors due to environmental noise, both of which tend to be nondeterministic in nature. For instance, carbon nanotube field-effect transistors (CNTFETs), an emerging alternative to CMOS, exhibit behavioral variations that are difficult to identify and control [122]. Other nano-devices such as memristors [66] and magnetic-tunnel junction devices [97] are naturally stochastic in some respect. The nondeterministic nature of such devices suggest that conventional binary-weighted coding of numbers may be unreliable for accurate computation. For decades, (weighted) binary encoding has been used as the main code for computation. Often reliable computation requires the use of error-correcting codes (ECC) [112] or replication of hardware, software and/or data [67] [105]. Most of these approaches impose high circuit overhead, and tend to be used only in the most cost-insensitive applications. For example, triple modular redundancy (TMR) [67] uses three replicas of the original circuit and a voter, and hence increases the area cost of the circuit by a factor of three.

The foregoing discussions suggest that unconventional computing methods that directly address the reliability issues should be considered as an alternative to conventional computing. Von Neumann took the first steps in designing reliable circuits using unreliable switches in the 1950s [134]. A fundamentally different way of encoding numbers was introduced in the 1960s by the emergence of *stochastic computing* (SC) [38] [103] [113]. In SC, a signal is encoded via a random bit-stream, i.e., a series of random pulses, where the probability of seeing a 1, or in other words, the rate of the pulses, is the value being carried. For example, the number 0.3 can be represented by the bit-stream $01001000100010010100...$, in which the probability of seeing a 1 is 0.3.

1.2 Stochastic Computing

Computing hardware has traditionally been partitioned into two broad classes: analog circuits acting on continuous data and digital circuits acting on discrete data, with real and in-

teger arithmetic being the corresponding mathematical methods. In the digital case where the fundamental data units are the bits 0 and 1, Boolean algebra plays a key role. Analog computing was eclipsed by digital in the mid-twentieth century due to the latter's greater generality, higher precision, and ease of use [80]. Nevertheless, analog computing continues to be found in applications that can exploit its performance advantages (high speed, complex basic operations, and error insensitivity) while tolerating its disadvantages. SC can be seen as a hybrid system that combines analog and digital features. This is similar to biological systems, where digital neural signals control analog functions like motion. In fact, there is an important similarity between neural signals and stochastically encoded numbers. Figure 1.2 shows this similarity: in both cases, the information is carried by the rate of the pulses. This similarity makes SC potentially useful for bio-related applications.

In SC, as mentioned, numbers are represented by random bit-streams that are interpreted as probabilities. For example, a bit-stream containing 25 percent 1's and 75 percent 0's denotes the *stochastic number* (SN) X with value $p_X = 0.25$, reflecting the fact that

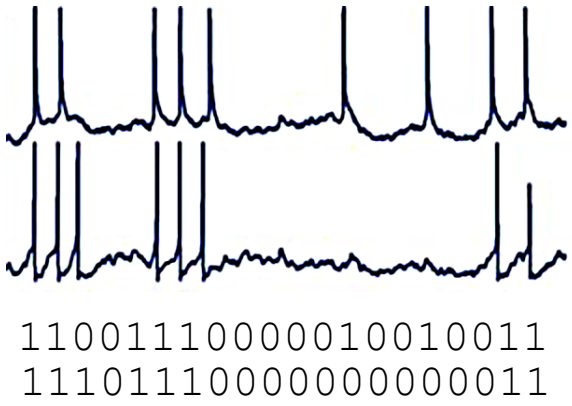


Figure 1.2: Similarity of biological signals and stochastic numbers; information is carried via pulses.

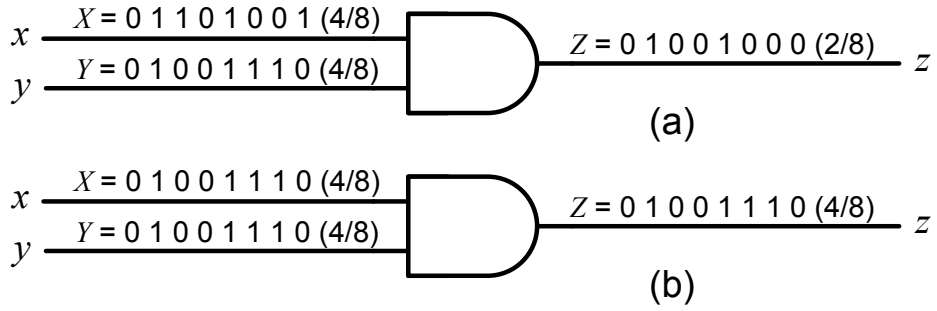


Figure 1.3: Stochastic multiplication: (a) accurate result with uncorrelated inputs; (b) inaccurate result due to correlated inputs.

the probability of observing a 1 at an arbitrary bit position is X. Neither the length nor the structure of X need be fixed; so bit-streams 1000, 0100 and 01000100 are all possible representations of 0.25. Note that the value p_X depends on the ratio of 1's to the length of the bit-stream, not on their positions, which can, in principle, be chosen randomly. The main attraction of SC when it was first introduced in the 1960s [38] [103] [113] is that it enables low-cost implementations of arithmetic operations using standard logic elements. For example, multiplication can be performed by a stochastic circuit consisting of a single AND gate. Consider two binary bit-streams that are logically ANDed together. If the probabilities of seeing a 1 on the input bit-streams are p_X and p_Y, then the probability of 1 at the output of the AND gate is $p_X \times p_Y$, assuming that the two bit-streams are suitably uncorrelated or independent. Figure 1.3a illustrates the multiplication of two SNs in this way. As can be seen, both inputs to the AND gate represent the number $4/8$. In the case of Figure 1.3a, we get an output bit-stream denoting $4/8 \times 4/8 = 2/8$.

Figure 1.3b shows one of the many possible alternative SC representations of the same values. In this case, the input bit-streams are highly correlated leading to an output bit-stream denoting $4/8$, which is inaccurate ($4/8 \times 4/8 \neq 4/8$). This example illustrates a key problem in SN processing.

In addition to their simplicity, SC circuits are error-tolerant. The value of an SN X is only determined by the number of 1's in the bit-stream, and not their position. This makes the representation highly redundant, and hence error-tolerant. As an example, consider the bit-stream 01001010 which is an SN of value $3/8$. In a noisy environment, the bits are susceptible to soft errors of the bit-flip type [13]. Should a bit-flip occur in the SN, its value will slightly change to $4/8$ (or $2/8$). Multiple bit-flips may even cancel each other out. On the other hand, bit-flips can greatly alter the value of a conventional binary number, if they occur on the most significant bits. For instance, if we consider the same number $3/8$ in conventional binary format 0.011, a single bit-flip causes a huge error if it affects a high-order bit. A change from 0.011 to 0.111, for example, changes the result from $3/8$ to $7/8$. SNs have no high-order bits as such since all bits of a stochastic bit-stream have the same weight. Figure 1.4 shows how SC circuits can outperform conventional image-processing circuits when environmental noise affects their inputs. The quality of output images of conventional circuits—even when noise reduction techniques are used—quickly drops with noise appearance, while SC circuits continue to produce acceptable results.

Due to its inherent probabilistic behavior and error tolerance, SC is suitable for utilizing nanoscale devices such as memristors and magnetic-tunnel junction devices. Knag et al.

[66] exploit the memristors' nondeterministic behavior to implement hybrid SC systems. In their designs, the memristors' stochasticity is essential in generating inexpensive SNs. Similarly, Onizawa et al. [97] use magnetic-tunnel junctions for efficient SN generation. Spintronics has also been suggested as another platform that exploits SC's properties [132].

Now let us look at the relation between the Boolean functions (BFs) of a combinational circuit and the stochastic function (SF) it implements. A single-output combinational circuit, such as the AND gate of Figure 1.3 corresponds to a BF that, in general, is of the form $z = f(x_1, ..., x_n)$, in which x_i's are the input Boolean variables and z is the output. Each Boolean variable y_i corresponds to an SN Y_i, whose value is represented by p_{Y_i}. The SF that corresponds to a BF is the relation between the value of the output SN and the values of the inputs SNs. For the AND gate of Figure 1.3 this function is $p_Z = F_{\text{AND}}(p_X, p_Y) = p_X \times p_Y$, as discussed before. For ease of reading, we occasionally denote the value of an SN Y_i with the same symbol Y_i. In such cases, the meaning of the symbol is known from the context. So the SF of the AND gate may also be denoted as $Z = F_{\text{AND}}(X, Y) = X \times Y$.

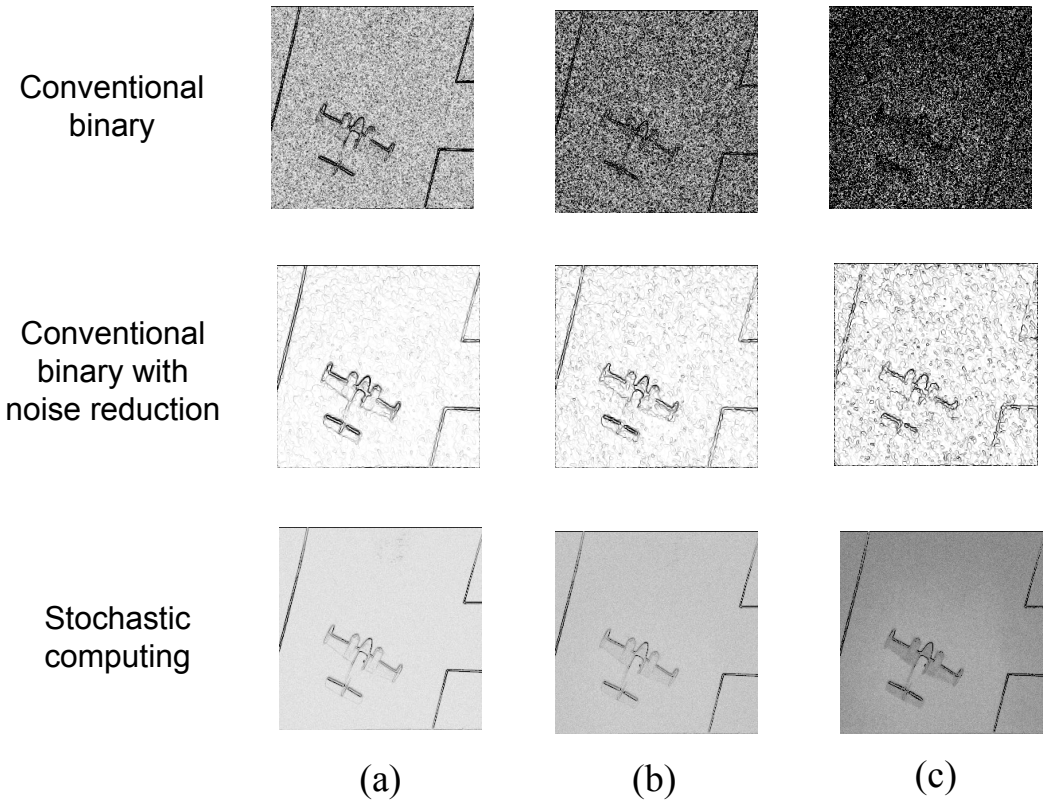


Figure 1.4: Edge-detection performance for three implementation methods with noise levels of (a) 5%, (b) 10% and (c) 20%.

Figure 1.5 shows an SC circuit implementing the SF $Z = \frac{1}{4} + \frac{1}{2}X_1X_2$. Again the number represented by each bit-stream is the probability of seeing a 1 in it; this is known as the *unipolar* format. There are several other possible formats, i.e., different ways of interpreting the value of a bit-stream, that will be discussed in Chapter 2. For example, the SNs X_1, X_2, Z appearing at x_1, x_2, z are $9/12, 8/12, 6/12$, respectively. The circuit has two primary inputs x_1 and x_2, and two auxiliary inputs r_1 and r_2. The auxiliary inputs are constant SNs of value $1/2$. The NAND gate of Figure 1.5 implements the stochastic function $Y_1 = 1 - X_1X_2$, which involves multiplication and subtraction (The relation between the elementary gates and the SFs they implement will be discussed in Chapter 2). The OR gate implements $Y_2 = R_1 + R_2 - R_1R_2$, and since $R_1 = R_2 = 1/2$, we have $Y_2 = 3/4$. Finally, the XOR gate implements the function $Z = Y_1 + Y_2 - 2Y_1Y_2 = \frac{1}{4} + \frac{1}{2}X_1X_2$. Again notice that the few simple gates of Figure 1.5 implement a relatively complicated SF.

SC can be categorized as an approximate computing technique. Approximate computing [77] has recently gained attention because it provides a promising solution to energy/power barriers of modern IC applications [35] [115] [116]. These applications are usually the ones that can naturally tolerate small errors. For instance, image processing and machine learning applications are naturally error-tolerant. Similarly, embedded systems that process sensory data do not need accurate computations because their input data is noisy. In these applications, it is possible to use hardware and software techniques that sacrifice accuracy for energy/power efficiency. For example, it is possible to reduce the supply voltage of a circuit in order to lower its power consumption at the cost of producing occasional incorrect results [121]. Another approach is to employ arithmetic circuits that

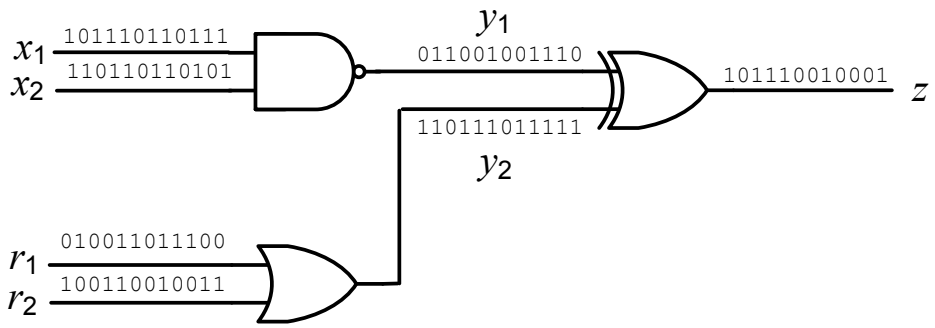


Figure 1.5: Stochastic computing circuit implementing the function $Z = \frac{1}{4} + \frac{1}{2}X_1X_2$. The SN represented by each bit-stream is the probability of seeing a 1 in a randomly chosen position.

execute approximate rather than exact algorithms; these circuits tend to be more power-efficient than their exact counter-parts [61]. Software approximation techniques also exist; for example loop perforation can reduce the number of iterations in a program loop, at the cost of less accuracy [115].

SC provides a clear accuracy-energy tradeoff: both the accuracy and the energy consumption of the circuit increase with length N of the SNs used during the computation. This makes SC suitable for scenarios that require multiple levels of accuracy, or perhaps require an adaptive approximation, because the same SC circuit provides multiple accuracy levels when used for a different number of clock cycles. One such scenario can occur in an embedded processor that processes sensor data such as images captured by a camera. In these cases, the SC circuit can perform low-cost pre-processing, image edge detection for instance, on the sensor data before handing it to the main processor. Figure 1.6 shows a high-level overview of this scenario, where a physical phenomenon is captured by a sensor and is converted into an analog signal. Usually, the analog signal is converted to a digital signal and is processed by a digital processor. However, it is possible to add an intermediate step to the analog-to-digital conversion process, in which the signal is represented by an SN and processed via SC circuits [3].

1.3 Dissertation Summary

This dissertation presents our research and contributions in the field of SC. Our main aim was to address theoretical questions such as "how to design an arbitrary SC circuit?" or "what is the role of correlation in SC?" We also analyzed the effect of different errors in SC, as well as its applicability to image-processing applications.

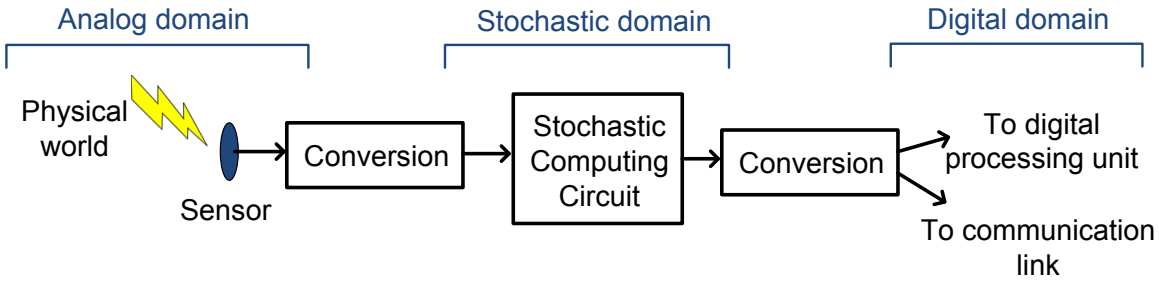


Figure 1.6: SC circuit used in an embedded system; SC circuit performs low-cost pre-processing before the sensor signal is converted to a digital number.

Our research started with a comprehensive survey [2] of the previous work from a modern perspective, where we argued that the small size, error resilience, and probabilistic features of SC may compete successfully with conventional methodologies in certain applications. We reviewed the key concepts of SN representation and circuit structure, and we discussed the challenges and the applications of stochastic computing.

We discovered that the notion of randomness in SNs was poorly understood in the literature and that little attention was given to the process of SN generation. To address this, we gave a rigorous definition for SN and excluded the notion of randomness from it. We also analyzed the effect of SN generation on the accuracy and runtime of SC circuits. Chapter 2 gives a detailed discussion of the basic concepts of SC (SNs, accuracy, correlation, etc.). It also surveys the history of the field and highlights SC's applications. Furthermore, Chapter 2 discusses the challenges of the field, some of which will be addressed in this dissertation. The subsequent chapters of this dissertation highly rely on the basic concepts defined in Chapter 2, so a reader unfamiliar to the field should read Chapter 2 first.

Since the early days of SC, many circuits have been designed for specific applications. However, most of the designs were ad hoc, or were simple assemblies of predesigned components; systematic methods of designing SC circuits were largely unknown. To address this, we defined the following synthesis problem: given a target function $Z = F(X_1, ..., X_n)$, can we find an SC circuit to implement it? Figure 1.7 illustrates the synthesis problem with an example. A major part of the dissertation focuses on solving the synthesis problem under different conditions.

Our main synthesis method, which is based on spectral transforms [8], starts from a polynomial target function and produces a combinational circuit whose stochastic function is the desired target function. We first prove that the Fourier transform of a Boolean func-

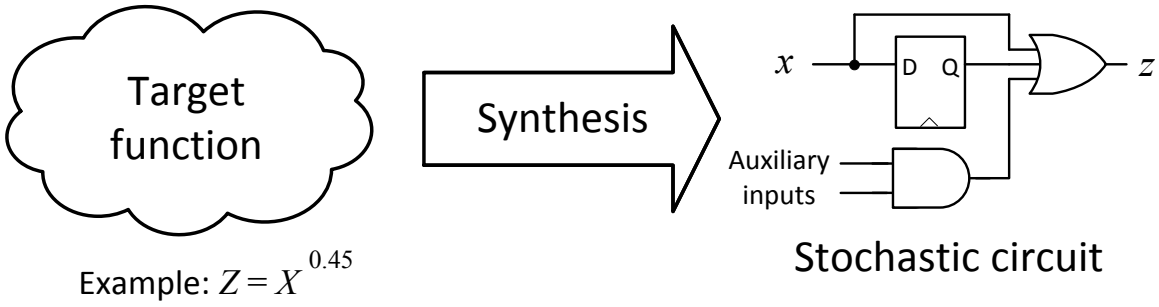


Figure 1.7: Example illustrating the SC synthesis problem: given a target function, find an SC circuit implementing it.

tion f reveals the stochastic behavior of f. Based on this discovery, we propose applying inverse Fourier transforms to a target stochastic function F in order to obtain a Boolean function f that implements it. The early version of our synthesis method appeared in 2012 [1], after which many optimization and improvement steps were added to the synthesis process. The resulting synthesis tool called STRAUSS (Spectral TRAnsform Use in Stochastic circuit Synthesis), will be discussed in Chapter 3.

Figure 1.3b, as discussed before, illustrates an example involving correlated SNs. The bit-streams appearing on the x and y inputs of the AND gate are identical, implying that they are highly correlated. This led to an unexpected result. When we introduced the AND gate as a multiplier, we mentioned that it performs multiplication only if the input SNs are independent. Similar correlation assumptions are made for the circuits that exist throughout the SC literature. It was generally believed that correlated SNs produce inaccurate results in SC circuits. We show, for the first time, how to quantify correlation of arbitrary SNs. For this purpose, we introduce a new measure called *stochastic computing correlation* (SCC) which is especially suitable for SC. Furthermore, we show that some Boolean functions are *correlation insensitive* (CI), meaning that correlation among their inputs will not affect their accuracy. This property can be used to reduce the cost of random number generation through sharing. In addition, we show that by deliberately using correlated inputs, we can change the functionality of a circuit and consequently we can implement certain useful functions very efficiently.

For example, an XOR gate with Boolean function $z = x \oplus y$ implements the stochastic function $Z = X + Y - 2XY$ when fed by independent (uncorrelated) inputs. However, if we apply highly correlated inputs to the XOR gate, as shown in Figure 1.8, the circuit implements a different function $Z = |X - Y|$, which turns out to be a useful function in image-processing tasks. The material related to correlation appears in Chapter 4 where we give rigorous definitions for correlation insensitivity and SCC. We also show how to systematically design circuits that involve correlation.

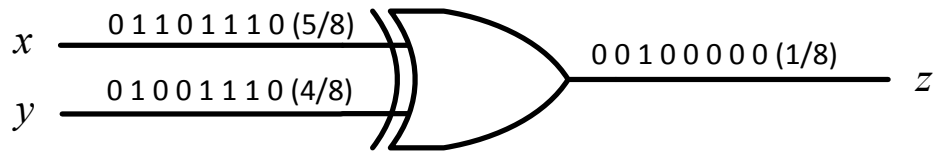


Figure 1.8: XOR gate with correlated inputs. It implements the stochastic function $Z = |X - Y|$, instead of $Z = X + Y - 2XY$ which is implemented by the same circuit using independent (uncorrelated) inputs.

Chapter 5 of this dissertation discusses different errors affecting SC circuits, including those induced by the environment. It was mentioned earlier that SC circuits are naturally tolerant of errors. However, to compare their behavior with that of conventional circuits, we have to quantify the effect of different errors. Chapter 5 discusses two methods of analyzing circuits that involve soft errors. There is also a brief discussion of the effect of voltage/frequency scaling on SC circuits.

The main application considered in our research is image processing, which will be covered in Chapter 6. We describe the design of an SC vision chip that employs efficient SC image-processing circuits. This design is suitable for real-time operation in low-energy/low-power applications such as medical implants. We show how SC circuits and image sensors can be integrated with no number conversion overhead. Furthermore, we show how an SC circuit can produce output images whose quality improves over time. This is achieved by exploiting a property of SC call progressive precision (PP) which will be introduced in Chapter 2. Finally, concluding remarks and future directions appear in Chapter 7.

Chapter 2

Stochastic Computing Basics

Stochastic computing (SC) was briefly introduced in the previous chapter. This chapter provides a more comprehensive discussion using several examples. We start by giving a formal definition for stochastic numbers and functions, and then we introduce the basic components and conversion units used in SC. Accuracy and precision of SC circuits will also be discussed before we survey SC's history and applications. More details of the history and applications can be found in [2], while the relation between Boolean functions and stochastic functions are from [7].

2.1 Stochastic Numbers and Functions

As mentioned earlier, An N-bit stochastic number (SN) X containing N_1 1's and $N - N_1$ 0's has the value $p_X = N_1/N \in [0, 1]$. Clearly, the stochastic representation of a number is not unique. SC uses a redundant number system in which there are $\binom{N}{N_1}$ possible representations for each number N_1/N. For example, with $N = 4$, there are six ways to represent $1/2$: 0011, 0101, 0110, 1001, 1010, and 1100. Moreover, a bit-stream of length N can only represent numbers in the set $\{0/N, 1/N, 2/N, ..., (N-1)/N, N/N\}$, so only a small subset of the real numbers in $[0, 1]$ can be expressed exactly in SC. More formally, we can define an SN as follows.

Definition 2.1: A binary sequence of length N is a *stochastic number* X of value $p_X = N_1/N$, where N_1 is the number of 1's in X.

For example, the SN on output line z of Figure 1.5 has the value $p_Z = 6/12$. Since p_X always lies in the real-number interval $[0, 1]$, it can be seen as the probability of observing

1 in any randomly selected position of X. This interpretation is called the *unipolar* (UP) format and represents real numbers over the unit interval $[0, 1]$, which is also the probability domain. The value p_X is also referred to as signal intensity, pulse rate, or frequency in different contexts. In neurobiology, for instance, a neural spike train can be modeled by a bit-stream X and its intensity can be represented by p_X [17]. In this dissertation, we occasionally use the same symbol X for an SN and its value. There will be no ambiguity since the meaning of the symbol will be known from the context.

Note that Definition 2.1 places no constraints on the randomness properties of an SN. In fact, bit-streams like 01010101 and 0000000011111111 are both acceptable representations of the number $1/2$ even though they both look very "non-random." The reason behind this selection will be discussed later in Section 2.3.

In addition to the unipolar format, several alternative SN formats have been proposed. Gaines [38] considers mapping the natural range of SNs, that is, the interval $[0, 1]$ to different symmetric ranges, and discusses the computation elements needed. One such mapping is from $x \in [0, 1]$ to the range $y \in [-1, 1]$ via the function $Y = 2X - 1$. This is the *bipolar* (BP) stochastic representation, which has the benefit of including negative numbers in a natural way. Table 2.1 shows several different SN formats and their relation to the UP format. The third format is a new one we call *inverted bipolar* (IBP) which is the inverse of BP. While, the IBP and BP formats are essentially equivalent, IBP is more convenient to use with spectral transforms, where the Boolean values 0 and 1 are replaced by $+1$ and -1, respectively. This small notational change greatly simplifies the analysis and synthesis of circuits in the spectral domain. The IBP format is mainly used in Chapter 3 of this dissertation. The last format shown in the table, is a format in which the value is represented by the ratio of 1's to 0's of the bit-stream. This gives the representation a large, albeit sparse, range [85].

Example 2.1: Consider the bit-stream 0110101101 of length 10 containing six 1's and four 0's. It represents the UP number 0.6, the BP number 0.2, the IBP number -0.2, and the ratio number 1.5. □

Example 2.2: To illustrate the full range of the different formats, we list all the non-redundant bit-streams of length $N = 8$, and display their values using different SN formats in Table 2.2. Note that only the number of 1's (and 0's) of a bit-stream, and not their position, is important. So the table excludes the redundant bit-streams that have equal number of 1's. Figure 2.1 illustrates the same SN values and also shows the range of each number

Table 2.1: Interpretations of a bit-stream of length N with N_1 1's and N_0 0's.

Format	**Number value**	**Number range**	**Relation to unipolar value p_X**
Unipolar (UP)	N_1/N	$[0, 1]$	p_X
Bipolar (BP)	$(N_1 - N_0)/N$	$[-1, +1]$	$2p_X - 1$
Inverted bipolar (IBP)	$(N_0 - N_1)/N$	$[-1, +1]$	$1 - 2p_X$
Ratio of 1's to 0's	N_1/N_0	$[0, +\infty]$	$p_X/(1 - p_X)$

Table 2.2: Representative bit-streams of length $N = 8$ and their values in different SN formats.

Bit-stream	**UP**	**BP**	**IBP**	**Ratio**
00000000	0	-1	$+1$	0
00000001	$1/8$	$-3/4$	$+3/4$	$1/7$
00000011	$2/8$	$-2/4$	$+2/4$	$1/3$
00000111	$3/8$	$-1/4$	$+1/4$	$3/5$
00001111	$4/8$	0	0	1
00011111	$5/8$	$+1/4$	$-1/4$	$5/3$
00111111	$6/8$	$+2/4$	$-2/4$	3
01111111	$7/8$	$+3/4$	$-3/4$	7
11111111	1	$+1$	-1	$+\infty$

format. As evident from the table and the figure, the ratio format provides a big but very sparse and irregular range, and hence, it is not widely used. □

In addition to the number formats defined in Table 2.1, several other ways of stochastic encoding exist. Gaines [39] defines a dual-rail representation, where two wires carry the information of the SN. In another recently proposed dual-rail format, the value is denoted by the ratio of the BP number represented by each rail [18]. This representation covers the range $[-\infty, +\infty]$. Similar multi-rail representations also exist in the literature [71]. In some cases, the different lines carrying a number have different weights, which makes the representation a hybrid of weighted binary and unweighted stochastic [27].

We already saw that SC multiplication $p_Z = F(p_X, p_Y) = p_X \times p_Y$ is performed by a single AND gate which has the Boolean function (BF) $z = x \wedge y$.[1] We now discuss the links between the BF of a combinational logic circuit and its stochastic function (SF). Unlike a

[1] When dealing with BFs, the $\vee$ and $\wedge$ symbols are usually used to denote the logical OR and logical AND functions, respectively. However, these symbols are occasionally replaced by $+$ and juxtaposition throughout the dissertation. Since Boolean variables are denoted by lowercase letters $x_1, x_2, ...$, the symbols will not be confused by addition and multiplication, which apply to numerical values $X_1, X_2, ...$

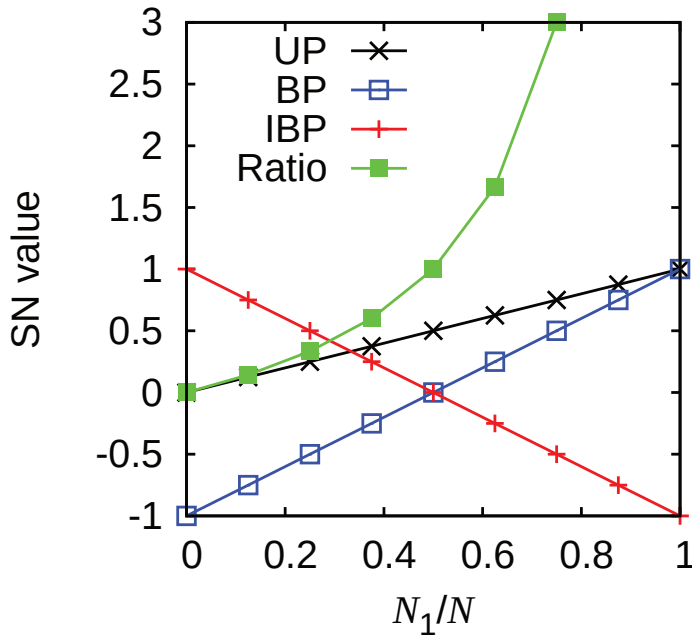


Figure 2.1: SN values of a bit-stream of length N with N_1 1's for different formats. The points in the graph correspond to the rows of Table 2.2.

BF that maps the discrete space $\{0,1\}^n$ to $\{0,1\}$ (n is the number of inputs of the BF), SFs are arithmetic functions over the real numbers and map $[0,1]^n$ to $[0,1]$ in the UP format.

Suppose some n-input single-output combinational circuit C realizes the BF $z(x_1, x_2, ..., x_n)$. This function has the canonical sum-of-minterms form

$$z(x_1, x_2, ..., x_n) = \bigvee_{i=0}^{2^n-1} c_i \wedge m_i \tag{2.1}$$

where the c_i's are 0-1 constants. The m_i's are *minterms* of the form $\tilde{x}_{1,i} \wedge \tilde{x}_{2,i} \wedge ... \wedge \tilde{x}_{n,i}$ where $\tilde{x}_{j,i}$ is either x_j or $\bar{x}_j$. For example, the sum-of-minterms representation of the AND gate is $z_{\text{AND}}(x_1, x_2) = m_3 = x_1 \wedge x_2$. For the XOR function we have $z_{\text{XOR}}(x_1, x_2) = m_2 \vee m_3 = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$.

Now suppose n SNs are applied to the inputs of C. If we use X_i to denote[2] the (unipolar) probability value of the SN on x_i, then $\bar{x}_i$ has the probability value $1 - X_i$. The following theorem gives C's output probability Z, and so defines its SF.

Theorem 2.1: Let $z(x_1, x_2, ..., x_n)$ be a Boolean function defined by Eq. (2.1). The stochastic function $Z(X_1, X_2, ..., X_n)$ implemented by z, assuming all input SNs are in-

[2]Using the alternative notation that replaces p_{X_i} with X_i for better readability.

dependent, is

$$Z(X_1, X_2, ..., X_n) = \sum_{i=0}^{2^n-1} c_i M_i \tag{2.2}$$

where $M_i = \widetilde{M}_{1,i}\widetilde{M}_{2,i}...\widetilde{M}_{n,i}$ with $\widetilde{M}_{j,i} = X_j$ $(= p_{X_j})$ if the corresponding minterm m_i of Eq. (2.1) has $\tilde{x}_{j,i} = x_j$; $\widetilde{M}_{j,i}$ is $1 - X_j$ $(= 1 - p_{X_j})$ if $\tilde{x}_{j,i} = \bar{x}_j$.

Proof. We want to find Z $(= p_Z)$, i.e., the value of the SN that corresponds to z. In unipolar format, this value corresponds to the probability of seeing a 1 in z, so $Z = P\{z = 1\}$, where $P\{A\}$ denotes the probability of event A. Following Eq. (2.1), we get

$$Z = P\left\{\left(\bigvee_{i=0}^{2^n-1} c_i \wedge m_i\right) = 1\right\} = P\left\{\bigvee_{i=0}^{2^n-1} (c_i \wedge m_i = 1)\right\}$$

Since the minterm functions (the m_i's) are disjoint, the events specified by different terms of the OR function are also disjoint. Based on this and the fact that c_i's are constants, we get

$$Z = \sum_{i=0}^{2^n-1} c_i P\{m_i = 1\} = \sum_{i=0}^{2^n-1} c_i P\left\{(\tilde{x}_{1,i} = 1) \wedge (\tilde{x}_{2,i} = 1) \wedge ... \wedge (\tilde{x}_{n,i} = 1)\right\}$$

Because the input SNs are independent, the ANDed events of the equation above become independent, yielding

$$Z = \sum_{i=0}^{2^n-1} c_i \left(P\{\tilde{x}_{1,i} = 1\} \times P\{\tilde{x}_{2,i} = 1\} \times ... \times P\{\tilde{x}_{n,i} = 1\}\right)$$

The probability of event $x_{j,i} = 1$ is X_j if $x_{j,i} = x_j$; it is $1 - X_j$ otherwise. So in the general case we have $P\{x_{j,i} = 1\} = \widetilde{M}_{j,i}$, yielding Eq. (2.2) □

This key result was first shown by Parker and McCluskey [100] using rather ad hoc notation. Theorem 2.1 implies that the SF Z has a polynomial form, in which all the variables X_j appear with a power of at most 1. This special type of polynomial is called a *multilinear* polynomial. Note that each M_i corresponds to a minterm and is the probability of the corresponding input combination. These probabilities have the form stated in Theorem 2.1 when the input SNs are independent. As will be shown in Chapter 4, if the input SNs are correlated, the M_i's may take a different form. For the XOR gate example with

independent inputs, Theorem 2.1 implies

$$Z_{\text{XOR}}(X_1, X_2) = X_1(1 - X_2) + (1 - X_1)X_2$$

which, when multiplied out, becomes

$$Z_{\text{XOR}}(X_1, X_2) = X_1 + X_2 - 2X_1X_2$$

The sum-of-minterms-style probability expression (2.2) can be seen as a *canonical representation* of the SF Z realized by the BF z. It thus captures z's stochastic behavior with respect to the basic UP format. When the X_i's are restricted to 0 and 1, and sum is interpreted as OR, Eq. (2.2) reduces to Eq. (2.1), so Z is effectively an interpolation of z in the real-number domain. Equation (2.2) is also easily converted to other widely used SN formats. To convert from UP to BP, for instance, we replace X_i by a new SN X_i' defined as $X_i' = 2X_i - 1$.

The canonical representation of Eq. (2.2) can also be expressed as the inner product of two vectors. The first is the truth-table vector $\boldsymbol{C}_z = [c_0 \quad c_1 \quad \ldots \quad c_{2^n-1}]$ defining z in terms of the constant coefficients in Eq. (2.1). The second is the input vector $\boldsymbol{M} = [M_0 \quad M_1 \quad \ldots \quad M_{2^n-1}]$ specifying the probability distribution of the input combinations, or equivalently, the stochastic minterm functions. We can now rewrite Eq. (2.2) as follows, where "$\cdot$" denotes the inner-product operation:

$$Z(X_1, X_2, ..., X_n) = \boldsymbol{C}_z \cdot \boldsymbol{M} = [c_0 \quad c_1 \quad \ldots \quad c_{2^n-1}] \cdot [M_0 \quad M_1 \quad \ldots \quad M_{2^n-1}] \tag{2.3}$$

The c_i elements in Eqs. (2.1), (2.2) and (2.3) are the same and belong to the binary set $\{0, 1\}$. Since SFs deal with real numbers, we can further generalize Theorem 2.1 by allowing the c_i's to be any numbers in the real interval $[0, 1]$. Such generalized c_i coefficients can be interpreted as constant SNs applied to the circuit when the corresponding minterm m_i is activated or set to 1. For example, if $c_0 = 0$, $c_1 = 0.5$, $c_2 = 0.5$, and $c_3 = 1$ (or in vector form $[0 \quad 0.5 \quad 0.5 \quad 1]$), then Eq. (2.2) for $n = 2$ becomes

$$Z(X_1, X_2) = 0.5X_1(1 - X_2) + 0.5(1 - X_1)X_2 + X_1X_2 = 0.5(X_1 + X_2) \tag{2.4}$$

which is well-known in the SC literature as the *scaled add* function. The coefficients $c_1 = c_2 = 0.5$ in (2.4) imply that when minterms m_1 and m_2 are activated, a constant SN of

value 0.5 should propagate to the output. Such constant probabilities can be obtained from (pseudo) random number sources. These sources often appear as auxiliary inputs in the corresponding circuit, as we will see in the next section.

2.2 Basic SC Components and Conversion Circuits

Figure 2.2 shows multipliers and adders for the top three number formats listed in Table 2.1. We have already shown how the AND gate implements the UP multiplier. We now explain how multiplication is performed in BP and IBP formats.

The XNOR gate of Figure 2.2 has the following minterms (in vector form): $[1 \quad 0 \quad 0 \quad 1]$, so according to Theorem 2.1, the UP SF implemented by the XNOR gate is

$$Z_{\text{XNOR}} = (1-X)(1-Y) + XY = 1 - X - Y + 2XY$$

To convert the SF to BP format, we need to apply the corresponding formula from Table 2.1 to every SN. So each SN X should be replaced by a new SN $X' = 2X - 1$, which means that $X = \dfrac{X'+1}{2}$. Consequently, we have:

$$\begin{aligned}
\frac{Z'_{\text{XNOR}}+1}{2} &= 1 - \frac{X'+1}{2} - \frac{Y'+1}{2} + 2\left(\frac{X'+1}{2}\right)\left(\frac{Y'+1}{2}\right) \\
Z'_{\text{XNOR}} + 1 &= 2 - X' - 1 - Y' - 1 + 1 + X' + Y' + X'Y' \\
Z'_{\text{XNOR}} &= X'Y'
\end{aligned}$$

So the XNOR gate is the BP multiplier. It is easily seen that the XOR gate of Figure 2.2 implements IBP multiplication.

The SC add operation for all three cases is performed by a multiplexer with an auxiliary select input r, where $p_R = 1/2$. When adding two UP numbers X and Y, the result $X + Y$ falls into the range $[0, 2]$, which is unacceptable for the range of UP numbers, i.e., $[0, 1]$. So it is necessary to scale the result by a factor such as $1/2$. This scaling factor is provided by the auxiliary input r. The final function implemented is then $Z = 0.5(X + Y)$, which is called the scaled add operation. The multiplexer of Figure 2.2d therefore acts as the scaled adder for the top three number formats of Table 2.1.

Circuits that convert binary numbers to SNs, and vice versa, are fundamental elements of SC. Figure 2.3a illustrates a widely used binary-to-stochastic conversion circuit, which we will refer to as a stochastic number generator (SNG). The conversion process involves

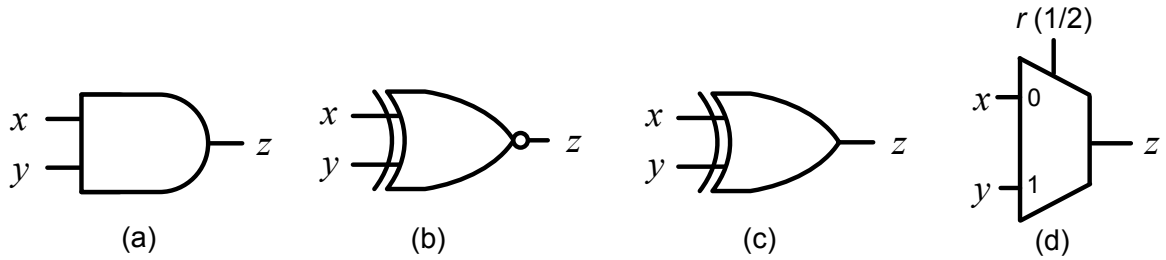


Figure 2.2: Basic SC components: (a) unipolar, (b) bipolar and (c) inverted bipolar multipliers; (d) stochastic adder for all three formats.

generating a k-bit random binary number in each clock cycle by means of a random or, more likely, a pseudo-random number generator, and comparing it to the k-bit input binary number B. The comparator produces a 1 if the random number is less than B, and a 0 otherwise. Assuming that the random numbers are uniformly distributed over the interval $[0, 1]$, the probability of a 1 appearing at the output of the comparator at each clock cycle is equal to $B/2^k$.

Linear feedback shift registers (LFSRs) [43] are the most common units used as the (pseudo) random number generator of Figure 2.3a. These are finite state machines (FSMs), i.e., sequential circuits that go through a specific sequence of states deterministically. Sequences generated by LFSRs pass many randomness tests, and so are very suitable for SC. As we will discuss later, however, it is possible to replace the LFSR with other deterministic number sources such as counters, and still produce acceptable SNs.

Converting an SN X to binary is much simpler. The SN's value p_X is carried by the number of 1's in its bit-stream form, so it suffices to count these 1's in order to extract p_X. Figure 2.3b shows a counter that performs this conversion. Note that the counter can also serve as an efficient storage register for X.

Example 2.3: Figure 2.4 shows a stochastic circuit that implements the arithmetic function $Z = X_1X_2X_4 + X_3(1 - X_4)$ [75]. The inputs X_1, X_2, X_3 and X_4 are provided in conventional binary form and must be converted to SNs via SNGs. We know that the AND gate is a multiplier, so (with high probability) it outputs $X_5 = 4/8 \times 6/8 = 3/8$. The probability of 1 at z is the probability of 1 at both x_4 and x_5 plus the probability of 0 at x_4 and a 1 at x_3. Hence, the SN appearing at z has the value $Z = X_1X_2X_4 + X_3(1 - X_4)$, and the counter at the output converts it to conventional binary form.

The result appearing at Z in Figure 2.4 represents $6/8$ exactly only if we get six 1's at the output in 8 clock cycles, otherwise the counter outputs an SN other than $6/8$. The probability of obtaining exactly six 1's is $P\{Z = 6/8\} = \binom{8}{6}(6/8)^6(2/8)^2 \simeq 0.31$, implying there is a 69% chance that we do not get six 1's, and the computation has some inaccuracy. The specific SNs in Figure 2.4 have been chosen to avoid inaccuracy. Indeed, even if we use a high-quality random number source such as the "one million random digits" table [111] to generate the SNs, we will probably still find some inaccuracy. For example, using the first four lines of this table to generate SNs, we obtain $X_1 = 01100010$, $X_2 = 00111111$, $X_3 = 11111111$ and $X_4 = 00000100$. Applying these numbers to the circuit in Figure 2.4 yields $Z = 11111011 = 7/8 \neq 6/8$. □

Figure 2.4 also illustrates the high cost of number conversion. The computation part of the circuit consists of only an AND gate and a multiplexer, while the conversion units include four random number generators, four comparators, and a counter. Since the four primary inputs must be independent, each SNG contains a separate random number generator. Qian et al. [109] report that in some image-processing applications, up to 80% of the stochastic circuit area is consumed by conversion units.

2.3 Accuracy and Precision

Accuracy concerns arise in SC for several reasons. The one previously discussed is due to the fluctuations inherent in random numbers. Correlations among the SNs being processed also lead to inaccuracies. Surprisingly, it is generally not desirable to use truly random number sources to derive or process SNs. As we will explain here, deterministic or pseudo-random number generators like LFSRs form the best driving sources for SNGs from both a

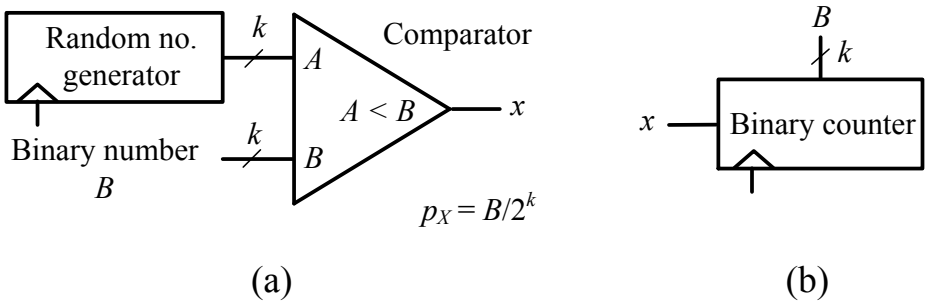


Figure 2.3: SC conversion units: (a) binary-to-stochastic converter, also referred to as a stochastic number generator (SNG), and (b) stochastic-to-binary converter.

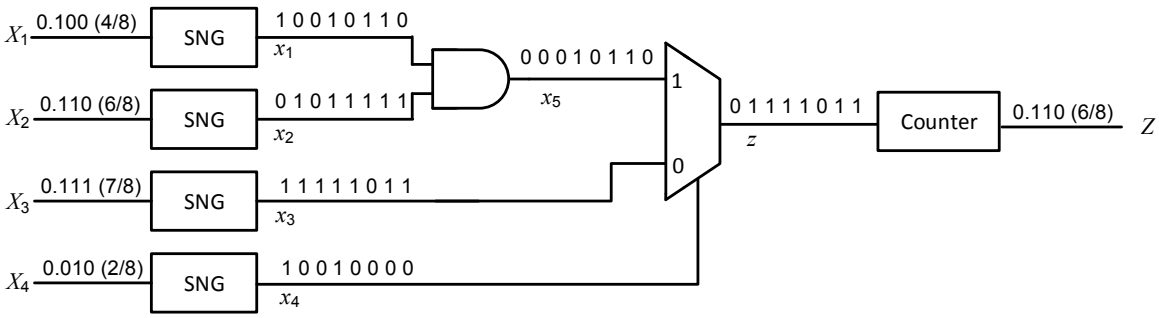


Figure 2.4: Stochastic circuit realizing the arithmetic function $Z = X_1X_2X_4 + X_3(1-X_4)$ along with its conversion units [75].

practical and theoretical point of view. They can be used to implement SC operations with high and, in some special cases, complete accuracy.

Inaccuracy in SC has several distinct sources: random fluctuations in SN representation, similarities (correlations) among the numbers that are being combined, and physical errors that alter the numbers. Jeavons et al. [57] provide a definition for independence among bit-streams used in SC.

Definition 2.2: Two bit-stream X and Y with unipolar values p_X and p_Y are *uncorrelated* or *independent* if and only if

$$p_X \times p_Y = p_{X \wedge Y}$$

where $X \wedge Y$ is a bit-stream obtained by performing bit-wise AND operation on X and Y, and $p_{X \wedge Y}$ denotes its unipolar value. Otherwise, the sequences are called correlated.

Example 2.4: This example shows how correlation can lead to inaccuracy. The eight-bit SNs 11110000 and 01010101, both representing $1/2$, are uncorrelated according to Definition 2.2. Their product, obtained by ANDing them (as in Figure 1.3), is $01010000 = 1/4$. In contrast, 11110000 and 00001111 are correlated, and their product $00000000 = 0$, which is far from the correct result. □

Quantifying correlation (of two SNs) is a difficult task which has received little attention in the past. We address this issue in depth in Chapter 4. To reduce such inaccuracies, SNGs are needed which produce SNs that are sufficiently random and uncorrelated. Once again, LFSRs are suitable choices because they are capable of generating multiple uncorrelated bit-streams. The preferred LFSRs have m flip-flops and cycle through $N = 2^m - 1$ distinct

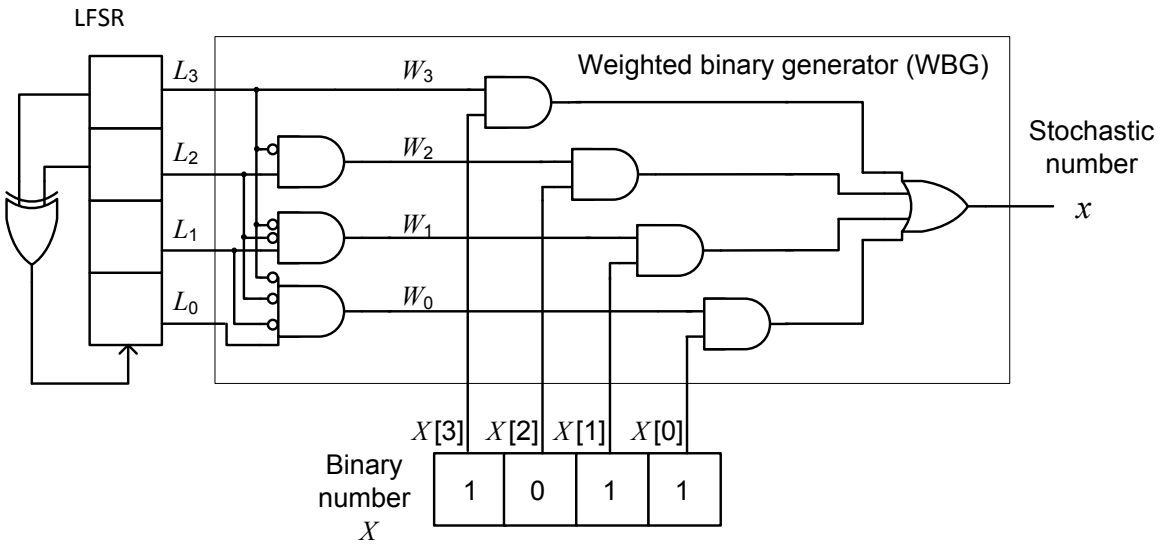


Figure 2.5: The weighted binary SNG proposed by Gupta and Kumaresan [49].

states, the maximum possible since the all-0 state is excluded. As mentioned, the generated bit-streams pass various randomness tests, although they are deterministic. For example, they contain (almost) equal numbers of 0's and 1's, as well as runs of 0's and 1's whose numbers and lengths correspond to those of a Bernoulli sequence. In addition, shifted versions of LFSR sequences have low correlation [43] [57]. Poppelbaum [104] noted that if the LFSRs are large enough, they resemble ideal random sources. Hence, noise-like random fluctuations appear as errors in the generated SNs. These errors can be reduced at a rate $N^{-1/2}$ by increasing the number length N. Much less is known about how this error size changes as SNs go through a sequence of operations.

Surprisingly, LFSRs can be used to generate SNs that are, in certain cases, guaranteed to be exact. This was demonstrated for multiplication by Gupta and Kumaresan [49] who introduced a new type of SNG that we call a weighted binary SNG. Figure 2.5 shows a 4-bit version, which converts a 4-bit binary number X to an SN of length 16. The pseudo-random source is a 4-bit LFSR to which the all-0 state is artificially added (details not shown). The SNG's behavior with $X = 11/16$ is illustrated in Table 2.3. The bit-streams L_i generated by the LFSR, the intermediate signals W_i, and the output SN are all shown. The key features of the Gupta-Kumaresan design, which immediately imply that the final SN exactly represents X, is that the W_3, W_2, W_1, and W_0 bit-streams have non-overlapping 1's, and weights of $1/2$, $1/4$, $1/8$, and $1/16$, respectively. Note that the order of the bits in the bit-streams of Table 2.3 is not important; we can reorder the columns of the table,

Table 2.3: Bit-streams generated by the circuit of Figure 2.5.

Signal	Bit-stream	Value
L_3	0 0 1 0 1 0 1 1 1 1 0 0 0 0 1 1	8/16
L_2	0 1 0 1 0 1 1 1 1 0 0 0 0 1 1 0	8/16
L_1	1 0 1 0 1 1 1 1 0 0 0 0 1 1 0 0	8/16
L_0	0 1 0 1 1 1 1 0 0 0 0 1 1 0 0 1	8/16
W_3	0 0 1 0 1 0 1 1 1 1 0 0 0 0 1 1	8/16
W_2	0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0	4/16
W_1	1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	2/16
W_0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0	1/16
x	1 0 1 0 1 0 1 1 1 1 0 1 1 0 1 1	11/16

Table 2.4: Bit-streams generated by the circuit of Figure 2.5 when the LFSR is replaced by a plain binary counter.

Signal	Bit-stream	Value
L_3	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1	8/16
L_2	0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1	8/16
L_1	0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1	8/16
L_0	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	8/16
W_3	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1	8/16
W_2	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0	4/16
W_1	0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	2/16
W_0	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1/16
x	0 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1	11/16

and the resulting bit-streams will still have the same values. Table 2.4 shows one possible ordering of the bit-streams, which is obtained by replacing the LFSR of Figure 2.5 with a plain binary counter. This is an important observation: deterministic number sources (such as a counter) can generate accurate SNs. The point is that only the number of 1's, and not their ordering, is important in the bit-streams. In fact, the same approach applies to the SNG of Figure 2.2b too; we can replace the LFSR with a counter. Note that the above scheme gives accurate results only when the period of the LFSR (or counter) used is equal to N (the length of the generated SN).

Gupta and Kumaresan further used their SNG to design a circuit that multiplies two SNs accurately. As shown in Figure 2.6 for the 4-bit case, it contains two weighted binary generators connected to a double-length LFSR. It is not hard to see that stochastic multiplication using this approach always yields exact results. Zelkin [138] employs an essentially similar SNG to design an accurate arithmetic unit for SNs.

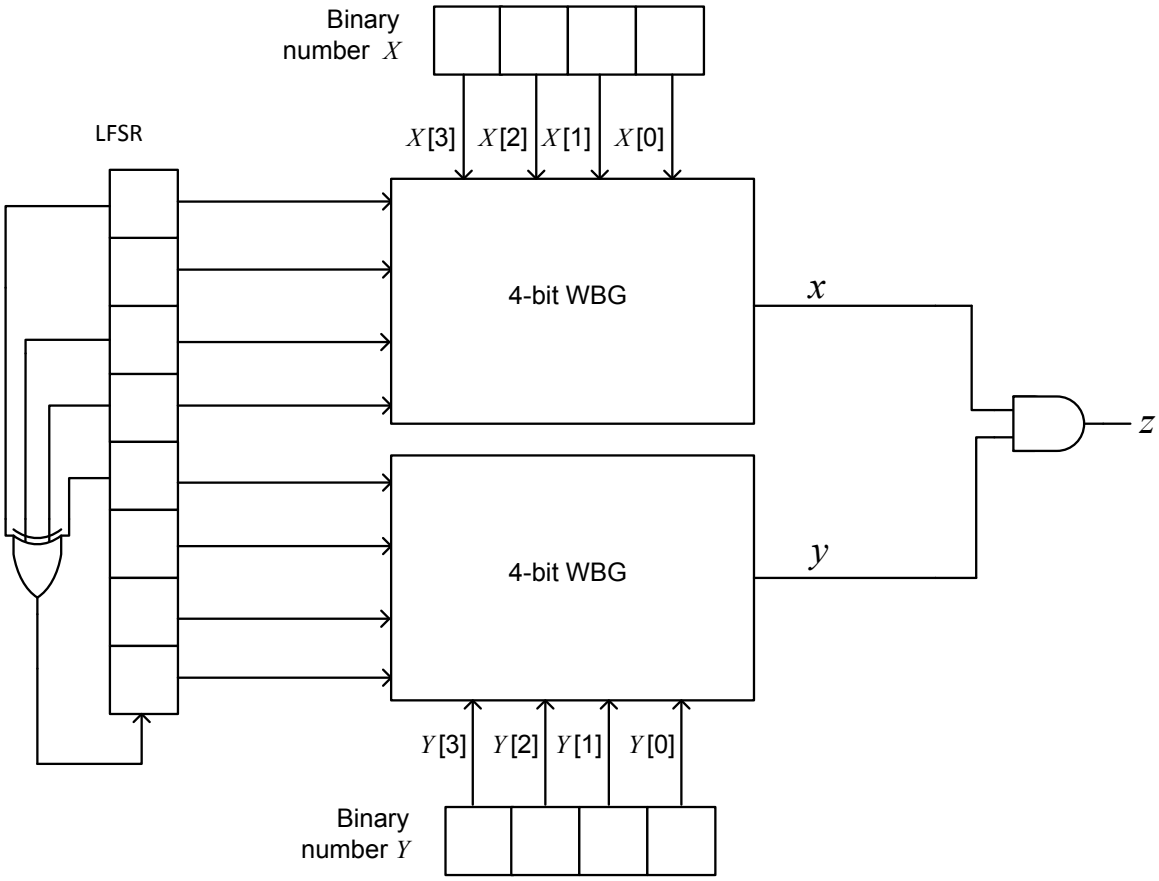
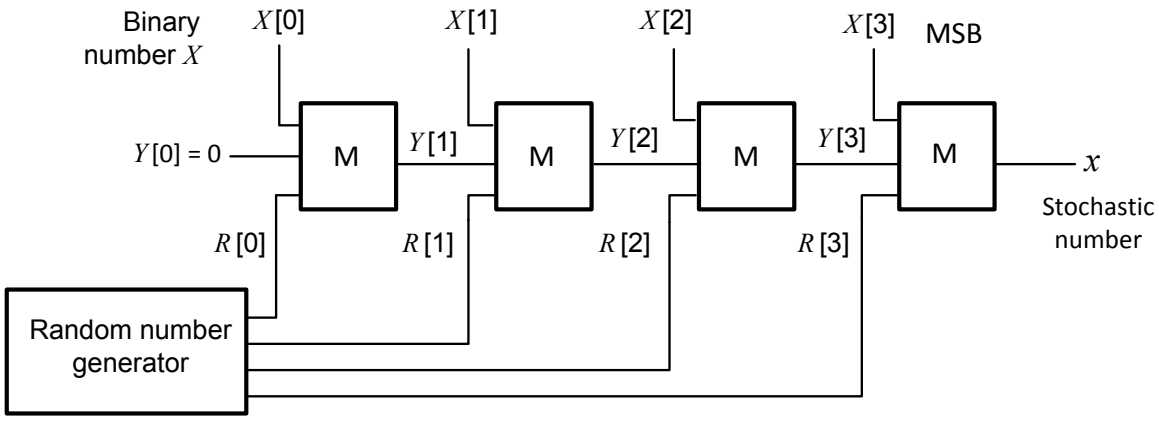


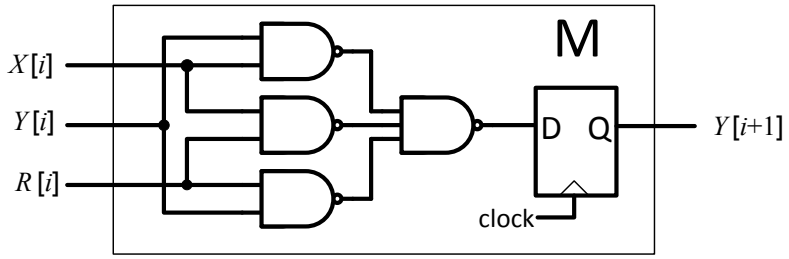
Figure 2.6: Accurate 4-bit stochastic multiplier of the type proposed by Gupta and Kumaresan [49].

The foregoing results reveal an important and perhaps unexpected aspect of SC: pseudo-random SNs can be better than random. This can be compared to the use of low-discrepancy numbers in quasi-Monte-Carlo sampling methods [123]. Quasi-Monte Carlo is similar to normal Monte Carlo, but uses carefully chosen deterministic samples instead of purely random ones. Moreover, Gupta and Kumaresan's work shows that small deterministic LFSRs can produce highly accurate results. In contrast, Gaines [38] and Poppelbaum [104] rely on very large LFSRs to ensure randomness in the generated SNs.

Jeavons et al. [57] address the SC accuracy problem by creating a mathematical framework in which another SNG type is introduced. Instead of using comparators as in the traditional SNG of Figure 2.3a, their design [130] employs a cascade of majority modules M, as illustrated in Figure 2.7. A random number generator feeds the M's with uncorrelated SNs of probability $1/2$, and Jeavons et al. [57] suggest using bits of an LFSR for



(a)



(b)

Figure 2.7: Four-bit SNG proposed by van Daalen et al. [130]: (a) overall structure; (b) module M.

this purpose. But similar to the previous SNGs, a counter can also be used as the number source. Individual bits of the binary number X to be converted are also fed to the M's, as shown in the figure, and one bit of the SN is generated per clock cycle. The SF implemented by each M is $Y[i+1] = 0.5(X[i] + Y[i])$, so for the 4-bit scenario of Figure 2.3a we have

$$p_X = \frac{1}{2}\left(X[3] + \frac{1}{2}\left(X[2] + \frac{1}{2}\left(X[1] + \frac{1}{2}X[0]\right)\right)\right) = \frac{1}{2}X[3] + \frac{1}{4}X[2] + \frac{1}{8}X[1] + \frac{1}{16}X[0]$$

Hence, the probability of seeing a 1 at x is equal to the fractional binary number represented by the $X[i]$ inputs.

Informally, the *precision* of a value is the number of bits needed to express that value. With m bits of precision, we can distinguish between 2^m different numbers. For instance, the numbers in the interval $[0, 1]$, when represented with 8-bit precision, reduce to the following 256-member set, $\{0/256, 1/256, 2/256, ..., 255/256, 256/256\}$, and their exact stochastic representation requires bit-streams of length 256. To increase the precision from 8 to 9 bits requires doubling the bit-stream length to 512, and so on. This exponential growth in data length with precision is responsible for the long computation times associated with SC. In general, an SN X of length N has $m = \log_2(N)$ bits of precision, which approximates that of an $\lceil m \rceil$-bit binary number.

The exponential length of SNs has another important consequence: storing SNs becomes inefficient. So it most SC circuits, storing SNs are avoided as much as possible. In applications where storage is absolutely necessary, SNs are typically converted to weighted binary numbers and are stored efficiently in a counter, as in Figure 2.4. They are converted back to the stochastic format when needed. This approach, however, is also costly due to excessive use of conversion circuits.

Now consider the 16-bit SNs 1111111110000000 and 0101010101010111; both represent $9/16$ and have 4-bit precision. There is a subtle difference between them, however. If we consider the first 8 bits of the sequences as an SN of lower precision (3-bit precision in this case), we obtain 11111111 and 01010101 representing $8/8$ and $4/8$, respectively. The latter provides a low-precision estimate of the full-length SN (i.e., $9/16$). This points to a potential advantage of SC: initial subsequences of an SN, if appropriately generated, can provide an estimate of a target number. We say an SN X of length N has *progressive precision* (PP) if all the SNs $\widehat{X}_k$ ($k \in \{1, 2, 3, ..., \log_2(N)\}$), composed of the first 2^k elements of X, are accurate. In other words, accuracy and precision increase steadily with SN length. SNs with good PP can therefore be seen as presenting their most significant bits first. Such behavior of SNs has been implicitly exploited in decoding applications [47]. In certain computations, this makes it possible to make decisions early by looking at the first few bits of a result. Figure 2.8 shows how p_X, the value of SN X fluctuates as N (the length of the bit-stream of X) increases for SNs of nominal value $p = 1/2$ generated by a typical SNG. SNs that rapidly converge to p are said to have good PP.

Despite the efforts of Jeavons et al. [57] and Gupta and Kumaresan [49], accuracy, or the lack thereof, continues to be a major concern in SC as low accuracy may degrade the energy efficiency promised by SC [89] [81]. Inaccuracies due to correlation, for example, worsen as the number of inputs increases, the SNs pass through multiple levels of logic,

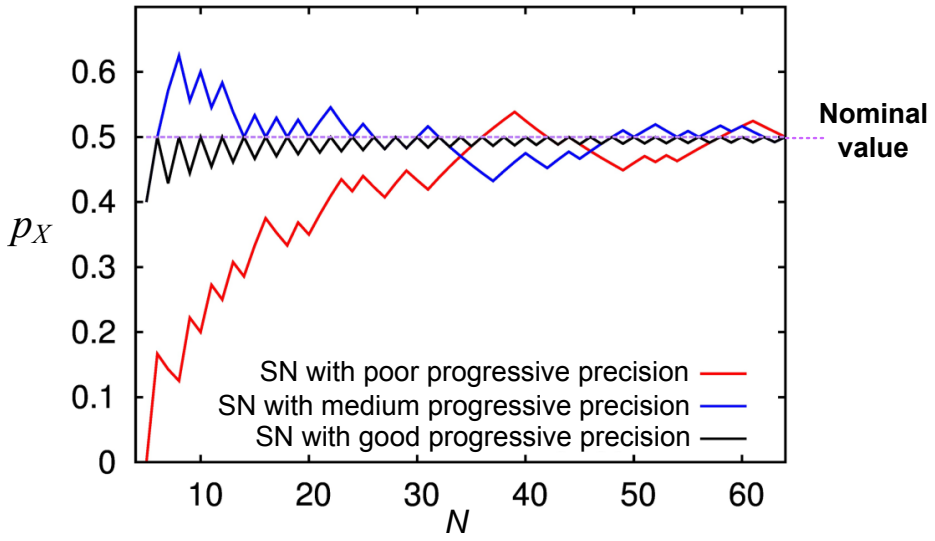


Figure 2.8: Fluctuations in p_X for 3 bit-streams as their length N increases.

or feedback is present. Reconvergent fanout, in particular, creates undesirable correlations when signals derived from a common input converge and interact. A possible but expensive solution to this problem is to convert these signals to binary and regenerate new and independent SNs from them on the fly [126]. We end this section by summarizing the known advantages and disadvantages of SC in Table 2.5

2.4 History and Applications

Over the years, SC has been recognized as potentially useful in specialized (and often embedded) systems, where small size, low power, or soft-error tolerance are required, and limited precision or speed are acceptable. Besides its intrinsic suitability for certain computation tasks, SC seems worth reexamining because it copes with some complex probabilistic issues that are becoming an unavoidable part of conventional technologies [68] and are as yet poorly understood.

Table 2.6 provides a brief historical perspective on SC from its early days until 2010. Von Neumann [134] defined fundamental ideas concerning probabilistic, error-tolerant design, and greatly influenced subsequent research. In the mid-1960s, further influenced by developments in both analog and digital computers, SC was defined and explored concurrently in the U.K. [38] [39] and the U.S. [103] [113]. Several of the few prototype stochastic machines ever actually implemented were built around that time, and they uncovered nu-

Table 2.5: Advantages and disadvantages of stochastic computing.

Feature	Advantages	Disadvantages
Circuit size and power	Tiny arithmetic components	Many random number sources and stochastic-binary conversion circuits
Operating speed	Short clock periods Massive parallelism	Very long bit-streams
Result quality	High error tolerance Progressive precision	Low precision Random number fluctuations Correlation-induced inaccuracies
Design issues	Rich set of arithmetic components	Theory not fully understood Little CAD tool support at present

Table 2.6: Timeline for the development of stochastic computing (1956-2010).

Dates	Items	References
1956	Fundamental concepts of probabilistic logic design.	[134]
1960–79	Definition of SC and introduction of basic concepts. Construction of prototype SC machines.	[38][39] [104]
1980–99	Advances in the theory of SC. Studies of specialized applications of SC, including artificial neural networks and hybrid controllers.	[57] [65] [129]
2000–10	Application to efficient decoding of error-correcting codes. New general-purpose architectures.	[42] [106]

merous shortcomings of the technology. Poppelbaum [104] observed that "short sequences are untrustworthy" and that a major drawback of SC is low bandwidth and therefore low computational speed.

It is interesting to note that the first—and also the last—International Symposium on Stochastic Computing and its Applications was held in Toulouse in 1978 [131]. Since then, interest in SC has greatly diminished as conventional binary circuits have become smaller, cheaper, faster, and more reliable. SC research has focused on a narrow range of specialized applications, such as neural networks, [16] [65], control circuits [82] [129], and reliability calculations [10] [21]. There were, however, some important theoretical discoveries [49] [57] relating to SN generation that have attracted little attention, but nevertheless have positive implications for SC.

There are other probabilistic methods in the computing literature that we do not consider here, some of which use the term "stochastic." They typically aim to achieve power-reliability trade-offs by means of probabilistic or statistical design, and differ substantially

from what we call SC. For example, Shanbhag et al. [121] and Chakrapani et al. [20] focus on supply-voltage overscaling and methods of reducing the effect of any resulting errors. Other examples of probabilistic computing hardware are found in Nepal et al. [93] and Vigoda [133]. The terms "stochastic numbers" and "stochastic arithmetic" appear in Alt et al. [11] which, however, is concerned with numerical errors in conventional binary computation.

Stochastic computing has been investigated for a variety of applications. Besides the basic operations of addition and multiplication, SC has been applied to a limited extent to division and square-rooting [129], matrix operations [83], and polynomial arithmetic [106] [109]. A more specialized application area for SC is reliability analysis [10] [21]. Since probabilities are fundamentally analog quantities, SC has been proposed for some analog and hybrid analog-digital computing tasks, often under the heading of digital signal processing [63] [101].

Neural (or neuromorphic) networks [16] [32] [65] [102] and control systems [33] [82] [15] are among the earliest and most widely studied applications of SC, and have close connections with analog computing. SC neural networks also appear as "pulse coded networks" in the literature [128] [114]. An illustrative example of a stochastic control system is found in [139], where a control unit for an induction motor is described that integrates several SC-based algorithms and a large neural network. The controller is implemented on an FPGA and is claimed to exhibit higher performance and lower hardware cost than conventional microprocessor-based designs for the same application. Figure 2.9 shows a simplified, high-level view of the motor controller. The stochastic integrators execute functions of the form $Y(n) = X(n) + Y(n-1)$ on SNs. The stochastic anti-windup controller incorporates a complex algorithm that limits any changes implied in the input speed command that might lead to improper motor operation. The stochastic neural network estimator implements in real time the key feedback-processing functions of the system, several of which are computation-intensive. An example is the hyperbolic tangent or "tansig" function, which is a typical transform function computed by an artificial neuron, and takes the form

$$tansig(x) = \frac{2}{1 - e^{-2x}} - 1$$

Zhang and Li [139] note that besides improved cost-performance figures, their SC-based design has advantages in terms of reduced design and verification effort.

Image processing is another potential application area for SC of great practical importance. Many imaging applications involve functional transformations on the pixels of an

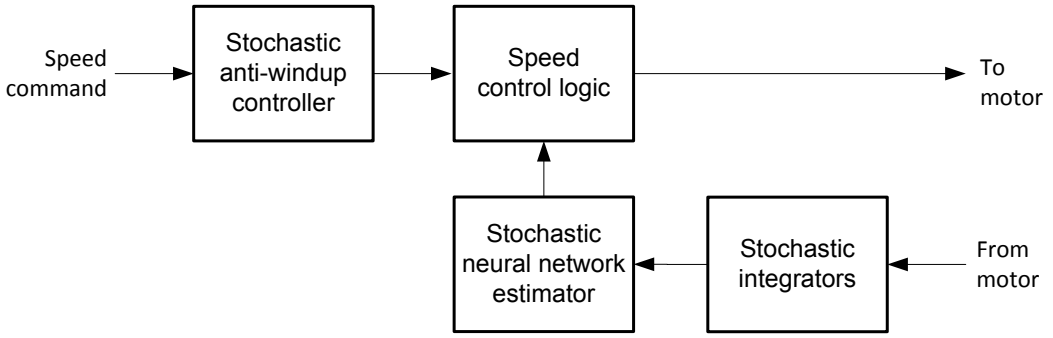


Figure 2.9: SC-based controller for an induction motor proposed by Zhang and Li [139].

input image which is transmitted as a long analog video signal resembling a digital bit-stream. The pixel-level functions are usually simple, but because of the large number of pixels involved, the overall transformation process is extremely computation-intensive. If these functions are implemented using SC, then low-cost, highly parallel image processing becomes possible, as demonstrated in an SC-based image-sensing chip [51].

The main reasons for the early interest in SC are the relative simplicity and robustness of SC-based arithmetic units and the possibility of having many units working in parallel. These benefits became less important as the transistors became cheaper, but as the foregoing motor-control application suggests, the benefits continue to be significant, even in some well-established applications. Furthermore, the past decade has introduced several entirely new applications for SC. One such application is the decoding of low-density parity check (LDPC) codes.

LDPC codes are powerful error-correcting codes which were introduced by Gallager in 1962 [41]. They enable data to be sent over noisy channels at rates close to the theoretical maximum (the Shannon limit). Because they are difficult to implement in practice, they were largely ignored until the 1990s when new research [79] and semiconductor technology developments made them economically viable. LDPC codes are attractive because there are no global relations among their bits, making it possible to have highly parallel and efficient decoding algorithms for very long codewords, such as the sum-product algorithm (SPA) [70]. LDPC codes are now utilized in communication standards such as WiFi [56] and digital video broadcasting [36].

Because of its use of extremely long codewords (often containing thousands of bits), LDPC decoding requires massive computational resources using conventional approaches [140]. Moreover, some of the better decoding algorithms are probabilistic or "soft" rather than deterministic. All these features suggest that LDPC decoding can take advantage of

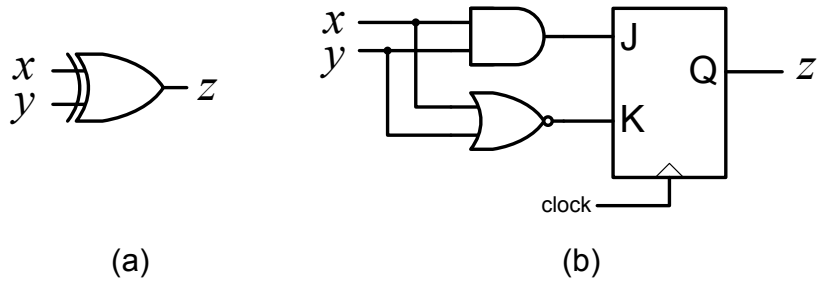


Figure 2.10: (a) Check and (b) update blocks used in stochastic LDPC decoding.

the compactness, error tolerance, and inherently probabilistic nature of SC circuits, as has been demonstrated [47] [91] [72].

LDPC decoding is the process of trying to find the closest valid codeword to a received and possibly erroneous codeword. The SPA algorithm mentioned above is a probabilistic algorithm for LDPC decoding based on belief propagation. Gross et al. [47] present a stochastic implementation of the SPA approach that employs the basic components shown in Figure 2.10. The XOR gate of Figure 2.10a implements the function $Z = X(1-Y) + Y(1-X)$ used in a step of the SPA algorithm known as the *check* step, while the circuit of Figure 2.10b implements the *update* step: $Z = XY/\big(XY+(1-X)(1-Y)\big)$. The simplicity of the elements shown in Figure 2.10 allows massively parallel decoding of LDPC codes.

While the probabilistic nature of SC makes it suitable for this decoding application, several other features of SC can be exploited in the decoder design. The low complexity of the basic components enables easy scaling to very large LPDC codes and effectively supports efficient parallel processing for such codes. Also, the progressive precision feature noted in Section 2.3 speeds up convergence of the decoding process. The early iterations of the algorithm can proceed faster by using rough values provided by the first bits of the SNs being processed. Furthermore, the overhead due to binary-to-stochastic number conversion is small. A recently reported IC implementation of an SC-based LDPC decoder [72] claims to achieve higher throughput with less chip area than conventional non-SC decoders.

2.5 Recent Developments

Since 2010, the year we started our research on SC, interest in the field has significantly increased. The promising new application to LDPC decoding has been an important milestone. Many new stochastic LDPC decoders have appeared in the literature [91] [72]. Furthermore, SC decoders for polar codes [137] and Viterbi decoders [22] have also been

designed. The probabilistic nature of SC, as well as its error tolerance, makes it a suitable choice for such designs.

Implementing artificial neural networks continues to be a major application of SC [18] [118] [58]. With the ever-increasing interest in the so-called spiking neural networks [86], further research in stochastic neural networks has also become relevant, because of the similarity between the encoding of SNs and that of the neural "spikes"; see Figure 1.2. SC neurons have low complexity, which allows the designers to put many of them in parallel and obtain a fully parallel neural network. Similarly, applications that involve matrix operations [127] can benefit from SC's low-complexity implementations.

Data recognition and mining [27] [90] and machine learning [48] are also among the new applications of SC. These applications of inherently error-tolerant, making approximate computing techniques, including SC, suitable for efficient implementations. Image processing also benefits from the same properties of SC. Since 2010, many new SC image-processing circuits have appeared in the literature [73] [3] [92], including a recently fabricated chip [37] that outperforms similar conventional designs. The theory of SC has also gained some attention recently; for instance a method of testing SC circuits has been proposed in [99].

General design methods for SC have received little attention since the 1970s. The topic was revisited recently by Qian et al. [109] and Li et al. [75], who introduced an SC architecture called Reconfigurable SC (ReSC) architecture that is capable of implementing many arithmetic functions. Function implementation in this style has a strong similarity to analog computing [80]. Its main idea [106] is to approximate a given function by a Bernstein polynomial [76]. A *Bernstein polynomial* of degree k has the form

$$Z = \sum_{i=0}^{k} b_k B_{i,k}(X)$$

where the b_i's are the coefficients of the polynomial, and $B_{i,k}(X)$ is a Bernstein basis polynomial of the form

$$B_{i,k}(X) = \binom{k}{i} X^i (1-X)^{k-i}$$

Figure 2.11 shows the ReSC architecture. The input variable X and the constant coefficients b_i are converted to SNs via the SNGs. The inputs of the adder are k independent SNs representing X for some realization of a polynomial of degree k. The probability of

obtaining a number i at the output of the adder is

$$P\{sum = i\} = \binom{k}{i} X^i (1 - X)^{k-i}$$

Now the probability of having a 1 at z is

$$P\{z = 1\} = b_0.P\{sum = 0\} + b_1.P\{sum = 1\} + ... + b_k.P\{sum = k\}$$

which reduces to

$$P\{z = 1\} = \sum_{i=0}^{k} b_k B_{i,k}(X)$$

In other words, the probability of outputting a 1 at z (i.e. the value of the SN Z) is a Bernstein polynomial of degree k defined by the coefficients b_i calculated at X.

Since 2010, general design methods for SC circuits gained more attention due to the need for automatic design of many different functions. We introduced a synthesis method called STRAUSS (Spectral TRAnsform Use in Stochastic circuit Synthesis) [1] [8] which will be covered in detail in the next chapter. STRAUSS and ReSC are only capable of designing combinational circuits. Sequential SC circuits are also mentioned in the literature. For instance, Figure 2.10b shows a sequential circuit implementing a relatively complex

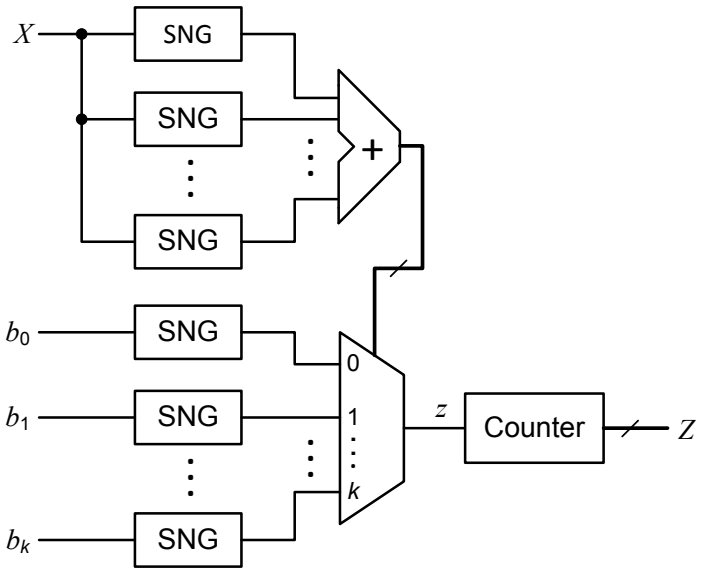


Figure 2.11: Reconfigurable stochastic arechitecture (ReSC) realizing a Bernstein polynomial of degree k [109].

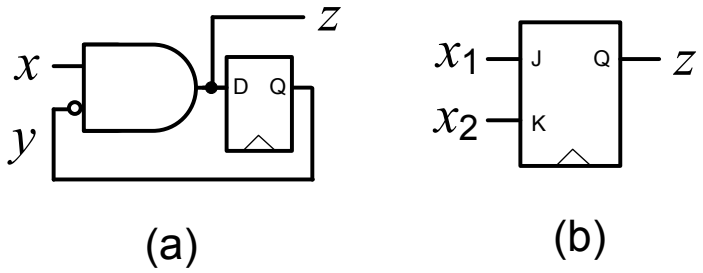


Figure 2.12: Sequential stochastic circuits implementing (a) $Z = X/(1 + X)$ and (b) $Z = X_1/(X_1 + X_2)$.

operation that involves division. However, sequential circuits are significantly more complicated to analyze and design than combinational circuits because introducing memory elements into stochastic circuits completely changes the picture, as we now show.

Consider the circuit of Figure 2.12a which combines an AND gate with a D-flip-flop. The AND acts as a stochastic multiplier implementing the function $Z = X(1-Y)$. The D-flip-flop simply shifts its input bit-stream by 1 bit, and implements the stochastic function $Y = Z$. Eliminating Y from the preceding equations, gives $Z = X/(1 + X)$, which is the SF implemented by the circuit of Figure 2.12a. This function does not have an appropriate polynomial form, and so cannot be directly implemented by combinational stochastic circuits. A similar example is the JK-flip-flop shown in Figure 2.12b, which has the SF $Z = X_1(X_1 + X_2)$, and is used to approximate stochastic division .

Figure 2.13 shows the general structure of an n-input sequential circuit with k flip-flops. The combinational block generates the output z and the next state variables $y_1^+, ..., y_k^+$ based on the inputs and the current state variables $y_1, ..., y_k$. The memory block merely copies the y_i^+ values to y_i at the active clock edge. The stochastic functions implemented by a sequential circuit C are defined by the stationary distribution Y of its states and the primary output Z, which can be derived by solving the Markov chain equations for C [39]. It can be shown [7] that sequential circuits implement a larger class of SFs than combinational circuits (which only implement certain polynomials; see Chapter 3), namely, rational functions of the form

$$Z(X_1, ..., X_n) = \frac{P(X_1, ..., X_n)}{Q(X_1, ..., X_n)}$$

in which P and Q are polynomials.

As noted, the stochastic function of sequential circuits can be obtained by solving the corresponding equations for Y and Z, but this can be very difficult when many state vari-

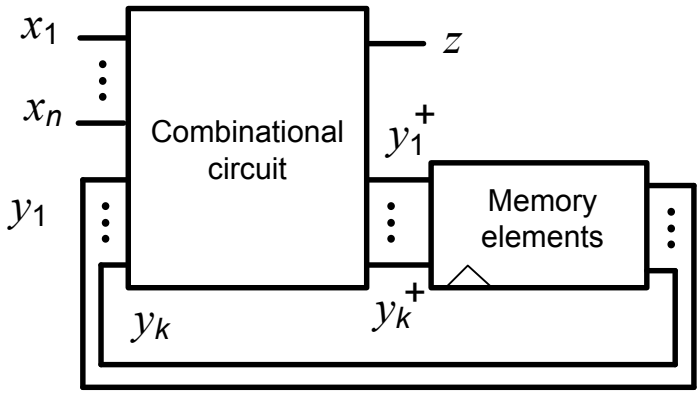


Figure 2.13: An n-input sequential circuit with k flip-flops.

ables are involved. To sidestep this problem, Gaines proposed restricting attention to FSMs with a chain structure in which the states are ordered and transitions only occur between adjacent states; jumping over states is not allowed [39]. This restriction allows easy Markov chain analysis. Figure 2.14 shows the state behavior of one such chain-structured FSM, the ADDIE (ADaptive Digital Element). Gaines also argued that state transitions should be local to avoid excessive fluctuations in SF values. ADDIEs have been used in various analog-style stochastic circuits such as filters [39]. Similar chain structured sequential circuits can implement non-polynomial functions such as $tanh$ and exp efficiently [16] [74].

Variations and extensions of Gaines's ADDIE model have been proposed over the years. A 2-dimensional extension of the chain-structured FSM was proposed by Li et al. [74]. A more general form of ADDIE was used by Saraf et al. [119] to implement SFs such as trigonometric functions. Evidently, sequential implementations can be more efficient than combinational for certain classes of SFs. However, many optimal combinational stochastic circuits exist. For example, the sequential edge-detection circuit designed in [73] is more than 20 times larger than the combinational edge detector of [3].

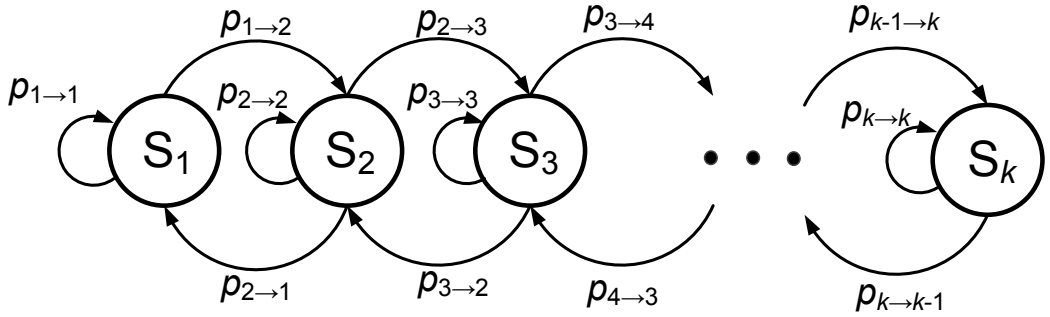


Figure 2.14: State diagram of a generalized ADDIE [39].

A drawback of sequential circuits is that they require a transition (warm-up) period before settling to the desired stationary distribution. During this period, which can be quite long, the circuit may produce inaccurate results. Another disadvantage of sequential circuits is that their behavior is affected by auto-correlation among the SNs. This refers to the correlation between an SN and its shifted or delayed versions. Auto-correlation imposes new requirements for the SNGs used for sequential circuits. Combinational circuits, being memoryless, are not affected by auto-correlation [24]. Unlike combinational circuits, a general design methodology for sequential stochastic circuits is not known. Most existing methods are limited to chain-structured designs.

As mentioned in Section 2.2, the conversion circuits can consume a lot of area, and in some cases can defeat the purpose of using SC. For this reason, there have been several efforts to reduce the cost of SNGs, especially the ones used for generating constant numbers. For instance, the ReSC design of Figure 2.11 uses $k + 1$ constant number generators, so reducing their cost can significantly improve its efficiency. Qian et al. [107][110] and Chen and Hayes [25] have introduced several methods of synthesizing circuits for constant number generation.

2.6 Summary

This chapter reviewed the basic concepts of SC, as well as its history and applications. We started by giving a rigorous definition of SNs that excludes the notion of randomness. This is important because it allows the use of deterministic number sources to increase an SC circuit's accuracy. We discussed the relation between BFs and SFs implemented by combinational circuits, and derived several basic components of SC. Then we highlighted SC's traditional and new applications, which include image processing, neural networks, and LDPC decoding. We also discussed the recent developments in the field and the challenges that need to be addressed, including accuracy issues, correlation problems, and the lack of general design methods. In the next chapters of this dissertation, we discuss our research that addresses many of the challenges discussed here.

Chapter 3

Design of Stochastic Circuits

One of the missing pieces of stochastic computing (SC) since its early days has been the lack of a general design method. While basic arithmetic operations such as multiplication and addition were known, many stochastic circuits used to be designed in an ad hoc fashion. This chapter presents a systematic method of designing stochastic circuits based on spectral transforms, in effect, targeting the synthesis problem defined in Chapter 1 (see Figure 1.7). We present a synthesis algorithm called STRAUSS, which produces low-cost SC circuits. Most of the material of this chapter is taken from our paper on STRAUSS [8] and an older version of the same synthesis method [1].

3.1 Spectral Transforms

We start by exploring a fundamental relation between SC circuits and spectral transforms like the Fourier transform [55] [62]. Such transforms have many applications in engineering. For instance, consider the time-domain impulse response of an analog filter. While it contains all the information about the filter's behavior, it is not easy to extract the response of the filter to a given input signal. The Fourier transform of the impulse response reveals the "hidden" frequency-domain behavior of the system, from which its response to a given input signal can readily be found. The spectral transforms of interest in this chapter map Boolean functions (BFs) from the logic domain to the domain of real numbers. We will see that they lead to a unique multilinear[1] representation of a given Boolean function (similar to the result of Theorem 2.1), which defines the underlying stochastic function (SF).

[1] A *multilinear* polynomial is a polynomial in which terms may contain products of variables, but no variable appears with a power of two or higher.

As mentioned earlier, a stochastic circuit C is a logic circuit that operates on (pseudo) random bit-streams, called stochastic numbers (SNs). Each wire x_i of C carries an SN X_i. The information conveyed by X_i, also conveniently denoted by X_i, is the rate or frequency of its 1-pulses and is independent of bit-stream length. Note that this is a slightly different notation from the one used in the previous chapters. Previously, the unipolar value represented by an SN X was denoted by p_X. However, in this chapter, we use X in a natural way to represent both the SN and its value.

We now introduce the spectral transforms used to analyze and synthesize stochastic circuits. An n-variable Boolean function (BF) $f(x_1, ..., x_n)$ maps $B_n = \{0,1\}^n$ to $B = \{0,1\}$. Here, B_n is seen as a 2^n-dimensional vector space, where each dimension corresponds to a row of f's truth-table (TT), or equivalently, to an n-variable minterm. For example, if n = 2, $f(x_1, x_2)$ has the 4-dimensional basis vectors $m_0 = (1,0,0,0), m_1 = (0,1,0,0), m_2 = (0,0,1,0)$, and $m_3 = (0,0,0,1)$, and can be written as

$$f(x_1, x_2) = \bigvee_{i=0}^{3} c_i \wedge m_i \tag{3.1}$$

This is the familiar sum-of-minterms expansion of f, where the c_i's are 0-1 coefficients that define f.

Example 3.1: If $f_1(x_1, x_2) = x_1 \vee \bar{x}_2$, then $f_1(x_1, x_2) = m_0 \vee m_2 \vee m_3$, or in the column-vector form that we use later:

$$\vec{f_1}(x_1, x_2) = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \vee \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \vee \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \begin{bmatrix}1\\0\\1\\1\end{bmatrix} \tag{3.2}$$

The last vector is essentially f_1's TT. To save space, we also write such vectors in the transposed form $[1 \quad 0 \quad 1 \quad 1]^{\mathrm{T}}$.

As Table 2.1 indicates, in the SC context we must deal with real numbers ranging over intervals such as $[0, 1]$ and $[-1, +1]$. Given a BF such as f_1 , we are interested in an equivalent real function $\widehat{F_1}$ defined on $[-1, +1]^n$ (or some other appropriate domain) that specifies the SC behavior of f_1 . This function can be obtained by interpolating the TT values in the real domain via a multilinear polynomial. For example, consider the TT vector in Eq. (3.2). By replacing 0's and 1's with $+1$'s and -1's, respectively, we see that f_1 produces the TT

vector $[-1 \quad 1 \quad -1 \quad -1]^{\mathrm{T}}$ for inputs $(x_1, x_2) = (1,1), (1,-1), (-1,1)$ and $(-1,-1)$, in IBP format. These four discrete "TT points" can be embedded in a continuous real-number function as follows:

$$\begin{aligned}\widehat{F}_1(X_1, X_2) = &\left(\frac{1+X_1}{2}\right)\left(\frac{1+X_2}{2}\right)(-1)\\ &+\left(\frac{1+X_1}{2}\right)\left(\frac{1-X_2}{2}\right)(+1)\\ &+\left(\frac{1-X_1}{2}\right)\left(\frac{1+X_2}{2}\right)(-1)\\ &+\left(\frac{1-X_1}{2}\right)\left(\frac{1-X_2}{2}\right)(-1)\end{aligned}$$

Observe that each term of the foregoing expression assumes the correct value 1 or -1 at each TT point. On expanding this expression, we get

$$\begin{aligned}\widehat{F}_1(X_1, X_2) = 0.25\big(&(1 + X_2 + X_1 + X_1X_2)(-1)\\ &(1 - X_2 + X_1 - X_1X_2)(+1)\\ &(1 + X_2 - X_1 - X_1X_2)(-1)\\ &(1 - X_2 - X_1 + X_1X_2)(-1)\big)\\ = -0.5 &- 0.5X_2 + 0.5X_1 - 0.5X_1X_2\end{aligned} \tag{3.3}$$

The polynomial (3.3) interpolates the TT values in the real numbers. It is linear with respect to variables X_1 and X_2, and is referred to as multilinear. Most importantly as we will see, it represents the stochastic behavior of the BF f_1. □

More generally, given a Boolean function $f(x_1, x_2, ..., x_n)$, if n independent SNs $X_1, X_2, ..., X_n$, defined in an SC format, such as UP, BP or IBP, are the input arguments of f, the output is another SN that is some function of $X_1, X_2, ..., X_n$. We denote this function by $\widehat{F}(X_1, X_2, ..., X_n)$ and refer to it as the SC behavior or stochastic function of f. We will see that $\widehat{F}$ has a unique multilinear form similar to that in (3.3), and can be determined by means of spectral transforms. This form is essentially the same as that given by (2.2) where the BF was defined over $\{0, 1\}$ rather than $\{-1, +1\}$.

The spectral transforms of interest execute a change of basis from the minterm space of a BF f to a real-valued space. We employ the Fourier transform $\mathcal{F}$ for BFs, which is also known as the discrete Walsh transform in Hadamard ordering [62] [95]. Spectral

transforms of this type have been considered previously for a wide variety of logic design and testing tasks [55] [62]. For BFs with large values of n, it may be impractical to deal with 2^n-dimensional spectra, although concise representations of spectra for functions with hundreds of variables are known [38]. This size issue is of much less concern in SC, however, because the values of n tend to be small, e.g., n = 2 in Figure 1.5.

To compute the Fourier transform of a BF given as a TT vector $\vec{f}$ or equivalent, we first replace 0 and 1 by $+1$ and -1, respectively. The Fourier transform $F = \mathcal{F}(f)$ is specified by

$$\vec{F} = \frac{1}{2^n} H_n \times \vec{f} \tag{3.4}$$

where $\vec{F}$ is the vector form of F denoting its spectral coefficients or spectrum, and H_n is the Walsh matrix (with natural or Hadamard ordering) of dimension 2^n defined recursively by

$$H_0 = \begin{bmatrix} +1 \end{bmatrix} \qquad \text{and} \qquad H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix}$$

Equation (3.4) is evaluated using the rules of linear algebra over real numbers. In the case of f_1 from Example 3.1, we get

$$\vec{F_1} = \frac{1}{4} H_2 \times \vec{f_1} = \frac{1}{4} \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} -1 \\ +1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ +0.5 \\ -0.5 \\ -0.5 \end{bmatrix}$$

The result is the spectrum of f, and the corresponding basis is

$$[+1 \quad +1 \quad +1 \quad +1]^{\mathrm{T}}, [+1 \quad -1 \quad +1 \quad -1]^{\mathrm{T}}, [+1 \quad +1 \quad -1 \quad -1]^{\mathrm{T}}, [+1 \quad -1 \quad -1 \quad +1]^{\mathrm{T}}$$

defined by the rows or columns of H_2 . (Note that H_n is symmetric.) These basis vectors resemble digital waveforms, and are analogous to harmonics in the sine-cosine Fourier transform used for time-frequency conversions. More specifically, they correspond to the four linear BFs: 1, x_2 , x_1, and $x_1 \oplus x_2$, where $\oplus$ denotes XOR. Recall that in the spectral domain, XOR is replaced by multiplication.

Analogous to the sum-of-minterms expansion (3.1) for f_1 in the Boolean domain, we write the transformed function as

$$F_1(X_1, X_2) = \sum_{i=0}^{3} C_i S_i \tag{3.5}$$

where S_i's are the basis vectors 1, X_2 , X_1 , and X_1X_2, in the spectral domain, and the C_i's constitute the spectrum of f_1. Hence, Eq. (3.5) becomes

$$F_1(X_1, X_2) = -0.5 - 0.5X_2 + 0.5X_1 - 0.5X_1X_2$$

This is a multilinear polynomial that interpolates (matches) the original BF f_1 at its four Boolean input coordinates (TT points), namely $(X1, X2) = (1, 1), (-1, 1), (1, -1), (-1, -1)$. This transformation is illustrated in Figure 3.1. Notice that the last expression above is exactly the same as (3.3), and the process of arriving at the two expressions is similar. So, we see intuitively that the Fourier transform of a BF defines its unique SC behavior. This leads to the following theorem, which is closely related to Theorem 2.1.

Theorem 3.1: If $\widehat{F}$ denotes the SC behavior of an n-variable Boolean function f in IBP format, and $F = \mathcal{F}(f)$ is its Fourier transform, then $\widehat{F} = F$.

Proof. We provide a proof by induction on n, the number of variables of f. Suppose $n = 1$, so f is a single-variable Boolean function. Using the Boole-Shannon expansion theorem [50], we can write $f(x) = c_0\bar{x} \oplus c_1x$ where $c_0 = f(0)$ and $c_1 = f(1)$ are constants. Now apply a bit-stream X of length N to the input x of f. If this bit-stream contains N_1

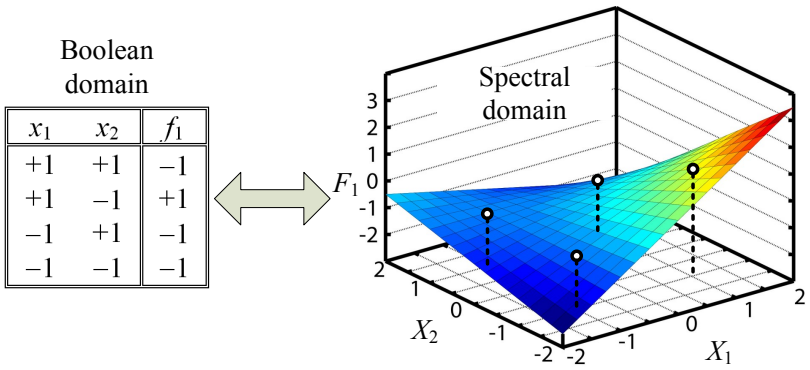


Figure 3.1: Illustration of the spectral transformation of the function f_1 of Example 3.1.

1's and N_0 0's, it represents the number $(N_0 - N_1)/N$ in IBP format. The function f therefore outputs another bit-stream with N_1 c_1's and N_0 c_0's representing the IBP number $(N_0C_0 + N_1C_1)/N$, where $C_0 = 1 - 2c_0$ and $C_1 = 1 - 2c_1$. Hence, for an arbitrary SN $X = 1 - 2N_0/N$, the output number is

$$\widehat{F}(X) = \frac{C_0 + C_1}{2} + \frac{C_0 - C_1}{2}X \tag{3.6}$$

which describes the SC behavior of f in the IBP format. Now the TT of f is $\vec{f} = [C_0 \quad C_1]^{\mathrm{T}}$, so its Fourier transform is

$$\vec{F} = \frac{1}{2}H_1 \times \vec{f} = \frac{1}{2}\begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix}\begin{bmatrix} C_0 \\ C_1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} C_0 + C_1 \\ C_0 - C_1 \end{bmatrix}$$

This corresponds to the polynomial

$$F(X) = \frac{1}{2}\big(C_0 + C_1 + (C_0 - C_1)X\big) \tag{3.7}$$

which is the same as (3.6), so the theorem holds for single-variable functions.

Now as the induction hypothesis, assume the theorem holds for all functions of up to $n-1$ variables. We want to show that it also holds for the n-variable function $f(x_1, ..., x_n)$. Again using Boole-Shannon expansion, we can write $f(x_1, ..., x_n) = f_0.\bar{x}_n \oplus f_1.x_n$, where $f_0 = f(x_1, ..., x_{n-1}, 0)$ and $f_1 = f(x_1, ..., x_{n_1}, 1)$ are functions of $n - 1$ or fewer variables. Figure 3.2 illustrates how f is decomposed in this way. Thus, f's SC behavior is, in terms

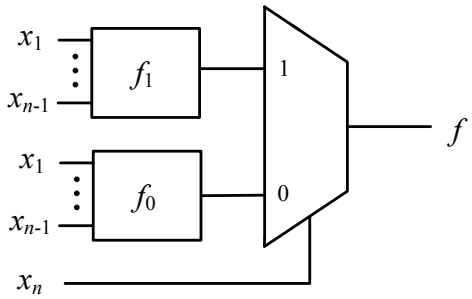


Figure 3.2: Circuit illustrating the application of Boole-Shannon expansion to the function f.

of the behavior of a 2-to-1 multiplexer (Figure 3.2)

$$\widehat{F}(X_1, ..., X_n) = \frac{\widehat{F_0} + \widehat{F_1}}{2} + \frac{\widehat{F_0} - \widehat{F_1}}{2} X_n \tag{3.8}$$

where $\widehat{F_0}$ and $\widehat{F_1}$ denote the SC behavior of f_0 and f_1, respectively.

We can express the TT of f in terms of the TTs of f_0 and f_1 as $\vec{f} = [\vec{f_0} \quad \vec{f_1}]^{\mathrm{T}}$. Accordingly, we can decompose the Fourier transform calculation (Eq. (3.4)) into

$$\vec{F} = \frac{1}{2^n} H_n \times \vec{f} = \frac{1}{2} H_1 \begin{bmatrix} \frac{1}{2^{n-1}} H_{n-1} \times \vec{f_0} \\ \frac{1}{2^{n-1}} H_{n-1} \times \vec{f_1} \end{bmatrix} = \frac{1}{2} H_1 \begin{bmatrix} \vec{F_0} \\ \vec{F_1} \end{bmatrix}$$

where $\vec{F_0}$ and $\vec{F_1}$ are the Fourier transforms of f_0 and f_1, respectively. The resulting polynomial is

$$F(X_1, ..., X_n) = \frac{1}{2}\big(F_0 + F_1 + (F_0 - F_1)X_n\big)$$

which is the same as (3.8). Hence $\widehat{F} = F$, and from the Principle of Induction we conclude that the theorem holds. □

Thus, given a combinational circuit or Boolean function, we can determine its IBP behavior by computing its Fourier transform. A simple interval conversion according to Table 2.1 is all that is required to determine its behavior in other SN formats. The next example illustrates this.

Example 3.2: As discussed before, the XOR gate of Figure 2.2c serves as an IBP multiplier. We can verify this as follows. XOR has the TT vector $\vec{f_2} = [+1 \quad -1 \quad -1 \quad +1]^{\mathrm{T}}$. Calculating its Fourier transform yields

$$\vec{F_1} = \frac{1}{4} H_2 \times \vec{f_1} = \frac{1}{4} \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

so $F_2(X_1, X_2) = X_1 X_2$. According to Theorem 3.1, the IBP behavior of XOR is

$$\widehat{F_2}(X_1, X_2) = F_2(X_1, X_2) = X_1 X_2$$

Similarly, we know that a 2-input AND gate acts as a multiplier in UP format. In this case, $\vec{f_3} = [+1 \quad +1 \quad +1 \quad -1]^{\mathrm{T}}$ and

$$\vec{F_1} = \frac{1}{4} H_2 \times \vec{f_1} = \frac{1}{4} \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} +1 \\ +1 \\ +1 \\ -1 \end{bmatrix} = \begin{bmatrix} +0.5 \\ +0.5 \\ +0.5 \\ -0.5 \end{bmatrix}$$

Thus,

$$F_3(X_1, X_2) = 0.5(1 + X_1 + X_2 - X_1 X_2) \tag{3.9}$$

Since we want the AND behavior in UP format, we must map (3.9) from IBP to UP. Let p_1 , p_2 and p_3 denote the UP values of X_1 , X_2 and F_3, respectively. Table 2.1 implies $X_1 = 1 - 2p_1$, $X_2 = 1 - 2p_2$, and $F_3 = 1 - 2p_3$. Hence,

$$1 - 2p_3 = 0.5(1 + 1 - 2p_1 + 1 - 2p_2 - 1 + 2p_1 + 2p_2 - 4p_1 p_2)$$

leading to the desired multiplication $p_3 = p_1 p_2$. □

Finally, we note that the Fourier transform is invertible and preserves all information about f. We can therefore retrieve the original TT form by applying the inverse Fourier transform $f = \mathcal{F}^{-1}(F)$ to the spectrum. Since H_n is related to own inverse (by a 2^n factor), this may be calculated as follows:

$$\vec{f} = H_n \times \vec{F} \tag{3.10}$$

This is the key link from the desired SC behavior F to a logic function f that, with appropriate modifications, implements F.

As mentioned, the elements of $\vec{F}$ correspond to polynomial terms in the spectral domain. For $n = 2$, these terms are $1, X_2, X_1$ and $X_1 X_2$, respectively. In the general case, we assign an n-bit binary number to each element of $\vec{F}$, starting from all 0's. So the elements of $\vec{F}$ are assigned to "000...0", "000...1", ..., "111...0", and "111...1", respectively. Now to find the polynomial term corresponding to each element, we get the assigned binary number and replace each 1 by the corresponding X_i for that position, and replace each 0 by 1. Hence, the terms corresponding to the first, second, and last elements of $\vec{F}$ are 1, X_n and $X_1 X_2 ... X_n$, respectively.

3.2 Synthesis based on Spectral Transforms

The spectral transforms discussed so far have several useful applications in the SC context. Besides analyzing Boolean functions and extracting their SC behavior, they can be used to systematically design combinational SC circuits. But before discussing our synthesis method (STRAUSS), a few preliminary concepts are needed.

Theorem 3.1 implies that a combinational circuit can implement a stochastic function $\widehat{F}$ given in multilinear polynomial form, i.e., one in which terms may contain products of variables of degree at most one. The idea is then to apply the inverse Fourier transform $\mathcal{F}^{-1}$ to $\widehat{F}$ and obtain the corresponding Boolean function $\vec{f}$ in vector truth-table form, as in Eq. (3.10). However, applying $\mathcal{F}^{-1}$ to an arbitrary target function $\widehat{F}$ does not necessarily yield a vector $\vec{f}$ whose elements are the TT values +1 and -1. In fact, we can have three possible outcomes:

1. All the elements of $\vec{f}$ are +1 or -1, in which case $\vec{f}$ is the truth-table of a Boolean function and is directly implementable by a logic circuit.

2. All elements of $\vec{f}$ lie in the interval $[-1, +1]$, but some have values $\{c_i\}$ other than +1 or -1. In that case, the function is still implementable, but requires auxiliary inputs and circuitry to generate the c_i's, as discussed in Section 3.3.

3. Some elements of $\vec{f}$ are larger than +1 or less than -1, in which case the function is not implementable. It is still possible to implement a related function $\widehat{F}'$ that is an approximate or scaled version of $\widehat{F}$.

We say polynomial $\widehat{F}$ is *SC-implementable* if we can synthesize a stochastic circuit whose behavior is defined by $\widehat{F}$. SC-implementable functions fall into the first two categories above, and are distinguished by having inverse Fourier transforms. The simple product function X_1X_2 is SC-implementable, whereas the unscaled sum $X_1 + X_2$ is not. We will refer to the latter as *SC-unimplementable*.

To illustrate the general case, consider the generic two-variable multilinear polynomial

$$\widehat{F}(X_1, X_2) = a_0 + a_1X_2 + a_2X_1 + a_3X_1X_2$$

Taking its inverse Fourier transform, we get

$$\vec{f} = H_2 \times \vec{F} = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} a_0 + a_1 + a_2 + a_3 \\ a_0 - a_1 + a_2 - a_3 \\ a_0 + a_1 - a_2 - a_3 \\ a_0 - a_1 - a_2 + a_3 \end{bmatrix}$$

In order for $\widehat{F}$ to be SC-implementable, the elements of $\vec{f}$ should be in the interval $[-1, +1]$. In other words, the following constraints should be satisfied:

$$\begin{aligned} -1 \leq a_0 + a_1 + a_2 + a_3 \leq +1 \\ -1 \leq a_0 - a_1 + a_2 - a_3 \leq +1 \\ -1 \leq a_0 + a_1 - a_2 - a_3 \leq +1 \\ -1 \leq a_0 - a_1 - a_2 + a_3 \leq +1 \end{aligned}$$

Constraints of this kind can be obtained for polynomials of n variables by applying the inverse Fourier transform to them. Concepts similar to SC-implementable polynomials are discussed by Qian and Riedel [106] [108]. They implement stochastic functions in the form of Bernstein polynomials, and define constraints on the coefficients of Bernstein polynomials in order to distinguish implementable functions. As we will show later, that approach can also be interpreted in terms of spectral transforms.

In order to synthesize a stochastic circuit for an arbitrary target function $\widehat{F}$, a few conversion steps are required. For instance, the inverse Fourier transform can only produce suitable Boolean functions when applied to multilinear functions [95]. Suppose $\widehat{F}$ is an ordinary polynomial of degree n

$$\widehat{F}(X) = a_0 + a_1 X + a_2 X^2 + ... + a_n X^n$$

implying that it has non-linear terms. We convert it to a multilinear polynomial $\widehat{P}(X_1, ..., X_n)$ in which the non-linear terms of $\widehat{F}$, such as $a_n X^n$, are replaced by multilinear terms like $a_n X_1 X_2 ... X_n$. The new variables $X_1, X_2, ..., X_n$ are assumed to be independent copies of the original variable X.

Procedure STRAUSS ($\widehat{F}$) – Returns a stochastic circuit implementing the target – function $\widehat{F}$

Step 1. Format the target function $\widehat{F}$ as a multilinear polynomial $\widehat{P}$

Step 2. Compute the inverse Fourier transform $\mathcal{F}^{-1}(\widehat{P})$ to obtain $\vec{f}$

Step 3. Generate constant IBP values for elements of $\vec{f}$ (Section 3.3)

Step 4. Optimize via standard combinational design procedures

Procedure 3.1: The main steps of the STRAUSS synthesis method.

There are many possible ways to select a multilinear polynomial $\widehat{P}$ that corresponds to $\widehat{F}$. A natural choice is one that is symmetric with respect to all its variables thus:

$$\widehat{P}(X_1,...,X_n) = a_0 + \frac{a_1}{n}(X_1 + X_2 + ... + X_n) + \frac{a_2}{\binom{n}{2}}(X_1X_2 + X_1X_3 + ... + X_{n-1}X_n) + ... + a_n(X_1X_2...X_n)$$

which is unique, because every term of the original polynomial is uniquely transformed into a set of symmetric terms. However, symmetry is not a necessity and asymmetric polynomials are also acceptable. For instance,

$$\widehat{P}(X_1,...,X_n) = a_0 + a_1X_1 + a_2X_1X_2 + ... + a_nX_1...X_n$$

is one of the possible asymmetric multilinear polynomials that correspond to $\widehat{F}$. Previous synthesis methods [106] [109] assume symmetry, but as we show that asymmetric multilinear polynomials may lead to better implementations.

Finally, to synthesize a function $\widehat{F}$ from $[0,1]^n$ to $[0,1]$ that is SC-unimplementable, we convert it to an SC-implementable polynomial by approximation. This step is a straightforward polynomial fitting problem and can be easily solved by tools such as MATLAB [84].

The main steps of STRAUSS are listed in Procedure 3.1. We first illustrate them with examples, and then discuss their details.

Example 3.3: Consider the problem of reverse engineering the IBP multiplier, so the given function is $\widehat{F}_2(X_1, X_2) = X_1X_2$. Since this already has the desired multilinear polynomial

form, we can skip Step 1 of STRAUSS and proceed to Step 2, where we use the inverse Fourier transform to obtain f_2's TT vector.

$$\vec{f_2} = H_2 \times \vec{F_2} = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}$$

The result $\vec{f_2}$ only has +1 and -1 as elements, and is clearly the TT of a 2-input XOR gate.

Now consider the same problem for the UP multiplier $\widehat{F}_3(X_1, X_2) = X_1X_2$. Note that this function will be different from $\widehat{F}_2$ because of the format change. Mapping $\widehat{F}_3$ to the IBP format yields

$$\widehat{P}_3(X_1, X_2) = 1 - 2\widehat{F}_3\left(\frac{1 - X_1}{2}, \frac{1 - X_2}{2}\right) = 0.5(1 + X_1 + X_2 + X_1X_2)$$

Applying the inverse Fourier transform to $\widehat{P}_3$ produces the TT $[+1 \quad +1 \quad +1 \quad -1]^{\mathrm{T}}$, which defines an AND gate. □

Example 3.4: Suppose we attempt to synthesize the arithmetic sum function $\widehat{F}_4(X_1, X_2) = X_1 + X_2$. This is also in multilinear form, so we proceed with the inverse Fourier transform.

$$\vec{f_4} = H_2 \times \vec{F_4} = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} +2 \\ 0 \\ 0 \\ -2 \end{bmatrix}$$

Two elements of $\vec{f_4}$ lie outside the interval $[-1, +1]$, which means that it is SC-unimplementable. The standard solution is scaled addition which substitutes $s(X_1 + X_2)$ for $X_1 + X_2$, where s is a scale factor that makes the function SC-implementable; in this

case $s = 1/2$. The new target function is $\widehat{F}_5(X_1, X_2) = \frac{1}{2}(X_1 + X_2)$, which yields

$$\vec{f_5} = H_2 \times \vec{F_5} = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} +1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

However, the result $\vec{f_5}$ has elements other than +1 and 1, namely 0, so a constant number generation step (Step 3) is required. This is discussed in the next section. □

3.3 Constant Number Generation

The third step of STRAUSS, as seen in Procedure 3.1, is constant number generation, a challenging problem in itself. We first discuss the intuition behind this step. Then, we formally define the problem and present several solutions.

Continuing Example 3.4 from the previous section, we derived the truth-table $\vec{f_5} = [+1 \quad 0 \quad 0 \quad -1]^{\mathrm{T}}$ for the stochastic add operation. We can interpret this TT as follows. If both inputs are 0, then a constant SN of value +1 (in IBP format) is sent out; if both inputs are 1, then a constant SN of value -1 (in IBP format) is sent out; if one input is 1 and the other is 0, a constant SN of value 0 is sent out. Producing an SN with values other than +1 or -1 requires additional inputs, i.e., random number sources. For the running example, we can replace the two-input TT $\vec{f_5}$, with the following three-input TT

$$\vec{f_5} = [+1 \quad +1 \quad +1 \quad -1 \quad +1 \quad -1 \quad -1 \quad -1]^{\mathrm{T}}$$

which is the Boolean function $f_5(x_1, x_2, r) = (x_1 \wedge r) \vee (x_2 \wedge r) \vee (x_1 \wedge x_2)$, where an auxiliary input r has been added to the function. (This happens to be the majority function.) The r input must be fed with the IBP stochastic number 0, i.e., a pure random bit-stream. Figure 3.3a shows a straightforward AND-OR implementation of f_5.

It is possible to optimize the synthesized circuit further by reordering the elements of $\vec{f_5}$. For example, the following TT has the same stochastic behavior of $\vec{f_5}$ but has a simpler implementation:

$$\vec{f_5} = [+1 \quad +1 \quad -1 \quad +1 \quad +1 \quad -1 \quad -1 \quad -1]^{\mathrm{T}}$$

This is the BF $f_5(x_1, x_2, r) = (x_1 \wedge r) \vee (x_2 \wedge \bar{r})$, which has the AND-OR implementation of Figure 3.3b. It is obvious that this new circuit is a 2-to-1 multiplexer with r as its select input. This is precisely the standard scaled adder in the SC literature [38].

Finding the best ordering of $+1$'s and -1's of a TT is a difficult optimization task. We now formally define the constant SN generation problem, and present a solution method for it.

Single SN generation problem: Given a constant number $c \in [-1, +1]$ and a desired precision m, we want to find an m-input Boolean function $f(r_1, ..., r_m)$ with $k = \lfloor (1-c)2^{m-1} \rfloor$ (or $k = \lceil (1-c)2^{m-1} \rceil$) minterms that has minimum cost.

When fed with pure random inputs, an m-input BF $f(r_1, ..., r_m)$ with $k = (1-c)2^{m-1}$ minterms, will output a 1 with a probability of $\frac{k}{2^m}$. Thus the IBP value generated at the output of f is $1 - 2(\frac{k}{2^m}) = c$. Note that the precision m only refers to the target constant c, and has no implications on the runtime for SN generation.

It should be noted that the number of auxiliary inputs introduced determines the precision of the constant numbers that can be generated. With $m + 1$ auxiliary inputs $r_1, ..., r_m, r_{m+1}$, we can only generate the following numbers

$$\left\{ \frac{-2^m}{2^m}, \frac{-2^m + 1}{2^m}, ..., \frac{-1}{2^m}, \frac{0}{2^m}, \frac{+1}{2^m}, ..., \frac{2^m - 1}{2^m}, \frac{2^m}{2^m} \right\}$$

Any other number c should be rounded to the closest number from the above set. So for an arbitrary real-valued c, one has to choose a value for m, taking into account that increasing m provides better precision and accuracy but also increases the cost of the circuit.

As the problem suggests, there may be many BFs that generate the same constant number, and our goal is to select one with minimum cost. We use literal count as our cost crite-

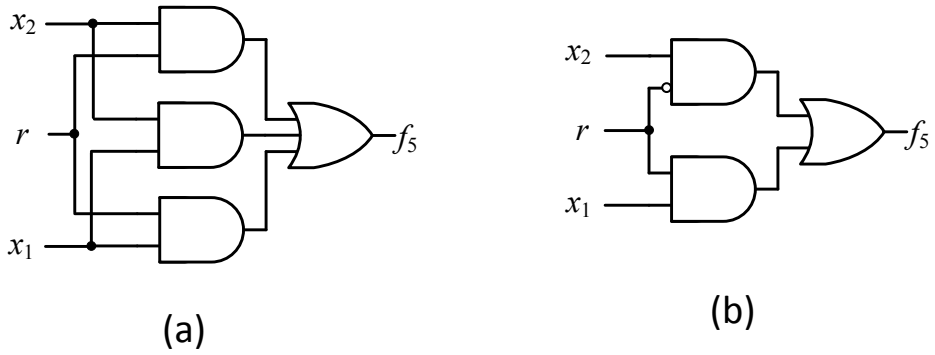


Figure 3.3: Two synthesized circuits for SC addition: (a) without optimization; (b) with optimization.

Procedure $SCG(m, k)$ – Returns m-input Boolean function with k minterms in – truth-table format

Step 1. (terminal cases)
If $k = 0$ then the function is constant 0, so return $[0 \quad 0 \quad ... \quad 0]^{\mathrm{T}}$
If $k = 2^m$ then the function is constant 1, so return $[1 \quad 1 \quad ... \quad 1]^{\mathrm{T}}$

Step 2a. If $k \leq 2^{m-1}$, fill the first half of the truth-table TT with zeros, and fill the second half using a recursive call to $SCG(m-1, k)$:
$$TT = [0 \quad 0 \quad ... \quad 0 \quad SCG(m-1, k)]^{\mathrm{T}}$$

Step 2b. If $k > 2^{m-1}$, fill the second half of the truth-table TT with ones, and fill the first half using a recursive call to $SCG(m-1, k) - 2^{m-1}$:
$$TT = [SCG(m-1, k-2^{m-1}) \quad 1 \quad 1 \quad ... \quad 1]^{\mathrm{T}}$$

Step 3. Return TT

Procedure 3.2: Overview of the single constant generation procedure SCG.

rion, because it is easy to use and reflects both transistor and gate count fairly accurately in standard CMOS logic [50]. For example, both of the following BFs $f_6(r_1, ..., r_m) = r_1$ and $f_7(r_1, ..., r_m) = r_1 \oplus ... \oplus r_m$ have 2^{m-1} minterms and thus generate the constant $c = 0$. But f_6 has a literal count of 1, while f_7 has a literal count of $2(m-1)$ using a chain or tree of XOR gates. Note that the cost of an XOR gate is twice the cost of an elementary gate.

Qian and Riedel [107] give a method of synthesizing a minimal two-level circuit that generates a given stochastic constant. Qian et al. [109] discuss several other methods that synthesize multi-level circuits. The method of [109] does not directly address the single SN generation problem defined in this chapter, so the optimality of that method will not be discussed here. We now present a recursive algorithm called *single constant generation* (SCG) to obtain a minimal multi-level constant generation circuit. This algorithm takes two parameters, the number of inputs m and the number of minterms k, and returns a truth-table (TT) of length 2^m with k minterms. If $k \leq 2^{m-1}$, i.e., k is at most half the TT's length, then SCG fills the first half of the TT with 0's, and makes a recursive call with parameters $m-1$ and k. This recursion can be interpreted as follows. SCG returns

$$f(r_1, ..., r_m) = r_1 \wedge f'(r_2, ..., r_m)$$

in which f' is a function with $m-1$ variables and k minterms. If $k > 2^{m-1}$, SCG fills the second half of the TT with 1's, and makes a recursive call with parameters $m-1$ and $k-2^{m-1}$. Similarly, this recursion step can be interpreted as the algorithm returning

$$f(r_1, ..., r_m) = r_1 \vee f'(r_2, ..., r_m)$$

After $m-1$ recursion steps, SCG returns a BF that is implemented by a chain of at most $m-1$ AND or OR gates. The steps of the SCG procedure are given in Procedure 3.2. This algorithm can also be used to solve the multiple constant SN generation problem, as discussed below.

Example 3.5: Consider finding a 7-input function $f_8(r_1, ..., r_7)$ with 77 minterms. We call $SCG(7, 77)$, which returns a TT with 64 ones in the second half, and 13 ones in the first half. The first half is now obtained by calling $SCG(6, 13)$. This recursion step corresponds to

$$f_8(r_1, ..., r_7) = r_1 \vee f'_8(r_2, ..., r_7)$$

in which f'_8 is the result of $SCG(6, 13)$. Continuing down the recursion path we see $SCG(5, 13)$, $SCG(4, 13)$, $SCG(3, 5)$, $SCG(2, 1)$, $SCG(1, 1)$, and $SCG(0, 1)$ which is a terminal case. The resulting function is hence

$$f_8(r_1, ..., r_7) = r_1 \vee r_2 r_3 (r_4 \vee r_5 \vee r_6 r_7)$$

which has minimal literal count. The corresponding optimal circuit implementation of f_8 is shown in Figure 3.4. □

The circuits generated by the SCG procedure are optimal in terms of literal count. The proof is straightforward; each input of the generated Boolean function appears at most once in the final expression. So if m inputs are required to generate a constant, the literal count of the generated circuit will be m, which is the minimum possible literal count for an m-

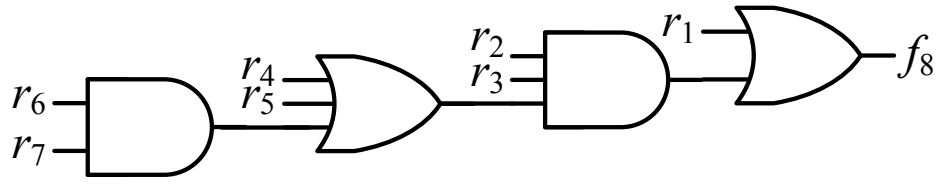


Figure 3.4: Optimal circuit implementation for function f_8 of Example 3.5.

input circuit. The constant number generators synthesized by the method of [107] have near-optimal two-level implementations, but in most cases their literal count is greater than m. Next, we generalize the problem to multiple constants.

Multiple SN generation problem: Given a set of constant numbers $\{c_0, ..., c_{2^n-1}\}$ where $c_i \in [-1, +1]$, we want to find 2^n minimum-cost m-input BFs $f_1, ..., f_{2^n-1}$ with $\lfloor (1 - c_0)2^{m-1} \rfloor, ..., \lfloor (1 - c_{2^n-1})2^{m-1} \rfloor$ minterms, respectively.

The multiple constant generation problem, which is Step 3 of STRAUSS (see Procedure 3.1), is a difficult one, because there are many possible opportunities for sharing gates between the circuits for the c_i's. Since exhaustively searching among them would be inefficient, we propose a heuristic algorithm based on the SCG procedure (Procedure 3.2). To generate all the constants, we call SCG for each c_i, and as SCG progresses, we keep a record of the constant SNs and circuits it has generated so far. The generated circuits are stored in a table and are reused if a previously generated SN is encountered. This approach is demonstrated in the following example which illustrates a complete synthesis computation using STRAUSS.

Example 3.6: Consider the target function $\widehat{F}_9(X) = 0.4375 - 0.25X - 0.5625X^2$. We want to use STRAUSS to synthesize a stochastic circuit that implements $\widehat{F}_9$. In Step 1 of Procedure 3.1, $\widehat{F}_9$ is reformulated as a multilinear polynomial $\widehat{P}_9$ because it contains a non-linear term X^2. This term is eliminated by introducing two new inputs X_1 and X_2 to replace X thus:

$$\widehat{P}_9(X_1, X_2) = 0.4375 - 0.125(X1 + X2) - 0.5625X_1X_2$$

This is only one of the many possible multilinear polynomials that are equivalent to the target function $\widehat{F}_9$, and is not necessarily the best choice.

For this example, we chose a symmetric multilinear polynomial, but as we will show in the next section, STRAUSS examines many possible polynomials (including asymmetric ones) and chooses one that leads to a lower cost. At Step 2 of STRAUSS, we have

$$\vec{f_9} = H_2 \times \vec{P_9} = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} +0.4375 \\ -0.1250 \\ -0.1250 \\ -0.5625 \end{bmatrix} = \begin{bmatrix} -0.375 \\ +1 \\ +1 \\ +0.125 \end{bmatrix}$$

All the elements of $\vec{f_9}$ are in the $[-1,+1]$ interval, implying that the function is SC-implementable. However, there exist elements other than $+1$ and -1, namely, $c_0 = -0.375$ and $c_3 = +0.125$, which require constant SN generation (Step 3). In fact $c_1 = c_2 = +1$ are also constant SNs, but they are trivial to generate.

To generate c_0, the SCG procedure (Procedure 3.2) is called with parameters $m = 4$ and $k = 11$. The choice of $m = 4$ stems from the fact that it is the least number of inputs for a BF capable of generating c_0. The parameter k comes from the formula $k = (1 - c_0)2^{m-1}$ discussed earlier. Calling $SCG(4, 11)$ leads to the recursive calls: $SCG(3, 3)$, $SCG(2, 3)$, $SCG(1, 1)$, and $SCG(0, 1)$, and the following BF is returned

$$f_{c_0}(r_1, r_2, r_3, r_4) = r_1 \vee \big(r_2(r_3 \vee r_4)\big)$$

Next, SCG is called with parameters $m = 4$ and $k = 7$ to generate c_3. Calling $SCG(4, 7)$ entails recursive calls $SCG(3, 7)$ and $SCG(2, 3)$, at which point the recursion stops because the results of the previous calls (during the circuit generation for c_0) are reused. The returned BF is

$$f_{c_3}(r_1, r_2, r_3, r_4) = r_1\big(r_2 \vee (r_3 \vee r_4)\big)$$

which shares a gate with f_{c_0}. Step 3 is now done, and we have a stochastic implementation of $\widehat{F}_9$, namely,

$$f_9(x_1, x_2, r_1, r_2, r_3, r_4) = \bar{x}_1\bar{x}_2 f_{c_0} \vee x_1 x_2 f_{c_3}$$

In the last step, f_9 is optimized via conventional CAD tools. Figure 3.5a shows the final gate-level implementation of f_9. Notice the shared OR gate ($r_3 \vee r_4$), which is used in both f_{c_0} and f_{c_3}. The inputs x_1 and x_2 must be fed with independent bit-streams that carry the same SN X. These can be generated by using two independent SNGs, or just by shifting one bit-stream in time and thus generating an independent copy of it [57]. The auxiliary inputs, on the other hand, must be supplied with pure random bit-streams. As shown in Figure 3.5, we connect the auxiliary inputs to a 4-bit LFSR, which generates four independent random bit-streams [57]. Figure 3.5b shows another SC implementation of $\widehat{F}_9$ using an older version of STRAUSS that was presented in [1]. As can be seen, the circuit generated by STRAUSS yields a smaller area based on literal count. In the next section, we discuss some other optimizations used in STRAUSS. □

3.4 Further Optimizations

Step 1 of STRAUSS involves converting the target function to a multilinear polynomial, which can be symmetric or asymmetric. All the previous examples used symmetric polynomials, but it is possible to further optimize the synthesized circuit by considering both symmetric and asymmetric polynomials.

We illustrate this with an example. Consider the target function

$$\widehat{F}_{10}(X) = \frac{1}{2}(X^3 + X)$$

Converting $\widehat{F}_{10}$ to symmetric multilinear form, we get

$$\widehat{P}_{10}(X_1, X_2, X_3) = \frac{1}{6}(X_1 + X_2 + X_3) + \frac{1}{2}X_1X_2X_3$$

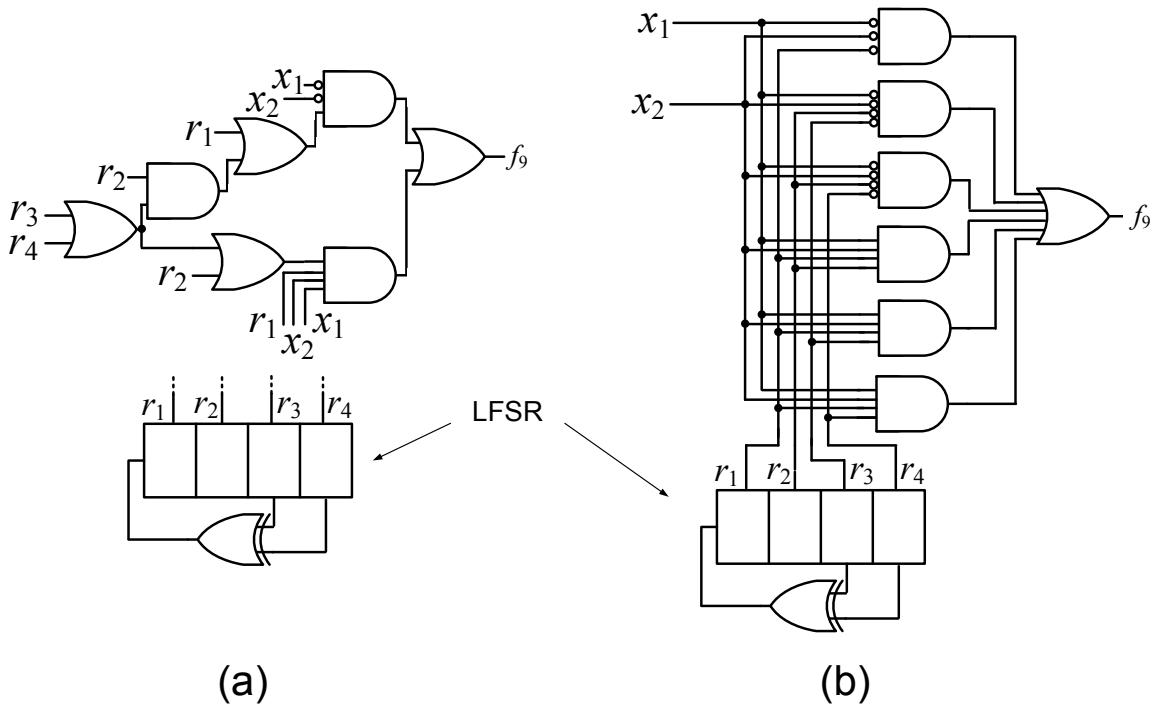


Figure 3.5: Stochastic implementation of $\widehat{F}_9(X) = 0.4375 - 0.25X - 0.5625X^2$ obtained by (a) STRAUSS and (b) the algorithm of [1].

which yields the following TT after applying the inverse Fourier transform

$$\vec{f}_{10} = [+1 \quad -\tfrac{1}{3} \quad -\tfrac{1}{3} \quad +\tfrac{1}{3} \quad -\tfrac{1}{3} \quad +\tfrac{1}{3} \quad +\tfrac{1}{3} \quad -1]^{\mathrm{T}}$$

Since this has elements other than 1 and -1, namely, $+\frac{1}{3}$ and $-\frac{1}{3}$, multiple-constant SN generation circuitry is required. However, if we choose the following asymmetric multilinear polynomial in the first step of STRAUSS

$$\widehat{P}'_{10}(X_1, X_2, X_3) = \frac{1}{2}(X_1 + X_2 - X_3) + \frac{1}{2}X_1X_2X_3$$

Then the inverse Fourier transform produces

$$\vec{f}'_{10} = [+1 \quad +1 \quad -1 \quad +1 \quad -1 \quad +1 \quad -1 \quad -1]^{\mathrm{T}}$$

which is simply the BF $f'_{10}(x_1, x_2, x_3) = x_1x_2 \vee x_1\bar{x}_3 \vee x_2\bar{x}_3$, and requires no constant-generation circuitry. This means that significant cost savings are possible if asymmetric polynomials are considered.

A given SC-implementable polynomial can be mapped to many different asymmetric multilinear polynomials with the same SC behavior, some of which may be SC-unimplementable. To distinguish between them, we need to apply the inverse Fourier transform (Step 2 of STRAUSS) and check if all the elements of the truth table are in the interval $[-1, +1]$. However, applying the transform to all possible polynomials is time-consuming, making this approach infeasible. So STRAUSS uses a different approach which is discussed below. An overview of this *Asymmetric Polynomial Selection* (APS) algorithm is given in Procedure 3.3.

Given a target polynomial, we start with a symmetric multilinear polynomial[2] and use the inverse Fourier transform to obtain a *symmetric truth-table* (STT). Because of its symmetry, the STT includes repeated elements. We then modify the repeated elements to obtain an asymmetric TT of better cost. If we keep the new elements in the $[-1, +1]$ interval, the newly obtained TT remains SC-implementable. And as long as we keep the average of the new elements the same as that of the old elements, the new TT will have the same SC behavior as the STT.

[2]There is only one way to convert any polynomial term into a set of multilinear terms that are symmetric with respect to all the input variables.

Procedure $APS(\widehat{F})$ – Given a polynomial $\widehat{F}$, find an asymmetric multilinear – polynomial of near-optimal cost

Step 1. Convert $\widehat{F}$ to a symmetric multilinear polynomial $\widehat{P}$

Step 2. Apply the inverse Fourier transform to $\widehat{P}$ and obtain a symmetric truth-table $\vec{f}$

Step 3. Pick a group of symmetric elements from $\vec{f}$, and replace them with new elements, all of which, except for at most one, must be +1 or -1

Step 4. Try all the possible orderings of the current group of elements, and select one of lowest cost

Step 5. Repeat Steps 3 and 4 for all elements of $\vec{f}$

Procedure 3.3: Summary of the asymmetric polynomial selection (APS) procedure.

As an example, consider a generic 3-input STT

$$\vec{f}_{11} = [c_1 \quad c_2 \quad c_2 \quad c_3 \quad c_2 \quad c_3 \quad c_3 \quad c_4]^{\mathrm{T}}$$

which has at most four distinct elements $c_1, ..., c_4$. We can replace the three c_2's with three new elements, c_2', c_2'', and c_2'''. If the new elements are within the interval $[-1, +1]$, and if $c_2' + c_2'' + c_2''' = 3c_2$, then the new TT

$$\vec{f}_{11}' = [c_1 \quad c_2' \quad c_2'' \quad c_3 \quad c_2''' \quad c_3 \quad c_3 \quad c_4]^{\mathrm{T}}$$

will have the same SC behavior as $\vec{f}_{11}$. Since all the inputs of the circuit have the same value X, the probabilities of getting any of the c_2 elements in $\vec{f}_{11}$ will be the same. So by changing the elements to c_2', c_2'', and c_2''' in $\vec{f}_{11}$ and keeping the average the same as before (by assigning $c_2' + c_2'' + c_2''' = 3c_2$), the SC behavior of the TT remains unchanged. Similarly, the three c_3's can also be replaced with new elements. The choice of the new elements directly affects the cost of the new TT. Elements such as +1 and -1 are desirable because they can be generated at no cost, while other elements require constant generation circuitry.

For example, consider the following STT

$$\vec{f}_{10} = [+1 \quad -\tfrac{1}{3} \quad -\tfrac{1}{3} \quad +\tfrac{1}{3} \quad -\tfrac{1}{3} \quad +\tfrac{1}{3} \quad +\tfrac{1}{3} \quad -1]^{\mathrm{T}}$$

which has three $-\frac{1}{3}$'s and three $+\frac{1}{3}$'s. As shown earlier, we can replace these elements with new elements and obtain

$$\vec{f'}_{10} = [+1 \quad +1 \quad -1 \quad +1 \quad -1 \quad +1 \quad -1 \quad -1]^{\mathrm{T}}$$

which has the same SC behavior. Notice that the three $-\frac{1}{3}$'s are replaced with two -1's and one +1, and the three $+\frac{1}{3}$'s are replaced with two +1's and one -1. Another choice of TT with the same behavior is

$$\vec{f''}_{10} = [+1 \quad 0 \quad 0 \quad 0 \quad -\tfrac{1}{3} \quad +\tfrac{1}{3} \quad 0 \quad -1]^{\mathrm{T}}$$

but it clearly has a higher cost because it requires several constant-generation circuits, while $\vec{f}_{10}$ requires none. It can be shown that given a set of symmetric elements, we can always find a new set of elements with the same average, all of which, except for at most one, are +1's or -1's.

Assume we have a set of k symmetric elements of value c. If $c > 0$, then we can replace the set with one +1 element and $k-1$ new elements c' of value $(kc-1)/(k-1)$. The new set has the same average as the old set:

$$+1 + (k-1)c' = 1 + (k-1)\frac{kc-1}{k-1} = kc$$

Similarly, if $c < 0$, we can replace the set with one -1 and $k-1$ new elements c' of value $(kc+1)/(k-1)$. By repeating this process on the new elements of the set, we obtain a set that has at least $k-1$ elements of value +1 or -1.

Another factor affecting cost is the order of the new elements in the TT. For example, the two +1's and one -1 replacing $+\frac{1}{3}$ can appear in three possible orders: $[+1 \quad +1 \quad -1]$, $[+1 \quad -1 \quad +1]$, and $[-1 \quad +1 \quad +1]$. Similarly, the new elements replacing $-\frac{1}{3}$'s can also be reordered. So the TT $\vec{f}_{10}$ can have 9 different orderings:

$$[+1 \quad +1 \quad -1 \quad +1 \quad -1 \quad +1 \quad -1 \quad -1]^{\mathrm{T}}$$
$$[+1 \quad +1 \quad -1 \quad +1 \quad -1 \quad -1 \quad +1 \quad -1]^{\mathrm{T}}$$
$$[+1 \quad +1 \quad -1 \quad -1 \quad -1 \quad +1 \quad +1 \quad -1]^{\mathrm{T}}$$
$$[+1 \quad -1 \quad +1 \quad +1 \quad -1 \quad +1 \quad -1 \quad -1]^{\mathrm{T}}$$

$$[+1 \quad -1 \quad +1 \quad +1 \quad -1 \quad -1 \quad +1 \quad -1]^{\mathrm{T}}$$
$$[+1 \quad -1 \quad +1 \quad -1 \quad -1 \quad +1 \quad +1 \quad -1]^{\mathrm{T}}$$
$$[+1 \quad -1 \quad -1 \quad +1 \quad +1 \quad +1 \quad -1 \quad -1]^{\mathrm{T}}$$
$$[+1 \quad -1 \quad -1 \quad +1 \quad +1 \quad -1 \quad +1 \quad -1]^{\mathrm{T}}$$
$$[+1 \quad -1 \quad -1 \quad -1 \quad +1 \quad +1 \quad +1 \quad -1]^{\mathrm{T}}$$

The number of different ways to order the TTs grows exponentially with the number of inputs, so searching among all of them is not possible for large circuits. For such cases, STRAUSS has a greedy search heuristic that usually finds a good ordering. The heuristic starts from the STT and selects a group of repeated elements (say c_2 in $\vec{f}_{11}$) and finds new elements (c_2', c_2'' and c_2''') to replace them. Then it tries all the possible orderings of this group, and selects one with the minimum cost. The algorithm proceeds to the next group of repeated elements (c_3 in $\vec{f}_{11}$). In so doing, the algorithm only examines $3+3=6$ orderings out of the $3 \times 3 = 9$ possible orderings. Thus, the search becomes feasible for larger circuits. Like most heuristics, this method does not always find an optimal solution, but our experiments show that a near-optimum polynomial is usually found. We also observed that in most cases, there are multiple optimum and many near-optimum polynomials, which favors the heuristic search.

It is worth noting that employing asymmetric polynomials reduces the precision needed in the constant generation circuit, and hence can lead to cost savings. A good example is the function $\widehat{F}_{10}$ discussed earlier in this section. A symmetric TT for this function is

$$\vec{f}_{10} = [+1 \quad -\tfrac{1}{3} \quad -\tfrac{1}{3} \quad +\tfrac{1}{3} \quad -\tfrac{1}{3} \quad +\tfrac{1}{3} \quad +\tfrac{1}{3} \quad -1]^{\mathrm{T}}$$

which requires constant generation circuitry for $\frac{1}{3}$ and $-\frac{1}{3}$. When applying the SCG procedure (Procedure 3.2) to $\vec{f}_{10}$, one has to choose a precision m, and round the numbers $\frac{1}{3}$ and $-\frac{1}{3}$ to the precision closest to m. However, an asymmetric implementation of $\widehat{F}_{10}$, namely,

$$\vec{f}_{10}' = [+1 \quad +1 \quad -1 \quad +1 \quad -1 \quad +1 \quad -1 \quad -1]^{\mathrm{T}}$$

needs no constant generation at all. It is as if the constants $\frac{1}{3}$ and $-\frac{1}{3}$ are implemented with unlimited precision and at no cost. We end this section with a complete example involving a multivariate target function.

Example 3.7: Consider the target function $\widehat{F}_{11}(X, Y, Z) = \frac{1}{64}(63 + X^2 + Y^2 + Z^2 - X^2Y^2 - X^2Z^2 - Y^2Z^2 + X^2Y^2Z^2)$. Since the maximum degree of the polynomial is two, we will need two independent copies of each input. The corresponding symmetric multilinear polynomial form of $\widehat{F}_{11}$ is

$$\begin{aligned}\widehat{P}_{11}(X_1, X_2, Y_1, Y_2, Z_1, Z_2) = \frac{1}{64}(&63 + X_1X_2 + Y_1Y_2 + Z_1Z_2 - X_1X_2Y_1Y_2 \\ &-X_1X_2Z_1Z_2 - Y_1Y_2Z_1Z_2 + X_1X_2Y_1Y_2Z_1Z_2)\end{aligned}$$

Applying the inverse Fourier transform on $\widehat{P}_{11}$ (Step 2 of STRAUSS), yields the symmetric TT vector

$$\begin{aligned}\vec{f}_{11} = [&1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,\tfrac{7}{8}\,\tfrac{7}{8}\,1\,1\,\tfrac{7}{8}\,\tfrac{7}{8}\,1\,1\,1\,1\,1\,1 \\ &1\,1\,1\,1\,\tfrac{7}{8}\,\tfrac{7}{8}\,1\,1\,\tfrac{7}{8}\,\tfrac{7}{8}\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1]^{\mathrm{T}}\end{aligned}$$

(This STT is of size 64 and is displayed in two lines due to spacing limits.) Steps 3 and 4 of STRAUSS yield the following Boolean function

$$f_{11}(x_1, x_2, y_1, y_2, z_1, z_2, r_1, r_2, r_3, r_4) = \big((x_1 \oplus x_2) \vee (y_1 \oplus y_2) \vee (z_1 \oplus z_2)\big) \wedge (r_1 \wedge r_2 \wedge r_3 \wedge r_4)$$

which requires four auxiliary inputs. A gate-level implementation of f_{11} is shown in Figure 3.6a.

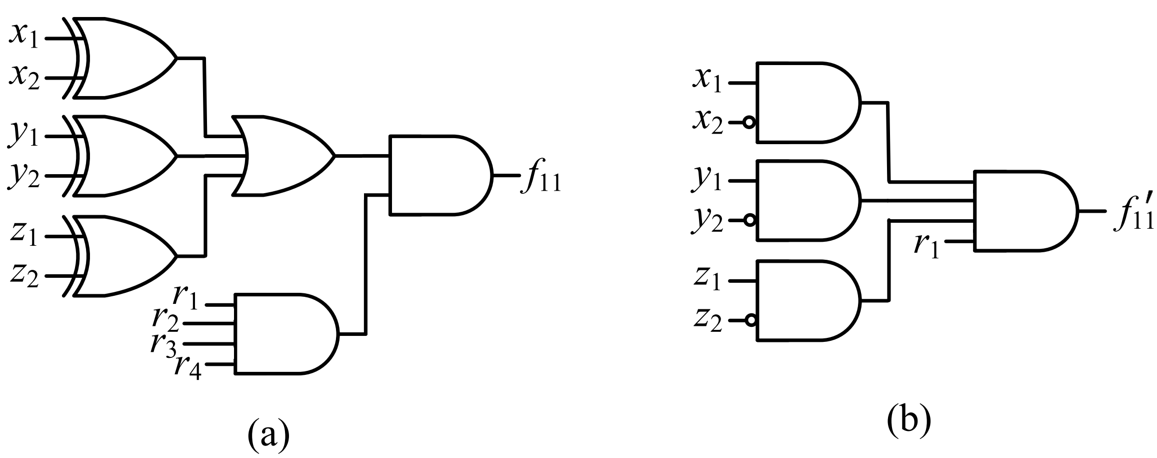


Figure 3.6: Stochastic implementations for $\widehat{F}_{11}$ of Example 3.7: (a) symmetric and (b) asymmetric.

It is possible to further optimize this circuit by applying the APS procedure to $\widehat{F}_{11}$. In the symmetric TT shown above, there are 8 symmetric elements with value $\frac{7}{8}$. APS identifies this symmetric group and replaces it with a group of seven elements of value 1 and one element of value 0. After trying different possible orderings, APS returns the following asymmetric TT (which is one of the 8 possible optimal TTs)

$$\vec{f}'_{11} = [1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1 \\ 1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1]^{\mathrm{T}}$$

This yields the Boolean function

$$f'_{11}(x_1, x_2, y_1, y_2, z_1, z_2, r_1) = x_1 \wedge \bar{x}_2 \wedge y_1 \wedge \bar{y}_2 \wedge z_1 \wedge \bar{z}_2 \wedge r_1$$

which requires only one auxiliary input. An implementation of f'_{11} is shown in Figure 3.6b. The asymmetric polynomial $\widehat{P}'_{11}$ corresponding to f'_{11} has more terms than the symmetric polynomial $\widehat{P}_{11}$ but, surprisingly, it yields a lower-cost implementation. Note that the APS procedure performs transformations on the TT only and does not deal with different (and in some cases complicated) polynomial terms. The first 16 terms of the asymmetric polynomial $\widehat{P}_{11}$ are shown below for illustration purposes only.

$$\begin{aligned}\widehat{P}'_{11}(X_1, X_2, Y_1, Y_2, Z_1, Z_2) = \frac{1}{64}(&63 - X_2 + X_1 + X_1X_2 - Y_2 - X_2Y_2 + X_1Y_2 \\ &+ X_1X_2Y_2 + Y_1 + X_2Y_1 - X_1Y_1 - X_1X_2Y_1 \\ &+ Y_1Y_2 + X_2Y_1Y_2 - X_1Y_1Y_2 - X_1X_2Y_1Y_2 + ...)\end{aligned}$$

□

3.5 Related Work and Experimental Results

Very little previous research has addressed the systematic design of SC circuits. Qian and Riedel [106] have developed a non-spectral method of implementing a single-variable SC target function $\widehat{F}(X)$ defined in UP format. They convert $\widehat{F}(X)$ to a Bernstein polynomial of the form

$$\widehat{P}(X) = \sum_{i=0}^{n} C_i \binom{n}{i} X^i (1 - X)^{n-i} \tag{3.11}$$

in which the C_i's are constant coefficients. This polynomial is then mapped to a specific style of logic circuit termed ReSC (Reconfigurable SC Architecture) [109] which was discussed in Section 2.5. It consists of an n-input adder that implements the terms $\binom{n}{i}X^i(1-X)^{n-i}$ (called the Bernstein terms) and a multiplexer that selects the C_i coefficients. Figure 3.7 a shows the ReSC implementation of Eq. (3.11). There are n independent inputs (x_i's) to the adder representing the variable X, and $n+1$ inputs to the multiplexer (c_i's) that represent the coefficients C_i in (3.11). The probability of a number k at the output of the adder is equal to $\binom{n}{k}X^k(1-X)^{n-k}$, i.e., the kth Bernstein term. Thus, the probability of having a 1 at f is equal to the probability getting a 0 at the adder and a 1 at c_0, plus the probability of getting a 1 at the adder and a 1 at c_1, plus ..., plus the probability of getting n at the adder and a 1 at c_n. These probabilities can be expressed as

$$F(X) = \sum_{i=0}^{n} C_i \binom{n}{i} X^i (1-X)^{n-i}$$

which is the same as (3.11). Note that the x_i inputs are independent SNs representing the number X, similar to the x_1 and x_2 inputs of Figure 3.5. This structure requires n SNGs to generate the x_i's and $n+1$ SNGs to generate the c_i's, which entails a significant area cost. Similar to STRAUSS, this design approach can be extended to multivariate functions [109], and to the bipolar format.

We now show that the method of [109] can be re-interpreted in terms of a spectral transform, a "Bernstein transform" $\mathcal{B}$ with a basis different to that of the Fourier transform $\mathcal{F}$.

Consider the 2-variable BF $f(x_1, x_2)$. Define a new spectral basis for it as follows: $B_0 = \frac{1}{4}(1+X_1)(1+X_2)$, $B_1 = \frac{1}{4}(1+X_1)(1-X_2)$, $B_2 = \frac{1}{4}(1-X_1)(1+X_2)$ and $B_3 = \frac{1}{4}(1-X_1)(1-X_2)$. (Note the difference between this and the spectral basis of the 2-variable Fourier transform, which is 1, X_1 , X_2 and X_1X_2 .) The resulting transform $\mathcal{B}$ of f can then be written as:

$$F(X_1, X_2) = \sum_{i=0}^{3} C_i B_i \tag{3.12}$$

This corresponds to the same multilinear expression generated by the Fourier transform. To see this, expand (3.12) thus:

$$\begin{aligned} F(X_1, X_2) = \frac{1}{4}\big(&C_0(1+X_1)(1+X_2) + C_1(1+X_1)(1-X_2) \\ &+ C_2(1-X_1)(1+X_2) + C_3(1-X_1)(1-X_2)\big) \\ = \quad \frac{1}{4}\big(&(C_0 + C_1 + C_2 + C_3) \\ &+ (C_0 - C_1 + C_2 - C_3)X_2 \\ &+ (C_0 + C_1 - C_2 - C_3)X_1 \\ &+ (C_0 - C_1 - C_2 + C_3)X_1X_2\big) \end{aligned}$$

which is the multilinear polynomial produced by the Fourier transform, as the following equation demonstrates:

$$\vec{F} = \frac{1}{4}\begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} C_0 + C_1 + C_2 + C_3 \\ C_0 - C_1 + C_2 - C_3 \\ C_0 + C_1 - C_2 - C_3 \\ C_0 - C_1 - C_2 + C_3 \end{bmatrix}$$

Despite their similarities, the circuits synthesized by STRAUSS are quite different from those designed with the ReSC architecture of [109]. Most importantly, ReSC does not benefit from asymmetric polynomials or constant input sharing. To illustrate this, we implemented the function $\widehat{F}_9$ of Example 3.6 using ReSC. Figure 3.7b shows the result (adjusted for the IBP format). Besides its adder and multiplexer, this design contains three SNGs, each consisting of a 4-bit comparator and a 4-bit random number generator (or LFSR), as in Figure 2.3a. Note, however, that this circuit can be optimized using standard combinational techniques; the middle SNG, for instance produces a 0, and so can be removed from the circuit.

To attempt a fair comparison, we used the Berkeley SIS synthesis tool [120] to optimize this design and those of Figure 3.5 and to map them to a generic library of gates. The library includes all the elementary gates and their relative area cost in terms of unit cells in a 0.35µm CMOS technology. Table 3.1 compares the three implementations of this function, along with several other representative circuits, with area cost reported in terms of unit cells. The methods that are being compared are: (i) ReSC from [109], (ii) the older

Table 3.1: Comparison between the proposed synthesis method STRAUSS and those of [1] and [109]; area is in unit cells from a generic library and includes the LFSRs and SNGs used for constant generation.

Target design	Qian et al. [109]	Alaghi and Hayes [1]	STRAUSS
$\widehat{F}_9$ from Example 3.6	113	70	70
$\widehat{F}_{10}(X) = \frac{1}{2}(X^3 + X)$	174	112	10
Gamma correction [109]	962	314	276
Average of 10 random target functions	426	151	125

version of STRAUSS from [1], and (iii) STRAUSS. The runtime of the circuits depends on the desired accuracy of the user. In general, longer runtime leads to better accuracy. The circuits compared in Table 3.1 achieve the same level of accuracy for a given runtime. These results show that STRAUSS synthesizes circuits that are significantly smaller than those designed by the techniques of [1] and [109]. The area reported in Table 3.1 does not include the conversion circuits that may be required for the primary inputs and outputs.

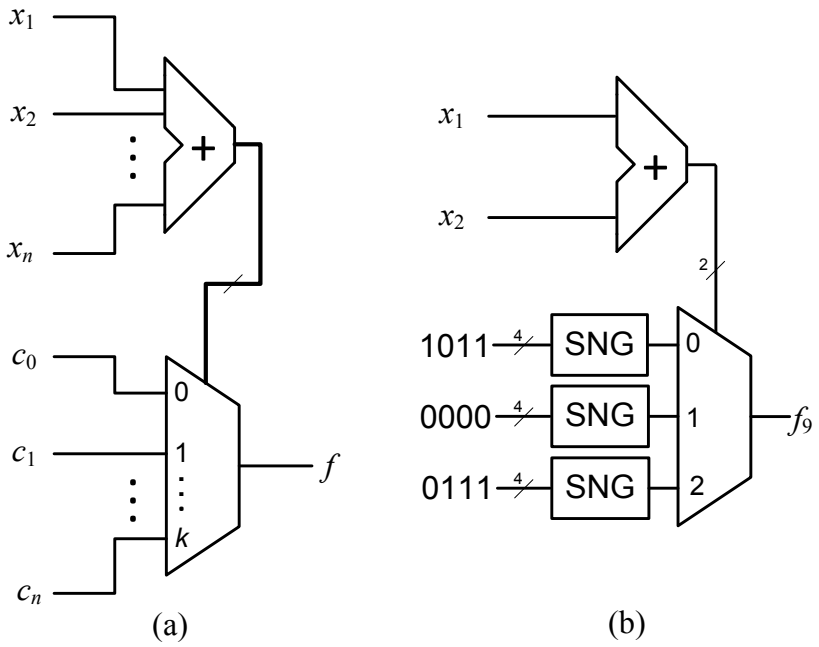


Figure 3.7: (a) ReSC architecture proposed in [109]; (b) Implementation of $\widehat{F}_9(X)$ from Example 3.6 using this architecture.

3.6 Summary

This chapter dealt with the problem of automatically designing (synthesizing) a stochastic circuit for a given target function. We have shown here that well-defined transforms linking the Boolean and the spectral domains exist, which provide fundamental theoretical insights into SC behavior. We have also successfully applied spectral transforms to the design of circuits in a way that naturally accommodates the most useful stochastic number formats. Furthermore, we have presented a novel and general synthesis technique STRAUSS for combinational circuit synthesis. Comparing this work to the major existing SC design method [109], we found that the results generated by STRAUSS can lead to significant cost savings.

Chapter 4

Correlation in Stochastic Computing

This chapter discusses the role of correlation in stochastic computing (SC). We start by introducing a concept called correlation insensitivity. Correlation insensitive (CI) functions are not affected by correlations among their inputs. This concept can be exploited in SC, as well as the closely related topic of statistical simulation (SS). For instance, inputs of a CI function can share random number sources without compromising the output accuracy, which in SC leads to significant cost savings.

After discussing CI functions, we show that systematic correlation in SC circuits can change their underlying function in a useful way. To illustrate this, we introduce a new correlation measure called SC correlation (SCC), that is suitable in quantifying the correlation of arbitrary SNs. Finally, we show that SCC adds a new dimension to SC circuit design, and in many cases leads to efficient implementation of useful functions. The material related to CI is published in [6], while that related to SCC appears in [4]

4.1 Statistical Simulation and Correlation Insensitivity

In this section, we introduce the problem of statistical simulation, which has direct links to SC and the problem of SN generation. After providing suitable definitions, we introduce correlation insensitivity, and how it affects SS. Then we show how SC can also exploit correlation insensitivity.

Each wire x in a stochastic circuit is associated with a Bernoulli random variable (BRV) X with parameter p_X, which is the probability of seeing a 1 on x. This is similar to the notion of an SN in SC. In fact, as shown in the next chapter, we can define SNs in the exact same way for error analysis purposes. The probability mass function (pmf) of X is

$f_X(k) = P\{X = k\}$, where $P\{A\}$ is the probability of event A. Since f_X is a binary function, $f_X(1)$ uniquely defines it. Note that $f_X(1) = p_X$.

The BRVs $X_1, ..., X_n$ associated with $x_1, ..., x_n$ may be correlated. Hence, their probabilistic behavior must be viewed as a joint probability distribution (or a joint pmf)

$$f_{X_1...X_n} = P\{X_1 = k_1, ..., X_n = k_n\}$$

The joint pmf specifies the probability of each of the 2^n possible combinations of X_i's. In the special case where all the X_i's are independent, the joint pmf reduces to the product of the marginal distributions of the individual BRVs.

$$f_{X_1...X_n} = f_{X_1}(k_1) \times ... \times f_{X_n}(k_n) \tag{4.1}$$

Definition 4.1: Let $z(x_1, ..., x_n)$ be the Boolean function realized by combinational circuit C. *Statistical simulation* (SS) is the process of estimating the pmf f_Z by applying samples from a joint pmf $f_{X_1...X_n}$ to C. Note that f_Z can be calculated exactly using the following equation:

$$f_Z(1) = \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} ... \sum_{k_n=0}^{1} \big(z(k_1, k_2, ..., k_n) \times f_{X_1X_2...X_n}(k_1, k_2, ..., k_n)\big) \tag{4.2}$$

which, along with $f_Z(0) = 1 - f_Z(1)$, completely specifies f_Z. Evaluating Eq. (4.2) is not practical, however, because of its exponential growth with n. SS is a practical alternative because it uses only a subset of the input combinations to estimate f_Z.

As an example, consider the $2n + 1$-input circuit C of Figure 3.7a. If the inputs are independent BRVs with parameter 1/2, i.e., unbiased BRVs, then SS of C involves generating samples from $2n + 1$ independent random sources, and recording the frequency of 1's and 0's on the output to estimate its pmf. As we will show later in this section, the multiplexer is correlation insensitive (CI) with respect to it data inputs, so we can use the same random source for all the data inputs, hence reducing the required random sources to $n + 1$.

In general, the input BRVs of a circuit can be biased, meaning that they can have a parameter other than 1/2, and they also can be non-independent (correlated). In such cases, the joint pmf of the inputs cannot be expressed in terms of individual pmfs, as in Eq. (4.1). So for SS purposes, we must sample directly from the joint pmf. The theoretical analysis

presented here applies to the general case. However, for ease of presentation, and since most applications employ independent inputs, we normally use examples that have independent RVs at their primary inputs.

Figure 4.1 shows a typical set-up for SS of a circuit with n independent inputs. The Bernoulli random variable generators produce BRVs with arbitrary parameter p_{X_i}. They each include a comparator (C) and an LFSR-based random number generator (RNG). In each clock cycle, C compares two k-bit binary numbers: the desired parameter p_{X_i} and a number r_i generated by the RNG. If $r_i < p_{X_i}$, then $x_i = 1$; otherwise, $x_i = 0$. Hence, $x_i = 1$ has probability p_{X_i}, approximately. To generate n independent BRVs, n number generators with independent RNGs are used. Note the similarity between a number generator and the SNG of Figure 2.3a.

Now let us look at another example of a CI function before formally defining the concept. Consider the function $z(x_1, x_4, x_5) = \bar{x}_1 x_4 \vee x_1 \bar{x}_5$ obtained by setting $x_2 = x_3 = 1$ in Figure 4.2. This is essentially the 2-to-1 multiplexer function, so it has the special property that inputs x_4 and x_5 cannot propagate to the output z simultaneously. In other words, x_4

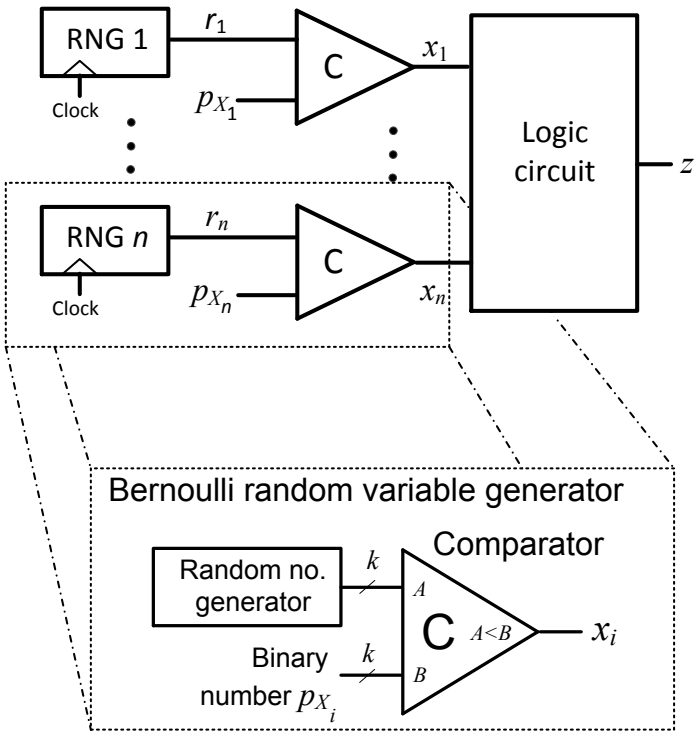


Figure 4.1: Statistical simulation set-up; random samples are generated at $r_1, ..., r_n$ to estimate probability distribution f_Z.

and x_5 never affect z at the same time. When $x_1 = 1$, the path from x_5 to z via G_3 and G_5 (highlighted in red, dashed) becomes active, and the path from x_4 through G_2 (highlighted in blue) is blocked. If $x_1 = 0$, the opposite happens. Assuming X_1 is independent of the other variables, we can write

$$f_Z(1) = f_{X_1}(0) f_{Z_{\bar{x}_1}}(1) + f_{X_1}(1) f_{Z_{x_1}}(1)$$

which is the probabilistic version of Boole-Shannon expansion (Theorem 4.1). $z_{\bar{x}_1} = x_4$ is the negative cofactor of z with respect to x_1 obtained by setting $x_1 = 0$ in z; $z_{x_1} = \bar{x}_5$ is the positive cofactor of z obtained by setting $x_1 = 1$. Hence,

$$f_Z(1) = f_{X_1}(0).f_{X_4}(1) + f_{X_1}(1).f_{X_5}(0)$$

implying that the output is a function of the marginal distributions f_{X_4} and f_{X_5} only, and not of their joint pmf $f_{X_4X_5}$, as if X_4 and X_5 were independent BRVs. This is because x_4 and x_5 do not appear in the x_1 cofactors of z simultaneously. A function like z is CI with respect to x_4 and x_5 because its output pmf is unaffected by correlations between X_4 and X_5.

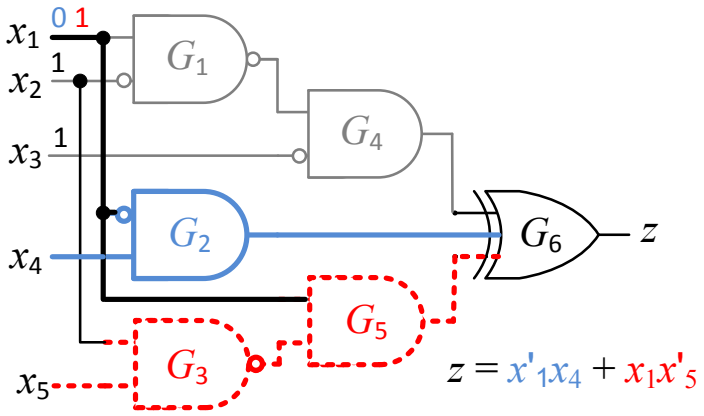


Figure 4.2: Five-input circuit; paths from x_4 (blue) and x_5 (red, dashed) are activated by different values of x_1.

Theorem 4.1: (*Probabilistic Boole-Shannon expansion*) For a Boolean function $z(x_1, ..., x_n)$, if BRV X_1 is independent of the remaining variables, then

$$f_Z(1) = f_{X_1}(0) f_{Z_{\overline{x}_1}}(1) + f_{X_1}(1) f_{Z_{x_1}}(1)$$

where Z_{x_1} and $Z_{\overline{x}_1}$ are the BRVs corresponding to the positive and negative cofactors, respectively, of z with respect to x_1.

Proof. The events $X_1 = 0$ and $X_1 = 1$ are complementary, so

$$\begin{aligned} f_Z(1) &= f_{Z|\overline{X}_1}(1).P\{X_1 = 0\} + f_{Z|X_1}(1).P\{X_1 = 1\} \\ &= f_{Z|\overline{X}_1}(1).f_{X_1}(0) + f_{Z|X_1}(1).f_{X_1}(1) \end{aligned} \tag{4.3}$$

$f_{Z|X_1}$ ($f_{Z|\overline{X}_1}$) is the conditional distribution of Z with respect to the event $X_1 = 1$ ($X_1 = 0$), and is defined as $f_{Z|X_1}(1) = f_{ZX_1}(1, 1)/f_{X_1}(1)$. Since X_1 is independent of the remaining variables, we have $f_{Z|X_1}(1) = f_{Z_{x_1}}(1)$. Similarly, $f_{Z|\overline{X}_1}(1) = f_{Z_{\overline{x}_1}}(1)$, so by Eq. (4.3)

$$f_Z(1) = f_{X_1}(0).f_{Z_{\overline{x}_1}}(1) + f_{X_1}(1).f_{Z_{x_1}}(1)$$

□

We are now ready to formally define CI.

Definition 4.2: A Boolean function $z(x_1, ..., x_n)$ is *correlation insensitive* with respect to variables x_1 and x_2 if the distribution (pmf) of Z only depends on the marginal probability distributions $f_{X_2X_3...X_n}$ and $f_{X_1X_3...X_n}$ and not the joint pmf $f_{X_1...X_n}$. We refer to x_1 and x_2 as *CI inputs* of z.

Equivalently, we could define $z(x_1, ..., x_n)$ as CI with respect to x_1 and x_2, if x_1 and x_2 do not appear in the cofactors of z with respect to $x_3, ..., x_n$, simultaneously. This equivalence is shown in the proof of Theorem 4.2.

Example 4.1: Consider again the function $z(x_1, x_4, x_5) = \bar{x}_1x_4 \vee x_1\bar{x}_5$ in Figure 4.2. We will show that correlations between X_4 and X_5 do not affect the output pmf, whereas

correlations between X_1 and X_4 do. According to Eq. (4.2)

$$f_Z(1) = \sum_{k_1=0}^{1}\sum_{k_4=0}^{1}\sum_{k_5=0}^{1}\big(z(k_1,k_4,k_5)f_{X_1X_4X_5}(k_1,k_4,k_5)\big)$$
$$= f_{X_1X_4X_5}(0,1,0) + f_{X_1X_4X_5}(0,1,1) + f_{X_1X_4X_5}(1,0,0) + f_{X_1X_4X_5}(1,1,0)$$

The first two terms marginalize X_5 and the last two terms marginalize X_4, yielding

$$f_Z(1) = f_{X_1X_4}(0,1) + f_{X_1X_5}(1,0)$$

Z's pmf is a function of $f_{X_1X_4}$ and $f_{X_1X_5}$ only, so by Definition 4.2, z is CI with respect to x_4 and x_5. The function z is not CI with respect to x_1 and x_4 (or x_1 and x_5), because terms like $f_{X_1X_4}$ (or $f_{X_1X_5}$) appear in the pmf of Z. The rest of the example shows how correlations between X_4 and X_5 leave Z unaffected, but those between X_1 and X_4 alter the distribution of Z.

Case 1: All the input BRVs X_1, X_4, X_5 are identically distributed and independent with $f_{X_i}(1) = 0.5$. Consequently,

$$f_Z(1) = f_{X_1X_4}(0,1) + f_{X_1X_5}(1,0) = f_{X_1}(0)f_{X_4}(1) + f_{X_1}(1)f_{X_5}(0) = 0.5 \qquad (4.4)$$

Case 2: The input BRVs have the same marginal distribution as before ($f_{X_i}(1) = 0.5$), but now X_4 and X_5 are highly correlated with the following distribution, while X_1 remains independent

$$f_{X_4X_5}(0,0) = f_{X_4X_5}(1,1) = 0.5 \quad \text{and} \quad f_{X_4X_5}(0,1) = f_{X_4X_5}(1,0) = 0$$

We can also use a vector notation to represent the joint distribution of multiple variables. In this case, we have $\vec{f}_{X_4X_5} = [0.5 \quad 0 \quad 0 \quad 0.5]$, where each element of the vector denotes to the value of $f_{X_4X_5}$ for a corresponding input combination (the first element corresponds to $X_4X_5 = 00$, the second correspond to $X_4X_5 = 01$, etc.). In the current case, the correlation between X_4 and X_5 leaves Z unaffected, following (4.4). This shows that Z is insensitive to correlations between X_4 and X_5.

Case 3: Again $f_{X_i}(1) = 0.5$ and X_5 is independent. This time assume X_1 and X_4 are highly correlated with joint distribution $\vec{f}_{X_1X_4} = [0.5 \quad 0 \quad 0 \quad 0.5]$. The pmf of Z changes

to

$$f_Z(1) = f_{X_1X_4}(0,1) + f_{X_1X_5}(1,0) = 0 + 0.25 = 0.25$$

indicating that z is *not* CI with respect to x_1 and x_4. □

When a circuit's output is CI with respect to two variables, SS can be performed as if those inputs were highly correlated, even when they are independent. For example, if the two inputs have the same probability, we can tie them together as if they were one. The same concept applies to an SC circuit: CI inputs can share the same random source.

Next, we show that CI can be expressed in terms of the *Boolean difference* (BD). The BD of z with respect to x_i is $dz/dx_i = z_{\overline{x}_i} \oplus z_{x_i}$, where $\oplus$ denotes XOR (exclusive-OR), and z_{x_i} and $z_{\overline{x}_i}$ are z's cofactors with respect to x_i [50]. For the function $z = \bar{x}_1x_4 \vee x_1\bar{x}_5$ of Example 4.1, $dz/dx_1 = x_4 \oplus \bar{x}_5$, $dz/dx_4 = \bar{x}_1$ and $dz/dx_5 = x_1$. If $dz/dx_1 = 0$, then x_1 is redundant, meaning that it has no influence on z. This enables X_1 to be marginalized out of Eq. (4.2), making f_Z a function of $f_{X_2...X_n}$ only. In this case, according to Definition 4.2, z is CI with respect to x_1 and x_j for any $j = 2, 3, ..., n$. This points to a method for identifying CI functions.

Theorem 4.2: Function $z(x_1, x_2, ..., x_n)$ with $n > 2$ variables is CI with respect to x_1 and x_2 if and only if for every cube (product term) c containing only variables $x_3, x_4, ..., x_n$, at least one input (x_1 or x_2) is redundant in the cofactor $z_c(x_1, x_2)$ or, equivalently

$$dz_c/dx_1 = 0 \quad \text{or} \quad dz_c/dx_2 = 0$$

Proof. (Sufficiency) For every combination of $x_3...x_n$, i.e., for every cube c containing only the variables $x_3, x_4, ..., x_n$, the cofactor z_c is a function of one of the variables x_1 or x_2, but not both. This means that $dz_c/dx_1 = z_c(0, x_2) \oplus z_c(1, x_2) = 0$ or $dz_c/dx_2 = z_c(x_1, 0) \oplus z_c(x_1, 1) = 0$, implying

$$\begin{aligned} &z_c(0,0) = z_c(1,0) \quad \text{and} \quad z_c(0,1) = z_c(1,1) \\ &\qquad\qquad\qquad \text{or} \\ &z_c(0,0) = z_c(0,1) \quad \text{and} \quad z_c(1,0) = z_c(1,1) \end{aligned} \tag{4.5}$$

Next, we express $f_Z(1)$ using Eq. (4.2) and unroll the first two sums (with respect to k_1 and k_2) to enumerate all the four possible cases of $z_c(0,0)$, $z_c(0,1)$, $z_c(1,0)$ and $z_c(1,1)$:

$$\begin{aligned} f_Z(1) = \sum_{k_3=0}^{1} ... \sum_{k_n=0}^{1} \big(& z(0,0,k_3,...,k_n) f_{X_1X_2...X_n}(0,0,k_3,...,k_n) + \\ & z(0,1,k_3,...,k_n) f_{X_1X_2...X_n}(0,1,k_3,...,k_n) + \\ & z(1,0,k_3,...,k_n) f_{X_1X_2...X_n}(1,0,k_3,...,k_n) + \\ & z(1,1,k_3,...,k_n) f_{X_1X_2...X_n}(1,1,k_3,...,k_n) \big) \end{aligned} \tag{4.6}$$

Equation (4.5) implies that for each iteration of the preceding summation, we can marginalize either X_1 or X_2, yielding

$$z(0,0,k_3,...,k_n) f_{X2...X_n}(0,k_3,...,k_n) + z(1,1,k_3,...,k_n) f_{X_2...X_n}(1,k_3,...,k_n)$$

or

$$z(0,0,k_3,...,k_n) f_{X1...X_n}(0,k_3,...,k_n) + z(1,1,k_3,...,k_n) f_{X_1...X_n}(1,k_3,...,k_n)$$

So the distribution of Z is a function of the marginal distributions $f_{X_2...X_n}$ and $f_{X_1X_3...X_n}$. From Definition 4.2, z is CI with respect to x_1 and x_2.

(Necessity) By way of contradiction, if z is CI with respect to x_1 and x_2, but there is a cube c with $dz_c/dx_1 \neq 0$ and $dz_c/dx_2 \neq 0$, then z_c is a non-degenerate function of both x_1 and x_2. By enumerating all possibilities for z_c—there are only 10 non-degenerate Boolean functions of two variables—we see that for the particular iteration of (4.6) that corresponds to c, none of the variables X_1 and X_2 can be marginalized. Hence, z is not CI with respect to x_1 and x_2, a contradiction from which the necessary condition follows. □

For Example 4.1, $dz_{x_1}/dx_4 = 0$, and $dz_{\overline{x}_1}/dx_5 = 0$, so in all the cofactors of z with respect to the remaining variable x_1, at least one of the variables x_4 and x_5 is redundant, confirming that z is CI with respect to x_4 and x_5.

It is possible to have Boolean functions that are CI with respect to more than two variables. In the extreme case, a function can be CI with respect to all its inputs, in which case, the function is just a constant and the output probability is either 0 or 1.

Definition 4.3: $z(x_1,...,x_n)$ is (*strongly*) *correlation insensitive* with respect to variable set $\mathbb{X} = \{x_1,...,x_k\}$, if it is CI with respect to every pair x_i, x_j in $\mathbb{X}$.

For example, the output of Figure 3.7a is CI with respect to every pair c_i, c_j, so it is strongly CI with respect to $\{c_1, ..., c_n\}$. Correlation insensitivity occurs in several other ways. A function may not be CI with respect to any pair of variables, but have a larger subset of variables that do not affect the output at the same time.

Definition 4.4: $z(x_1, ..., x_n)$ is *weakly correlation insensitive* (WCI) with respect to variables $\mathbb{X} = x_1, ..., x_k$ if the distribution of the BRV Z is only a function of the k marginal probability distributions in which one of the BRVs $X_1, ..., X_k$ is marginalized out, i.e., $f_{X_2X_3...X_kX_{k+1}...X_n}, f_{X_1X_3X_4...X_kX_{k+1}...X_n}, ..., f_{X_1X_2X_3...X_{k-1}X_{k+1}...X_n}$. X is called a WCI set of z.

Weakly CI generalizes CI, so WCI sets can be expected to occur more often than (strong) CI sets. A CI input-pair is a WCI set of size 2, because if the function z is CI with respect to x_1 and x_2, then the BRV Z is a function of $f_{X_2...X_n}$ and $f_{X_1X_3...X_n}$ only, so by Definition 4.4, $\{x_1, x_2\}$ is also a WCI set of z. Similarly, (strong) CI sets of variables are special cases of WCI sets.

Identifying WCI sets is harder than finding CI pairs because more variables are involved. If $z(x_1, ..., x_n)$ is WCI with respect to $\{x_1, x_2, x_3\}$, then for every cube c containing $x_4, ..., x_n$, at least one variable from the set $\{X_1, X_2, X_3\}$ must be marginalized out in the corresponding iteration of Eq. (4.2). This is only possible if some variable is redundant in z_c, or if z_c is CI with respect to at least one pair of its variables. Hence, z is WCI with respect to $\{x_1, x_2, x_3\}$ if and only if for every cube c containing the variables $x_4, ..., x_n$, the cofactor z_c is CI with respect to $\{x_1, x_2\}$, $\{x_1, x_3\}$ or $\{x_2, x_3\}$.

Theorem 4.3: z is WCI with respect to $\mathbb{X} = \{x_1, ..., x_k\}$ if and only if for every cube c containing the variables $x_{k+1}, ..., x_n$, z_c is WCI with respect to at least one subset of size $k-1$ of $\mathbb{X}$.

References [6] [136] discuss methods of quickly identifying CI in large circuits, and show how it improves the quality and speed of SS. For the purpose of this dissertation, we confine ourselves to showing how CI affects SC. Figure 4.3 illustrates the concept with a small example. The original design of the SC adder of Figure 4.3a requires two independent SNGs to generate the input SNs. However, as we showed in this section, z is CI with respect to x and y, so a correlation between the corresponding SNs X and Y will not affect

Z. Consequently, we can use the same RNG for the purpose of generating X and Y, and hence arrive at the circuit of Figure 4.3b, which has a reduced cost.

4.2 Correlation of Stochastic Numbers

In this section, we discuss how the correlation between two bit-streams should be measured in the context of SC. Correlation refers to statistical similarity between two phenomena. As discussed in detail in [44], the correlation of two sequences (bit-streams) is measured by some form of covariance or inner product operation. With appropriate normalization, a correlation value of $+1$ means maximum similarity, a correlation value -1 means minimum similarity (maximum difference), and a correlation of 0 means the sequences are uncorrelated. While many measures of similarity exist [28], they are not very useful in the SC context. The standard definition of correlation (also known as Pearson correlation [28]) $\rho(X, Y)$, in particular, is unsuitable because it imposes constraints on the expected value of the bit-streams. For example, $\rho = +1$ implies that the bit-streams must be identical. A suitable similarity measure should be independent of, or orthogonal to, the data values; in other words, it should not impose constraints on the data. We therefore propose a new correlation measure defined as follows.

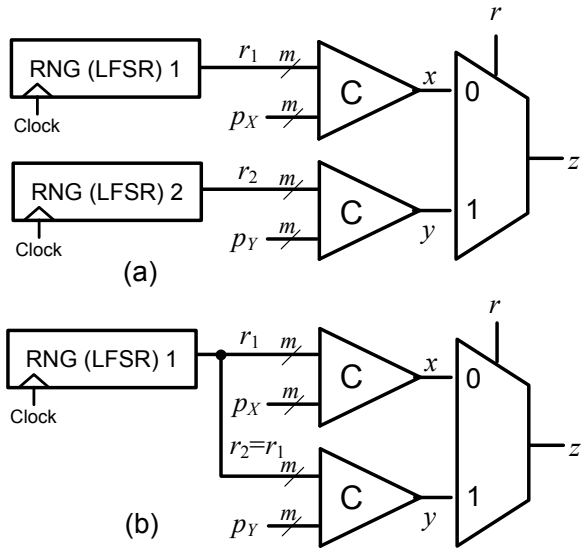


Figure 4.3: Exploiting correlation insensitivity in an SC adder; (a) with independent RNGs and (b) with a shared RNG.

Definition 4.5: The *SC correlation* $SCC(X,Y)$ of two SNs X and Y is given by

$$SCC(X,Y) = \begin{cases} \dfrac{p_{X\wedge Y} - p_X p_Y}{\min(p_X, p_Y) - p_X p_Y} & \text{if } p_{X\wedge Y} > p_X p_Y \\ \dfrac{p_{X\wedge Y} - p_X p_Y}{p_X p_Y - \max(p_X + p_Y - 1, 0)} & \text{otherwise} \end{cases} \tag{4.7}$$

The starting point in constructing $SCC(X,Y)$ is obtaining the bit-wise AND function $X \wedge Y$ (a kind of inner product) of the SNs, that is, finding $p_{X\wedge Y}$. This is then centralized by the uncorrelated value $p_X p_Y$ yielding $p_{X\wedge Y} - p_X p_Y$ Finally, the centralized value is normalized by dividing it by the maximum possible values. The centralization makes SCC consistent with the definition of independence in [57] when $SCC(X,Y) = 0$, i.e., when $p_{X\wedge Y} - p_X p_Y = 0$. The normalization guarantees that for two maximally similar (or different) SNs X and Y, we get $SCC(X,Y) = +1$ (or -1). Unlike the standard correlation measure $\rho(X,Y)$, SCC does not vary with the SN values. All the intermediate values of SCC are linearly interpolated between the independent case and the maximum similarity (or difference) case. For example, $SCC(X,Y) = 0.5$ means that $p_{X\wedge Y}$ is half-way between $p_X p_Y$, i.e., the independent case, and $\min(p_X, p_Y)$, i.e., the maximum overlap case.

We can also define SCC using the notation of [28], which allows easy comparison with other correlation concepts. For two n-bit SNs X and Y, denote the number of overlapping 1's by a, the number of overlapping 1's of X and 0's of Y by b, the number of overlapping 0's of X and 1's of Y by c, and the number of overlapping 0's on both SNs by d. Clearly, $a + b + c + d = n$. We then have the following definition which is equivalent to Definition 4.5:

$$SCC(X,Y) = \begin{cases} \dfrac{ad - bc}{n\min(a+b, a+c) - (a+b)(a+c)} & \text{if } ad > bc \\ \dfrac{ad - bc}{(a+b)(a+c) - n\max(a-d, 0)} & \text{otherwise} \end{cases} \tag{4.8}$$

The numerator $ad - bc$ is common to many similarity measures including Pearson correlation

$$\rho(X,Y) = \frac{ad - bc}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$$

and it captures the overlap of 0's and 1's in the two bit-streams. The denominator, on the other hand, is simply a normalization factor. While Pearson correlation is normalized by the variance of the bit-streams, SCC is normalized so that the bit-streams with maximum

Table 4.1: Some SNs with their SCC and standard correlation values.

Stochastic numbers	**SC correlation** $SCC(X,Y)$	**Standard correlation** $\rho(X,Y)$
$X = 11110000 \quad Y = 11001100$	0	0
$X = 11110000 \quad Y = 11110000$	$+1$	$+1$
$X = 11110000 \quad Y = 00001111$	-1	-1
$X = 11111100 \quad Y = 11110000$	$+1$	$+0.58$
$X = 11111100 \quad Y = 00001111$	-1	-0.58
$X = 11111100 \quad Y = 11100001$	0	0
$X = 11000000 \quad Y = 11111100$	$+1$	$+0.33$

(minimum) overlap of 1's and 0's lead to $SCC = +1$ (-1), independent of the values of the SNs.

Table 4.1 shows examples of bit-streams with their ρ and SCC values. Note that ρ and SCC are the same for independent SNs, and for SNs with equal values. When the SNs have different values, SCC consistently gives the value $+1$ (or -1) for maximum (minimum) overlap of 1's and 0's between the bit-streams, while ρ gives different values. This shows that, unlike ρ, SCC is not affected by the values of the bit-streams.

As mentioned earlier, the function of a stochastic circuit can effectively be changed by enforcing correlations among its inputs. The XOR gate of Figure 1.8 illustrates this. Figure 4.4a shows the stochastic functions implemented by the same XOR gate at different levels of SCC. In all cases, the output of the function remains the same at the four corners, but the function changes greatly for the intermediate values. Figure 4.4b shows the same function for various fixed values of p_Y and SCC.

4.3 Combinational Circuits

Every stochastic circuit implements a real-valued function F, which is interpreted as its stochastic behavior. For example, the AND gate of Figure 1.3 implements the multiplication function $p_Z = F(p_X, p_Y) = p_X p_Y$, assuming $SCC(X,Y) = 0$. The inputs of F are the values of the SNs p_X and p_Y . Hence, to obtain the stochastic behavior of a logic circuit, we need to determine its corresponding probability function F.

We saw earlier that a circuit's functionality can change in the presence of correlation. The AND gate, for example, implements $p_Z = F(p_X, p_Y) = \min(p_X, p_Y)$ if $SCC(X,Y) = +1$. Based on Definition 4.5 and the subsequent discussion, the SC function of the AND gate for the case of, say $SCC(X,Y) = 0.5$, is half-way between its

SC function for $SCC(X,Y) = 0$ and $SCC(X,Y) = +1$. In general, we can express a circuit's functionality as a linear combination of its functions at $SCC(X,Y) = 0$ and $SCC(X,Y) = +1$ or -1. Hence for any SCC, p_Z can be written as

$$p_Z(p_X, p_Y) = \begin{cases} \big(1 + SCC(X,Y)\big).F_0(p_X, p_Y) \\ \quad -SCC(X,Y).F_{-1}(p_X, p_Y) & \text{if } SCC(X,Y) > 0 \\ \big(1 - SCC(X,Y)\big).F_0(p_X, p_Y) \\ \quad +SCC(X,Y).F_{+1}(p_X, p_Y) & \text{otherwise} \end{cases} \tag{4.9}$$

where $F_0(p_X, p_Y)$, $F_{-1}(p_X, p_Y)$ and $F_{+1}(p_X, p_Y)$ are the functions of the same circuit at $SCC(X,Y) = 0$, -1 and $+1$, respectively. For the AND gate example, we have $F_0(p_X, p_Y) = p_X p_Y$, $F_{-1}(p_X, p_Y) = \max(p_X + p_Y - 1, 0)$, and $F_{+1}(p_X, p_Y) = \min(p_X, p_Y)$.

In order to derive the stochastic behavior of a circuit C with correlated inputs, we use the vector notation defined in Section 2.1. This notation is closely related to the notation of probabilistic transfer matrices [69] which will be discussed in the next chapter. In this notation, we use a vector $\boldsymbol{I}$ to represent a set of correlated inputs. Each element i_k of $\boldsymbol{I}$ denotes the probability of one input combination. BFs (and elementary gates) are represented by a vector $\boldsymbol{M}$ that shows their truth-table. Using this notation, we obtain the probability

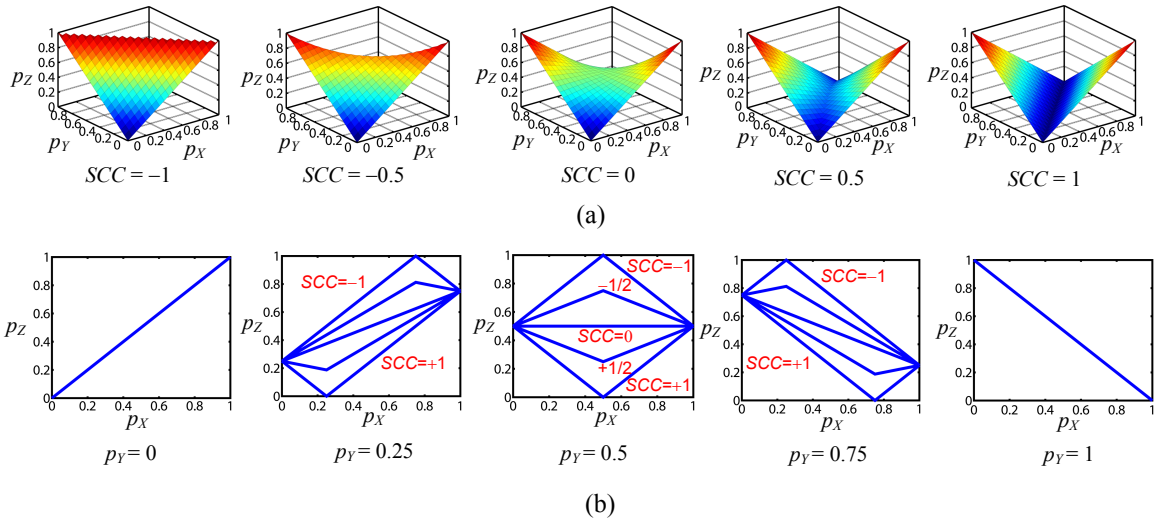


Figure 4.4: Functions implemented by an XOR gate (a) with different input SCC values, and (b) with fixed values of p_Y.

Table 4.2: Functions used in the elements of input vector $\boldsymbol{I}$ of a two-input circuit.

	$F_0(p_X, p_Y)$, $SCC = 0$	$F_{-1}(p_X, p_Y)$, $SCC = -1$	$F_{+1}(p_X, p_Y)$, $SCC = +1$
i_0	$(1-p_X)(1-p_Y)$	$\max(1-p_X-p_Y, 0)$	$\min(1-p_X, 1-p_Y)$
i_1	$(1-p_X)p_Y$	$\min(1-p_X, p_Y)$	$\max(p_Y-p_X, 0)$
i_2	$p_X(1-p_Y)$	$\min(p_X, 1-p_Y)$	$\max(p_X-p_Y, 0)$
i_3	$p_X p_Y$	$\max(p_X+p_Y-1, 0)$	$\min(p_X, p_Y)$

of seeing a 1 at the output z through $p_Z = \boldsymbol{I} \cdot \boldsymbol{M}$, where "$\cdot$" denotes the inner product (See Eq. (2.2)).

If C has two input bit-streams X and Y with correlation SCC, construct the input vector $\boldsymbol{I} = [i_0 \quad i_1 \quad i_2 \quad i_3]$, in which the i_k's are expressed in the form of Eq. (4.9). The F_0's, F_{-1}'s, and F_{+1}'s corresponding to each i_k are defined in Table 4.2. From these, we can extract C's stochastic behavior by computing the inner product of the vector $\boldsymbol{I}$ and the circuit's truth-table vector $\boldsymbol{M}$. For instance, if C is an XOR gate,

$$p_Z = \begin{bmatrix} i_0 & i_1 & i_2 & i_3 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix} = i_1 + i_2$$

yields the functions illustrated in Figure 4.4a for some representative values of SCC.

Only correlation corresponding to the special case $SCC = 0$ has been considered in the literature. As we have just seen, other cases such as $SCC = +1$ lead to useful results. The vector formulation of two-input SC circuits discussed above points to a method for synthesizing stochastic circuits with correlated inputs. In this approach, a target function $F(p_X, p_Y)$ is approximated by a function $p_Z = F'(p_X, p_Y)$ defined by

$$p_Z = \begin{bmatrix} i_0 & i_1 & i_2 & i_3 \end{bmatrix} \cdot \begin{bmatrix} t_0 & t_1 & t_2 & t_3 \end{bmatrix} \tag{4.10}$$

in which the t_k's are the parameters of F' to be determined and the i_k's are expressed in the form of Eq. (4.9) and Table 4.2. The process of approximation is to find the best t_k's and the best SCC for which the following error function is minimized.

$$\varepsilon = \iint_{0,0}^{1,1} \big(F(p_X, p_Y) - F'(p_X, p_Y)\big)^2 dp_X dp_Y \tag{4.11}$$

This is done by adjusting the t_k's and the SCC using standard optimization techniques. Because of the limited number of parameters and the well-behaved error function, this problem is relatively easy to solve. Once the parameters are found, F' can be realized

by a logic circuit with the overall multiplexer-style structure shown in Figure 4.5, The constant SNs needed for the four t_k's can be generated by up to four copies of the circuit in Figure 2.3a. The resulting circuit can then be further optimized using conventional logic synthesis methods and tools.

Generating two SNs X and Y with a desired level of SCC is a problem that has not been studied before. We propose to use the circuit structure shown in Figure 4.6, which generates X by means of a standard SN generator and, depending on the sign SCC_{sign} and magnitude SCC_{magn} of SCC, mixes uncorrelated and correlated versions of Y together. For example, if $SCC = +1$, then Y is generated from the same random number source used by X, so X and Y become highly correlated. Note that Figure 4.6 is a *programmable* structure, and not all the components shown are needed in every design. For example, if a circuit requires X and Y to be generated with $SCC = +1$, then the select inputs xy of the multiplexer are set to 01, implying that random number generators 2 and 3, and their associated circuits can be removed. Procedure 4.1 summarizes our proposed *correlated combinational circuit* (CCC) synthesis algorithm.

As an example, consider the problem of synthesizing a circuit for the target function $F(p_X, p_Y) = \min(p_X, p_Y)$ In Step 1 of CCC, $p_Z = F'(p_X, p_Y)$ is prepared in the form of Eq. (4.10), and in Step 2 the error function ε is prepared in the form of Eq. (4.11). Then, the t_k's and SCC are adjusted until ε is minimized. For the running example, $\varepsilon = 0$, i.e., the exact target function, can be achieved by assigning $t_0 = 0$, $t_1 = 0$, $t_2 = 0$, $t_3 = 1$ and $SCC = +1$. On plugging these values into the circuit of Figure 4.5, we obtain that of Figure 4.7a, i.e., an AND gate. Observe that the AND implements $\min(p_X, p_Y)$ only if its inputs have $SCC = +1$. In order to generate X and Y, we use the circuit of Figure 4.6 and plug in $SCC = +1$ ($SCC_{\text{sign}} = 0$, $SCC_{\text{magn}} = 1$), yielding the circuit of Figure 4.7b. This

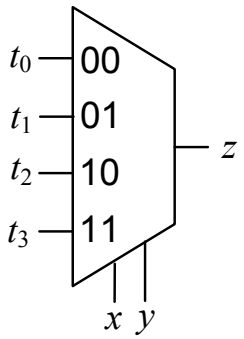


Figure 4.5: High-level structure of a synthesized two-input circuit with correlated inputs, prior to simplification.

Procedure CCC($F(p_X, p_Y)$) – synthesizes F

Step 1. Determine a suitable approximating function $p_X = F'(p_X, p_Y)$ according to Eq. (4.10) in which the input vector I is defined by Eq. (4.9) and Table 4.2.

Step 2. Determine the error function ε given by Eq. (4.11).

Step 3. Minimize ε by adjusting the t_k and SCC parameters in F'.

Step 4. Insert these parameters into the structure of Figure 6, and use standard logic synthesis methods to optimize the resulting circuit.

Procedure 4.1: Procedure CCC to synthesize a stochastic function $F'(p_X, p_Y)$ with correlated inputs.

circuit is smaller than one employing two of the independent SN generators in Figure 2.3a, so generating correlated SNs is cheaper than generating independent ones in this case.

Table 4.3 shows examples of circuits synthesized by the CCC algorithm. Most of the target functions are useful non-linear functions that have no efficient stochastic implementation when the inputs are uncorrelated. The last synthesized function in the table is the multiplexer-based scaled adder in which correlation of the input data does not matter. Another type of SC adder, a saturating adder, is also shown. This circuit adds its inputs

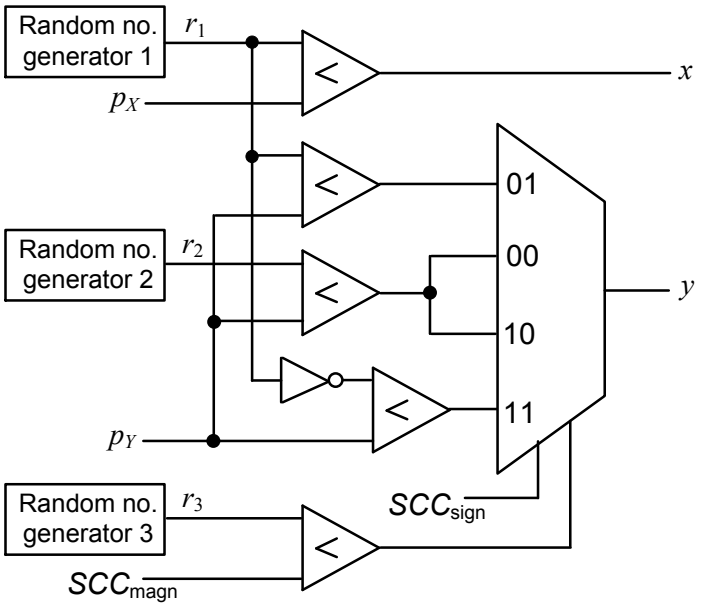


Figure 4.6: Generating SNs with a specified SCC.

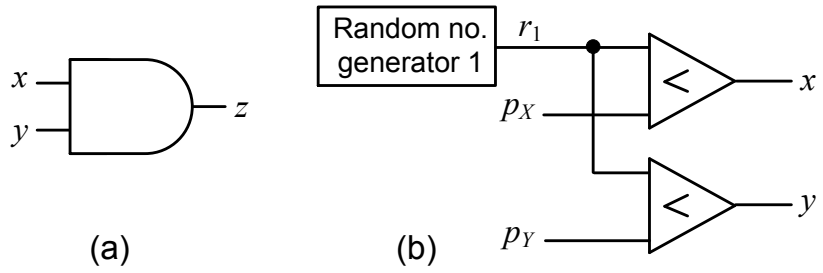


Figure 4.7: Implementing the function $F(p_X, p_Y) = \min(p_X, p_Y)$: (a) synthesized stochastic circuit, and (b) corresponding SN generator.

Table 4.3: Examples of synthesized stochastic circuits exploiting various correlation levels.

Target function	$[t_0 \quad t_1 \quad t_2 \quad t_3]$	SCC	**Synthesized circuit**
$p_Z = \min(p_X, p_Y)$	$[0 \quad 0 \quad 0 \quad 1]$	$+1$	AND gate with positively correlated inputs
$p_Z = \max(p_X, p_Y)$	$[0 \quad 1 \quad 1 \quad 1]$	$+1$	OR gate with positively correlated inputs
$p_Z = \lvert p_X - p_Y \rvert$	$[0 \quad 1 \quad 1 \quad 0]$	$+1$	XOR gate with positively correlated inputs; implements absolute-valued subtraction
$p_Z = \min(p_X + p_Y, 1)$	$[0 \quad 1 \quad 1 \quad 1]$	-1	XOR gate with negatively correlated inputs; implements saturating addition
$p_Z = \frac{1}{2}(p_X + p_Y)$	$[0 \quad \frac{1}{2} \quad \frac{1}{2} \quad 1]$	Any	Multiplexer with arbitrary correlation among its data inputs; implements scaled add

without scaling until the saturating value 1 is reached. Finally, observe that in some cases, the circuits synthesized by CCC are the same as the standard designs. For example, the smallest SC multiplier is the AND gate of Figure 1.3, which requires uncorrelated inputs. This shows that the CCC is capable of replicating circuits synthesized by existing methods, because $SCC = 0$ is also allowed in CCC.

Table 4.4 compares the circuits synthesized by CCC and those designed by the spectral synthesis method of [1], which makes the usual independent-inputs assumption, i.e., $SCC = 0$. In addition to the circuits of Table 4.3, a few other functions were implemented. Since it is normally impossible to implement real-valued functions exactly, some are approximated before synthesis. Area is estimated by mapping the circuits to a generic library of cells using 0.35μm CMOS technology [120]. For a fair comparison, we also report the measured mean error between the synthesized and target functions F' and F. The results indicate that in most cases, the circuits synthesized by CCC are smaller and more accurate than those designed by the method of [1].

When dealing with more than two signals, considering their pairwise SCC values may be insufficient, as higher-order correlations can exist among groups of three or more of the signals. To handle such cases, we suggest using vectors that are large enough to embed

Table 4.4: Comparison between circuits synthesized by CCC and those synthesized by the spectral method of [1].

Target function	**Synthesis method and correlation assumption**	**Area (μm^2)**	**Mean error (%)**
$p_Z = \min(p_X + p_Y, 1)$ Saturating adder	Method of [1] ($SCC = 0$)	1,628	10
	CCC with $SCC = -1$	1,091	0
$p_Z = \max(p_X - p_Y, 0)$ Saturating subtracter	Method of [1] ($SCC = 0$)	1,663	10
	CCC with $SCC = +1$	1,118	0
$p_Z = p_X \times p_Y$ Multiplier	Method of [1] ($SCC = 0$)	1,646	0
	CCC with $SCC = 0$	1,646	0
$p_Z = \frac{1}{2}(p_X + p_Y)$ Scaled adder	Method of [1] ($SCC = 0$)	1,857	0
	CCC with $SCC = +1$	1,320	0
$p_Z = (p_X p_Y)^{0.45}$	Method of [1] ($SCC = 0$)	1,980	12
	CCC with $SCC = +1$	1,443	7
$p_Z = (1 - p_X - p_Y)^2$	Method of [1] ($SCC = 0$)	2,306	15
	CCC with $SCC = -1$	1,760	9
A multivariate polynomial	Method of [1] ($SCC = 0$)	2,086	9
	CCC with $SCC = \frac{1}{2}$	2,473	4

all the signal correlations of interest. Circuits with many inputs can also be designed by decomposing the target function into subfunctions of two variables. The CCC method can then be used to synthesize the pieces and put them back together. For example, consider the four-variable function $p_Z = \frac{1}{2}(|p_X - p_Y| + |p_V - p_W|)$, which performs the useful image-processing task of edge detection [3]. It decomposes into two absolute-valued subtraction functions p'_Z and p''_Z, which are then combined by a scaled add to produce p_Z:

$$p'_Z = |p_X - p_Y|, p''_Z = |p_V - p_W|, p_Z = \frac{1}{2}(p'_Z + p''_Z)$$

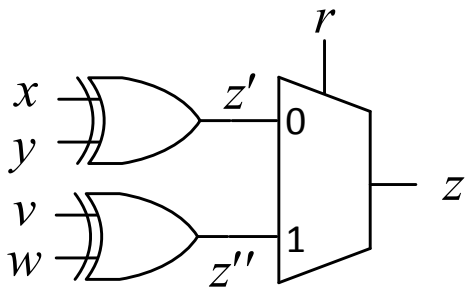


Figure 4.8: Stochastic circuit for image edge detection [3].

All three of these functions can be synthesized by CCC, as indicated in Table 4.3. Figure 4.8 shows the result; the XOR gates perform absolute-valued subtraction, while the multiplexer performs scaled addition. Note that the select input of the multiplexer is fed by an auxiliary input r with $p_R = \frac{1}{2}$.

4.4 Summary

In this chapter, we analyzed how correlation among SNs affects SC circuits. Most of the previously designed SC circuits assume that their input SNs are independent (uncorrelated), because it was believed that correlation leads to inaccurate computation. An example of this issue was illustrated in Figure 1.3b, where correlated inputs hindered the multiplication process. So the usual solution has been to avoid correlation entirely at the cost of introducing many independent stochastic number sources or re-randomizers. However, we showed that correlation is not always undesirable. We introduced the concept of correlation insensitivity, which was originally applied to a closely related problem in statistical simulation. CI functions are not affected by correlations among their inputs, which means that the inputs can be fed by correlated bit-streams without affecting the probabilistic behavior of the circuit. In the context of SC, CI inputs of a circuit can share random number sources (such as LFSRs), which leads to significant cost savings.

Furthermore, we showed that contrary to what one would intuitively expect, correlation can serve as a resource in designing stochastic circuits. To exploit this, we introduced a new correlation measure called SCC, which is especially suited to quantifying the similarity of SNs. Then we showed that systematic correlation can change the functionality of an SC circuit in useful ways. In effect, correlation can be seen as a new dimension in SC circuit design. Based on this, we discussed an extension of STRAUSS that exploits correlation. Using this method, we demonstrated how to implement several useful functions such as saturating addition and subtraction that had no previous efficient SC implementations. We reported a comparative study indicating that the circuits with correlated inputs are generally smaller and more accurate than combinational designs with independent inputs.

Chapter 5

Errors Affecting Stochastic Computing

In this chapter, we discuss and analyze errors that affect stochastic computing (SC). First, we categorize the errors, and then we focus on environment-induced or soft errors, where we show SC outperforms conventional binary circuits in terms of error tolerance. To analyze the effect of soft errors, we employ the well-known probabilistic transfer matrix (PTM) tools [69] (a brief introduction will be provided). Finally, we show the SC circuits are also tolerant of errors induced by aggressive voltage/frequency scaling; a frequently used power reduction technique for contemporary circuits. This chapter's material has been published in [23] and [9].

5.1 Error Categories

There exist various error types that can affect the course of stochastic computation. Qian et al. [109] provide a good error classification for their ReSC architectures. We generalize their classification to cases where soft errors exist. Table 5.1 shows this classification. As we will explain next, in some cases it is hard to draw a line between some categories, as they may be closely related.

Approximation and Quantization Errors: When using a design method like STRAUSS [8], a given target function is usually transformed into polynomial form before implementation. However, not every target function has a good polynomial fit, especially if the function is highly non-linear and the fitted polynomial has a low degree. A straightforward solution to address this error is to increase the degree of the fitting polynomial. This, however, comes at a great cost as shown in Chapter 3. A similar error appears during the design flow (using STRAUSS, for instance), when constant numbers are needed. The precision

Table 5.1: Categories of errors affecting SC, and their possible causes and solutions.

Error Category	Possible Cause	Possible Solution
Approximation, Quantization	Non-linear target functions, Low-degree polynomial approximation, Low-precision constant number generation	Increase polynomial degree, Increase number of bits in constant number generation
Random fluctuation	Inherent randomness, Short bit-stream length	Increase bit-streams length, Use deterministic or low-discrepancy sequences
Insufficient randomness	Correlation, Auto-correlation, Poor random source selection	Increase random sources, De-correlate correlated signals, Use re-randomizers to remove auto-correlation, Use better number sources (larger LFSRs)
Soft errors	Environmental noise, Component variability, Voltage/frequency scaling	Use circuit-level error-resilience techniques, Balance 0-to-1 and 1-to-0 errors

selected during the single constant generation (SCG) procedure (Procedure 3.2) quantizes the original constant, and hence may produce a quantization error in the implemented function. Similar to the previous case, this error can be addressed by increasing the number of bits used in the SCG process.

Random Fluctuations: Stochastic numbers (SNs) are naturally probabilistic, and so their representation of a value is also probabilistic. When representing a number p_Z using a bit-stream of length N, we are in fact looking at N independent Bernoulli trials with probability p_Z of being successful (generating 1). The value represented by the bit-stream, i.e., the number if 1's in it, will hence have a binomial distribution with variance $p_Z(1-p_Z)/N$. So the number represented fluctuates, and the fluctuation is decreased by increasing the length N of the bit-stream. As we will show in the next chapter, it is also possible to reduce the fluctuation error by using deterministic number sources such as low-discrepancy sequences.

Insufficient Randomness: This error occurs when the number of random sources used in a circuit is fewer than what is required. For instance, errors caused by correlation fall into this category, because they can usually be corrected by using an additional random source for one of the correlated signals. When feedback is present in the circuit, auto-correlation within a signal can also produce this error, which again can be addressed by increasing the randomness in the circuit, e.g., by adding re-randomizers in the feedback paths [126]. A

poor selection of random sources may also lead to insufficient randomness. For instance, using an LFSR with a period smaller than N (the bit-stream length) will cause the LFSR to repeat its sequence during the course of the computation, and hence produce errors. While there are similarities between this error type and the random fluctuation error, we distinguish this category by the following test. If the error is resolved by increasing the bit-stream length N, then it is classified as the random fluctuation error, otherwise it is from the insufficient randomness category.

Soft Errors: This category includes the errors that occur because of physical fluctuations in circuit parameters, and are not unique to SC—they appear in non-SC circuits, too. Soft errors of the bit-flip type fall into this category. These are errors that usually appear because of noise induced by the environment, and they alter the circuit by flipping the state of a wire. Errors due to circuit variations, or to aggressive voltage/frequency scaling, also fall into this category. Soft errors are difficult to analyze and quantify. A big portion of this chapter deals with analyzing SC in situations where soft errors exist, especially in the presence of high error rates such as those encountered in avionics or spacecraft instrumentation [40]. Addressing this type of error involves using circuit-level techniques, e.g., radiation hardening against cosmic rays. In the case of SC, it is possible to exploit the redundant coding of SNs to reduce the error. This can be done by balancing the number of 0-to-1 and 1-to-0 errors, so that they cancel each other out.

Before getting into the analysis of soft errors, we need to introduce the PTM tools that we employ.

5.2 Probabilistic Transfer Matrices

A convenient tool for analyzing the probabilistic behavior of logic circuits is the probabilistic transfer matrix (PTM) and its associated algebra [69]. PTMs were introduced to analyze the reliability of conventional logic circuits. Their 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, an n-input m-output combinational circuit's normal (fault-free) behavior 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

$$J_{\text{AND}} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.1}$$

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 Eq. (5.1), the top row tells us that the AND produces output $z = 0$ with probability 1, and output $z = 1$ with probability 0, in response to the input $xy = 00$. By choosing suitable probability values, PTMs can be constructed to represent a remarkably wide range of error scenarios [69]. For example, the effect of a bit-flip error e with rate p_e on the output of the AND gate multiplier is represented by the PTM

$$M_{\text{AND}} = \begin{bmatrix} 1 - p_e & p_e \\ 1 - p_e & p_e \\ 1 - p_e & p_e \\ p_e & 1 - p_e \end{bmatrix} \tag{5.2}$$

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 sum 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 or circuit C is represented by a PTM that describes C's logic function and error status; see Figure 5.1. An n-output fanout gate F_n copies an input signal to its n outputs. Figure 5.1c shows the ITM for the 2-output fanout gate F_2. Wire permutations such as crossing wires are represented by the swap or crossover gate. The ITM for an adjacent wire swap is shown in Figure 5.1d; any permutation of wires can be modeled by a series of adjacent swaps. A wire corresponds to a 2×2 PTM; the ITM case is just the 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.

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.

For example, consider again the faulty AND gate multiplier with $p_X = 0.2$ and $p_Y = 0.3$, and $p_e = 0.05$. To determine the probability of getting a 1 at the gate's output, we first form the input vectors, namely, $M_x = [0.8 \quad 0.2]$ and $M_y = [0.7 \quad 0.3]$. These are then combined via the tensor product

$$M_{xy} = M_x \otimes M_y = \begin{bmatrix} 0.56 & 0.24 & 0.14 & 0.06 \end{bmatrix} \tag{5.3}$$

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.

$$M_{z^*} = M_{xy} \times M_{\text{AND}} = \begin{bmatrix} 0.56 & 0.24 & 0.14 & 0.06 \end{bmatrix} \times \begin{bmatrix} 0.95 & 0.05 \\ 0.95 & 0.05 \\ 0.95 & 0.05 \\ 0.05 & 0.95 \end{bmatrix} = \begin{bmatrix} 0.896 & 0.104 \end{bmatrix}$$

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

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

Figure 5.1: 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.

From this, we conclude that p_{z^*}, i.e., the probability of seeing a 1 in z^*, is 0.104 . Note that the PTM M_{AND} of the AND gate implicitly incorporates the bit-flip error, as is also the case in (5.2).

5.3 Effect of Soft Errors on Stochastic Numbers and Circuits

An SN X, as defined in Chapter 2, is a bit-stream carrying a probability value p_X that denotes 1 (success) or 0 (failure). It 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 [124]. Since probabilistic behavior can be easily modeled and analyzed in terms of Bernoulli RVs, we now use them to give a formal definition of SNs 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. The sampling process is, in fact, a very basic form of stochastic computing. For instance, assume that the AND gate multiplier 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 hence estimating p_Z. The expected rate of success at z can be calculated by the *expected value* operator denoted as $\mathbb{E}[Z]$. 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 $\widehat{p}_Z$, which we refer to as the *estimated* value in contrast with the *exact* value p_Z. The estimated value $\widehat{p}_Z$ is obtained by sampling the circuit/RV N times and recording the number N_1 of 1's appearing at the output; this yields $\widehat{p}_Z = \frac{N_1}{N}$. 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 $\widehat{p}_Z = \frac{2}{8} = 0.25$.

In general, $\widehat{p}_Z$ can be any of the 2^N different bit-streams derived from random sources, which allows p_Z and $\widehat{p}_Z$ to differ, sometimes significantly, from one another. This differ-

ence between p_Z and $\widehat{p}_Z$ 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) $E_Z = \mathbb{E}[(p_z - \widehat{p}_Z)^2]$. In the case of the Bernoulli RVs of interest here, we have [124]

$$E_Z = \frac{p_Z(1-p_Z)}{N} \tag{5.4}$$

Thus the MSE of an SN estimate can be reduced by increasing the number of samples i.e., the bit-stream length N. Note that E_Z is a function of p_Z and N only, implying that no matter what the circuit is (whether the AND gate multiplier or any other circuit), once the expected rate of success p_Z at the output is calculated, we can use Eq. (5.4) to calculate its MSE. Also note that the MSE changes with the value p_Z of the SN. It is maximized when $p_Z = 1/2$, and it becomes zero for $p_Z = 0$ or 1. We will now see that soft errors affect SNs differently because their effect is minimized when $p_Z = 1/2$.

Besides the random fluctuations inherent in the selection of a particular bit-stream to represent Z in a stochastic circuit C, various nondeterministic physical phenomena 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 [13] that e causes bit-flips in z, which affect 0's and 1's 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. The circuit's fault-free output z then changes to an erroneous function z^*, as illustrated in Figure 5.2a for the AND gate 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 5.2b 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 gate multiplier from 0.06 to 0.104. An analytical method of calculating the expected value p_{Z^*} and its MSE will be developed next.

Consider an SN 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

$$p_{X^*} = \mathbb{E}[X^*] = p_X + p_e(1 - 2p_X) \tag{5.5}$$

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 $\widehat{p}_{X^*}$ and the exact value p_X

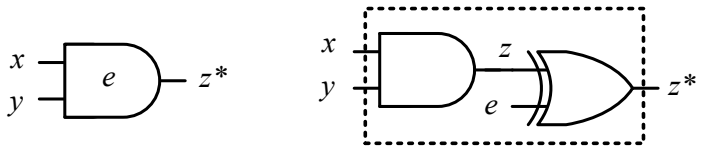
$$E_{X^*} = \mathbb{E}\big[(\widehat{p}_{X^*} - p_X)^2\big]$$

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 Eq. (5.5), we should be able to find p_{X^*} and hence E_{X^*} analytically. Assuming the estimated value $p_{X^*} = \frac{1}{N}\sum_{i=1}^{N} X_i^*$ obtained by summing N independent samples of X^*, we get

$$\begin{aligned} E_{X^*} &= \mathbb{E}\big[(\widehat{p}_{X^*} - p_X)^2\big] = \mathbb{E}\big[\widehat{p}_{X^*}^2 + p_X^2 - 2p_X\widehat{p}_{X^*}\big] \\ &= \mathbb{E}\big[\widehat{p}_{X^*}^2\big] + \mathbb{E}\big[p_X^2\big] - \mathbb{E}\big[2p_X\widehat{p}_{X^*}\big] \\ &= \frac{N^2 p_{X^*}^2 + N p_{X^*}(1 - p_{X^*})}{N^2} + p_X^2 - 2p_X p_{X^*} \\ &= (p_{X^*} - p_X)^2 = \frac{p_{X^*}(1 - p_{X^*})}{N} \end{aligned} \tag{5.6}$$

There are two terms in the final result of (5.6). The first term 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 (5.5) into (5.6) thus:

$$E_{X^*} = p_e^2(1 - 2p_X)^2 + \frac{1}{N}\Big(p_X(1 - p_X) + p_e(1 - p_e)\big(1 - 4p_X(1 - p_X)\big)\Big) \tag{5.7}$$



(a) (b)

Figure 5.2: 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.

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

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

Using the above equations, we compare the effect of bit-flips on a 5-bit binary number and an SN of length 32. Figure 5.3a 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 5.3b 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 5.3 by Monte Carlo simulation.

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

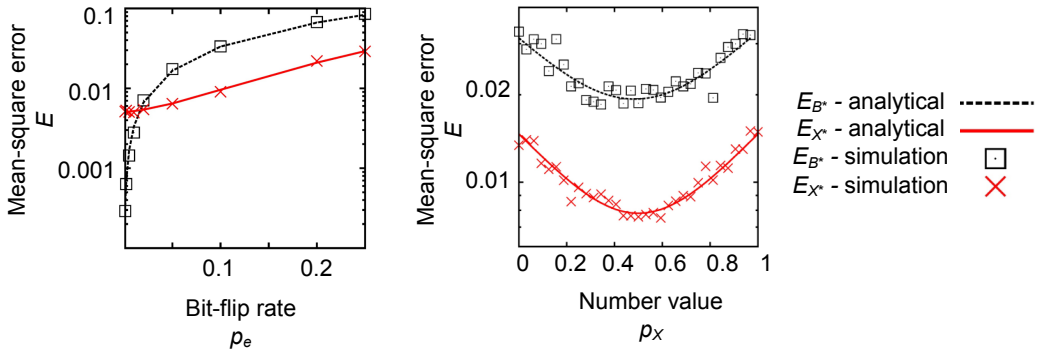


Figure 5.3: MSE of a stochastic number E_{X^*} and a binary number E_{B^*} in the presence of bit-flips calculated using analytical and simulation methods; (a) for different values of p_e and (b) for different values of p_X.

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 of Figure 5.2 which is multiplying two SNs X and Y and has output error p_e. Generalizing Eq. (5.3) gives the 1×4 input vector

$$M_{xy} = \begin{bmatrix} (1-p_X)(1-p_Y) & (1-p_X)p_Y & p_X(1-p_Y) & p_X p_Y \end{bmatrix}$$

Now, consider two cases: first, the AND gate is error-free, and second, it contains the error e defined by the PTM in (5.2). In the error-free case, the output vector is

$$M_{xy} \times J_{\text{AND}} = \begin{bmatrix} 1-p_Z & p_Z \end{bmatrix} = \begin{bmatrix} 1-p_X p_Y & p_X p_Y \end{bmatrix}$$

indicating that the probability of output 1 is $p_X p_Y$. If an error e is present, then using M_{AND} from (5.2), 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 Eq. (5.6).

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 [69]. 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. 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.

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 5.4ab. The special input r is a constant scaling factor of value 0.5. The corresponding circuits with XOR gates added for error injection are shown in Figure 5.4cd. 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 5.5 presents error data obtained by PTM analysis and circuit simulation for the two basic gate types AND and NOT, as well as the scaled adder circuits of Figure 5.4. 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 by soft errors. The MSE is given by $E_{Z^*} = \mathbb{E}[(\widehat{p}_{Z^*} - p_Z)^2]$. As Figure 5.5 shows, the analytical and simulation results are quite consistent.

We also constructed PTMs M_{MAJ} and M_{MUX} for the circuits of Figure 5.4ab level by level from the PTMs of their component gates, including wiring gates, according to the method of [69]. The vertical lines on the figure separate the circuits into levels. In high-

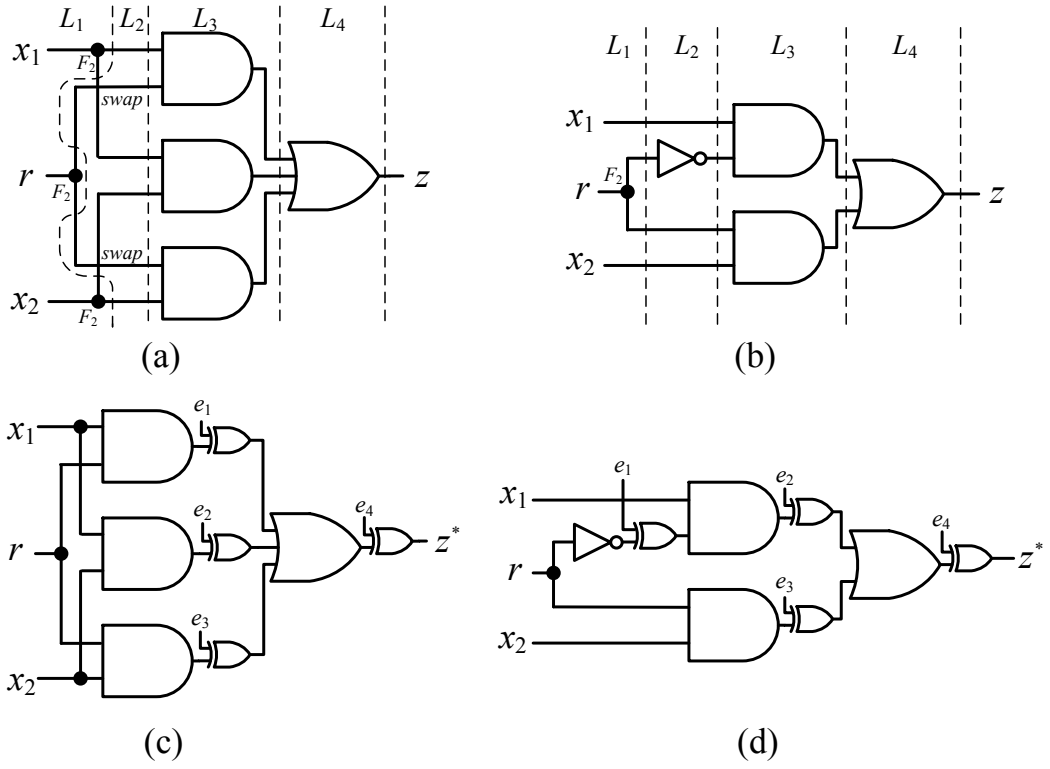


Figure 5.4: Stochastic circuits for the scaled addition $p_Z = 0.5(p_{X_1} + p_{X_2})$: (a) majority-based, (b) multiplexer-based, (c) majority-based with error injection, and (d) multiplexer-based with error injection; dashed vertical lines separate levels of the circuits.

level symbolic form, we obtain the PTM expressions

$$M_{\text{MAJ}} = (F_2 \otimes F_2 \otimes F_2) \times (I \otimes \text{swap} \otimes \text{swap} \otimes I) \times (\text{AND2}_{p_e} \otimes \text{AND2}_{p_e} \otimes \text{AND2}_{p_e}) \times (\text{OR3}_{p_e})$$

$$M_{\text{MUX}} = (I \otimes F_2 \otimes I) \times (I \otimes \text{NOT}_{p_e} \otimes I \otimes I) \times (\text{AND2}_{p_e} \otimes \text{AND2}_{p_e}) \times (\text{OR2}_{p_e})$$

Each parenthesized term in these equations corresponds to a circuit level, and the overall PTM of a circuit is obtained by multiplying the PTMs of all its levels. For example, the first level L_1 of the majority circuit consists of three F_2 gates in parallel, so $L_1 = (F_2 \otimes F_2 \otimes F_2)$. Similarly, we obtain $L_2 = (I \otimes \text{swap} \otimes \text{swap} \otimes I)$, $L_3 = (\text{AND2}_{p_e} \otimes \text{AND2}_{p_e} \otimes \text{AND2}_{p_e})$, and $L_4 = (\text{OR3}_{p_e})$. The PTM for the majority gate is then $M_{\text{MAJ}} = L_1 \times L_2 \times L_3 \times L_4$. We use the same method to construct the PTM for the multiplexer.

Fully expanded to binary form, M_{MAJ} and M_{MUX} become 8×2 matrices, which we derived from the above equations with the aid of MATLAB [84]. The ITMs J_{MAJ} and J_{MUX}

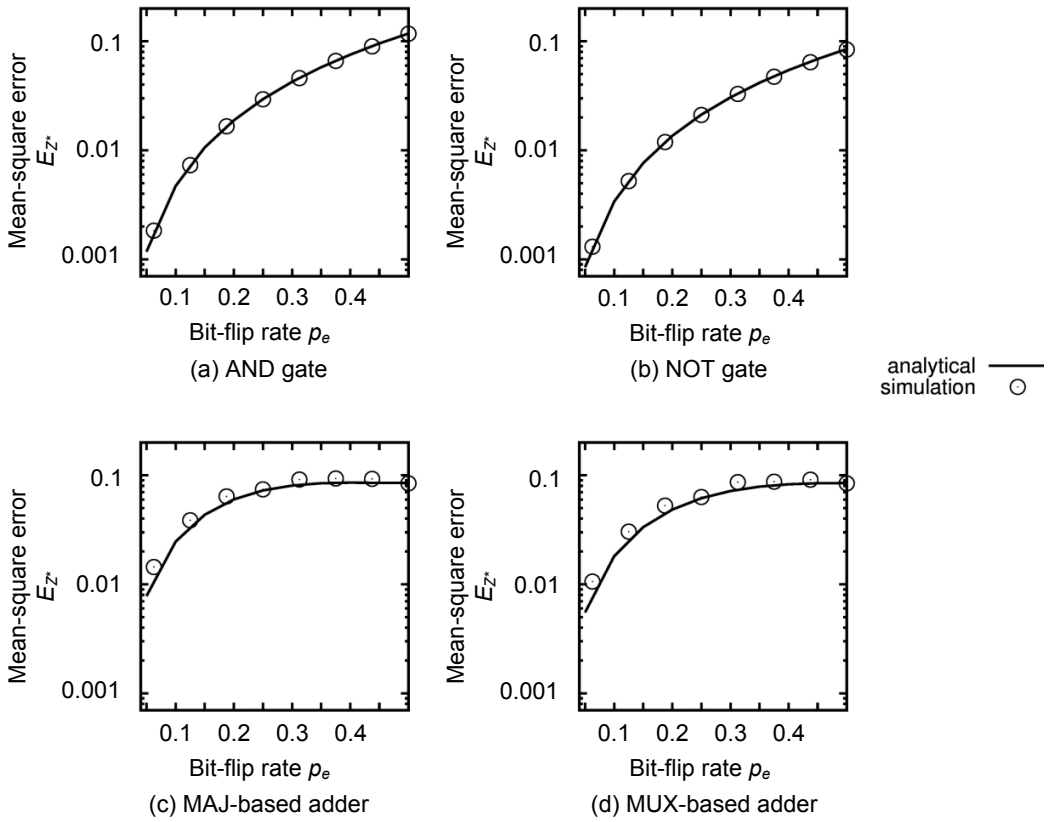


Figure 5.5: MSE at the outputs of representative stochastic circuits in the presence of soft-errors calculated using analytical and simulation methods.

for the two circuits, which have $p_e = 0$, take the form

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

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

When errors are present, the 01 entries of J_{MAJ} and J_{MUX} must be replaced by complex polynomial expressions involving the variable p_e to obtain J_{MAJ} and J_{MUX} in expanded form.

Knowing both the erroneous PTMs and the ITMs, we can calculate the corresponding MSEs; see Figure 5.5. 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.

5.4 Case Study: Image Edge Detection

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. In this section, we compare the error tolerance of two edge-detection circuits. The SC edge-detection circuit was introduced in the previous chapter. More details of its designs and that of a conventional edge-detection circuit will appear in the next chapter (see Figure 6.6).

We use PTMs to analyze the behavior of these circuits of 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 $M_{\text{Conv.}}$, 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 $M_{\text{in}} \times M_{\text{SC}}$ and $M'_{\text{in}} \times M_{\text{Conv.}}$. 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 5.6 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 relatively 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 5.5.

We have already discussed SC's inherent error tolerance, but for the current example, we give two more reasons as to why the SC circuit outperforms the conventional one. First, the conventional circuit is significantly bigger than the SC circuit and has more gates that are subject to noise. So at any point of time, the probability of seeing at least one bit-flip in the conventional circuit is higher than in the SC circuit. Second, the conventional circuit operates in only one (long) clock cycle. Should a bit-flip appear, the conventional circuit's output will become erroneous. On the other hand, the SC circuit distributes computation over multiple (short) clock cycles which increases the chance of error cancellation.

Figure 5.7 compares the output image quality of the two edge detectors of Figure 6.6 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. Note that this experiment is different from the noise injection experiments previously done in [109] and [3] where noise was only injected into the input

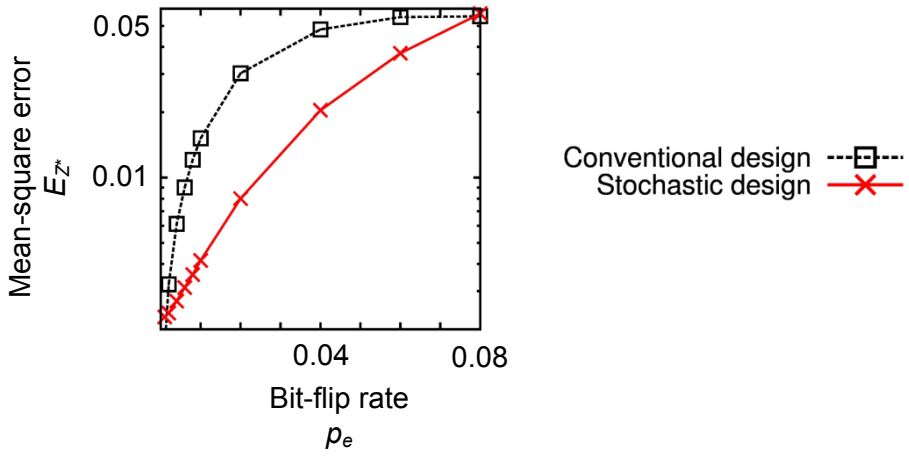


Figure 5.6: MSE of stochastic and conventional edge-detection circuits in the presence of soft-errors.

image signals and the image-processing circuitry was assumed to be error-free. In this later experiment, noise is injected into the stochastic circuits themselves to demonstrate their fault-tolerant behavior. 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.

5.5 Effect of Voltage/Frequency Scaling on Stochastic Circuits

In this section, we briefly discuss the effect of voltage/frequency scaling on SC circuits, and we show how the errors caused by aggressive scaling can be reduced. Voltage scaling, i.e., reducing the supply voltage of a circuit, reduces the circuit's energy consumption but increases its latency. If latency overhead is allowable by the application context, aggressive voltage scaling can be applied at the cost of occasionally erroneous results. Thus, voltage scaling allows designers to trade accuracy for energy. This approach has been extensively studied in the non-SC literature [53] [54] [60], and methods of tolerating and/or correcting timing errors have been proposed. However, the probability of timing violations

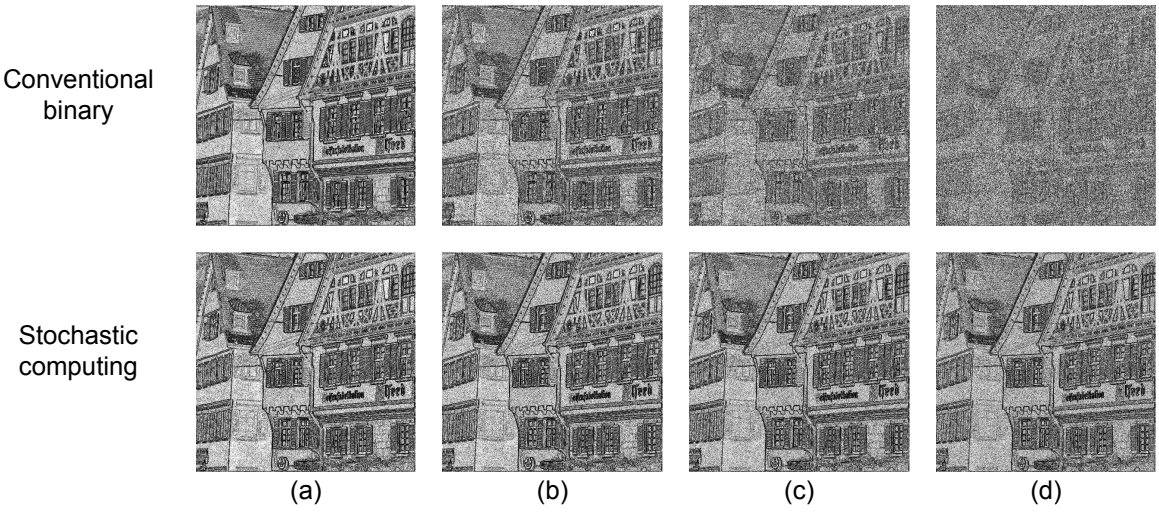


Figure 5.7: Comparison of stochastic and conventional edge detection for different soft-error rates (bit-flips percentages) in the edge-detection circuits: (a) 0.1%, (b) 0.5%, (c) 1% and (d) 2%.

increases rapidly with voltage scaling, necessitating complicated error-correcting methods. Frequency scaling is the process of overclocking a circuit in order to reduce its runtime at the cost of similar timing errors.

As demonstrated in this chapter, SC circuits can resist high rates of soft errors, but some level of error eventually appears in the output. However SC circuits' error tolerance against timing errors (caused by voltage/frequency scaling) is much higher. In other words, timing errors have very little effect on the operation of an SC circuit. To show the intuition behind this claim, consider an SC circuit e.g., a multiplier, where the bit-stream 001100100110 with value $5/12$ appears at its output. By applying aggressive voltage scaling, we can reduce the energy consumption of the circuit at the cost of timing errors that appear as transition delays; a 0-to-1 (1-to-0) transition does not make it in time to be captured at the end of the clock cycle. This means that a sequence 011 may appear as 001 in the output. Assuming an extreme case where *all* the transitions experience a one-clock-cycle delay, we get 000110010011 which also represents $5/12$; i.e., zero error.

More generally, timing errors cause the circuit to miss some of the 0-to-1 (or 1-to-0) transitions, leading to a $1/N$ change in the magnitude of Z. (N is the bit-stream length.) Since the numbers of 0-to-1 and 1-to-0 transitions are almost the same for any bit-stream, these timing errors tend to cancel each other out. This error cancellation is maximized if the rates of 0-to-1 and 1-to-0 errors are exactly the same. Figure 5.8 shows the average error on an SN for different rates of 0-to-1 and 1-to-0 timing errors. It can be seen that the error magnitude is reduced when the rates of delay errors are the same.

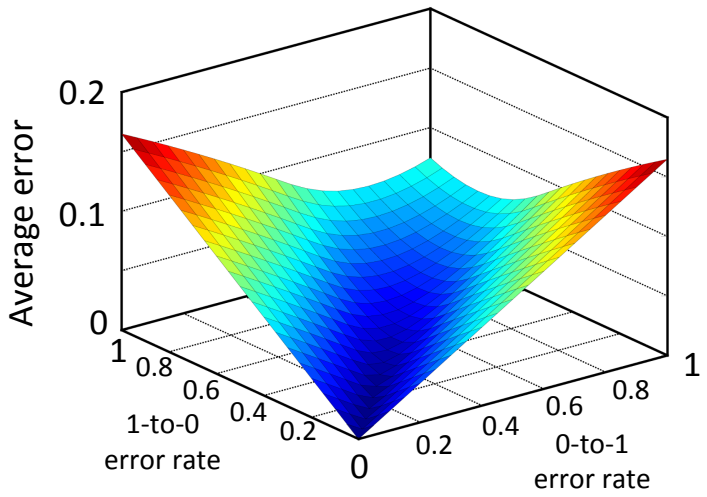


Figure 5.8: Error in the magnitude of an SN for different 0-to-1 and 1-to-0 timing error rates.

Voltage/frequency scaling adds new dimensions to the accuracy-energy tradeoff possibilities for SC circuits. As mentioned earlier, the length N of a stochastic computation controls the output accuracy. It also governs the total energy consumed by the circuit. Thus, by decreasing N, one can trade accuracy for energy or power savings. This natural tradeoff has been successfully used in the past [3]. By allowing voltage/frequency scaling, we provide SC circuits with three "control knobs"—(i) supply voltage V_{dd}, (ii) clock frequency f, and (iii) bit-stream length N—that control their accuracy and energy/power consumption. Finding the best combination of the knobs is thus a new problem.

In [9], we pose and answer the following question: "Given an SC circuit, what is the lowest energy required for computation with an average error goal of E_{goal}?" Previous methods search for the minimum N for which the average error is less than E_{goal}. However, as discussed, it is possible to adjust all three parameters (supply voltage V_{dd}, clock frequency f, and SN length N) concurrently in order to find the best answer to the question. We will refer to the triplet (V_{dd}, f, N) as an *operating point* of an SC circuit. Now we formalize the above question in the following problem statement.

Minimum-Energy Operating Point Problem. Given an SC circuit, find the operating point (V_{dd}, f, N) that has the minimum energy consumption, while satisfying the accuracy requirement of an average error less than E_{goal}.

We solve this problem using a straightforward search within the operating-point space. We try different operating points and, for each, we evaluate the accuracy and energy of the corresponding circuit using circuit simulations and statistical calculations. Then we will choose the point that has the lowest energy while satisfying the accuracy requirements.

As illustrated in Figure 5.8, the accuracy of SC circuits can be improved if the rate of 0-to-1 and 1-to-0 timing errors are balanced. This observation provides an opportunity to optimize SC circuits against timing errors. We saw that different paths of the circuit scale at a different rate when the supply voltage is reduced [9]. This causes the input signals to propagate unevenly and have misaligned arrival times at the output, which leads to unbalanced 0-to-1 and 1-to-0 error rates. To overcome this problem, we use logical and physical design techniques to balance the path delays.

Circuits with a chain structure (such as the one showed in Figure 3.4) tend to have unbalanced path delays. In contrast, circuits with a tree structure are more balanced and hence are more tolerant of timing errors induced voltage/frequency scaling. So the employed synthesis algorithm has an impact on the circuits ability to resist voltage scaling. For example,

circuits synthesized by Qian et al.'s ReSC [109] are more balanced than those synthesized by STRAUSS [8].

Circuit-level modifications such as gate sizing and buffer insertion can also balance the timing errors in SC circuits. For instance, buffers can be added to a path that is shorter than then rest. Similarly, wire detours can be added to adjust a path's delay. These modifications, however, increase the circuit's energy consumption so there is a tradeoff in using them, making it a difficult optimization problem. In [9], we formulated and solved this optimization problem for several representative circuits.

After optimizations, the representative SC circuits tolerated aggressive (up to 40%) voltage reduction with no significant loss of accuracy. Circuits with balanced structures, e.g., the edge-detection circuit introduced in the previous section, can tolerate voltage scaling without extensive optimization. This eliminates the overhead of gate sizing and buffer insertion and produces more efficient circuits.

A comparison between another optimized SC image processing circuit (gamma correction from [109]) and its conventional binary counterpart appears in Figure 5.9. Both implemented using the 28nm fully depleted silicon on insulator (FDSOI) technology. Once again, the SC circuit shows better tolerance against aggressive voltage scaling. In contrast, the binary circuits' output quality quickly drops, even with modest voltage changes. The error-tolerance of SC circuits enable significant energy savings. In the example of

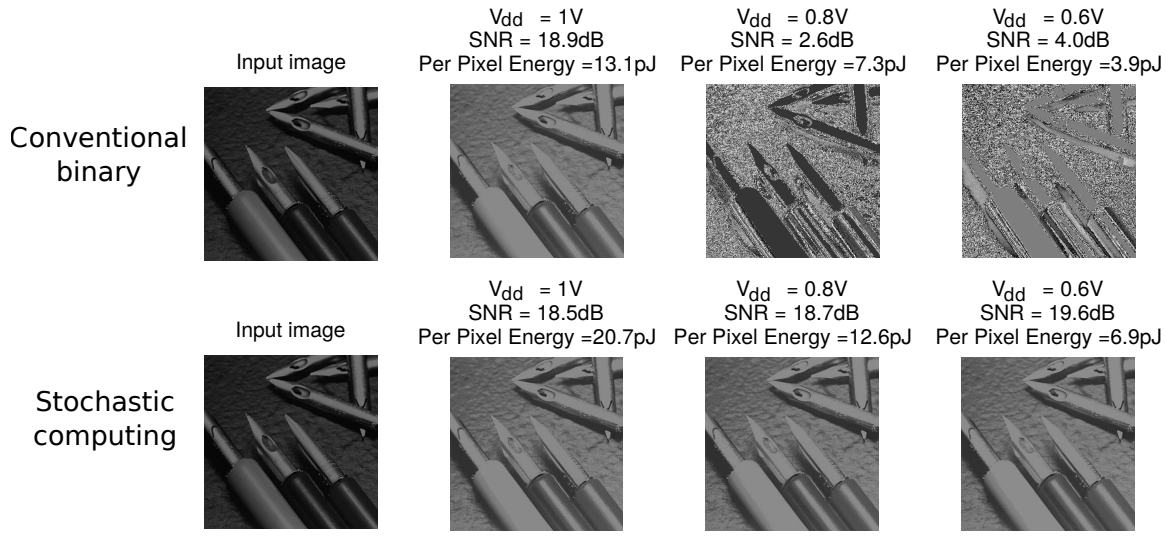


Figure 5.9: Voltage scaling results of gamma correction executed by conventional circuit and stochastic circuit.

Figure 5.9, the SC circuit at $V_{dd} = 0.6$V achieves the same accuracy as the conventional circuit at $V_{dd} = 1$V, while the SC circuit consumes about 44% less energy. Note that the signal-to-noise ratio (SNR) of SC is not monotonic when the voltages are scaled. This is because the random bit-stream sensitizes different paths when delays scale.

5.6 Summary

In this chapter we categorized the errors that affect SC circuits into four groups: (i) approximation and quantization, (ii) random fluctuation, (iii) insufficient randomness, and (iv) soft errors. We briefly discussed their causes and their possible solutions. We then focused on soft errors and presented methods of analyzing SC circuits in their existence. In particular, we showed how PTM tools can be used to evaluate the error behavior of SC circuits. We observed that SC circuits are error-tolerant and can outperform conventional binary circuits in noisy environments with high error rates. We also showed how SC circuits can be optimized to resist timing errors that are induced by voltage/frequency scaling. This allows us to reduce the supply voltage of the circuit, and hence reduce its energy consumption, without compromising its accuracy. In an image-processing case study, we obtained a threefold energy reduction with no significant accuracy loss.

Chapter 6

Stochastic Image Processing

This chapter discusses the application of stochastic computing (SC) to image processing. We present a stochastic vision chip that eliminates the heavy cost of conversion units by operating on sensor data directly. We also discuss how SC's features can be exploited in real-time image processing. We demonstrate the design of several representative image-processing circuits, including an efficient edge-detection circuit, and compare them to conventional binary counterparts. Finally, we show how progressive precision (PP) and be exploited and guaranteed in SC circuits. To show this, we employ low-discrepancy (LD) sequences that are commonly used in quasi-Monte Carlo (QMC) simulations. This chapter's material has been published in [3] and [5].

6.1 Vision Chip Overview

Vision chips are loosely classified as analog or digital (pulse domain), depending on the type of circuitry used in the preprocessing stage to convert the sensed analog input signals to digital form for final processing. Typical preprocessing circuits are analog-to-digital converters (ADCs), noise filters, and edge detectors [12] [45] [59]. These steps may require many operations per pixel, and consume most of the power of the system [135]. The design of vision chips is challenging since it involves complex trade-offs among chip area, power, speed and accuracy. It also requires some degree of parallel processing, which can be at the level of individual pixels, groups of pixels, or the overall system [135].

We propose to use SC in order to overcome the challenges associated with vision chips. SC and real-time image processing share some key properties. They both handle streaming analog data (image intensities or probabilities), process the data digitally, and have

good noise tolerance. Several proposed vision chips encode the sensed light signals using pulse-frequency modulation (PFM) [51] [59] [125]. This means that pixel information is conveyed by the frequency of a pulse train, as in biological neural networks and SC circuits. SC thus has the potential to meet most of the challenging requirements of the retinal implant application mentioned earlier: streaming neural-style data, small circuit size, extremely low power, and insensitivity to noise.

It has been shown that SC can outperform conventional binary in some processing tasks involving stored images [73] [109]. Ma et al. [78] demonstrate that SC is useful in fault-tolerant image processing. To apply SC to stored images, data conversion between the weighted-binary and stochastic domains is necessary, and as we showed in Chapter 2, the conversion is very costly. However, we will demonstrate here that this cost can be avoided in real-time image processing. The use of SC in real-time vision chips was briefly discussed by Hammadou et al. in 2003 [51], but otherwise has received very little attention.

Vision chips vary widely in how their sensors and processing circuits are laid out [87] [19]. In the simplest form, one processor handles all the pixels in series and no parallel processing occurs. At the other extreme, each pixel has a processing element (PE) of its own, providing maximum parallelism. Since conventional digital PEs can be large, this approach does not scale well [135]. For real-time applications, one processor per pixel is desirable and, as we show, is achievable using stochastic computing techniques.

We propose a vision chip with maximum parallelism using stochastic processing elements (SPEs) that are small and scale well. Our designs are also applicable to cases where processing circuits are shared among pixels. Figure 6.1 shows a high-level view of the proposed chip and its SPEs. For clarity, a 4×4 pixel array is shown, but it is possible to have many more pixels on chip. In addition to the SPEs, the chip has shared resources that manage random number generation and include a few counters based on LFSRs (linear feedback shift registers). The area cost of these resources is minor since they are small and their cost does not change with the pixel count.

As Figure 6.1 shows, vision chips have image sensors that convert the perceived light intensity to an analog electrical voltage. To enable digital processing, this analog signal must be converted to digital form using a conventional ADC or, in the SC case, an analog-to-stochastic converter. As noted earlier, the cost of an analog-to-stochastic converter is similar to that of a conventional ADC, which is depicted in Figure 6.2. In the conventional case, the analog voltage from the sensor is converted to a digital number through a comparison with a ramp voltage. The ramp voltage is generated by a counter and a digital-

to-analog converter (DAC). The analog comparator, fed by the sensor voltage and the ramp voltage, directly triggers a second counter which produces the desired digital output. In the SC case, the sensor voltage is converted to a stochastic number (SN) by comparing it to a random voltage generated by an LFSR-based counter and a DAC. An SN appears at the output of the comparator and can then be processed by an application-specific stochastic circuit, such as an edge detector. The second counter is used to convert the final result to weighted binary form. It should be clear from Figure 6.2 that analog-to-stochastic conversion imposes little overhead as it employs essentially the same ADC circuits found in any digital vision chip.

Although analog comparators are well understood, they still present some circuit design challenges; for instance, low-area comparators are susceptible to noise. It is feasible to place comparators of suitable quality and size at every pixel [34]. In conventional digital

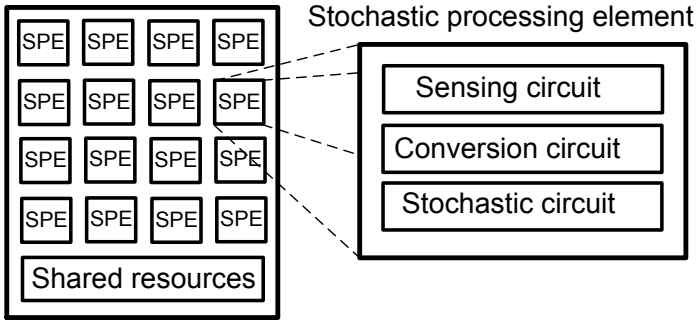


Figure 6.1: Top-level view of an SC-based vision chip and its stochastic processing elements (SPEs).

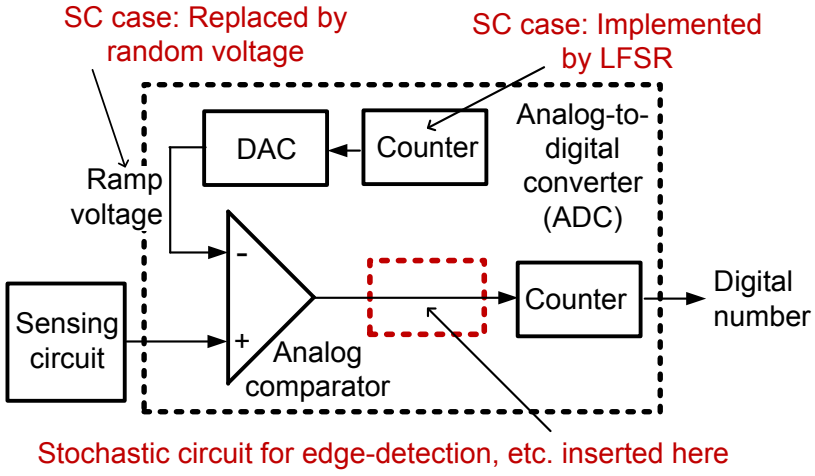


Figure 6.2: Conventional ADC circuit for a vision chip with the changes (in red) needed for analog-to-stochastic conversion.

image processors, a noise reduction step such as median filtering is needed [73]. In the proposed SC vision chip, however, the impact of noise is minimal thanks to the error tolerance of SNs, and a separate noise suppression step is unnecessary.

6.2 Image Processing Operations

This section discusses two basic image preprocessing categories, namely pixel-wise operations and windowing operations, examples of which are implemented later. We then present two ways to produce images with progressive quality improvement, which greatly speed up stochastic processing.

Pixel-wise operations modify a pixel's intensity value X independent of the values at other pixels. They typically implement a real-valued function $F(X)$ that adjusts intensity values. A well-known example is *gamma correction*, which is used to compensate for non-linearities in recording or display devices, or to increase pixel contrast [109]. One of the simpler gamma-correction functions is $F(X) = X^{0.45}$. To synthesize stochastic circuits that implement functions like $F(X)$, we use our STRAUSS method [8]. As we have seen in Chapter 3, this approach produces efficient circuits for a broad class of arithmetic functions.

A second category of image-processing operations of concern are windowing operations, where a weighted moving-average operation is performed on a small window of pixels, either to extract features of the image or to modify its quality. Examples of such operations are edge detection, sharpening and blurring [45]. The pixel windows are typically of size 2×2, 3×3, or 5×5. In order to design operations of this type, we mainly use the components of Figure 2.2. An m-to-1 multiplexer with a random select input performs averaging operations of the form $Z = \frac{1}{m}(X_1 + X_2 + ... + X_m)$; if negative weights are present, subtraction can be implemented by XOR or NOT gates.

Generating images that progressively improve is an important feature of image processing because it enables a trade-off between accuracy and computation that can be exploited in several ways. Image standards such as JPEG2000 [29] encode images of various qualities simultaneously. A conventional method of reducing the quality of an image is to reduce its number of pixels, i.e., its resolution. Figure 6.3 shows an image at several resolution levels; clearly, the quality diminishes as the resolution decreases.

Processing images with multiple resolutions imposes some computational overhead. However, we show by an example, that in the SC case, this overhead is minimal. Assume that a given image is to be processed at its original resolution, and at a lower-quality level

with 16 times less resolution. In the latter case, intensity signals from 16 neighboring pixels of the original image are averaged to produce a super-pixel.

Figure 6.4 shows this averaging process implemented by a 16-to-1 multiplexer. This circuit processes each input individually, and records its results in the corresponding counter. Meanwhile, it performs the same computation on the low-resolution super-pixel and records that result in a separate counter. As seen in the figure, the overhead of a super-pixel computation is only the additional counter.

As noted earlier, SNs have PP, meaning that short sub-sequences of an SN can provide low-precision estimates of its value. This property can also be used to obtain images of different qualities because we can have SN-encoded pixels with different precisions. This approach is orthogonal to the previous spatial-resolution method, and, since it is an inherent property of SC, it comes at essentially no cost. One simply uses the values appearing at the output counters of Figure 6.4 at successive points of time.

Figure 6.5 shows how this property can be exploited in image processing. An edge-detection operation is being performed on an image. The input image has 8 bits of precision

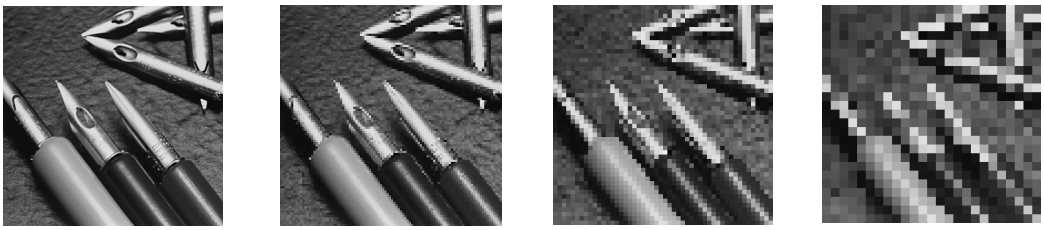


Figure 6.3: An image at four different resolution levels: (a) 400×400, (b) 100×100, (c) 50×50, and (d) 25×25 pixels.

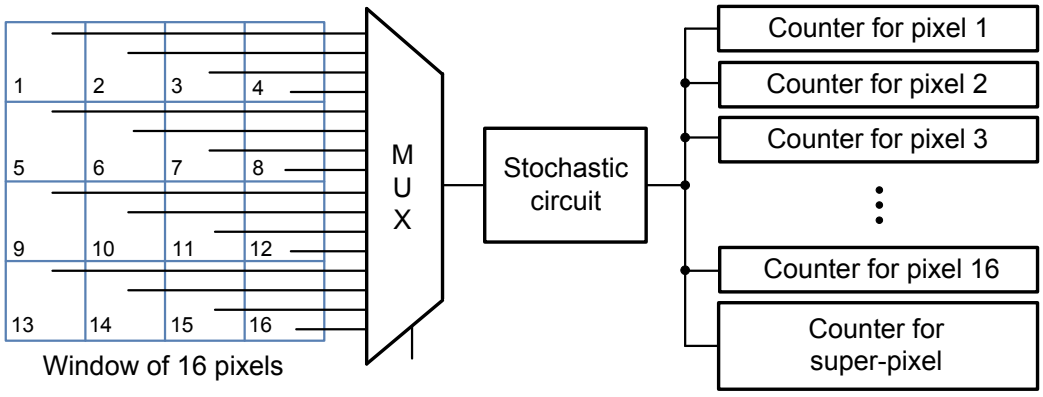


Figure 6.4: Stochastic processing of 16 pixels individually and as a super-pixel.

(the precision of an image corresponds to its gray-scale resolution [45]), and hence requires SN bit-streams of length 256. However, if the output image is checked at different points of time, it can be seen that as early as 4 clock cycles into the computation, many edges of the input image are detected, and after 32 clock cycles, almost all the edges are detected.

6.3 Implementations and Results

In this section, we demonstrate the design of image-processing circuits, including a high-efficiency edge detection circuit. Edge detection is useful in image processing and computer vision because it allows objects to be extracted from an image by highlighting their edges. As mentioned in Chapter 1, this can be useful in retinal implants because it generates high-contrast images of the environment that greatly help a vision-impaired person to navigate correctly and avoid obstacles.

Many edge-detection algorithms are known [45], and a few have been implemented with (non-real-time) SC [73]. Here we use the Roberts cross algorithm [45]. It computes a moving average of intensity values on a window of size 2×2 for each pixel $X_{i,j}$ at row i and column j of the image, and generates an output value $Z_{i,j}$ according to the following formula.

$$Z_{i,j} = \frac{1}{2}\big(|X_{i,j} - X_{i+1,j+1}| + |X_{i,j+1} - X_{i+1,j}|\big) \tag{6.1}$$

A stochastic implementation of this operation has been proposed by Li and Lilja [73], but it uses relatively large sequential circuits to approximate the absolute value function. Instead, we use the simple combinational SC components of Figure 2.2, which lead to a design that is more than 20 times smaller than that of [73], but results in similar (or even better) performance. Figure 6.6a shows the proposed stochastic circuit for edge detection. It uses just two XOR gates to implement the subtractions in Eq. (6.1) and a multiplexer to perform

Figure 6.5: Progressive precision results for edge detection: (a) input image; output image after (b) 4, (c) 32, and (d) 256 clock cycles.

the addition. As discussed in Section 4.3 (Table 4.3), the input SNs must be maximally correlated, i.e., they must have maximum overlap of 1's and 0's. This is assured by using a common random number source in their conversion circuitry. The multiplexer's select input is fed with a random input r, which is produced at minimal cost since it is shared among the SPEs. In contrast, the corresponding binary design (Figure 6.6b), assuming 8-bit precision, contains several big arithmetic units such as adders and subtracters.

As proof-of-concept, we implemented and validated the SC edge-detection circuit (and the other examples in this section) on a Xilinx Virtex-5 FPGA chip. Figure 6.7 illustrates the FPGA board (XUPV5-LX110T) used for our experiments, along with a representative image-processing task, namely, gamma correction. As seen in the figure, gamma correction corrects the brightness of the image and improves its visibility. It is important to note that the FPGA implementation was only used to verify the functionality of the circuits, and not for performance comparison. The output generated by the circuit is validated by comparing it to an expected output generated via conventional approaches.

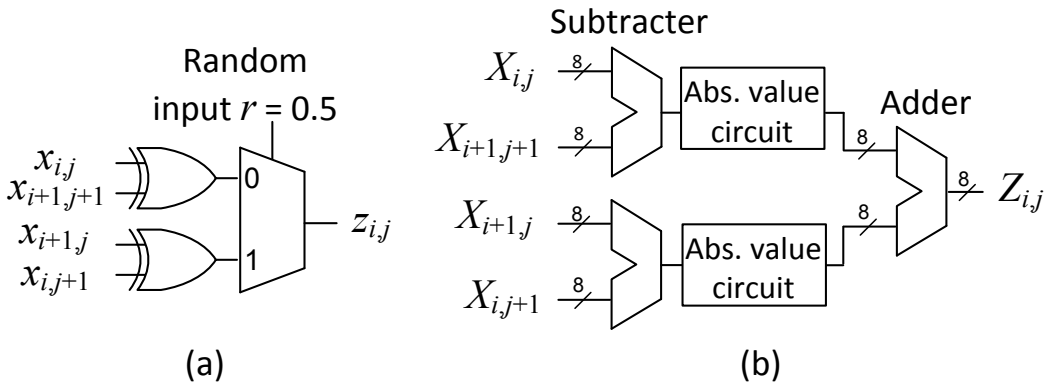


Figure 6.6: Edge detectors: (a) stochastic and (b) conventional designs.

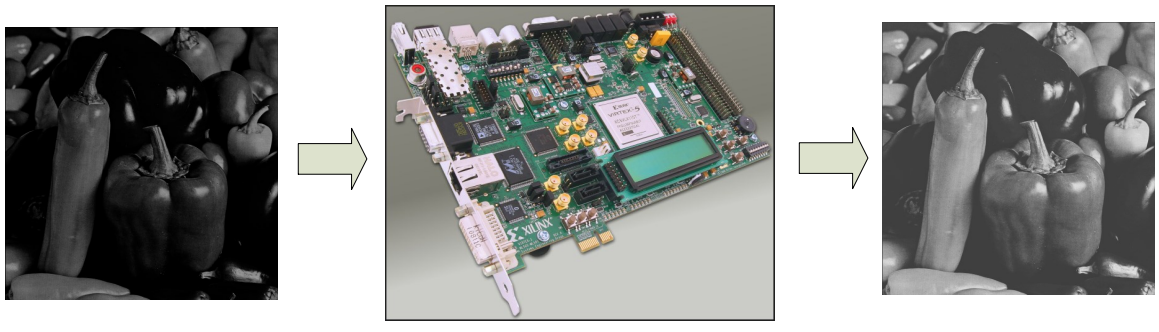


Figure 6.7: FPGA setup for emulating image-processing tasks, in this case, gamma correction.

Table 6.1: Synthesis results for the edge-detection circuits.

Implementation	Area (μm^2)	Delay (ns)	Power @20MHz (μW)	Energy (nJ)	Area $\times$ Delay ($\mu m^2 \times ps$)
Conventional weighted-binary	6928	19.49	7767.9	0.39	135.03
Previous SC design [73]	4312	1300	2213.7	28.34	5607.67
SC design proposed here	200	58.88	88.7	1.14	11.78

In order to compare the proposed edge-detection design with previous work, we used the SIS synthesis CAD tools [120]. Table 6.1 summarizes the results. All the numbers are estimated values based on a generic library of cells using 0.35μm CMOS technology. The delay of each circuit is obtained by multiplying the clock period by the number of cycles required to perform the operation. The conventional binary design shown in Figure 6.6b implements Eq. (6.1) in a single clock cycle. The SC designs, on the other hand, require 256 cycles for the full 8 bits of precision. The dynamic power consumption of the circuits are also estimated using the SIS tools [120].

The results reveal that the proposed edge-detection circuit is strictly better than the previous SC design [73]. Our SC design is about 30 times smaller than the conventional designs, and only 3 times slower. From the area-delay numbers, we see that the SC design has a significant cost advantage. The dynamic power consumption of each circuit is also reported in Table 6.1, which indicates that for a given clock frequency (in this case 20MHz), SC has significantly lower power consumption. However, since a stochastic circuit with fixed precision runs for a longer time, its energy consumption eventually becomes higher than that of a conventional design. We do not report the leakage power/ energy of these circuits, but since leakage power is directly proportional to area, we can conclude that the leakage power of the SC circuits is lower than the conventional case.

Also of interest is the performance of the edge-detection circuit when producing images with progressive quality improvement. The examples in Figure 6.5 suggest that the runtime of the edge-detection circuit can be further reduced (by a factor of 8) without compromising accuracy. This implies that edge detection requires less precision (than the original 8 bits), so for a fair comparison, we also implemented a low-precision version of the conventional edge-detection circuit. As can be seen from Table 6.2, the stochastic design is strictly better than the conventional one. Also, the stochastic edge-detection is so efficient that

Table 6.2: Synthesis results for low-precision edge-detection circuits.

Implementation	Area (μm^2)	Delay (ns)	Power @20MHz (μW)	Energy (nJ)	Area $\times$ Delay ($\mu m^2 \times ps$)
Conventional weighted-binary	4344	7.66	5156.8	0.26	33.28
SC design proposed here	200	7.36	88.7	0.14	1.47

can operate in real-time (15 frames per second) at 1nW power consumption. This number might be further reduced by switching to sub-threshold technologies.

Besides edge detectors, we designed several other stochastic image-processing circuits and evaluated their performance compared to alternative designs. We used STRAUSS [8] to obtain the gamma-correction circuit in Figure 6.8a. A flip-flop is used in this design in order to produce a second uncorrelated copy of the input bit-stream. The function implemented approximates the target function $Z_{i,j} = X_{i,j}^{0.45}$ but produces acceptable results, as seen in Figure 6.7. Figure 6.8b shows a conventional binary implementation of the same function. Like our edge-detection circuit, the stochastic gamma-correction circuit has a random input of probability 0.25 produced by a vision chip's shared resources.

Table 6.3 shows the synthesis results of our gamma-correction circuit, along with a conventional design of the same precision, and a previous SC design [109]. The design of [109] has better accuracy, but is much costlier. The table also includes synthesis results for two other image-processing tasks, namely, blurring and gradient calculation. These results are consistent with those we obtained for edge detection.

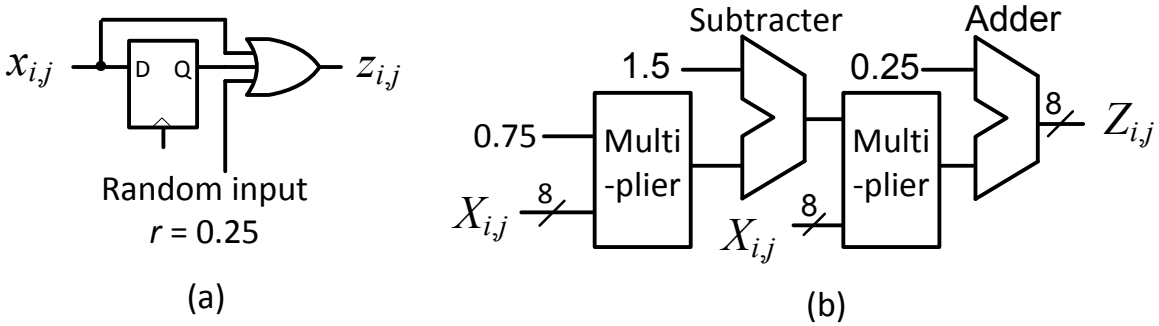


Figure 6.8: Gamma correctors: (a) stochastic and (b) conventional.

Table 6.3: Synthesis results for gamma-correction, blurring, and gradient calculation circuits.

Task	Design method	Area (μm^2)	Delay (ns)	Power @20MHz (μW)	Energy (nJ)	Area $\times$ Delay ($\mu m^2 \times ps$)
Gamma correction	Conventional binary	10576	27.2	21486	1.07	287.77
	Previous SC method [109]	1416	5365	970.3	49.68	7597.9
	This work	168	15.4	55.8	0.71	2.58
Blurring	Conventional binary	19464	32.6	13196	0.66	634.92
	This work	664	589	433.4	5.55	390.96
Gradient calculation	Conventional binary	1520	3.4	716.6	0.04	5.20
	This work	72	51.2	26.9	0.34	3.69

6.4 Guaranteeing Progressive Precision

Progressive precision (PP) was defined earlier as a property of SNs in which initial subsequences provide an estimation of the SNs' value. We also showed how it can lead to early edge detection (see Figure 6.5). This section analyzes this property more rigorously and presents methods of guaranteeing and exploiting it.

First, we will need to revisit the definitions of precision and accuracy. The *precision* of an SN X of length N may be defined as $\log_2 N$ bits, while the *accuracy* of X is its closeness to a target value denoted by p_X^*, which may be stated in terms of acceptable error bounds. To increase precision by 1 bit, an SN's length must be doubled. Accuracy targets, however, may demand even longer bit-streams. PP enables accuracy to be traded for speed: if the first $N' < N$ result bits of an N-bit stochastic computation provide a sufficiently good approximation to a desired result, the computation can be stopped early.

Technically, all SNs have PP to some extent. Some SNs are said to have "good" PP, implying that their first bits provide a good estimation of the final value, other have a "bad" (or poor) PP. For example, consider the following two SNs

$$X : 1010101010101110$$
$$Y : 1111111110000000$$

both representing the number $9/16$. X has good PP because its first 8 bits (10101010) represent $4/8 \simeq 9/16$. Y on the other hand has bad PP because its first 8 bits represent the number $1 \not\simeq 9/16$. Figure 6.9 shows how good and bad PP affect edge detection. SNs with good PP reveal edges early into the computation. In order to guarantee good PP, we need to quantify it. First, we introduce an error measure called bit-error.

Definition 6.1: Let X be an SN of length N with value p_X and let p_X^* denote its exact value. Then the *bit-error* of X is defined as $\varepsilon_X = N.|p_X - p_X^*|$.

The bit-error indicates how many bits p_X is away from p_X^* for the precision level corresponding to N. The exact value p_X^* is often known from the context. For instance, the output of a stochastic multiplier with inputs X and Y has the exact value $p_Z^* = p_X p_Y$, which serves to measure the output error ε_Z. We are now ready to define a measure to quantify PP.

Definition 6.2: An N-bit SN is k-PP if the bit-error of its initial sub-sequence of length 2^i is at most k for all i.

For example, let $X = 0111111111110000$ and let the exact value $p_X^* = 10/16$. The initial sub-sequences of length 2, 4, 8, and 16 are 01, 0111, 01111111, and 0111111111110000, respectively. The corresponding bit-errors of the sub-sequences are, from Definition 6.1: 0.25, 0.5, 2 and 1, respectively. This means that X is 2-PP because the

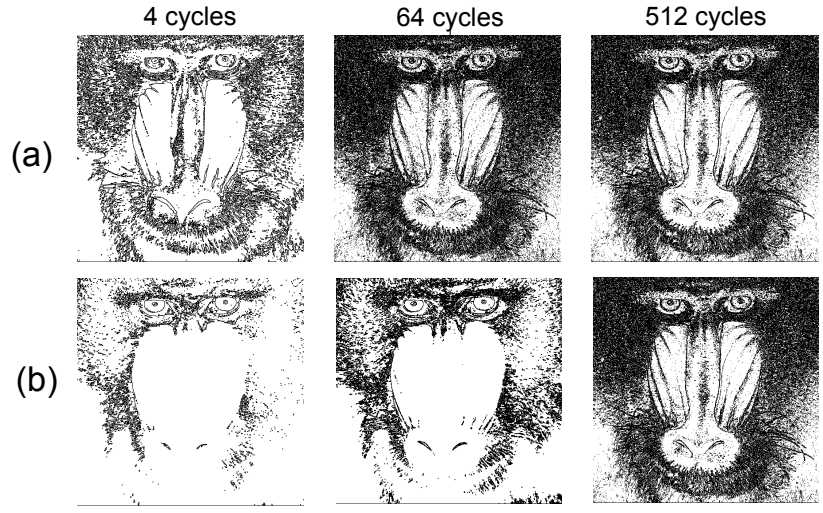


Figure 6.9: (a) Good PP and (b) bad PP in edge detection.

maximum bit-error of its initial sub-sequences is 2. The number $Y = 0111110111110000$, on the other hand, is 1-PP because the bit-errors of the initial subsequences have values 0.25, 0.5, 1 and 0, respectively. We say Y has better PP than X because the errors of its initial sub-sequences are lower.

A key insight of our approach is establishing a link between SC and Monte Carlo (MC). To illustrate this, we interpret an SC circuit as implicitly defining a small MC problem. Consider the circuit C in Figure 6.10 which is a stochastic multiplier supplied by two inputs X and Y derived from SNGs like that of Figure 2.3a. The MC problem defined by C is the following: given the input probability p_X and p_Y, find an estimate p_Z of the exact value $p_Z^* = p_X p_Y$ by applying independent uniform random samples to r_1 and r_2. The direct MC approach [52] to solving this problem is to generate N independent random samples at r_1 and r_2. It can be shown that the expected value of the estimated p_Z is indeed $p_Z^* = p_X p_Y$ and its variance is $p_Z^*(1-p_Z^*)/N$. The variance reflects the random fluctuations around p_Z^*, and implies that p_Z converges toward p_Z^* at the rate $O\left(1/\sqrt{N}\right)$.

The $O\left(1/\sqrt{N}\right)$ rate of convergence happens when random or pseudo-random sources (like LFSRs) are used in Figure 6.10. Unfortunately, the SNs generated by LFSRs have a relatively poor PP because they have uneven spacing of 1's and 0's [44]. In the example of Figure 6.10, LFSRs produce outputs with 20-PP when $N = 512$. To generate SNs with better PP, we introduce a new class of random number sources that produce *low-discrepancy* (LD) sequences, in which 1's and 0's are uniformly spaced, so they do not suffer from random fluctuations. They are widely used in quasi-Monte Carlo (QMC) sampling [94] [31], but have not been previously applied to SC. Other benefits of LD include deterministic error bounds and fast convergence. We present circuits employing LD sequences that are faster and more accurate than existing SC designs. For example, in image edge detection using LD sequences with good PP, as shown in Figure 6.9a, many edges are detected after

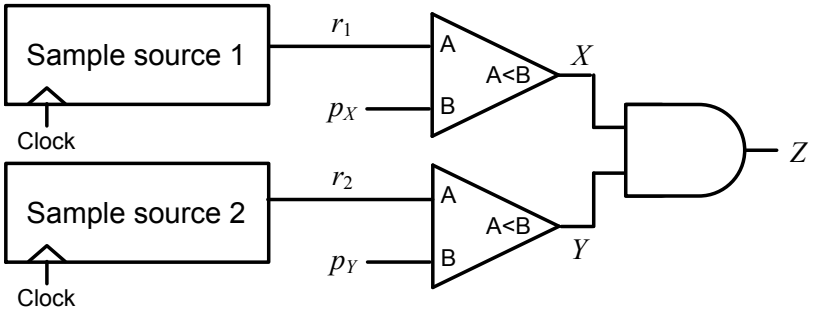


Figure 6.10: SC multiplication viewed as a Monte Carlo problem.

only 4 clock cycles, so the computation can stop early in most cases, as opposed to full-scale computation with 512 clock cycles. But if bit-streams with bad PP are used, we get the output edges shown in Figure 6.9b, where many edges remain undetected even after 64 clock cycles.

Figure 6.11a illustrates the PP behavior of three different types of SNs as they converge toward the target. The SNs generated from random and pseudo-random (LFSR-based) sample sets converge slowly and fluctuate a lot before settling at or near the target value. These are examples of bad PP. The SN generated using low-discrepancy sequences, on the other hand, quickly and monotonically converges to the target value. To further illustrate the convergence behavior of the different SNs, we plot the average error of several SNs as their length increases; see Figure 6.11b. The error of the LD-based SNs drops much faster than the others. This implies that the initial subsequences of the LD case provide a good early estimate of the target value, implying good PP.

To generate LD sequences in SC, we propose to use Halton sequences [94]. To obtain samples for a Halton sequence, the interval $[0, 1]$ is divided into b equal partitions, where b is a prime number. Then $b - 1$ samples are selected from the partitions, after which each partition is recursively divided into another b partitions and are sampled in a similar

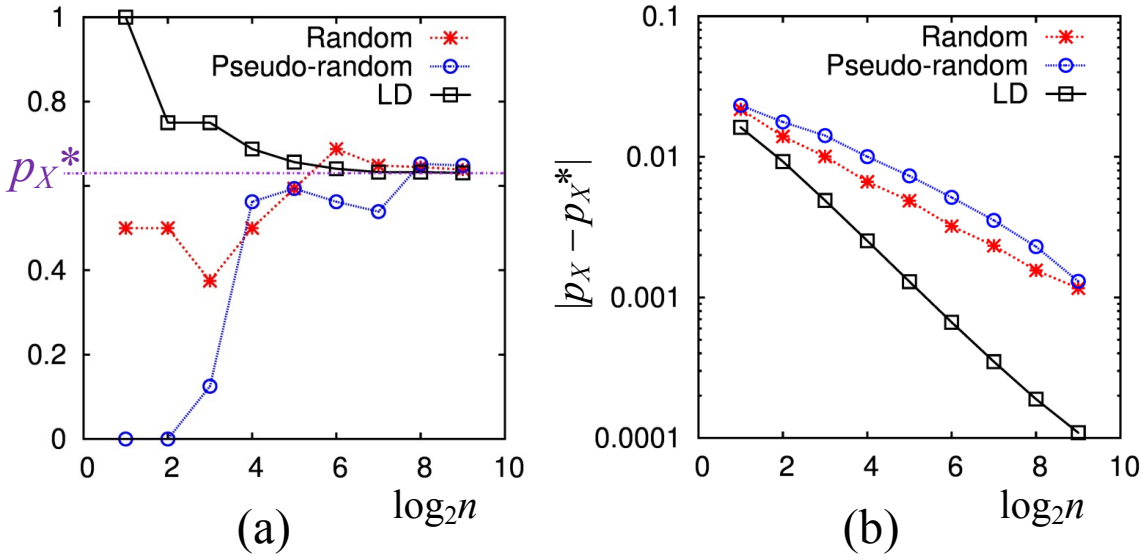


Figure 6.11: SN generation illustrating PP; (a) numerical value p_X for $p_X^* = 0.63$, and (b) average absolute error $|p_X - p_X^*|$ for all p_X^*'s.

fashion. The following sequence shows the first 8 samples of a Halton sequence with $b = 2$

$$\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}, \frac{1}{16}, \frac{9}{16}, \cdots$$

which is also known as the van der Corput sequence [94]. For $b = 3$ the first 8 samples are

$$\frac{1}{3}, \frac{2}{3}, \frac{1}{9}, \frac{4}{9}, \frac{7}{9}, \frac{2}{9}, \frac{5}{9}, \frac{8}{9}, \frac{1}{27}, \cdots$$

Halton sequences are among the less complicated LD sequences, and they show good performance for problems like those posed by stochastic circuits. Figure 6.12 shows the structure of our Halton sequence generator. It consists of a binary-coded base-b counter, where b is a prime number. The order of the output digits is reversed and the resulting number is converted to a binary number so that it fits into the SNG framework of Figure 2.2a. For example, for b = 3, the (binary- coded ternary) counter generates the sequence:

$$000, 001, 002, 010, 011, ..., 220, 221, 222$$

Then, the order of the digits is reversed thus:

$$000, 100, 200, 010, 110, ..., 022, 122, 222$$

(which requires no logic) and the reordered digits are converted to equivalent binary numbers. Each base-3 digit is converted using a digit converter, then the results are summed to generate the desired sequence of binary numbers

$$00000, 01011, 10101, ..., 10100, 11111$$

When $b = 2$, Figure 6.12 reduces to a simple binary counter. For n inputs, we need n copies of the circuit of Figure 6.12 with different prime bases. The 2-input circuit of Figure 6.10, requires two Halton sequence generators having $b = 2$ and 3. The output Z then becomes a 3-PP SN, in contrast with the 20-PP SNs produced by pseudo-random sequences.

So LD sequences can guarantee a good PP at the output. Now let us see how this good PP can be exploited in the example of Figure 6.10. Suppose that with the given inputs p_X

and p_Y, we are interested in the function

$$F_1(p_X, p_Y) = \begin{cases} 1 & \text{if } p_Z^* = p_X p_Y \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Assuming 8 bits of precision are required, a conventional non-stochastic design would use an 8-bit multiplier along with a comparator to implement the decision $p_Z^* \geq 0.5$. It would also perform the desired operation in one clock cycle. On the other hand, an SC implementation would use a circuit like that of Figure 6.10, where the AND gate implements multiplication. An additional circuit (such as a counter acting as a stochastic-to-binary converter and a comparator) would be needed to implement the decision. Using LFSR sequences, the SC design would be expected to need $N = 1,024$ clock cycles to produce a satisfactory result with an error rate of less than 1%.

Applying LD sequences to an SC multiplier produces a 3-PP output Z, so the initial sub-sequences of Z are at most 3 bits away from the target result. Suppose the initial subsequence of length $16 \ll N = 1,024$ is $Z_{16} = 0010000101001000$, denoting $p_{Z_{16}} = 4/16$, which means that $p_Z^* = 4/16 \pm 3/16 < 0.5$, and hence $F_1(p_X, p_Y) = 0$. In this case, the computation of F_1 can stop after 16 clock cycles. In general, the computation can stop after

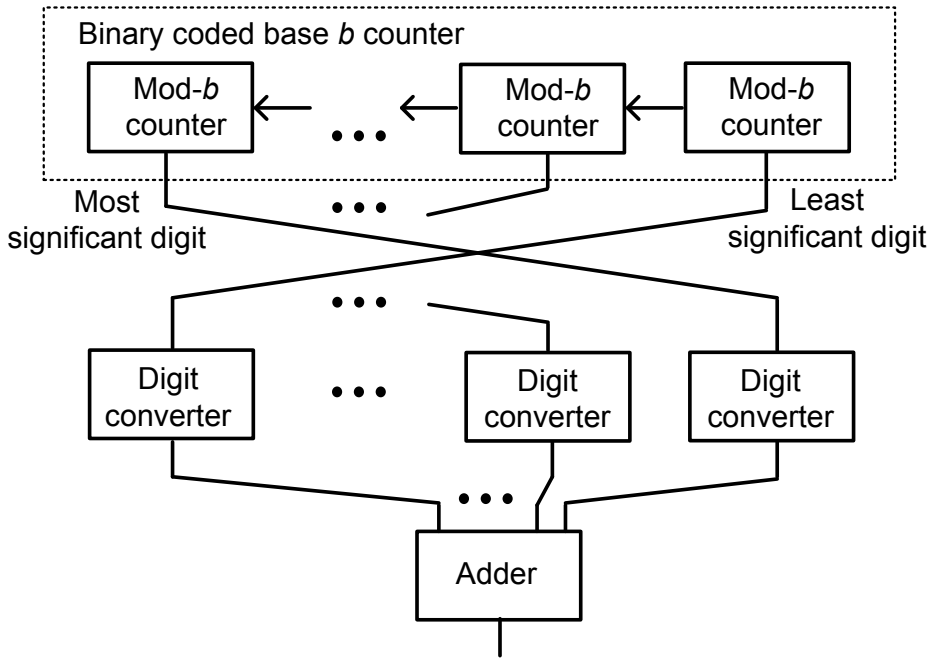


Figure 6.12: SN generation illustrating PP; (a) numerical value p_X for $p_X^* = 0.63$, and (b) average absolute error $|p_X - p_X^*|$ for all p_X^*'s.

N' clock cycles if $p_{Z_{N'}} > 0.5 + 3/N'$ or $p_{Z_{N'}} < 0.5 - 3/N'$. It turns out that for uniformly distributed values of p_X and p_Y, the average runtime per input of the design exploiting PP is only 59 cycles.

For uniformly distributed values of p_X and p_Y, nearly 40% of the input combinations lead to a computation that can be stopped after 16 clock cycles. These inputs correspond to the largest shaded region in Figure 6.13. Nearly 33% of the input combinations can be stopped after 32 clock cycles. It turns out that only 3% of the input combinations need a full-precision computation. The average runtime per input of the design exploiting PP is thus 59.31 cycles, a more than fivefold runtime reduction over existing SC designs. Applying the same approach to the edge-detection circuit of Section 6.3 we achieve a tenfold runtime reduction, making it even more efficient.

6.5 Summary

In this chapter, we have shown that SC is practical for real-time image processing. Complex image-processing tasks can be implemented with only a few gates, thus enabling on-chip massively parallel processing at the pixel level. We presented designs for SC image-processing circuits that outperform existing designs (both conventional and SC) in most aspects. In particular, we designed an edge-detection circuit that is strictly better than equivalent conventional designs. This highly efficient circuit seems ideal for vision chips

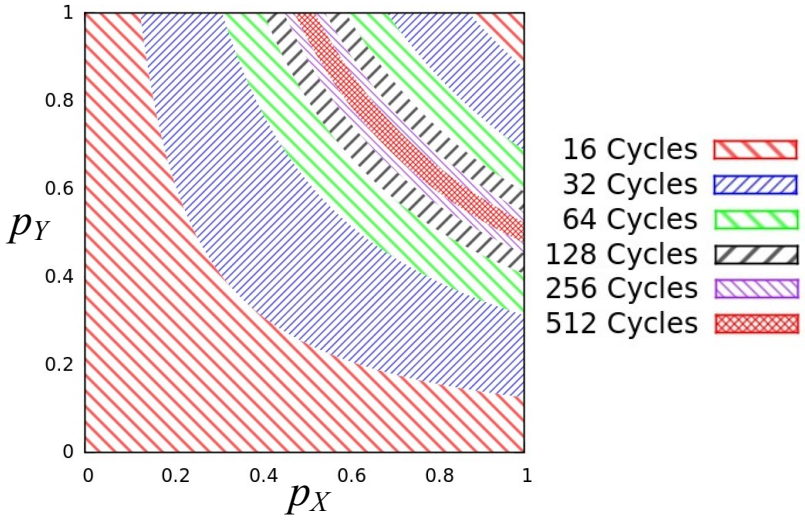


Figure 6.13: Input regions for which computation of F_1 can stop after 2^k clock cycles, for $3 < k < 9$.

and retinal implants. We also designed other representative image-processing circuits and made detailed comparisons with alternative implementations. The SC circuits are, in general, significantly smaller than conventional designs, and are more efficient in terms of power consumption and area-delay product. Furthermore, we demonstrated a method of exploiting PP in SC circuits by employing LD sequences that lead to fast output convergence. This leads to significant runtime reduction, and hence lower energy consumption in SC circuits.

Chapter 7

Concluding Remarks

The preceding chapters presented some of the major challenges posed by stochastic computing (SC) as well as our solutions to address them. We now summarize our contributions and point to several possible directions to extend this work.

7.1 Summary of Contributions

Our research started in 2010 with a survey of SC's history and applications. SC was first introduced in the 1960s as a hybrid analog-digital computing technique that enabled designers to implement complicated arithmetic functions with only a handful of transistors. But as the transistors became cheaper over time, this benefit faded away and circuits that operated on weighted binary numbers became more attractive. Even though the weighted binary circuits were less efficient in hardware cost and chip area, they were preferred over SC circuit due to their high speed and precision. For this reason, SC was only considered for a limited set of applications, such as neural networks.

With the development of modern mobile applications and embedded systems, IC design challenges have now shifted to the energy/power consumption of the circuits, because most of these circuits operate on batteries (or harvested energy). In some applications such as medical implants, other physical limitations, such as the heat amount generated by the circuits, define their power consumption. For these reasons, we concluded that SC should be revisited, because the area efficiency of SC circuits can make them a viable alternative to conventional binary circuits. In addition, SC's natural probabilistic behavior and error tolerance seems well-suited to nanoscale CMOS devices, as well as the emerging "beyond CMOS" nanotechnologies such as memristors [66], carbon nanotube field-effect transistors (CNTFETs) [122], and magnetic-tunnel junction device [97].

In our survey paper [2], we identified the main challenges of SC which became the focus of our subsequent research. First of all, SC lacked a comprehensive design method. Most of the previous designs were ad hoc, or based on pre-designed small components. Second, SC suffered from low precision, low accuracy and low speed, which limited the scope of SC's applications. In addition, circuits that convert numbers from/to SNs are costly, and if used extensively, they can defeat the purpose of employing SC. All of these challenges point to a bigger question: is there an application that (i) avoids the cost of conversion units, (ii) does not require high precision and speed, and (iii) can benefit from SC's good properties such as fault tolerance and area efficiency? Our research has addressed many of these challenges.

We introduced a general design method called STRAUSS (Spectral TRAnsform Use in Stochastic circuit Synthesis) which, as evident from its name, is based on spectral transforms. We proved that there is a key relation between the Fourier transform of a Boolean function (BF), and the stochastic function (SF) implemented by it: the Fourier transform of a BF (corresponding to a circuit C) produces a polynomial which is the SF implemented by C. This makes the Fourier transform an important tool because it allows us to directly extract the stochastic behavior of an arbitrary combinational circuit.

Furthermore, the Fourier transform is reversible, meaning that it preserves all the information of the original BF. Consequently, we can apply the inverse Fourier transform to a polynomial and obtain its corresponding BF. This became the basis of our STRAUSS design method. Given a target SF F, we approximate it with a polynomial P (if not in that form already), and then apply the inverse Fourier transform to P to obtain its corresponding BF f. We then use standard circuit optimization techniques to implement f. We showed the details of the process of arbitrary functions, and developed several optimization techniques such as sharing constant number generator circuits and using asymmetric polynomials. Finally, we showed that for several representative target functions, the circuits design using STRAUSS were significantly more efficient than those designed by previous methods. STRAUSS enables SC circuits to be included in standard (or approximate [117]) hardware-software co-design approaches.

We then focused on a major and poorly understood source of inaccuracy in SC circuits, namely, correlation. Avoiding correlation requires many independent stochastic number generators (SNGs) which are very costly. We identified and analyzed a class of BFs called correlation insensitive (CI), which are not affected by correlations among their input stochastic numbers (SNs). This property allows us to avoid the extra cost of SNGs

by sharing random number sources between the inputs. We showed that exploiting CI can significantly reduce circuit cost.

If a function is not CI, systematic correlation changes its functionality. For example, the SC AND-gate multiplier implements $Z = X$ instead of the expected $Z = X \times X = X^2$ when fed by two SNs with identical bit patterns. This change of functionality was usually regarded as inaccuracy and was generally avoided by producing independent (uncorrelated) SNs. However, we determined that the assumption "correlation leads to inaccuracy" is not always true. For instance, we showed that the XOR gate with BF $z = x \oplus y$ implements the SF $Z = X + Y - 2XY$ when fed by independent SNs, but it also implements $Z = |X - Y|$ (absolute-valued subtraction) when fed by suitably correlated inputs. The latter turns out to be a useful function in several applications. In fact, the case of XOR with correlated inputs has a double bonus, because not only does it provide a useful function, it also requires fewer random sources leading to lower overall cost.

Building on this idea, we extended our synthesis method (STRAUSS) to allow the design of circuits with correlated inputs. First, we introduced a new correlation measure to quantify the correlation between SNs. We found that standard correlation measures such as Pearson correlation [28] are not suited for SC because they impose constraints on the values of the SNs. Our correlation metric called SC correlation (SCC) measures the similarity of two SNs, without constraining their values. SCC adds a new dimension to the design space of stochastic circuits and allows efficient implementation of useful functions such as min/max functions, saturating addition, and an absolute-valued subtraction.

The contributions mentioned above helped us design efficient and accurate stochastic circuits. Based on them, we found and established an important application of SC to real-time image-processing for retinal implants. These are tiny circuits that have strict energy/power constraints. We designed several efficient image-processing circuit, including an edge-detection circuit that exploits most of the techniques discussed earlier, such as CI and progressive precision (PP). We also showed that costly SNGs can be avoided if SC circuits are used in pre-processing sensor data, because they can operate on stochastically encoded signals that exist within analog-to-digital converters (ADCs). We presented a method of guaranteeing PP SC circuits by employing low-discrepancy (LD) sequences instead of LFSR sequences. This slight modification leads to significant runtime reduction, and hence addresses one of the biggest challenges of SC.

Finally, we analyzed SC's robustness and error tolerance against various error types. We saw that SC circuits are tolerant of soft errors caused by cosmic radiations or manufactur-

ing variability. We also found that SC circuits tolerate errors induced by voltage/frequency overscaling techniques. Voltage/frequency overscaling is usually used to reduce the energy/power of a digital circuit at the cost of accuracy. The accuracy of conventional circuits quickly drops with scaling due to timing violations. However, SC's redundant encoding makes it robust against aggressive scaling. We showed that in a representative image-processing circuit, even a 40% voltage reduction—which saves energy by a factor of 3—leads to no loss of accuracy. The results suggest that SC has a great potential in energy/power-limited applications.

7.2 Future Directions

Despite the recent increased interest in SC, there are still many open research problems in the field. In this section, we highlight some of these problems, and point to directions that we believe are important.

Besides the early prototype machines and the few recently fabricated chips [37] [72], most current SC designs are verified via digital simulation or FPGA emulation, which hide important circuit-level details from the designers. So there is a great need for more fabricated SC systems for different applications to provide real-world data on their performance. In general, building big SC systems is still a major challenge. While the theory of designing small arithmetic circuits is fairly well developed, many problems still arise when we connect multiple SC circuits. These problems become more complicated when there is a feedback loop present, because the feedback may cause (auto-)correlation within the SNs and hence change the functionality of the circuit. For instance, the iterative decoding of LDPC codes requires multiple use of the same components, whose outputs are fed back to their inputs. In such cases, extra care must be taken when re-using SNs; in the worst case the SNs must be "decorrelated" or "re-randomized", imposing significant costs [126].

Storing SNs also becomes an issue in bigger systems such as instruction-set processors, as intermediate results are required to be stored often. In a conventional processor, intermediate results are efficiently stored in registers (or memory); the weighted binary encoding insures dense storage. However, storing SNs is very expensive as their length grows exponentially with their precision. Most existing SC circuits avoid storage either by converting the results to binary [109] (which is also costly) or by operating on a stream of data [3], both of which limit the application scope of SC. So to have SC processors, we

need to directly address the storage problem, and design efficient SC storage units. This would significantly expand the application scope of SC.

The recent interest in memristors [66] and magnetic-tunnel junction devices [97] as SN generators suggests opportunities to use them as storage units. For example, Onizawa et al. [97] propose an efficient analog-to-stochastic converter, which alongside an analog integrator, can be an SN storage unit. The integrator accumulates the SN pulses and converts them to an analog signal that is proportional to the rate of the pulses. The analog signal is then fed to the analog-to-stochastic converter for regeneration. Memristors may also be used in a similar way and are hence good candidates for SN storage.

Another direction to continue this line of research is to extend the discussed design methods (e.g., STRAUSS [8]) to sequential circuits. The existing sequential circuit design methods are restricted to very specific finite-state machine structures [74], as we saw in Chapter 2. In [4] we showed how arbitrary SC sequential circuits can be analyzed by converting their state transition function into a set of linear equations. The solution to this set of equations produces the SF implemented by the circuit. The reverse engineering problem, i.e., coming up with a set of sequential equations whose solution is a desired target function, is an interesting and non-trivial problem. The number of different equation sets possible rapidly increase with the number of inputs and the number of states in the circuit, making it impractical to perform exhaustive search. Saraf et al. [119] have taken important steps in generalizing the previous design methods, but their approach is still quite limited. Building on their work, we have shown that circuits with completely different state-machine structures can produce the same SF [7]. For examples, the circuits shown in Figure 7.1 implement the same SF $Z(X) = (2X - 2)/(X - 2)$. This observation suggests that we may be able to shrink the search space for a target set of equations by eliminating the equivalent sets.

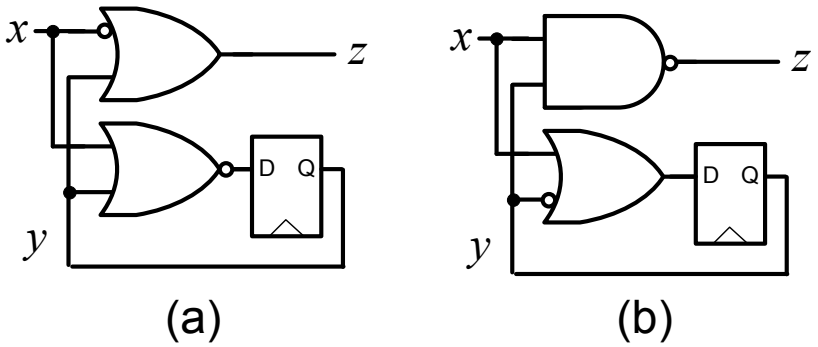


Figure 7.1: Two sequential circuits implementing $Z(X) = (2X - 2)/(X - 2)$.

SC applications used to be limited to a few specific domains, such as neural networks, image processing, and LDPC decoding. We showed that SC has a great promise in real-time image processing applications and medical implants. However, SC's application domain still remains small, compared to the domain of digital circuits. We believe that the theory and the techniques presented in this dissertation bring fresh perspective to the SC field. In particular, energy/power-constrained mobile embedded systems can greatly benefit from our methods. As we showed in Chapter 5, SC's error tolerance is a powerful property that can be exploited to reduce the energy consumption of the circuits. In short, we think that finding and establishing new SC applications is an interesting and promising research direction.

Finally, we emphasize that SC has a great potential in medical applications that involve computation on neural signals. The encoding methods of neural signals and SNs are similar, which has the potential to allow SC circuits to operate on neural signals directly. The methods and the concepts presented in this dissertation equip the IC designers with many cost-saving and error-reducing techniques, and thus enable the use of SC in medical applications that cannot be tackled efficiently with conventional approaches.

BIBLIOGRAPHY

[1] A. Alaghi and J. P. Hayes, "A spectral transform approach to stochastic circuits," *Proc. ICCD*, pp. 315–312, 2012. doi:10.1109/ICCD.2012.6378658

[2] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedded Computing Systems*, pp. 92:1–92:12, 2013. doi:10.1145/2465787.2465794

[3] A. Alaghi, C. Li and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," *Proc. DAC*, pp. 136:1–136:6, 2013. doi:10.1145/2463209.2488901

[4] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," *Proc. ICCD*, pp. 39–46, 2013. doi:10.1109/ICCD.2013.6657023

[5] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," *Proc. DATE*, pp. 76:1–76:4, 2014. doi:10.7873/DATE.2014.089

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

[7] A. Alaghi and J. P. Hayes, "On the functions realized by stochastic computing circuits," *Proc. Great Lakes Symp. VLSI*, pp. 311–336, 2015. doi:10.1145/2742060.2743758

[8] A. Alaghi and J. P. Hayes, "STRAUSS: spectral transform use in stochastic circuit synthesis," *IEEE Trans. on CAD*, 2015. doi:10.1109/TCAD.2015.2432138

[9] A. Alaghi et al., "Optimizing stochastic circuits for accuracy-energy tradeoffs," *Proc. ICCAD* (to appear), 2015.

[10] H. Aliee and H. R. Zarandi, "Fault tree analysis using stochastic logic: a reliable and high speed computing," *Proc. Reliability and Maintainability Symp.*, pp. 1–6, 2011. doi:10.1109/RAMS.2011.5754466

[11] R. Alt, J. L. Lamotte and S. Markov, "On the solution to numerical problems using stochastic arithmetic," *Proc. Symp. Scientific Computing, Computer Arithmetic and Validated Numerics*, p. 6, 2006. doi:10.1109/SCAN.2006.35

[12] F. Andoh et al., "A digital pixel image sensor for real-time readout," *IEEE Trans. Electron Dev.*, 47, 11, pp. 2123–2127, 2000. doi:10.1109/16.877174

[13] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Trans. Device and Materials Reliability*, 5, 3, pp. 305–316, 2005. doi:10.1109/TDMR.2005.853449

[14] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, 25, pp. 10–16, 2005. doi:10.1109/MM.2005.110

[15] D. Braendler, T. Hendtlass and P. O'Donoghue, "Deterministic bit-stream digital neurons," *IEEE Trans. Neural Networks*, 13, 6, pp. 1514–1525, 2002. doi:10.1109/TNN.2002.804284

[16] B. D. Brown and H. C. Card, "Stochastic neural computation. I. Computational elements," *IEEE Trans. Computers*, 50, 9, pp. 891–905, 2001. doi:10.1109/12.954505

[17] E. N. Brown et al., "Multiple neural spike train data analysis: state-of-the-art and future challenges," *Nature Neuroscience*, 7, pp. 456–461, 2004. doi:10.1038/nn1228

[18] V. Canals et al., "A new stochastic computing methodology for efficient neural network implementation," *IEEE Trans. Neural Networks and Learning Systems*, 2015. doi:10.1109/TNNLS.2015.2413754

[19] Centeye Inc., *Introduction to Current Centeye Vision Chips*, http://centeye.com/technology/vision-chips/, Feb. 2011.

[20] L. N. Chakrapani et al., "Ultra-efficient (embedded) SoC architectures based on probabilistic CMOS (PCMOS) technology," *Proc. DATE*, pp. 1–6, 2006. doi:10.1109/DATE.2006.243978

[21] H. Chen and J. Han, "Stochastic computational models for accurate reliability evaluation of logic circuits," *Proc. Great Lakes Symp. VLSI*, pp. 61–66, 2010. doi:10.1145/1785481.1785497

[22] T. H. Chen and J. P. Hayes, "Design of stochastic Viterbi decoders for convolutional codes," *Proc. DDECS*, pp. 66–71, 2013. doi:10.1109/DDECS.2013.6549790

[23] T. H. Chen, A. Alaghi and J. P. Hayes, "Behavior of stochastic circuits under severe error conditions," *it - Information Technology*, pp. 182–191, 2014. doi:10.1515/itit-2013-1042

[24] T. H. Chen and J. P. Hayes, "Analyzing and controlling accuracy in stochastic circuits," *Proc. ICCD*, pp. 367–373, 2014. doi:10.1109/ICCD.2014.6974707

[25] T. H. Chen and J. P. Hayes, "Equivalence among stochastic logic circuits and its application," *Proc. DAC*, pp. 131:1–131:6, 2015. doi:10.1145/2744769.2744837

[26] T. H. Chen and J. P. Hayes, "Design of stochastic Viterbi decoders for convolutional codes," *Proc. Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 66–71, 2013. doi:10.1109/DDECS.2013.6549790

[27] V. K. Chippa et al., "StoRM: a stochastic recognition and mining processor," *Proc. ISLPED*, pp. 39-44, 2014. doi:10.1145/2627369.2627645

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

[29] C. Christopoulos, A. Skodras and T. Ebrahimi, "The JPEG2000 still image coding system: an overview," *IEEE Trans. Consumer Electronics*, 46, 4, pp. 1103–1127, 2000. doi:10.1109/30.920468

[30] H. Chun, Y. Yang and T. Lehmann, "Safety ensuring retinal prosthesis with precise charge calance and low power consumption," *IEEE Trans. Biomedical Circuits and Systems*, vol. 8, no. 1, pp. 108–118, 2014. doi:10.1109/TBCAS.2013.2257171

[31] I.L. Dalal, D. Stefan and J. Harwayne-Gidansky, "Low discrepancy sequences for MC simulations on reconfigurable platforms," *Proc. ASAP*, pp. 108–113, 2008. doi:10.1109/ASAP.2008.4580163

[32] J. A. Dickson, R. D. McLeod and H. C. Card, "Stochastic arithmetic implementations of neural networks with in situ learning," *Proc. Intl. Conf. Neural Networks*, pp. 711–716. doi:10.1109/ICNN.1993.298642

[33] A. Dinu, M. N. Cirstea, and M. McCormick, "Stochastic implementation of motor controllers," *Proc. IEEE Symp. Industrial Electronics*, pp. 639–644, 2002. doi:10.1109/ISIE.2002.1026366

[34] P. Dudek and P. J. Hicks, "A general-purpose processor-per-pixel analog SIMD vision chip," *IEEE Trans. Ccts. and Sys. I*, 52, 1, pp. 13–20, 2005. doi:10.1109/TCSI.2004.840093

[35] H. Esmaeilzadeh et al., "Architecture support for disciplined approximate programming," *In Proc. ASPLOS*, pp. 301–312, 2012. doi:10.1145/2150976.2151008

[36] ETSI, *European Telecommunications Standards Institute Standard TR 102 376 V1.1.1: Digital Video Broadcasting (DVB). User guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications*, 2005. http://www.etsi.org.

[37] D. Fick et al., "Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2D edge detection and noise filtering," *Proc. CICC*, pp. 1–4, 2014. doi:10.1109/CICC.2014.6946130

[38] B. R. Gaines, "Stochastic computing," *Proc. AFIPS Spring Joint Computer Conf.*, pp. 149–156, 1967. doi:10.1145/1465482.1465505

[39] B. R. Gaines, "Stochastic computing systems," *Advances in Information Systems Science*, pp. 37–172, 1969. doi:10.1007/978-1-4899-5841-9_2

[40] J. Gal-Edd and C. C. Fatig, "L2-James webb space telescope operationally friendly environment?" *Proc. Aerospace Conf.*, pp. 105–110, 2004. doi:10.1109/AERO.2004.1367595

[41] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, 8, pp. 21–28, 1962. doi:10.1109/TIT.1962.1057683

[42] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, 39, 3, pp. 299–301, 2003. doi:10.1049/el:20030217

[43] S. W. Golomb, *Shift Register Sequences*, Aegean Park Press, Laguna Hills, CA, 1981.

[44] S.W. Golomb and G. Gong, *Signal Design for Good Correlation*, New York: Cambridge Univ. Press, 2004.

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

[46] D. Graham-Rowe, "A bionic eye comes to market," *MIT Technology Review*, March 2011. Retrieved July 2015.

[47] W. J. Gross, V. C. Gaudet and A. Milner, "Stochastic implementation of LDPC decoders," *Proc. Asilomar Conf. Signals, Systems and Computers*, pp. 713–717, 2005. doi:10.1109/ACSSC.2005.1599845

[48] S. Gupta and K. Gopalakrishnan, "Revisiting stochastic computation: approximate estimation of machine learning kernels," *Workshop on Approximate Computing Across the System Stack*, 2014.

[49] P. K. Gupta and R. Kumaresan, "Binary multiplication with PN sequences," *IEEE Trans. Acoustics, Speech and Signal Processing*, 36, 4, pp. 603–606, 1988. doi:10.1109/29.1564

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

[51] T. Hammadou et al., "A 96×64 intelligent digital pixel array with extended binary stochastic arithmetic," *Proc. ISCAS*, pp. IV:772–IV:775, 2003. doi:10.1109/ISCAS.2003.1206298

[52] J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods*. London: Methuen, 1964.

[53] K. He, A. Gerstlauer and M. Orshansky, "Low-energy signal processing using circuit-level timing-error acceptance," *Proc. ICICDT*, pp. 1–4, 2012. doi:10.1109/ICICDT.2012.6232873

[54] R. Hegde and N.R.Shanbhag, "Soft digital signal processing," *IEEE Trans. VLSI*, 9, 6, pp. 813–823, 2001. doi:10.1109/92.974895

[55] S. L. Hurst, D. M. Miller and J. C. Muzio, *Spectral Techniques in Digital Logic*, Academic Press, 1985.

[56] IEEE, *IEEE Standard 802.11n for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks*, 2009, *http://standards.ieee.org*

[57] P. Jeavons, D. A. Cohen and J. Shawe-Taylor, "Generating binary sequences for stochastic computing," *IEEE Trans. Information Theory*, 40, 3, pp. 716–720, 1994. doi:10.1109/18.335883

[58] Y. Ji et al., "A hardware implementation of a radial basis function neural network using stochastic logic," *Proc. DATE*, pp. 880–883, 2015. doi:: 10.7873/DATE.2015.0377

[59] K. Kagawa et al., "Pulse-domain digital image processing for vision chips employing low-voltage operation in deep-submicrometer technologies," *IEEE Jour. Sel. Topics in Quantum Electronics*, 10, 4, pp. 816–828, 2004. doi:10.1109/JSTQE.2004.833888

[60] A. B. Kahng et al., "Slack redistribution for graceful degradation under voltage overscaling," *In Proc. ASP-DAC*, pp. 825–831, 2010. doi:10.1109/ASPDAC.2010.5419690

[61] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," *Proc. DAC*, pp. 820–825, 2012. doi:10.1145/2228360.2228509

[62] M. G. Karpovsky, R. S. Stankovic and J. T. Astola , *Spectral Logic and its Applications for the Design of Digital Devices*, Wiley-Interscience, 2008.

[63] J. F. Keane and L. E. Atlas, "Impulses and stochastic arithmetic for signal processing," *Proc. Intl. Conf. on Acoustics, Speech and Signal Processing*, pp. 1257–1260, 2001. doi:10.1109/ICASSP.2001.941153

[64] S. K. Kelly et al., "A hermetic wireless subretinal neurostimulator for vision prostheses," *IEEE Trans. Biomedical Eng.*, 58, 11, pp. 3197–3205, 2011. doi:10.1109/TBME.2011.2165713

[65] Y. C. Kim and M. A. Shanblatt, "Architecture and statistical model of a pulse-mode digital multilayer neural network," *IEEE Trans. Neural Networks*, 6, 5, pp. 1109–1118, 1995. doi:10.1109/72.410355

[66] P. Knag et al., "A native stochastic computing architecture enabled by memristors," *IEEE Trans. Nanotech.*, 13, 2, pp. 283–293, 2014. doi:10.1109/TNANO.2014.2300342

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

[68] S. Krishnaswamy, I. L. Markov and J. P. Hayes, "Tracking uncertainty with probabilistic logic circuit testing," *IEEE Design and Test of Computers*, 24, 4, pp. 312–321, 2007. doi:10.1109/MDT.2007.146

[69] S. Krishnaswamy, G. F. Viamontes, I. L. Markov and J. P. Hayes, "Probabilistic transfer matrices in symbolic reliability analysis of logic circuits," *ACM Trans. Design Autom. Electron. Sys.*, 13, pp. 8:1–8:35, 2008. doi:10.1145/1297666.1297674

[70] F. R. Kschischang, B. J. Frey and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, 47, 2, pp. 498–519, 2001. doi:10.1109/18.910572

[71] R. Kuehnel, "Binomial logic: extending stochastic computing to high-bandwidth signals," *Proc. Asilomar Conf. Signals, Systems and Computers*, pp. 1089–1093, 2002. doi:10.1109/ACSSC.2002.1196952

[72] X. R. Lee, C. L. Chen, H. C. Chang and C. Y. Lee, "A 7.92 Gb/s 437.2 mW stochastic LDPC decoder chip for IEEE 802.15.3c applications," *IEEE Trans. Circuits and Systems I*, vol. 62, no. 2, pp. 507–516, 2015. doi:10.1109/TCSI.2014.2360331

[73] P. Li and D. J. Liljia, "Using stochastic computing to implement digital image processing algorithms," *Proc. ICCD*, pp. 154–161, 2011. doi:10.1109/ICCD.2011.6081391

[74] P. Li and D. J. Liljia, W. Qian, K. Bazargan and M. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," *Proc. ICCAD*, pp. 480–487, 2012. doi:10.1145/2429384.2429483

[75] X. Li, W. Qian, M. D. Riedel, K. Bazargan and D. J. Lilja, "A reconfigurable stochastic architecture for highly reliable computing," *Proc. Great Lakes Symp. VLSI*, pp. 315–320, 2009. doi:10.1145/1531542.1531615

[76] G. G. Lorentz, *Bernstein Polynomials 2nd Ed.*, Chelsea, New York, 1986.

[77] J. T. Ludwig, S. H. Nawab and A. P. Chandrakasan, "Low-power digital filtering using approximate processing," *IEEE JSSC*, 31, 3, pp. 395–400, 1996. doi:10.1109/4.494201

[78] C. Ma, S. Zhong and H. Dang, "High fault tolerant image processing system based on stochastic computing," *Proc. Intl. Conf. Computer Science and Service System*, pp. 1587–1590, 2012. doi:10.1109/CSSS.2012.397

[79] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronic Letters*, 33, 6, pp. 457-458, 1997. doi:10.1049/el:19970362

[80] B. J. MacLennan, "Analog computation," *Encyclopedia of Complexity and System Science*, Springer, pp. 271–294, 2009.

[81] R. Manohar, "Comparing stochastic and deterministic computing," *Computer Architecture Letters*, 2015. doi:10.1109/LCA.2015.2412553

[82] S. L. T. Marin, J. M. Q. Reboul and L. G. Franquelo, "Digital stochastic realization of complex analog controllers," *IEEE Trans. Industrial Electronics*, 49, 5, pp. 1101–1109, 2002. doi:10.1109/TIE.2002.803233

[83] P. Mars and H. R. McLean, "High-speed matrix inversion by stochastic computer," *Electronic Letters*, 12, 18 pp. 457–459, 1976. doi:10.1049/el:19760347

[84] MathWorks, Inc., *MATLAB and Statistics Toolbox Release 2012b*, Natick, Massachusetts.

[85] S.-J. Min, E.-W. Lee and S.-I. Chae, "A study on the stochastic computation using the ratio of one pulses and zero pulses," *Proc. ISCAS* 6, pp. 471–474, 1994. doi:10.1109/ISCAS.1994.409628

[86] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomput.* 74, 1-3, pp. 239–255, 2010. doi:10.1016/j.neucom.2010.03.021

[87] A. Moini, *Vision Chips*, Kluwer, 1999.

[88] W. Mokwa, "Retinal implants to restore vision in blind people," *Proc. Intl. Conf. Transducers*, pp. 2825–2830, 2011. doi:10.1109/TRANSDUCERS.2011.5969883

[89] B. Moons and M. Verhelst, "Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies," *IEEE Jour. Emerging and Selected Topics in Circuits and Systems*, 4, 4, pp. 475–486, 2014. doi:10.1109/JETCAS.2014.2361070

[90] A. Morro et al., "Ultra-fast data-mining hardware architecture based on stochastic computing," *PLoS ONE*, 10, 5, 2015. doi:10.1371/journal.pone.0124176

[91] A. Naderi, S. Mannor, M. Sawan and W. J. Gross, "Delayed stochastic decoding of LDPC codes," *IEEE Trans. Signal Processing*, 59, 11, pp. 5617–5626, 2011. doi:10.1109/TSP.2011.2163630

[92] M. H. Najafi and M. E. Salehi, "A fast fault-tolerant architecture for Sauvola local image thresholding algorithm using stochastic computing," *IEEE Trans. VLSI*, 2015. doi:10.1109/TVLSI.2015.2415932

[93] K. Nepal, R. I. Bahar, J. Mundy, W. R. Patterson and A. Zaslavsky, "Designing logic circuits for probabilistic computation in the presence of noise," *Proc. DAC*, pp. 485–490, 2005. doi:10.1109/DAC.2005.193858

[94] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, SIAM Series in App. Math., 32, Philadelphia, 1992. doi:10.1137/1.9781611970081

[95] R. O'Donnell, "Some topics in the analysis of Boolean functions," *Proc. ACM STOC Conf.*, pp. 569–578, 2008. doi:10.1145/1374376.1374458

[96] J. Ohta et al., "Large-scale integration-based stimulus electrodes for retinal prosthesis," *Artificial Sight*, Biological and Medical Physics, Biomedical Engineering, pp. 151–168, 2007. doi:10.1007/978-0-387-49331-2_8

[97] N. Onizawa et al., "Analog-to-stochastic converter using magnetic-tunnel junction devices," *Proc. NANOARCH*, pp. 59–64, 2014. doi:10.1109/NANOARCH.2014.6880490

[98] N. L. Opie et al., "Heating of the eye by a retinal prosthesis: modeling, cadaver and in vivo studies," *IEEE Trans. Biomed. Engin.*, 59, pp. 339–345, 2012. doi:10.1109/TBME.2011.2171961

[99] A. Paler, A. Alaghi, I. Polian and J. P. Hayes, "Tomographic testing and validation of probabilistic circuits," *Proc. ETS*, pp. 63–68, 2011. doi:10.1109/ETS.2011.43

[100] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Trans. Computers*, C-24, 6, pp. 668–670, 1975. doi:10.1109/T-C.1975.224279

[101] D. Pejic and V. Vujicic, "Accuracy limit of high precision stochastic watt-hour meter," *IEEE Trans. Instrum. Meas.*, 49, 3, pp. 617–620, 2000. doi:10.1109/19.850404

[102] E. M. Petriu et al., "Instrumentation applications of multibit random-data representation," *IEEE Trans. Instrum. Meas.*, 52, 1, pp. 175–181, 2003. doi:10.1109/TIM.2003.809492

[103] W. J. Poppelbaum, C. Afuso and J. W. Esch, "Stochastic computing elements and systems," *Proc. AFIPS Fall Joint Computer Conf.*, pp. 635–644, 1967. doi:10.1145/1465611.1465696

[104] W. J. Poppelbaum, "Statistical processors," *Advances in Computers*, pp. 187–230, 1976. doi:10.1016/S0065-2458(08)60452-0

[105] B. Pratt et al., "Fine-grain SEU mitigation for FPGAs using partial TMR," *IEEE Trans. Nuclear Science*, vol. 55, pp. 2274–2280, 2008. doi:10.1109/TNS.2008.2000852

[106] W. Qian and M. D, Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," *Proc. DAC*, pp. 648–653, 2008. doi:10.1145/1391469.1391636

[107] W. Qian and M. D. Riedel, "Two-level logic synthesis for probabilistic computation," *Intl. Workshop on Logic and Synthesis*, 2010.

[108] W. Qian and M. D. Riedel, "Uniform approximation and Bernstein polynomials with coefficients in the unit interval," *European Journal of Combinatorics*, 32, 3, pp. 448–463, 2011. doi:10.1016/j.ejc.2010.11.004

[109] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Computers*, 60, 1, pp. 93–105, 2011. doi:10.1109/TC.2010.202

[110] W. Qian et al., "Transforming probabilities with combinational logic," *IEEE TCAD*, 30, 9, pp. 1279–1292, 2011. doi:10.1109/TCAD.2011.2144630

[111] RAND Corp., *A Million Random Digits with 100,000 Normal Deviates*, Glencoe, IL: Free Press, 1955, Reprinted by RAND Corp. in 2001.

[112] T. R. N. Rao. *Error Coding for Arithmetic Processors*, Academic Press, Orlando, 1974.

[113] S. T. Ribeiro, "Random-pulse machines," *IEEE Trans. Electronic Computers*, EC-16, 3, pp. 261–276, 1967. doi:10.1109/PGEC.1967.264662

[114] R. Rojas, *Neural Networks, A Systematic Introduction*, Springer-Verlag, Berlin, New York, 1996.

[115] M. Samadi et al., "SAGE: self-tuning approximation for graphics engines," *Proc. MICRO*, pp. 13–24, 2013. doi:10.1145/2540708.2540711

[116] M. Samadi et al., "Paraprox: pattern-based approximation for data parallel applications," *SIGARCH Comput. Arch. News*, 42, 1, pp. 35–50, 2014. doi:10.1145/2654822.2541948

[117] A. Sampson, J. Bornholt and L. Ceze, "Hardware-software co-design: not just a cliché," *Summit on Advances in Programming Languages*, pp. 262–273, 2015. doi:10.4230/LIPIcs.SNAPL.2015.262

[118] K. Sanni et al., "FPGA implementation of a deep belief network architecture for character recognition using stochastic computation," *Proc. Conf. Information Sciences and Systems*, pp. 1–5, 2015. doi:10.1109/CISS.2015.7086904

[119] N. Saraf et al., "Stochastic functions using sequential logic," *Proc. ICCD*, pp. 507–510, 2013. doi:10.1109/ICCD.2013.6657094

[120] E. M. Sentovich et al., "SIS: A system for sequential circuit synthesis," *Univ. of California, Berkeley, Tech. Report*, UCB/ERL M92/41, Electronics Research Lab, 1992.

[121] N. R. Shanbhag, R. A. Abdallah, R. Kumar and D. L. Jones, "Stochastic computation," *Proc. DAC*, pp. 859–864, 2010. doi:10.1145/1837274.1837491

[122] M. M. Shulaker et al., "Carbon nanotube computer," *Nature*, vol. 501, pp. 526–530, 2013. doi:10.1038/nature12502

[123] A. Singhee and R. A. Rutenbar, "Why quasi-Monte Carlo is better than Monte Carlo or latin hypercube sampling for statistical circuit analysis," *IEEE Trans. CAD*, 29, 11, pp. 1763–1776, 2010. doi:10.1109/TCAD.2010.2062750

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

[125] F. Taherian and D. Asemani, "Design and implementation of digital image processing techniques in pulse-domain," *Proc. Asia Pacific Conf. Ccts. and Sys. (APCCAS)*, pp. 895–898, 2010. doi:10.1109/APCCAS.2010.5775031

[126] S. S. Tehrani et al., "Majority-based tracking forecast memories for stochastic LDPC decoding," *IEEE Trans. Signal Processing*, 58, 9, pp. 4883–4896, 2010. doi:10.1109/TSP.2010.2051434

[127] P. S. Ting and J. P. Hayes, "Stochastic logic realization of matrix operations," *Proc. Euromicro Conference Digital System Design*, pp. 356–364, 2014. doi:10.1109/DSD.2014.75

[128] M. S. Tomlinson Jr., D. J. Walker and M. A. Sivilotti, "A digital neural network architecture for VLSI," *Proc. Intl. Joint Conf. Neural Networks*, pp. 545–550, 1990. doi:10.1109/IJCNN.1990.137764

[129] S. L. Toral, J. M. Quero and L. G. Franquelo, "Stochastic pulse coded arithmetic," *Proc. ISCAS*, pp. 599–602, 2000. doi:10.1109/ISCAS.2000.857166

[130] M. van Daalen et al., "Device for generating binary sequences for stochastic computing," *Electronic Letters* 29, 1, pp. 80-81, 1993. doi:10.1049/el:19930052

[131] Various authors, *Proc. Intl. Symp. Stochastic Computing and its Applications*, Toulouse, 1978.

[132] R. Venkatesan et al., "Spintastic: spin-based stochastic logic for energy-efficient computing," *Proc. DATE*, pp. 1575–1578, 2015. doi:10.7873/DATE.2015.0460

[133] B. Vigoda, *Analog Logic: Continuous-Time Analog Circuits for Statistical Signal Processing*, Ph.D. dissertation, Massachusetts Institute of Technology, 2003.

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

[135] H. Yamashita and C. G. Sodini, "A CMOS imager with a programmable bit-serial column-parallel SIMD/MIMD processor," *IEEE Trans. Electron Dev.*, 56, 11, pp. 2534–2545, 2009. doi:10.1109/TED.2009.2030718

[136] C.-C. Yu, A. Alaghi and J. P. Hayes, "Scalable sampling methodology for logic simulation: reduced-ordered Monte Carlo," *Proc. ICCAD*, pp. 195–201, 2012. doi:10.1145/2429384.2429422

[137] B. Yuan and K. K. Parhi, " Reduced-latency LLR-based SC List Decoder for Polar Codes," *Proc. GLSVLSI*, pp. 107–110, 2015. doi:10.1145/2742060.2742108

[138] B. Zelkin, "Arithmetic unit using stochastic data processing," *U.S. Patent 6,745,219 B1*, 2004.

[139] D. Zhang and H. Li, "A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms," *IEEE Trans. Industrial Electronics*, 55, 2, pp. 551–561, 2008. doi:10.1109/TIE.2007.911946

[140] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "An efficient 10GBASE-T ethernet LDPC decoder design with low error floors," *IEEE JSSC*, 45, 4, pp. 843–855, 2010. doi:10.1109/JSSC.2010.2042255