

Warren J. Gross · Vincent C. Gaudet
Editors

Stochastic Computing: Techniques and Applications



Springer

Stochastic Computing: Techniques and Applications

Warren J. Gross • Vincent C. Gaudet
Editors

Stochastic Computing: Techniques and Applications

 Springer

Editors

Warren J. Gross
Department of Electrical
and Computer Engineering
McGill University
Montréal, QC, Canada

Vincent C. Gaudet
ECE Department
University of Waterloo
Waterloo, ON, Canada

ISBN 978-3-030-03729-1 ISBN 978-3-030-03730-7 (eBook)
<https://doi.org/10.1007/978-3-030-03730-7>

Library of Congress Control Number: 2018967965

© Springer Nature Switzerland AG 2019

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*To Amy, Ester, Margaret, and Katherine,
who inspire us at every step*

Foreword

It is a pleasure to wholeheartedly recommend to you the book before you. Professor Gross and Professor Gaudet are long-standing authorities in and contributors to the topic of Stochastic Computing. You will be richly rewarded in the intellectual journey they and their invited authors present here.

Stochastic Computing, as conceived by von Neumann in 1953, has a long distinguished body of research literature associated with it that continues today and is expertly reviewed by the invited authors. This is a fascinating, intellectually rich topic with wide-ranging applicability to a surprisingly large number of applications. And increasingly so, given advances in low-cost, highly accessible technology-binding options such as FPGAs are now commonly found in a range of environments from IoT devices to global cloud computing infrastructure.

My research group has had a long-standing research interest in the area of approximate computing. Many detection and estimation problems, decoders for error-correcting codes and machine learning systems, involving noisy observations, can often afford computational approximations that deliver acceptable compromises in overall system performance. The benefits to doing so are in the rewards delivered to other design variables important to production-quality hardware system realizations such as implementation area, testability, etc.

I first became aware of the topic of stochastic computing by reading the 1974 Ph.D. thesis of Gary Black (supervised by Prof. K.C. Smith) of the University of Toronto. The simplicity of the mathematics and its ready application to approximate analog computation was fascinating to me on first encounter. Translating analog variables into and out of the stochastic domain and expanding the types of operators was a challenge I presented to my former M.A.Sc. student, Ravi Ananth.¹ He explored an intriguing application of stochastic computing techniques to providing real-time monitoring of analog power supply state variables as encountered in IBM

¹Ravi Ananth, M.A.Sc. Thesis, University of Toronto, 1996 http://amicus.collectionscanada.gc.ca/s4-bin/Main/ItemDisplay?l=0&l_ef_=-1&id=22483.50643&v=1&lvl=1&coll=18&rt=1&itm=16138573&rsn=S_WWWhdabTK1KF&all=1&dt=AW+lananthl&spi=-&rp=1&v=1

mainframe computers and patented a number of techniques. Ravi made the research investigation a thrilling ride that has captured my continued ongoing interest in the subject matter over the years.

This area has developed both in depth of insight and in breadth of application due to the efforts of many researchers globally over the years, including many important contributions by Profs. Gross and Gaudet and their respective graduate students and research collaborators.

Now is an exciting time to start to reacquaint yourself with the latest developments in the field of stochastic computing, and this book is the best place to start.

Department of Electrical
and Computer Engineering
University of Toronto
Toronto, ON, Canada

Glenn Gulak

Preface

Stochastic computing has fascinated both of us for several years. After seeing many publications in the 1960s, the area largely fell silent until the early 2000s. Although we both had been introduced to the area by our doctoral advisor, Glenn Gulak, it really came into our consciousness after seeing a publication by Howard Card and one of his colleagues in a 2001 issue of the *IEEE Transactions on Computers*. Since then, stochastic computing has come back to life, especially in terms of its application to error-control decoding (and, more broadly, in belief propagation networks), image processing, and neural networks.

This manuscript represents an attempt to present a contemporary view of the field of stochastic computing, as seen from the perspective of its leading researchers. We wish to thank Kenneth C. Smith and Brian Gaines, both retired faculty members, who worked in the area during their illustrious careers and who have shared with us their memories of the previous boom in activity in stochastic computing in the 1960s.

There are three main parts to this book. The first part, comprising Chaps. 1 to 3, provides a history of the technical developments in stochastic computing and a tutorial overview of the field for both novice and seasoned stochastic computing researchers. In the second part, comprising Chaps. 4 to 9, we review both well-established and emerging design approaches for stochastic computing systems, with a focus on accuracy, correlation, sequence generation, and synthesis. The last part, comprising Chaps. 10 and 11, provides insights into applications in machine learning and error-control coding.

We hope you enjoy reading the book. May you discover new ways to make computation more efficient and effective and, above all, exciting!

Montréal, QC, Canada
Waterloo, ON, Canada
September 2018

Warren J. Gross
Vincent C. Gaudet

Contents

Introduction to Stochastic Computing	1
Vincent C. Gaudet, Warren J. Gross, and Kenneth C. Smith	
Origins of Stochastic Computing	13
Brian R. Gaines	
Tutorial on Stochastic Computing	39
Chris Winstead	
Accuracy and Correlation in Stochastic Computing	77
Armin Alaghi, Paishun Ting, Vincent T. Lee, and John P. Hayes	
Synthesis of Polynomial Functions	103
Marc Riedel and Weikang Qian	
Deterministic Approaches to Bitstream Computing	121
Marc Riedel	
Generating Stochastic Bitstreams	137
Hsuan Hsiao, Jason Anderson, and Yuko Hara-Azumi	
RRAM Solutions for Stochastic Computing	153
Phil Knag, Siddharth Gaba, Wei Lu, and Zhengya Zhang	
Spintronic Solutions for Stochastic Computing	165
Xiaotao Jia, You Wang, Zhe Huang, Yue Zhang, Jianlei Yang, Yuanzhuo Qu, Bruce F. Cockburn, Jie Han, and Weisheng Zhao	
Brain-Inspired Computing	185
Naoya Onizawa, Warren J. Gross, and Takahiro Hanyu	
Stochastic Decoding of Error-Correcting Codes	201
François Leduc-Primeau, Saied Hemati, Vincent C. Gaudet, and Warren J. Gross	

Contributors

Armin Alaghi University of Washington, Seattle, WA, USA

Jason Anderson The Edward S. Rogers Sr. Department of Electrical & Computer Engineering, University of Toronto, Toronto, ON, Canada

Bruce F. Cockburn The Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada

Siddharth Gaba University of Michigan, Ann Arbor, MI, USA

Brian R. Gaines University of Victoria, Victoria, BC, Canada

University of Calgary, Calgary, AB, Canada

Vincent C. Gaudet University of Waterloo, Waterloo, ON, Canada

Warren J. Gross McGill University, Montréal, QC, Canada

Jie Han The Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada

Takahiro Hanyu Tohoku University, Sendai, Miyagi, Japan

Yuko Hara-Azumi School of Engineering, Tokyo Institute of Technology, Tokyo, Japan

John P. Hayes University of Michigan, Ann Arbor, MI, USA

Saied Hemati Department of Electrical and Computer Engineering, University of Idaho, Moscow, ID, USA

Hsuan Hsiao The Edward S. Rogers Sr. Department of Electrical & Computer Engineering, University of Toronto, Toronto, ON, Canada

Zhe Huang Fert Beijing Institute, BDBC and School of Microelectronics, Beihang University, Beijing, China

Xiaotao Jia Fert Beijing Institute, BDBC and School of Microelectronics, Beihang University, Beijing, China

Phil Knag University of Michigan, Ann Arbor, MI, USA

François Leduc-Primeau École Polytechnique de Montréal, Montréal, QC, Canada

Vincent T. Lee University of Washington, Seattle, WA, USA

Wei Lu University of Michigan, Ann Arbor, MI, USA

Naoya Onizawa Tohoku University, Sendai, Miyagi, Japan

Weikang Qian Shanghai Jiao Tong University, Shanghai, China

Yuanzhuo Qu The Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada

Marc Riedel University of Minnesota, Minneapolis, MN, USA

Kenneth C. Smith University of Toronto, Toronto, ON, Canada

Paishun Ting University of Michigan, Ann Arbor, MI, USA

You Wang Fert Beijing Institute, BDBC and School of Microelectronics, Beihang University, Beijing, China

Chris Winstead Department of Electrical and Computer Engineering, UMC 4120, Utah State University, Logan, UT, USA

Jianlei Yang Fert Beijing Institute, BDBC and School of Computer Science and Engineering, Beihang University, Beijing, China

Yue Zhang Fert Beijing Institute, BDBC and School of Microelectronics, Beihang University, Beijing, China

Zhengya Zhang University of Michigan, Ann Arbor, MI, USA

Weisheng Zhao Fert Beijing Institute, BDBC and School of Microelectronics, Beihang University, Beijing, China

Abbreviations

A/D	Analog-to-Digital (Converter)
ASL	All-Spin Logic
B2IS	Binary-to-Integral Stochastic (Converter)
B2S	Binary-to-Stochastic (Converter)
BLSI	Brainware LSI
CMOS	Complementary Metal Oxide Semiconductor
CORDIC	COordinate Rotation DIgital Computer
D/A	Digital-to-Analog (Converter)
DBN	Deep Belief Network
DS	Delayed Stochastic
ECC	Error-Control Code
EM	Edge Memory
EPO	Energy per Operation
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
IM	Internal Memory
IT	Inferior Temporal
LD	Low Discrepancy (Sequence)
LDPC	Low-Density Parity-Check (Code)
LFSR	Linear Feedback Shift Register
LGN	Lateral Geniculate Nucleus
LLR	Log-Likelihood Ratio
LR	Likelihood Ratio
LSI	Large-Scale Integrated (Circuit)
LSZ	Least-Significant Zero
MIMO	Multiple-Input Multiple-Output
MNIST	Mixed National Institute of Standards and Technology (Data Set)
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MT	Middle Temporal
MTJ	Magnetic Tunnel Junction (Device)
MUX	Multiplexer

PCMOS	Probabilistic CMOS
PSOC	Probabilistic System-on-a-Chip
PUF	Physically Unclonable Function
PWM	Pulse-Width Modulation
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RHS	Relaxed Half-Stochastic
RMSE	Root-Mean-Square Error
RNG	Random Number Generator
RRAM	Resistive Random Access Memory
SC	Stochastic Computing (or Stochastic Computation)
SEU	Single Event Upset
SNG	Stochastic Number Generator
SNN	Spiking Neural Network
SPA	Sum-Product Algorithm
SPC	Single Parity Check (Code)
STT	Spin-Transfer-Torque (MTJ)
TFF	Toggle Flip-Flop
TPA	Throughput per Area
TRNG	True Random Number Generator
TSMC	Taiwan Semiconductor Manufacturing Company
VC	Voltage-Controlled (MTJ)
V1	Primary Visual Cortex

Introduction to Stochastic Computing



Vincent C. Gaudet, Warren J. Gross, and Kenneth C. Smith

Abstract In addition to opening the book, this chapter serves as a brief overview of the historical advances in stochastic computing. We highlight four distinct eras, including early work from the 1950s, fundamental work in the 1960s, an era of slower progress in the 1970s, 1980s, and 1990s, and a return to prominence in the 2000s.

Introduction

In addition to opening the book, this Chapter serves as an overview of the historical advances in stochastic computing. We highlight four distinct eras in its evolution: Section “The Early Years (1950s)” describes early work mainly led by John von Neumann in the 1950s; Section “Exciting Times (1960s)” describes the fundamental advances from the 1960s that form the basis for what we know today as stochastic computing; Section “Into the Darkness (1970s, 1980s and 1990s)” talks about the decline in stochastic computing research in the 1970s through 1990s and provides some insights into why this may have happened; then, Section “Rebirth (2000s and Beyond)” points to advances from the past 20 years that brought stochastic computing back to life. Finally, in Section “Overview of the Book” we provide a high-level overview of the rest of this textbook.

V. C. Gaudet (✉)
University of Waterloo, Waterloo, ON, Canada
e-mail: vcgaudet@uwaterloo.ca

W. J. Gross
McGill University, Montréal, QC, Canada
e-mail: warren.gross@mcgill.ca

K. C. Smith
University of Toronto, Toronto, ON, Canada

The Early Years (1950s)

We trace the origins of stochastic computing to a series of lectures delivered by John von Neumann at the California Institute of Technology in 1952 [1]. These lectures were carefully recorded by mathematician Richard S. Pierce, and the notes were later published by von Neumann as a chapter in a book edited by Claude Shannon in 1956 [2]. The lectures and paper describe several fundamental concepts such as using redundancy to deal with component error. This redundancy is inspired by biological/neural signaling models; it is based on bundles of wires likened to nerves, and is also motivated by a desire to gain a better understanding of the human brain. This bundling of wires is used as part of a mechanism to compute both addition and a variation on multiplication ($1-xy$). The addition circuit (Fig. 1) appears similar in form to the stochastic multiplexer often used for weighted summation, with the exception that it appears that the subset of wires taken from each input bundle is predetermined rather than randomly selected. The multiplication (Fig. 2) is based on the Sheffer stroke (represented as a circle labeled with an S and with two “bubbles” at the input—essentially a NAND gate), and is reminiscent of a stochastic multiplier. Note that the bundles are represented in space—in stochastic computing, the bundles are typically represented as sequences in time rather than in space (or a combination of both).

The work by von Neumann has been widely cited, but mainly in the context of performing accurate calculations using unreliable components (see, e.g., [3–5]); citations do not appear to expand upon the stochastic computing aspects. Accurate computing using unreliable components is still a very active field to this day, especially in the context of post-CMOS and nanoscale technologies. The aspect of von Neumann’s paper that is most closely associated with stochastic computing, namely, computing using randomly generated sequences of bits, does not re-appear in the open literature until 1967.

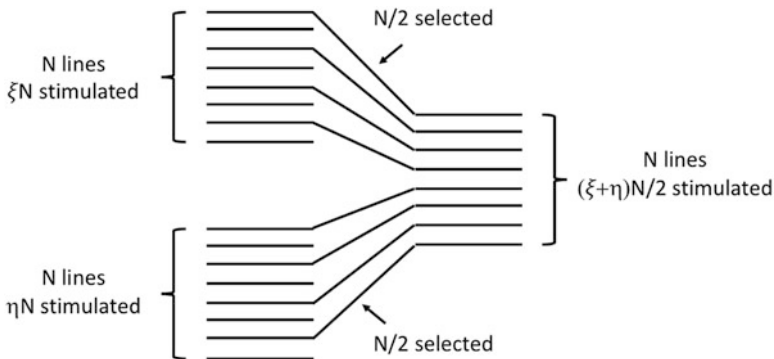


Fig. 1 Addition circuit proposed in [1], redrawn for clarity

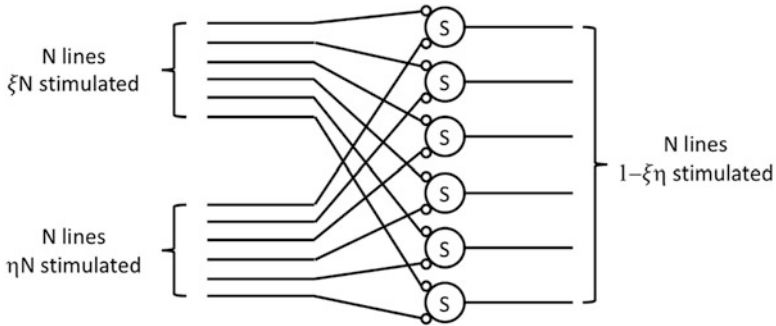


Fig. 2 Multiplication circuit proposed in [1] and based on the Sheffer stroke, redrawn for clarity

Exciting Times (1960s)

Many of the fundamental concepts of stochastic computing were developed in the 1960s. These efforts were led by several prolific individuals with interesting backgrounds and intertwined efforts. Their fascinating stories are described in the paragraphs below. Many of the recollections come from Kenneth C. Smith, who spent several years at the University of Illinois around that time. Also, a very special thank you goes to Brian Gaines for his insights—it was wonderful to meet one of the people whose work inspired our own!

W.J. (Ted) Poppelbaum

Wolfgang Johann (Ted) Poppelbaum (1924–1993) began an illustrious career as a faculty member at the leading-edge of computer-related circuits and systems at the Digital Computer Laboratory (DCL), University of Illinois, Urbana in 1955. This followed a postdoctoral year there in the solid-state research group under Professor John Bardeen, working on electrolytic analogs of junction transistors. From his early interest in mathematics and science, Poppelbaum had graduated with a Ph.D. in Physics (working on electrical and thermal conductivity in crystals) from the University of Lausanne in 1953. At DCL, he joined the thrust toward the creation of Illinois' third computer to be called Illiac II (preceded by ORDVAC and Illiac I), creating the Circuits and Systems Research Group. His overall success was based on being well read, with an unusual ability to recognize the importance and significance of new developments, conceiving new research directions involving basic components systems and applications, and performing early critical experiments to verify the integrity of his ideas. Correspondingly, he was renowned for his ability to obtain and deliver successful contract research.

This approach to discovery explains the timescale of Poppelbaum's interest in stochastic computing: While his first major publication in this area was in 1967

[6] sponsored by the Office of Naval Research (ONR), he had identified as early as 1963 the need in stochastic systems for multiple independent random-sequence generators and had done initial experiments using avalanche diodes for this purpose as performed by his graduate student Chushin Afuso [7] (working as a research assistant on topics unrelated to his thesis). Subsequently, Afuso realized that a system using a single random generator was possible, which he referred to as a Synchronous Random Pulse Sequence (SRPS) system (described in J.W. Esch's doctoral thesis [8]).

Chushin Afuso

Chushin Afuso (born in 1933) was an important contributor to Poppelbaum's research in several areas, including stochastic computing. Interestingly, Afuso, who was born in the Ryukyus (of which Okinawa is the largest island), embodies an ancient history of the area. While initially an independent Kingdom covered by both China and Japan, it took on a unique mixture of Chinese and Japanese cultures, as represented by Afuso's given (Chinese) name. In 1959, Afuso came from employment at the University of the Ryukyus to take a Master's degree at the University of Illinois under Poppelbaum, completing his degree in 1960, and then returning to the University of the Ryukyus, where he published a paper on difference amplifiers in a local venue [9]. Later in 1964, he returned for a Ph.D. under Poppelbaum, also working as a research assistant on a variety of topics including by 1965 initial work on a "noise computer" [7], and in 1967 contributing to the co-authored paper with Poppelbaum and Esch [6]. In 1968, he finished his PhD titled, "Analog Computing with Random Pulse Sequences" [10]. Upon graduation, he spent some time as an Assistant Professor at the University of Washington, St. Louis, returning to the University of the Ryukyus where he participated in a variety of areas including multiple-valued logic PLAs. In 1993, he co-authored a paper with Nagata on the topic of semiconductor testing at the International Symposium on Multiple-Valued Logic [11].

John William Esch

John William Esch (born in 1942) graduated from Electrical Engineering, University of Wisconsin, Madison, with a variety of experiences having worked as a programmer in the College of Letters and Science and as an electronics technician for the Space and Astronomy Laboratory, and later participated in a NASA grant with the Institute of Space Physics at Columbia University. In 1965, he began working on his Master's in Electrical Engineering at the University of Illinois with a research assistantship from the Department of Computer Science under Poppelbaum. In 1967, he received his Master's degree and then contributed to the

1967 paper with Poppelbaum and Afuso [6]. In 1969, he completed his Ph.D. thesis titled “RASCEL – A Programmable Analog Computer Based on a Regular Array of Stochastic Computing Element Logic” [8], which described an elaborate hardware implementation of a stochastic computer. Subsequently, he was employed by Sperry then Unisys in the software area.

Sergio Telles Ribeiro

Sergio Telles Ribeiro (born in 1933) was born in Brazil, completed his Master’s degree at the University of Illinois in 1961 on BJT saturation, and his Ph.D. titled “Phase Plane Theory of Transistor Bistable Circuits” [12] under Poppelbaum in 1963. He notably published a paper on pulsed-data hybrid computers in 1964 [13]. By the time of his publication on “Random-Pulse Machines” in 1967 [14], he was working for Carson Laboratories, Bristol, Connecticut. In his 1967 paper [14], an intriguing footnote indicates that in 1963 he had recognized the potential importance of stochastic computing as described by von Neumann in 1956 [2], and encouraged the Poppelbaum group to pursue research in the area, reporting Afuso’s work on stochastic generators documented in a 1965 progress report.

Brian Gaines

Brian R. Gaines (1938-) began his technical career as a polymath at the age of 12, fascinated by science including math, physics, chemistry, and philosophy. Fortunately, his father did not share the same passion and requested a departure from chemistry. Thus, Brian converted to an intense interest in electronics, building various measuring instruments including his own oscilloscope. This hobby led to early employment following high school at the Standard Telephone Cables Ltd. (STC) in the semiconductor laboratory, a connection which he maintained for many years. Subsequently, during his time at Cambridge taking math, his talents in electronics allowed him to work as an electronics assistant for a professor working in applied psychology. Intrigued with his work in cognitive psychology, the issue of pursuing an advanced degree in psychology arose, resulting in a requirement for an appropriate first degree which he subsequently pursued. Later, with all of his degrees from Cambridge, and as a chartered engineer and chartered psychologist, he led a diverse career including chairmanship of Electrical Engineering Science at Essex, Professor of Industrial Engineering at the University of Toronto, and finally Killam Memorial Research Professor at the University of Calgary, from which he retired in the late 1990s to Vancouver Island. Along the way, Gaines’ work in human learning and the potential for machine intelligence led to his interest in what became stochastic computing, publishing many of the seminal works [15–18].

Reed Lawlor

In 1971 Reed C. Lawlor (1909–1990) was granted a patent titled “Computer Utilizing Random Pulse Trains” [19] which was applied for in 1968 using two primary references: one of the papers by Ribeiro [14], and one by G. White titled “The Generation of Random-Time Pulses at an Accurately Known Mean Rate and Having a Nearly Perfect Poisson Distribution” [20]. Interestingly, Lawlor was a patent attorney in Los Angeles, CA with diverse interests including the role of computing in the application of the law. He is particularly renowned for a paper written 1963 on potential roles for computing in the law in which a particular concern of what would become artificial intelligence (AI) and machine learning (ML) [21]. Indeed, in 2016 a Scottish technology futures publication [22] quoted the following statement from Lawlor’s 1963 paper: “Computer technology is the cornerstone of the second Industrial Revolution. Will computers revolutionize the practice of law and the administration of justice, as they will almost everything else.” Lawlor had received a degree in physics from the University of California, Los Angeles in 1932, followed by his LLB degree from Blackstone College of Law in Chicago. As a practicing patent attorney in Los Angeles, he was an active member of the California Bar Association with many advisory roles, including Chairman of the Association’s Special Committee on Electronic Data Retrieval, principal investigator of an NSF supported grant at UCLA on technological aids for legal research, as well as contribution to many books on law and technology. It is unclear why the patent was applied for, nor how it was used.

1967: A Major Milestone in Stochastic Computing

1967 was an unusual year! Besides being the 100th anniversary of the founding of Canada (marked by hosting a hugely successful world fair called Expo 67 in Montreal), it was also a milestone in the publication of papers in stochastic computing.

Brian Gaines (1938-) born in the United Kingdom, working at Standard Telecommunication Laboratories, published “Stochastic Computing”; Ted Poppelbaum (1924–1993) born in Germany, working at the University of Illinois, Urbana, published (with Afuso and Esch) “Stochastic Computing Elements and Systems”; Sergio Telles Ribeiro (1933-) born in Brazil, working at Carson Laboratories, Connecticut, published “Random-Pulse Machines”. While Gaines’ and Poppelbaum’s work was done independently, Ribeiro obtained his Ph.D. under Poppelbaum in 1963, on a different topic (phase plane theory of transistor bistable circuits), but soon started publishing in related topics.

Into the Darkness (1970s, 1980s and 1990s)

The 1970s started off well for stochastic computing. Poppelbaum continued his work in the area [23, 24]. Kenneth C. Smith, by then at the University of Toronto, supervised Gary Black's doctoral thesis on random pulse-density modulation [25]. There was also some discussion at the International Symposium on Multiple-Valued Logic (ISMVL), including a panel that included Gaines, Poppelbaum, and Smith at the 1976 ISMVL in Logan, Utah [26]. Finally, a dedicated conference on stochastic computing and its applications was held in 1978 in Toulouse, France [27]; it appears to have been well-received, with a 420-page conference record. However, it was to be the only conference on stochastic computing to be held until 2016.

Progress on stochastic computing became more sporadic in the late 1970s and 1980s. In many ways, the continued progress in semiconductor scaling and high performance of these technologies gave more traditional digital approaches a competitive edge over stochastic computing.

However, by the 1990s, there appeared to be a slight upsurge in interest, fueled by the advent of field-programmable gate arrays (FPGAs). In his Master's thesis completed under Glenn Gulak at the University of Toronto, Ravi Ananth proposes field-programmable stochastic processors [28]. There was also work on sequence generation [29]. The 1990s also saw significant interest in efficient hardware implementations of artificial neural networks. There were several papers on digit-serial approaches including pulse-width modulation (PWM) [30] and pulse-density modulation (PDM) [31–33]. The latter can be seen as a form of stochastic computing, as pointed out by Howard Card and his colleagues [34, 35].

Indeed, Howard Card, by then a Distinguished Professor at the University of Manitoba, was one of the instigators of the renewed interest in stochastic computing.

Rebirth (2000s and Beyond)

Howard Card (1947–2006) earned his B.Sc. and M.Sc. degrees at the University of Manitoba and then went on to complete a Ph.D. at the University of Manchester in 1971. He subsequently became an Assistant Professor at the University of Waterloo, and then an Associate Professor at Columbia University. He served as a consultant for the IBM T.J. Watson Research Centre and was an instructor at AT&T Bell Labs. He returned to the University of Manitoba in 1980 to take on a Professorship in Electrical Engineering, and was eventually honoured as a Distinguished Professor. He profoundly cared about his students, and in 1983 he received the Olive Beatrice Stanton Award for Excellence in Teaching. Card believed that “if a thing is worth doing, it's worth doing to excess.” That mantra characterized all aspects of his life, including his mentorship, teaching and research.

In 2001, Card published two landmark papers on stochastic computing, both in the IEEE Transactions on Computers [36, 37]. Much of today’s work on stochastic computing can be traced back to this work.

In recent years, stochastic computing has been applied to many domains, and its theoretical work continues to advance. Like-minded researchers held workshops at the University of Waterloo in 2016, and again as part of the IEEE GlobalSIP conference in Montreal in 2017—these represent the first workshops on stochastic computing since the 1978 conference, announcing that stochastic computing is back!

Overview of the Book

This book is intended as a contemporary reference on stochastic computing. Chapter “[Origins of Stochastic Computing](#)” is a history of the early days of stochastic computing, as told by one of its pioneers, Brian Gaines. Chapter “[Tutorial on Stochastic Computing](#)”, written by Chris Winstead, is an introductory tutorial on stochastic computing. While it is primarily targeted to the stochastic computing newcomer, the chapter also attempts to find and define common language that sets the stage for the rest of the book.

Chapter “[Accuracy and Correlation in Stochastic Computing](#)”, written by John Hayes and his colleagues Armin Alaghi, Paishun Ting, and Vincent Lee, sheds light on two of the most important yet closely related pitfalls in stochastic computing, namely accuracy and correlation. Indeed, much of the stochastic computing literature focuses on minimizing sequence length, and on generating sequences where bits are independently from each other. In the chapter, the authors present their own approaches to tackle these problems, and present a case study based on image processing.

Chapter “[Synthesis of Polynomial Functions](#)” by Marc Riedel and his colleague Weikang Qian, looks at an approach for the synthesis of arbitrary polynomial functions that is based on Bernstein polynomials. The approach is validated using examples from image processing. Chapter “[Deterministic Approaches to Bitstream Computing](#)”, also by Marc Riedel, looks beyond the comfort zone of Bernoulli sequences, and investigates the use of other encoding mechanisms for stochastic-like computing. Indeed, he demonstrates that many of the perceived advantages of stochastic computing do not require sequences that are random in the strictest sense, but rather, sequences that are inspired by analog computation. Again, examples are derived from the world of image processing.

Chapters “[Generating Stochastic Bitstreams](#),” “[RRAM Solutions for Stochastic Computing](#)” and “[Spintronic Solutions for Stochastic Computing](#)” study several approaches for generating stochastic sequences. Chapter “[Generating Stochastic Bitstreams](#)”, by Jason Anderson and his colleagues Hsuan Hsiao and Yuko Hara-Azumi, reviews several techniques, starting with pseudo-random approaches based on linear feedback shift registers (LFSRs) and Sobol sequences; the chapter then

shifts its focus towards other sources of randomness such as in timing variations, and then introduces us to post-CMOS approaches. This serves as a transition to chapters “[RRAM Solutions for Stochastic Computing](#)” and “[Spintronic Solutions for Stochastic Computing](#)”, which look into the post-CMOS world. Chapter “[RRAM Solutions for Stochastic Computing](#)”, by Zhengya Zhang and his colleagues Phil Knag, Siddharth Gaba, and Wei Lu, looks at memristive devices and their inherent randomness as a way to generate stochastic sequences. Then, in chapter “[Spintronic Solutions for Stochastic Computing](#)”, Jie Han and his colleagues Xiaotao Jia, You Wang, Zhe Huang, Yue Zhang, Jianlei Yang, Yuanzhuo Qu, Bruce Cockburn, and Weisheng Zhao, look at similar phenomena in spintronic devices such as magnetic tunneling junctions. These last two chapters really show us that stochastic computing may have a very significant impact in the future world of nanotechnology.

The last two chapters look into applications of stochastic computing. Chapter “[Brain-Inspired Computing](#)”, by Naoya Onizawa and his colleagues Warren Gross and Takahiro Hanyu, reports very compelling applications in the domain of machine learning, postulating that we are entering the era of brainware LSI, or “BLSI.” Finally, chapter “[Stochastic Decoding of Error-Correcting Codes](#)”, by François Leduc-Primeau, Saied Hemati, Vincent Gaudet, and Warren Gross reviews progress over the past 15 years in the area of stochastic decoding of error-control codes.

References

1. J. von Neumann, *Lectures on probabilistic logics and the synthesis of reliable organisms from unreliable components*, California Institute of Technology, notes by R. S. Pierce, Jan. 1952.
2. J. von Neumann, “Probabilistic logics and the synthesis of reliable organisms from unreliable components,” *Automata Studies*, pp. 43-98, 1956.
3. P. Elias, “Computation in the Presence of Noise,” *IBM Journal of Research and Development*, vol. 2, no. 4, pp. 346-353, 1958.
4. A.A. Mullin, “Reliable Stochastic Sequential Switching Circuits,” *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 77, no. 5, pp. 606-611, Nov. 1958.
5. J.D. Cowan, “Toward a Proper Logic for Parallel Computation in the Presence of Noise,” *Bionics Symposium*, Dayton, OH,
6. W.J. Poppelbaum, C. Afuso, and J. W. Esch, “Stochastic Computing Elements and Systems,” *American Federation of Information Processing Societies, Fall Joint Computer Conference*, vol. 31, pp. 635-644, Books, Inc., New York, 1967.
7. W.J. Poppelbaum and C. Afuso, *Noise Computer*, University of Illinois, Dept. Computer Science, Quarterly Technical Progress Reports, April 1965-January 1966.
8. J. Esch, *Rascal – A Programmable Analog Computer Based on a Regular Array of Stochastic Computing Element Logic*, doctoral thesis, University of Illinois, 1969.
9. C. Afuso, “Two Wire System Computer Circuits Using Transistor Difference Amplifier,” *The Science Bulletin of the Division of Agriculture, Home Economics & Engineering*, The University of the Ryukyus, no. 9, pp. 308-321, Dec. 1962. Abstract Available: <http://ir.lib.u-ryukyu.ac.jp/handle/20.500.12000/23153>. Accessed on Aug. 2, 2018.
10. C. Afuso, *Analog Computing with Random Pulse Sequences*, doctoral thesis, University of Illinois report #255, 1968.
11. Y. Nagata and C. Afuso, “A Method of Test Pattern Generation for Multiple-Valued PLAs,” *International Symposium on Multiple-Valued Logic*, pp. 87-91. 1993.

12. S.T. Ribeiro, *Phase Plane Theory of Transistor Bistable Circuits*, doctoral thesis, University of Illinois, 1963.
13. S. T. Ribeiro, "Comments on Pulsed-Data Hybrid Computers," *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 5, pp. 640-642, Oct. 1964.
14. S. T. Ribeiro, "Random-Pulse Machines," *IEEE Transactions on Electronic Computers*, vol. EC-16, no. 3, pp. 261-276, June 1967.
15. B. R. Gaines, "Stochastic Computing," *American Federation of Information Processing Societies, Spring Joint Computer Conference*, vol. 30, pp. 149-156, Books, Inc., New York, 1967.
16. B. R. Gaines, "Techniques of Identification with the Stochastic Computer," *International Federation of Automatic Control Symposium on Identification*, Prague, June 1967.
17. B. R. Gaines, "Stochastic Computer Thrives on Noise," *Electronics*, vol. 40, no. 14, pp. 72-79, July 10, 1967.
18. B. R. Gaines, "Stochastic Computing," *Encyclopaedia of Information, Linguistics and Control*, pp. 766-781, Pergamon Press, New York and London, 1968.
19. R.C. Lawlor, *Computer Utilizing Random Pulse Trains*, U.S. patent 3,612,845, priority date July 5, 1968, granted Oct. 12, 1971.
20. G. White, "The Generation of Random-Time Pulses at an Accurately Known Mean Rate and Having a Nearly Perfect Poisson Distribution," *Journal of Scientific Instruments*, vol. 41, no. 6, p. 361, 1964.
21. R.C. Lawlor, "What Computers can do: Analysis and Prediction of Judicial Decisions," *American Bar Association Journal*, vol. 49, no. 4, pp. 337-344, April 1963.
22. W. Peakin, "Alexa, Guilty or Not Guilty?" posted Nov. 13, 2016. Available: <<http://futurescot.com/alex-guilty-not-guilty/>>, Accessed: August 31, 2018.
23. W.J. Poppelbaum, "Statistical Processors," *Advances in Computers*, vol. 14, pp. 187-230, 1976.
24. P. Mars and W. J. Poppelbaum, *Stochastic and Deterministic Averaging Processors*, Peter Peregrinus Press, 1981.
25. G.A. Black, *Analog Computation Based on Random Pulse Density Modulation*, doctoral thesis, University of Toronto, 1974.
26. —, "Applications of Multiple-Valued Logic," panel session at the *International Symposium on Multiple-Valued Logic*, chair: M.S. Michalski, panelists: B.R. Gaines, S. Haack, T. Kitahashi, W.J. Poppelbaum, D. Rine, and K.C. Smith, Logan, UT, May 1976.
27. —, *First International Symposium on Stochastic Computing and Its Applications*, Toulouse, France, 420 pages, 1978.
28. R. Ananth, *A Field Programmable Stochastic Computer for Signal Processing Applications*, Master of Applied Science thesis, University of Toronto, 1992.
29. P. Jeavons, D.A. Cohen, J. and Shawe-Taylor, "Generating Binary Sequences for Stochastic Computing," *IEEE Transactions on Information Theory*, vol. 40, pp. 716-720, 1994.
30. A.F. Murray, S. Churcher, A. Hamilton, et al., "Pulse Stream VLSI Neural Networks," *IEEE Micro*, vol. 13, no. 3, pp. 29-39.
31. Y. Hirai, "PDM Digital Neural Network System," in K. W. Przytula and V.K. Prasanna, *Parallel Digital Implementations of Neural Networks*, pp. 283-311, Englewood Cliffs: Prentice Hall, 1993.
32. J.E. Tomberg and K. Kaski, "Pulse Density Modulation Technique in VLSI Implementation of Neural Network Algorithms," *IEEE Journal of Solid-State Circuits*, Vol. 25, no. 5, pp. 1277-1286, 1990.
33. L. Zhao, *Random Pulse Artificial Neural Network Architecture*, Master of Applied Science Thesis, University of Ottawa, May 1998. Available: https://www.collectionscanada.gc.ca/obj/s4/f2/dsk2/tape17/PQDD_0006/MQ36758.pdf. Accessed on August 2, 2018.
34. J.A. Dickson, R.D. McLeod, and H.C. Card, "Stochastic Arithmetic Implementations of Neural Networks with In Situ Learning," *International Conference on Neural Networks*, pp. 711-716, 1993.

35. H. Card, "Limits to Neural Computations in Digital Arrays," *Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1125-1129, 1997.
36. B.D. Brown and H.C. Card, "Stochastic Neural Computation. I. Computational Elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891-905, 2001.
37. B.D. Brown and H.C. Card, "Stochastic Neural Computation. II. Soft Competitive Learning," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891-905, 2001.

Origins of Stochastic Computing



Brian R. Gaines

Abstract In the early 1960s research groups at the University of Illinois, USA, and Standard Telecommunication Laboratories (STL), UK, each independently conceived of a constructive use of random noise to implement analog computers in which the probability of a pulse in a digital pulse stream represented a continuous variable. The USA group initially termed this a *noise computer* but shortly adopted the UK terminology of *stochastic computer*. The target application of the USA group was visual pattern recognition, and that of the UK group was learning machines, and both developed trial hardware implementations. However, as they investigated applications they both came to recognize that the technology of their era did not support stochastic computing systems that could compete with available computational technologies, and they moved on to develop other computing architectures, some of which derived from the stochastic computing concepts. Both groups published expositions of stochastic computing which provided a comprehensive account of the technology, the architecture of its functional modules, its potential applications and its then current limitations. These have become highly cited in recent years as new technologies and issues have made stochastic computing a competitive technology for a number of significant applications. This chapter provides a historical a historical analysis of the motivations of the pioneers and how they arrived at the notion of stochastic computing.

Introduction

The possibility of emulating analog computers using digital hardware by representing a continuous number as the probability of the presence of a digital pulse in a train of pulses was conceived independently by Sergio Ribeiro and Brian Gaines in the

B. R. Gaines (✉)
University of Victoria, Victoria, BC, Canada
University of Calgary, Calgary, AB, Canada
e-mail: gaines@uvic.ca; gaines@ucalgary.ca

early 1960s. Ribeiro was a graduate student of Ted Poppelbaum in the Information Engineering Laboratory (IEL) at the University of Illinois, Champaign, Illinois, USA, and Gaines was a graduate student of Richard Gregory in the Department of Experimental Psychology, Cambridge University, UK and also a consultant to John Andreae's learning machines group at Standard Telecommunications Laboratory (STL), UK.

The US and UK groups both implemented digitally-based analog computers using probabilistic pulse trains, the IEL group initially terming this a *noise computer* but shortly adopting the terminology of the STL group, *stochastic computer*, which became the common designation in later research. As both groups evaluated applications of stochastic computing, for IEL primarily image processing and for STL navigational aids and radar tracking, it became apparent that the stochastic computer based on the digital circuitry then available was not competitive with alternative techniques. They began to develop other computer architectures to address those applications such as *burst* and *bundle* processing [58], and *phase computers* [37] and *microprogrammed* computers [21], respectively.

Both groups published extensively on stochastic computing in the late 1960s [24, 26, 30, 61, 68] which stimulated research in other research groups world-wide and many of those publications continue to be widely cited in the current renaissance of stochastic computing as they provide tutorial material on the fundamentals and the commonly adopted terminology for stochastic computer components, representations and applications. They also contain critical commentaries on the strengths and weaknesses of stochastic computing which are still applicable today.

Ted Poppelbaum

When I was asked to contribute a foreword to this collection of articles on the current state of the art in stochastic computing and its applications, my first reaction was sorrow that Ted Poppelbaum was no longer available to co-author it with me. Ted died in 1993 at the age of 68 and did not live to see the massive resurgence of stochastic computing research in the past decade.

Wolfgang (Ted) Johan Poppelbaum was born in Germany in 1924 and studied Physics and Mathematics at the University of Lausanne from 1944 to 1953. In 1954 he joined the Solid State Research Group under Bardeen at the University of Illinois and researched an electrolytic analog of a junction transistor. In 1955 he joined the faculty of the Department of Computer Science and became a member of the Digital Computer Laboratory developing the circuits for the ILLIAC II and later computers. In 1960 he received a patent for his invention of the transistor *flip-flop* storage module [59]. In 1972 he became Director of the Information Engineering Laboratory and received a major Navy contract to support his research on statistical computers and their applications. He retired in 1989.

Ted had many and varied projects in his laboratory. His 1973 report [57] on the achievements and plans of the Information Engineering Laboratory summarizes

some 45 distinct projects during the post-Illiack II phase from 1964 to 1973. They are grouped under the categories: Storage/Hybrid Techniques; Stochastic and Bundle Processing; Displays and Electro-Optics; Communication/Coding; World Models and Pattern Recognition; Electronic Prostheses.

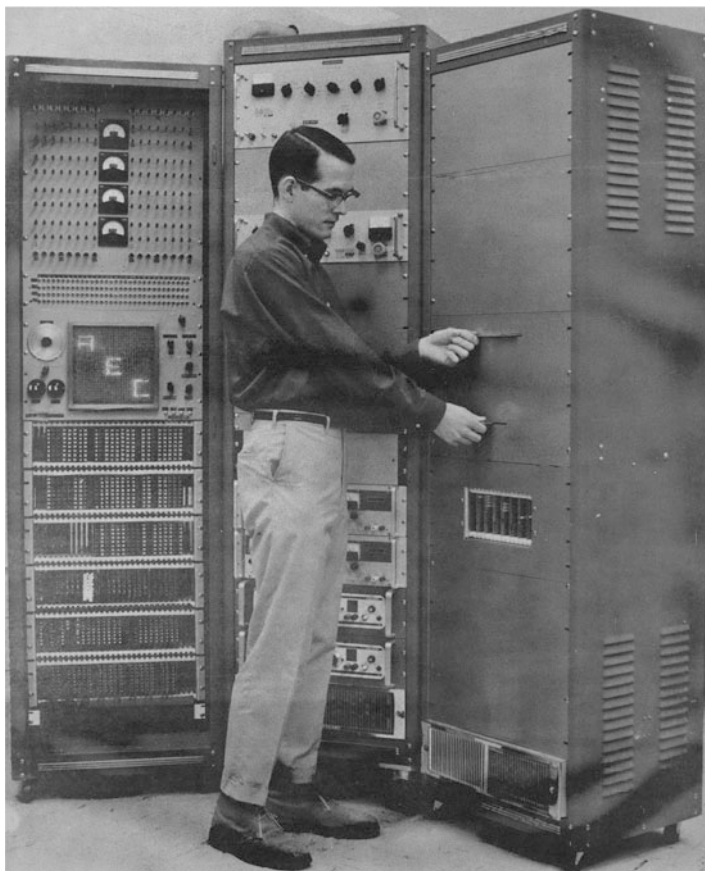


Fig. 1 Paramatrix: online digital/analog processing of picture information, Information Engineering Laboratory, University of Illinois, 1965

Ted and I became aware of our common interests in stochastic computing in 1967 as we both commenced publishing about the research and he invited me to present a paper on stochastic computing [18] at the IEEE Convention in March 1968 in New York where he was organizing a session on *New Ideas in Information Processing*. I also visited his laboratory, saw the many systems he had developed including the *Paramatrix* image processor (Fig. 1) which was one of his target applications for stochastic computing, and met John Esch who had built the *RASCEL* stochastic computing system.

Ted and I found we had a common background in solid state electronics and computer innovation, and discussed them at length as if we had been colleagues for many years. I met with him again and introduced him to John Andreae and David Hill at the IFIP conference in August 1968 in Edinburgh (Fig. 2). We kept in touch intermittently and planned a joint book on stochastic computing but I had moved on to other projects and introduced him to Phil Mars at Robert Gordon Institute in Aberdeen who was actively pursuing stochastic computing research. They co-published *Stochastic and Deterministic Averaging Processors* in 1981 [47].



Fig. 2 Three pioneers of computational intelligence: from left to right, John Andreae (learning machines), David Hill (speech recognition), Ted Poppelbaum (stochastic computing in image processing), IFIP Congress, August 1968, Edinburgh

Ted published several additional major articles that placed stochastic computing in the context of other computing technologies, notably his surveys in *Advances in Computers* in 1969 on *what next in computer technology?* [60], in 1976 on *statistical processors* [58] and in 1987 on *unary processing* [62]. His 1972 textbook on *Computer Hardware Theory* [56] that was widely used in engineering courses includes a chapter on *analog, hybrid and stochastic circuits*.

Sergio Ribeiro, Cushin Afuso and John Esch

Sergio Telles Ribeiro was born in Brazil in 1933, received an Engineering degree there in 1957 and taught electronics at the Institute of Technology and Aeronautics. In 1960 he received a fellowship from the Brazilian Government to study in the USA and entered the University of Illinois, receiving his masters in 1961 and his doctorate in 1963. His doctoral topic was a *phase plane theory of transistor bistable*

circuits [67] reflecting Ted's continuing interest in the circuit he had invented and its dynamic behavior that determined the speed and reliability of its operation.

After his doctorate Ribeiro continued as a research assistant working with Ujhelyi on the electronic deflection [65] and intensity modulation [79] of laser beams, and in 1964 they joined Carson Laboratories to pursue the industrial applications of that research. In July 1966 he submitted a paper to the IEEE Transaction on Computers on *random pulse machines* [68] that has become one of the classics in the stochastic computing literature.

It appears that Ribeiro's research on study of the architecture and potential of random pulse machines was theoretical. He notes in footnote 2 that "In the spring of 1963 while working with Dr. W.J. Poppelbaum at the University of Illinois the author suggested that a research program be undertaken to investigate theoretical and practical aspects of random-pulse systems." He thanks Dr. Carson for his support of the writing of the paper without implying that it is a project at Carson Laboratories.

Ribeiro had left Ted's laboratory before I visited and I never met him and have not been able to trace any publications by him after a Carson Laboratories 1966 patent for a display device based on his research with Ujhelyi [80]. There is no specific information about how Ribeiro came to be interested in random pulse computing. However, there is some strong circumstantial evidence that indicates how the notion may have occurred to him.

In 1964 Ribeiro [66] published a correspondence item in the IEEE Computer Transactions critiquing Schmid's [71] 1963 paper on a providing analog-type computation with digital elements. He corrects some errors in Schmid's discussion, suggests improvements in his implementation and then, whilst discussing the utility of pulse rate computers, suggests that studies of artificial neurons show that the implementation could be simple. Ribeiro cites three papers on electronic neuron models [7, 48, 50] from the *Bionics* session at 1961 National Electronics held in Chicago, about an hour away from Champaign, suggesting he may have attended that meeting, and a 1963 paper [44] from the IEEE Transactions of Biomedical Electronics suggesting he continued to follow the related literature.

However, none of the cited papers mention the notion that neurons had stochastic behavior which was common in the neurological literature going back to at least to Lashley in 1942 [42, p. 311]. In 1962 Jenik [40, 41] showed that the rates of the non-coherent pulse trains of two neurons afferent to a third were multiplied in its efferent train. Ribeiro might have become aware of such analyses or he might have considered the optoelectronic approximate multiplier described in one of the neuron model papers [7] and realized that if the pulse streams were independent random events then the output of an AND-gate would be the product of their generating probabilities.

In his 1967 paper Ribeiro mentions neurons in his abstract and index terms, commences the introduction with a presentation of Von Neumann's book on *The Computer and the Brain* [81], discusses the neural analogy extensively throughout, and has a *Bionics* subsection in his references with 12 citations. However, he does

not specifically attribute the source of his introduction of the notion of random pulses into Schmid's architecture to any specific material that he cites.

In 1964 Ted initiated a research program to study the computational potential of random-pulse systems by making it the topic of Afuso's doctoral research in 1964 and that of Esch in 1967. Cushin Afuso was born in 1933 in Japan, studied for his masters at the University of Illinois in 1959–1960, and returned for his doctorate in 1964–1968. He states that his 1968 dissertation, *Analog computation with random pulse sequences* [1] is “is a feasibility study of a stochastic computing system” taking the operations of an analog computer as his target and showing how multipliers, dividers, adders and subtractors may be implemented.

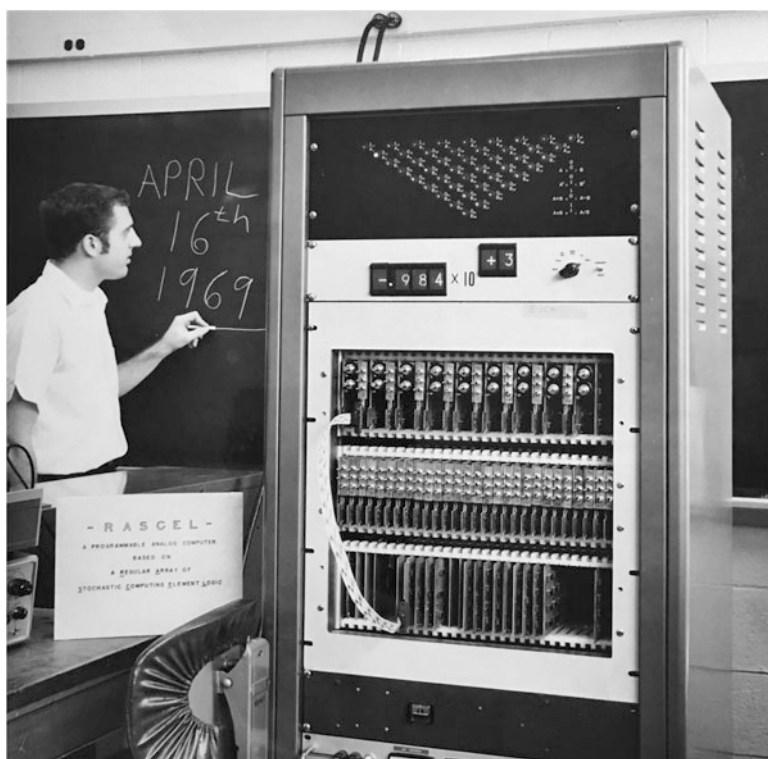


Fig. 3 John Esch presenting his RASCEL stochastic computer, Information Engineering Laboratory, University of Illinois, 1969

John W. Esch was born in the USA in 1942, studied for his masters at the University of Illinois in 1965–1967, and for his doctorate in 1967–1969. He states in his 1969 dissertation, *RASCEL - A programmable analog computer based on a regular array of stochastic computing element logic* [11] (Fig. 3) that “in February of 1967 this author joined Afuso and worked with him to extend the SRPS system to a sign-magnitude number representation and to develop a programmable arithmetic

computing element.” He went on to a career in the IT industry becoming a major contributor to knowledge representation technologies and literature and developing *conceptual graphs* innovations, implementations and applications [12, 13].

Neither Afuso nor Esch mention the neural analogy in their theses and their research was focused on developing a stochastic computer that emulated the functionality of an analog computer. It seems that the Schmid’s version of the digital differential analyzer and research on artificial neurons jointly provided the original inspiration for Ribeiro’s ideas but the later implementation became an engineering project with an initial focus on circuits to generate random pulse trains and a later one on how to emulate an analog computer.

Yiu Kwan Wo undertook graduate research with Ted commencing in 1967 receiving his masters in 1970 and his doctorate in 1973 for a thesis entitled *APE machine: A novel stochastic computer based on a set of autonomous processing elements* [85]. However, his work does not extend the stochastic computing aspect of Afuso and Esch’s research but instead focuses on radio-frequency transmission of data between the modules supporting inter-operation and reconfiguration without physical interconnection, an intriguing possibility in its own right.

Brian Gaines

It should be easier to describe my own research and the influences on it, and I do have some detailed recollections, but, after five decades, much has been forgotten and I have had to go back to files of notes, documentation, reports papers, memoranda and correspondence from the 1960s that I have dragged around the world for over 50 years but not previously opened—there were many surprises.

I was born in the UK in 1940, studied at Trinity College, Cambridge, from 1959 to 1967 for my bachelors in mathematics and theoretical physics and doctorate in psychology. Electronics became my primary hobby when I was 12 after my father banned analytical chemistry when hydrogen sulphide permeated the house. The availability of low-cost government surplus electronics components and systems after the war made it feasible to create a professional laboratory at home and I built my first oscilloscope from the components of a rocket test set when I was 14.

My school library had several of Norbert Wiener’s books and I became fascinated by his notion of cybernetics as the common science of people and machines and his portrayal of what is was to be a mathematician. The headmaster taught a small group of students philosophy in the library and I audited his lectures becoming very interested in Kant and the issues of human understanding of the world and of the nature of scientific knowledge. I found Ashby’s writings on cybernetics and admired the way that he solved very general problems using algebraic techniques and I also found Carnap’s logical structure of the world and Wittgenstein’s tractatus provided formal approaches to the issues that Wiener and Kant had raised.

I was on the science side at school and obtained a state scholarship in mathematics to attend University in 1958 and applied to Trinity College, Cambridge

but they made it a condition of acceptance that I also qualify in Latin and delay entry until 1959. I went to the Latin teacher's home for an hour on Saturdays for 3 months to prepare for the examination, and spent the interim year working as a laboratory assistant at ITT's semiconductor research laboratory¹ working on phosphorus and boron diffusion, epitaxial growth of silicon and the fabrication of gallium arsenide tunnel diodes. I also designed and built a nanoamp measuring instrument to determine the greatly reduced leakage current in transistors as we experimented with the planar process and was surprised to find it still in routine use at the end of a 74n integrated circuit family production line when I visited STC at Footscray again some 5 years later.

When I went up to Cambridge I planned to continue my activities in electronics and took with me many samples of the transistor and tunnel diodes that I had fabricated. At the *Societies Fair* in my first term I asked the chairman of the Wireless Society, Steve Salter, whether he knew anyone who might be interested in using them as I hope to find a home in some electronics laboratory. Steve was Richard Gregory's instrumentation engineer and introduced me to Richard who agreed that I could act as Steve's electronics assistant. Richard's primary research was how the brain reconstructed depth information from the disparate images of the separated eyes. I built an oscilloscope with two cathode ray tubes and prisms that allowed the eyes to be stimulated separately. This enabled me to display the 3D projection of a rotatable 4D cube and I studied how the projection was perceived as the subject manipulated it.

In 1961 saw an advertisement in *Nature* for staff for a new *learning machines* project at STL,² ITT Europe's primary research laboratories, about an hour away from Cambridge. I applied to John Andreae, the Project Leader, to be employed there in vacations and became his part-time assistant in mathematics, electronics and computing. In particular, I worked on the interpretation of neural net simulations and on the theoretical foundations of the STeLLA³ learning robot [5] which John was simulating on the KDF9 and his electronics assistant, Peter Joyce, had built in the laboratory (Fig. 4).

Richard and John's laboratories were my focus of attention during my Cambridge years. Trinity only required me to attend a 1 h tutorial with a college Fellow once a week and work on questions from past examination papers, and eventually take the part II mathematics tripos exam to qualify for a degree. Lectures were offered by the university and open to all students but not compulsory or assessed. I largely went to philosophy topics that interested me and lectures by renowned visitors such as Murray Gell-Mann and Richard Feynman in cutting-edge research areas where it was fascinating to meet those who were creating new theories of the nature of matter.

¹Standard Telecommunications Company (STC), Footscray, Kent.

²Standard Telecommunication Laboratories (STL), Harlow, Essex.

³Standard Telecommunication Laboratories Learning Automaton (STeLLA).

In June 1962 I took the part II tripos examination in mathematics, and asked the state scholarship committee if I could have a further year of funding to study for the mathematics tripos part III. However, Richard was a consultant to the Ministry of Defence and had been offered funding for a graduate student to undertake a study of the *adaptive training* of perceptual-motor skills. He offered me the opportunity but Oliver Zangwill, the head of the department, said I needed a psychology degree to do so. My scholarship sponsors approved this variation, and Richard asked Alan Watson, the eminent behavioral psychologist, to be my tutor. My positivist leanings suited him well and he set me to write essays for him on a very wide range of topics in psychology, debating my extremely behavioristic mathematical interpretations.

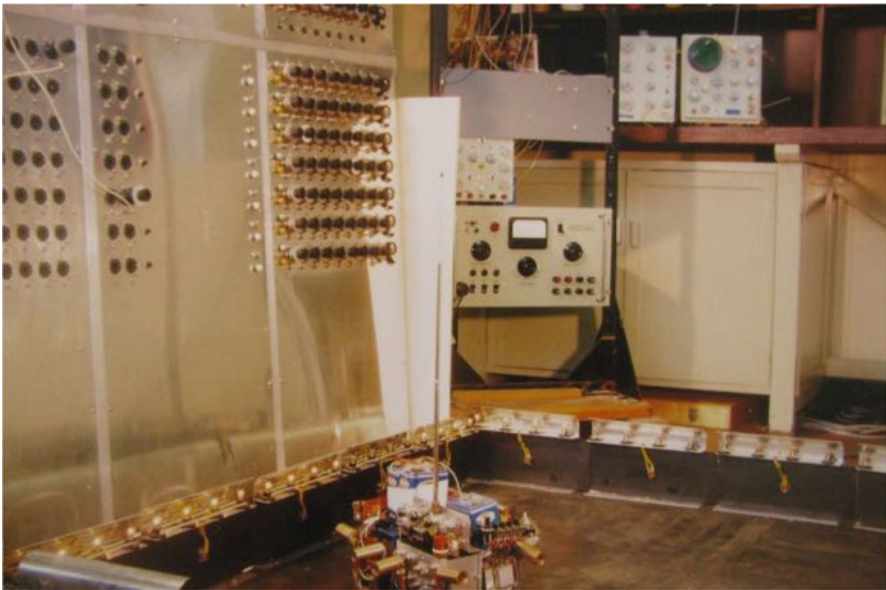


Fig. 4 Andreae's STeLLA, Standard Telecommunications Laboratories, Harlow, Essex, UK, 1963: in the foreground, robot with its sensors in its environment; in the background, racks of electronic equipment and potentiometers for adjusting weights implementing the learning algorithms

In June 1963 I took the part II tripos examination in psychology and became Richard's graduate student funded through the contract. *Adaptive training* is a technique to generate a learning progression for a skill by automatically adjusting the task difficulty based on the trainee's performance, thus turning a simulator into a teaching machine [29]. Common sense suggests it should be effective but nearly all studies to date had negative outcomes. I analyzed the situation Ashby-style assuming that a skill was constituted as a number of dependent sub-skills ordered such that the probability of learning one was low if prior ones had not been learned and showed that in such a situation adaptive training should be very effective even

if one has no knowledge of the sub-skill structure or trainee's state in terms of it. I examined previous laboratory studies and felt that the tasks investigated had been insufficiently challenging. The task of interest to the sponsor was classified but the training literature suggested that a tandem-propeller submarine involving managing position by controlling the rate of change of acceleration was extremely difficult and I decided to simulate that.



Fig. 5 Brian Gaines working with the analog computer and stereoscopic oscilloscope that he built, Department of Experimental Psychology, Cambridge, 1964

I built a 10-amplifier analog computer using commercially available operational amplifiers (Fig. 5) but needed an analog multiplier to adjust the parameters that varied the difficulty of the task which was not within the available budget. I designed what I termed a *chopping multiplier* where one signal was intermittently switched by a comparator whose inputs were the other signal and a uniformly distributed waveform. The latter could have been random but the most convenient to generate was a sawtooth. The output was smoothed by a resistor-capacitor feedback loop to generate an estimated product of the two signals. This reduced the bandwidth but it was still more than adequate, and in practice I found it useful to leave some ripple as a disturbance to create a challenging control task. The electronic multiplier tended to drift and for the final study I designed a motorized potentiometer multiplier where the bandwidth for one signal was high but for the other was low but more than adequate for the performance-adaptive adjustment. The experience of designing multipliers sensitized me to the issues of providing them as analog computer modules.

The notion of stochastic computing came from three distinct influences. From Wiener and Ashby I came to see random processes as a source of variety and

innovation that became structured through the various filtering processes discussed by Kant, Carnap and Wittgenstein. From my experiences in constructing analog computer multipliers the simplicity of the multiplication of probabilities of the conjunction of uncorrelated events seemed to have engineering potential. From Richard I developed an interest in the neurological basis of depth perception and proposed that the representation of visual intensity by neuronal discharges could be used to extract depth information by spatial correlation through a simple conjunctive processes if the discharges were essentially asynchronous and hence uncorrelated.

In addition, my studies of adaptive training had three components: a theoretical one to show that a very general framework for what is was for any system to *learn a skill* showed that adaptive training accelerated learning; an empirical one of training humans; and an empirical one of training learning machines undertaking the same task as the humans. For the last I used a digital version of Rosenblatt's [69] *perceptron* which did not have the same convergence properties as an analog version. I had noticed this when analyzing Novikoff's [53] proof of perceptron convergence as one of steepest descent. I had previously deconstructed Pontryagin's [55] *maximum principle*⁴ to understand why there was no discrete version even though there were several erroneous attempts in the literature to derive one. It seemed to me that a discrete perceptron would have similar problems because it could only approximate steepest descent and might enter a non-convergent limit cycle. I hypothesized that random variation in the training sequence might overcome this as might random variation in the weight changes and showed empirically that the limit cycles did prevent convergence and theoretically that randomness in the training sequence or weight changes could overcome this [30]. However, even though I envisioned a discrete perceptron with random weight changes I did not at that time extend the notion to more general applications. I also did not implement at that time a stochastic perceptron but I found the issues of training one that had problems of convergence very useful to my analysis of the dynamics of training both people and learning machines [17].

All these notions came together when I visited STL in May 1965 and found that Peter Joyce had designed a digital module that John Andreae termed an *ADDIE*⁵ that enabled the reinforcing weight adjustments for the STeLLA learning robot to be made automatically rather than by manually adjusting potentiometers. The weight update equation was in the form of a running average, $w' = \alpha w + (1 - \alpha)x$, and Peter had approximated this with a complex of integrated circuits. I noted that the component count could be greatly reduced and the approximation improved if the variables were represented as the generating probability of a digital pulse train, and sketched out circuit diagrams for *ADDIE*'s with various resolutions using 74n series

⁴I became interested in Pontryagin's work because one of my experiments in Richard's laboratory was to replicate the results in a memorandum by Bartlett where he had investigated reaction times in a tapping task with variations in target difference. His results were consistent with the hypothesis that people made Pontryagin-type *bang-bang* movements using maximum acceleration following by maximum deceleration, and I was later able to demonstrate this in my control task[20].

⁵Adaptive Digital Data Integrating Element (*ADDIE*).

integrated circuit up-down counters, flip-flop arrays with logic gates to generate a pseudo-random number sequence and adders acting as binary number comparators.

John approved further investigation and during the next week I developed the statistical theory for the behaviour of an ADDIE, Peter breadboarded the circuit, and we were able to confirm that theory and practice conformed and provided a module providing the functionality required in the STeLLA architecture. I realized the ADDIE emulated an operational amplifier with a negative feedback loop in my analog computer, that a logic gate acted in the same way as my *chopping multiplier* and that the $[0, 1]$ range of probabilities could be used to emulate $[-1, +1]$, $[0, \infty]$ and $[-\infty, +\infty]$ ranges through appropriate transformations.

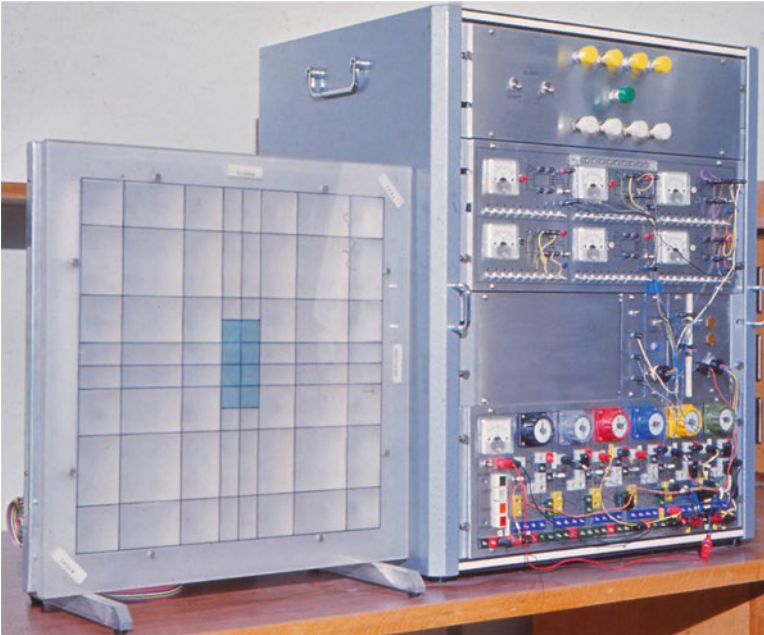


Fig. 6 On right, stochastic analog computer designed by Brian Gaines and built by Peter Joyce; on left, visual display of STeLLA learning controller's trajectory to the specified region of state space, Standard Telecommunication Laboratories, 1965

This led me to propose that the ADDIE be generalized to a complete *stochastic analog computer* with objective of providing all the computational support required by the STeLLA architecture. When it came to a later presentation to other ITT researchers John suggested that the term *analog* could be confusing as the computer was a digital one emulating analog functionality and we abbreviated the name to *stochastic computer*. Peter rapidly constructed a six-integrator stochastic computer with both analog and digital input-output capabilities (Fig. 6) and we were able to test its performance in a variety of applications, not only those relevant to learning machines such as the ADDIE, a stochastic perceptron, Bayesian prediction, and so

on, but also the simulation of dynamic systems, digital control, solution of partial differential equations, and so on. STL's primary output was ITT patents and I worked with our resident agent who had trained as a mathematician and understood the principles of the computer to file in March 1966 a comprehensive patent that had 54 claims covering the computer, representations and applications [27].

Once the patent was filed ITT approved the publication of details of the stochastic computer. The first public disclosure was at the IFAC Congress in June 1966 where I presented a paper with John on *A Learning Machine in the Context of the General Control Problem* [35] which updated his at the 1963 Congress [5]. The stochastic computer was the focus of my discussion contribution reporting progress since the paper was submitted which was transcribed in the published proceedings [35].

The first full paper I wrote on stochastic computing was requested late in 1966 by Roger Meetham at the National Physical Laboratory (NPL) who had heard a non-disclosure presentation that John and I gave to some NPL researchers and requested an article for the *Encyclopaedia of Linguistics, Information and Control* that he was editing. The paper was written early in 1966 and approved for submission by ITT in April but the book did not appear until 1969 [25].

The IFAC discussion [35], encyclopaedia entry [25], an internal presentation in December 1965 [16] and the patent [27] together provide a good account of how we perceived the stochastic computer at the time of its invention and before we were aware of a similar invention at the University of Illinois.

The magazine, *Electronics*, had published a short news item in December 1966 noting that "at the University of Illinois, a group of computer researchers has designed a series of analog computer circuits that depend on noise and therefore needn't be protected from it" and providing circuit examples [10]. I asked the editor if they would like an article on the similar research at STL and he took my draft, redrew all my diagrams as hand-drawn sketches to make them appear doodles from a research notebook, and retitled it as *Stochastic computer thrives on noise* [24].

I submitted a paper to the analog computing session Spring Joint Computer Conference in Atlantic City [26] as part of my first trip to the USA where I visited IBM, Bell Laboratories and Xerox research laboratories, under a research-liaison agreement between the major electronics companies. The doyens of analog and hybrid computers, Granino Korn and Walter Karplus also presented and, in leading the discussion on my paper, Walter remarked that he had never expected to see further radical innovations in analog computing.

At the conference exhibition I met Gene Clapper from IBM who was exhibiting his research on character recognition and speech recognition based on digital perceptrons [9]. He remarked that he had been surprised to find character recognition less accurate but ascribed it to a lower variety in the training sequences, and we discussed the role of noise in aiding the convergence of perceptrons. I also presented a paper [30] at the IFAC conference on system identification in Prague which focused on the modelling applications of the stochastic computer such as gradient techniques, the digital perceptron and Bayesian predictor.

My December 1965 presentation at STL aroused interest in several other research groups, notably those developing navigational aids and air traffic control systems and the written version was also distributed to other ITT companies in Europe. It aroused great interest, not only as a new computing technique but also because computing research had been discouraged in ITT after early commercial losses but several ITT manufacturing companies had become involved in both computer manufacture and application through government funding in their own countries. There was a need for computer expertise to be developed in the research laboratories.

I visited ITT companies in France, Germany, Sweden and Norway and found some potential applications of stochastic computers but was also able to contribute knowledge of general-purpose computers from my own experience and also from our usage of large-scale integration logic circuits in the 74n TTL family where usage had previously been discouraged because of early experience of unreliability. We were asked to investigate the feasibility of digital versions of navigational aids for the Decca and Omega systems, and for radar automatic-tracking systems, and I was also tasked with developing plans for a computer research division that I might lead when my university research was complete in 1967.

It rapidly became apparent that stochastic computing in itself would be too slow to be competitive in the proposed applications but I realized that the low-cost multiplication with AND-gates only required uncorrelated pulse streams, not necessarily random ones, and developed a digital version of my *chopper multiplier* where one pulse stream was uniformly distributed and the other clumped so that the two were uncorrelated. It was again possible to develop a complete analog computing framework and I termed it a *phase computer* [37] since the processing was now cyclic and the results were accumulated over each phase. Peter Joyce quickly produced a general-purpose prototype structured similarly to an analog computer (Fig. 7) and we were able to test this in a variety of applications such as a Decca navigator, a radar tracker and a digital PID control system.

In June 1966 John Andreae relocated to New Zealand to take up a chair in electrical engineering and I took over as part-time project leader but kept closely in touch with him by correspondence which has provided a useful record of what was happening at STL in late 1966 and early 1967. We were awarded a British government contract from the Ministry of Technology Advanced Computer Project for *Pulse Rate Modular Computing* and Ray Shemer joined the project to develop the phase computer and investigate its applications, also registering as an external doctoral student with the University of London [72, 73].

I was invited to be the keynote speaker at the IFAC conference on *Pulse Rate and Pulse Number Signals in Automatic Control* in Budapest in 1968 and presented a paper on *varieties of computer — their applications and inter-relationships* [34] which analyzed the relations between general-purpose and special-purpose computer architectures and applications. From the other presentations and discussions I also came to understand how what we had developed fit into a much larger sphere of applications of pulse rate computing. There was extensive discussion of the paper largely focusing on the relative merits of general-purpose and special-

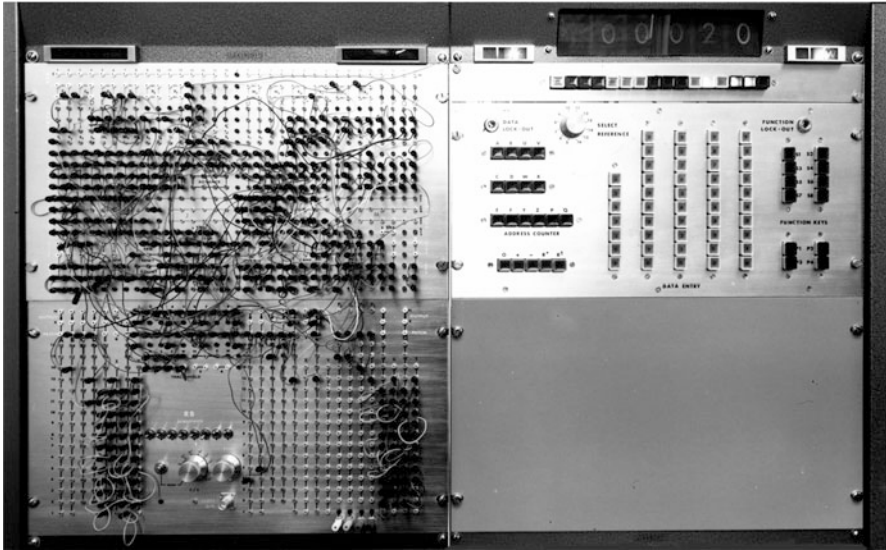


Fig. 7 Phase computer designed by Brian Gaines and built by Peter Joyce, Standard Telecommunication Laboratories, 1966

purpose computers and I still remember the joking, but perceptive, comment of one discussant that buying a general-purpose machine was safer because if it turns out to be unsuitable you can always find another use for it.

I was asked to summarize the conference by the editor of *Automatica* and Ray and I wrote an overview that concluded “Whatever the state of the special-purpose/general-purpose controversy, it is clear that the advent of low-cost integrated circuits has opened up a tremendous range of possibilities for new developments in control hardware; the re-development of DDA-like incremental computing techniques is one of the more attractive possibilities which is likely to lead to practical applications” [38]. I was also asked by the editor of the *IEEE Computer Transactions* to write a commentary on Ribeiro’s 1967 paper and concluded that “the main obstacle to the practical application of the stochastic computer is, at present, the generation of the random variables required in a reliable and economical manner. It may well be that we should look to truly random physical processes, such as photon-photon interactions, to provide the hardware foundation for stochastic computing systems” [23].

In May 1966 ITT decided to tender for the Boeing 747 simulators required in the UK on the basis of the simulation capabilities of LMT, their French company, but needed to establish a British company to manage the tendering process and support the products if they were successful. I was told I was to be appointed chief engineer of the new company rather than head of the new advanced developments and computing division. I had already arranged for that division to have research links with the Electrical Engineering Science department at the newly formed University

of Essex, and when I expressed my dismay at no longer having that opportunity to Barrie Chaplin the incoming Head of Department he offered me a position as one of the four initial faculty with a contract that encouraged me to develop industry links. I accepted, remaining a consultant to STL and the phase computer development for some years but becoming increasingly involved with other activities.

After my 1967 presentation at the Spring Joint Computer Conference, Julius Tou asked me for a survey of stochastic computing for *Advances in Information Technology* and I extended my encyclopaedia article to what was essentially a 136 page *brain dump* of all that I knew about stochastic computer. It was published in 1969 [28] and came to establish much of the stochastic computing terminology in the literature.⁶ I intended that to be my final paper on stochastic computing but was asked in 1976 to provide an overview of progress for an IEE colloquium on *parallel digital computing methods* [32]. In 1987 I was asked to introduce the topic at the first IEEE International Conference on *Neural Networks* [33], a paper that attracted a large audience and substantial discussion despite my noting that the research reported was 22 years old and has become cited as a primary source in many papers on random pulse neural networks.

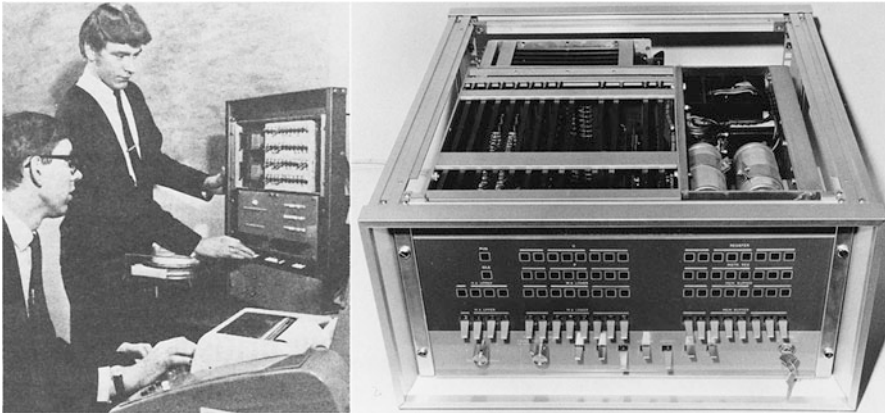


Fig. 8 Minic 8-bit microprogrammed microcomputer designed by Brian Gaines and built by Tony De’ath, Essex University, 1968; on left, the university prototype; on right, the commercial version

At Essex University my research became focused on the design of general-purpose and hybrid computers and associated operating systems, compilers and applications, human factors of computing and artificial intelligence. I received a contract from the RAF to continue my research on adaptive training in flight

⁶Earl Hunt was one of the first to cite this chapter (in the context of von Neumann’s book [81]) in his 1971 paper on “what kind of computer is man?” and comes to the conclusion that man is a stochastic computer. Earl unfortunately died in 2016 just before the advent of stochastic deep learning neural networks [45] and the assessment of how the behaviour of deep networks emulated human visual perception [54] that begins to validate his conjecture.

simulators, and decided to add hybrid capabilities to my analog computer to support more flexible training strategies. I had a PDP9 in my laboratory but needed a small dedicated computer, could not find a suitable one and decided to build my own which would give me full control over both hardware and software. I designed an 8-bit microprogrammable computer using the 74n series integrated circuits supporting a main memory from 1 to 64 Kbytes and a drum backing store, and my technician, Tony De'ath, built a prototype (Fig. 8). The experience of designing a conventional computer was very similar to that of designing the stochastic and phase computers, even though the architectures appear so different—all such digital computers are finite-state automata and the design process is akin to writing programs in the same language but satisfying somewhat differing requirements.

When I was evaluating a 64 Kbyte Sperry drum as a permanent storage device for Minic, the sales manager, Dave Seale, and his engineer, Tony Maine, said they would be interested in forming a company to manufacture the computer. I agreed and we formed Microcomputer Systems with Dave as CEO, Tony as chief engineer and myself as part-time technical director. We discussed funding with several potential investors but eventually went with the George Kent, an industrial instrumentation company who stated in their press release that “the Minic’s concept is far in advance of anything in the UK or the United States” [78, 82]. That may appear an exaggeration given DEC’s PDP7, 8, 9 and 10 architectures, but Minic seems to be the first microcomputer for sale at below £2000 and also the first with customer-programmable microprogramming. It was a precursor of the personal computers yet to come based on the Intel 8080, Zilog Z80, and so on, single-chip microcomputers.

The machine tool company, Alfred Herbert, needed an industrial computer for numerically-controlled machines and became our biggest customer. I had provided for 256 microcode instructions but only used 128 for the computer itself, and we were able to encode the stepping machine geometry in the remaining microcode using the phase computer techniques in a way that minimized the interface needed between computer and machine tool [74]. Alfred Herbert was able to sell its *Batchmatic* machine tool controller for £3500, half the price of their major competitor.

My graduate student, Peter Facey, wrote an emulator for MINIC on our central PDP10 computer [14], and I programmed the MINIC operating system, assembler, loader and a general-purpose system and application language, MINSYS [15, 19, 22], entirely in the emulator so that they were available before the first machines were manufactured. I also programmed some of the initial applications such as an 8-bed intensive care monitoring system for University College Hospital that operated on a 1 Kbyte MINIC with a 64K drum and output charts of blood pressure, temperature, and so on, on a Tektronix storage oscilloscope.

We received a British government contract from the Ministry of Technology Advanced Computer Project for the development of a more powerful version of MINIC, codenamed MINIC-S, and intended for industrial applications requiring high-performance computing and IO, security of operation and high reliability. The original MINIC was used to provide the IO processor, or processors, and the

MINIC-S had a descriptor and capability architecture targeted on the constructs required by high-level language compilers and the hardware assurance that running programs could not unintentionally interfere with one another [36, 39, 84]. A prototype was built and an operating system, assembler, linking loader, FORTRAN compiler and part of an ALGOL-68 compiler were written. However, in 1974 George Kent was acquired by Brown Boveri who decided to use DEC computers rather than manufacturer their own and MINIC manufacture and MINIC-S development were cancelled.

These computers may seem remote from the issues of stochastic computing but for me they illustrate the continuity between special-purpose and general-purpose computers. The MINIC modules were the same as those in the stochastic and phase computers but the microprogram used them to emulate a von Neumann architecture stored-program computer. If we had continued the development of the stochastic computer then its modules would have been similarly controlled to provide a programmable version of the analog computer patchboard. The principles of program control apply to all computer architectures, even neural networks. There is always a need to be able to configure general-purpose modules for special-purpose tasks.

I became aware of an analogous phenomenon in humans through discussions with the Russian psychologist, Alexander Luria, during his visits to the Psychology Department at Cambridge. I had been very impressed by his investigations on the effect of linguistic behaviour on the performance of perceptual-motor skills [46], and investigated verbal instruction as a priming technique for both my human and learning machine subjects [31]. One role of language is to provide program control, or at least behaviour priming, in both human and artificial neural networks.

The Invention of Stochastic Computing

Ted and I took for granted the independent simultaneous invention of stochastic computing at the University of Illinois and STL and never discussed it or tried to ascertain who was 'first.' We became aware of earlier pulse rate computers and of statistical linearization techniques in polarity coincidence correlators [86] and saw noise/stochastic computing as an extension of such techniques.

Multiple discovery and invention [51] is a common and well-studied phenomenon across many disciplines [83] and the usual explanation is that those involved were stimulated by the availability of the same, or similar, information. I have tried to ascertain that common inspiration for Ribeiro's and my research, and have suggested that it is the overlapping neural analogy in Ribeiro's considering artificial neurons as modules of pulse rate computers, and my considering the multiplicative processes implementing correlation in the interaction of the pulse streams of natural neurons.

In addition, the history of stochastic computing also exhibits another phenomenon of multiple discovery/invention where later researchers are unaware of

previous work. One of my colleagues at STL, David Hill, found in a patent search in the early 1970s that an invention filed by William G. Thistle in 1962 entitled *Integrating Apparatus* [77] that carried out computations using random pulse trains.

Thistle was an electronics engineer conducting research for the Canadian Ministry of Defence at the Canadian Armament Research and Development Establishment in Québec. David contacted him for further information and received both the patent and an internal report entitled *A novel special purpose computer* [76]. He sent me copies at the time and I recollect reading the patent and noting it was related to stochastic computing but have only now read the report in detail whilst writing this paper.

The abstract of Thistle's report states: "A type of computer is described for the real time evaluation of integrals of the form $I = \int y dx$, where x and y are functions of time. It is believed to be novel in its use of quasi-random processes, operating on pulse trains, to achieve the desired result. The method may be extended to cases where y is a function of several variables dependent on time. Accuracies comparable to analog methods are obtainable without the drift problems usually associated with analog methods."

Thistle describes techniques for addition, subtraction, multiplication, division and integration using random pulse trains, provides circuit diagrams, and described an application to a simple navigation system. His computer encompasses the basic architecture of the stochastic computers developed at the University of Illinois and STL and would constitute *prior art* from a patent perspective.

His report was not widely circulated. The distribution list shows that only 3 copies were issued (to the superintendents of systems and of electronics, and the chief superintendent) and 25 were lodged in the documents library. Thistle has three other patents (for power supplies and a gas discharge matrix display), and seems to have no other publications although there will likely be other internal reports. It is probable that much of his work was associated with classified systems.

A google scholar search on his name returns two of his patents, one of which is the US version of his *Integrating Apparatus* retitled *Computer for evaluating integrals using a statistical computing process*. His patent is not cited in other patents as prior art, and it seems unlikely that, even today, a content-based automated search would be able to link his text to the stochastic or pulse rate computing literature. As far as I know, Thistle's research is completely unrecognized and has had no influence, and there is no indication of how he came to invent a stochastic computer, but it deserves recognition in the history of computing as the earliest documented invention of a fully-functional stochastic computer.

Thistle's invention is also relevant to another question frequently asked about discoveries and inventions, what would have happened if neither the Illinois or STL teams had developed stochastic computers, would others have done so? The answer is clearly *yes*—it had already happened but no one knew. There was also research in neurology where it became known empirically, possibly as early as the 1950s, that the coincidence of neurons firing could result in a common afferent neuron firing and that this might be the basis of motion detection [64]. This led to an

empirical analysis of the jitter in neural firing that was shown to be sufficient for the afferent neuron to be acting as a pulse frequency multiplier [75]. Thus, *stochastic bit-stream neural networks* [8] were conceived from biological studies uninfluenced by stochastic computing (even though the similarity to the stochastic computer is often noted in that literature, e.g. [43]).

Conclusions

In the three decades after Ted and I completed our research in stochastic computing research continued elsewhere but at a low intensity. We received papers to referee, were asked to be thesis examiners, and were aware that there was continuing activity by colleagues across the world, such as Phil Mars in the UK, Sadamu Ohteru in Japan, Robert Massen in Germany (who in 1977 wrote the first book on *stochastic computer technology* [49]) and others, but no major growth in interest. However, in the recent past there has been a significant growth in research as illustrated in Fig. 9 which shows the citations to my 1969 survey (a more robust estimator based on a basket of several commonly cited articles shows a similar pattern). This book provides a much-needed overview of this burgeoning literature through tutorials and overviews by some of those who make have major contributions to its growth.

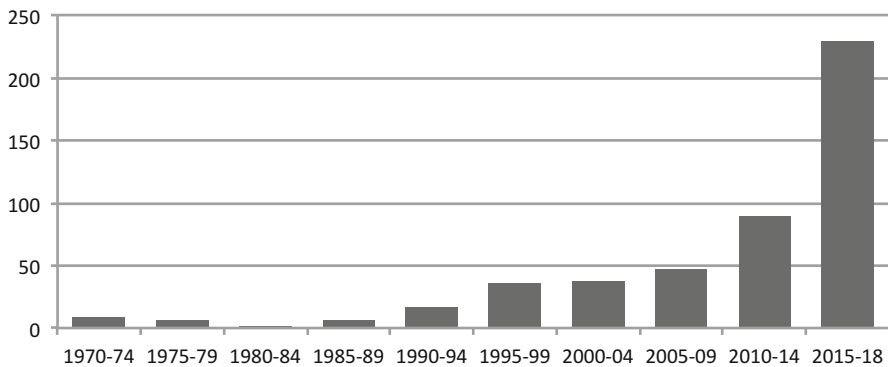


Fig. 9 Counts of the citations of *Stochastic computing systems* [28]

In the conclusions of my 1969 survey I noted three aspects of stochastic computing that seem to me to remain relevant to current research:

- “The stochastic computer has as yet had no major practical impact on data processing systems. Equally, the analogy between nervous processes and stochastic computing has not been carried to a stage where the stochastic computer can be justified solely as a model of the central nervous system. Indeed, present

justification for interest in stochastic computing is of a scientific nature—it extends the range of known data-processing systems.”

- “If true stochastic processes are utilized, e.g., from light photons or radioactive sources, then it would seem better to establish computing elements working directly with the original sources; photon/photon interactions brought about by certain electron systems would seem to offer a great potential for *natural* stochastic computation.”
- “The design of learning machines and pattern recognizers which take full advantages of the properties peculiar to stochastic computers offers the greatest promise for the future development and exploitation of the systems surveyed.”

To these, in the light of the ensuing developments in Ted and my laboratories, I would add:

- The essence of the stochastic computer is that, by representing a number as the frequency of occurrence of a binary pulse stream, simple modules can be used to perform significant computations, but, as shown by the phase computer, the pulse streams do *not* necessarily have to be truly random, only uncorrelated.
- The maintenance of this lack of correlation such that the output of a computation may be used as the input to another is a major issue in stochastic computer applications (which makes the apparent simplicity of stochastic computing misleading). Whilst a general solution might be feasible (e.g. using technologies with intrinsic stochastic behaviour), the requirement may also be addressed by application-specific techniques to manage correlation (e.g. [2]) that take advantage of the structure of the computations required.
- Hybrid architectures that combine the modular, parallel processing of the parallel computer and programmability of the general-purpose computer, particularly to control module interconnection will become increasingly significant.
- There are computational problems where randomness plays a significant role in computing the optimal or the least complex solution, and these merit special attention as actually requiring a stochastic implementation.

I will not attempt to present the current state of the art in stochastic computing as it relates to these issues. Very perceptive surveys by those deeply involved in current research are available [3] and the contributions to this book provide in-depth studies of the state of the art. Ted would be interested to see his interest in image processing addressed in the research that in many way triggered the resurgence of interest stochastic computing when it was shown to be as effective, error-tolerant, and requiring less energy usage than competing image processing technologies [4, 63]. My early interests are addressed by the applications of stochastic neural networks to deep learning [45] and other neuromorphic applications [6], by the wide range of alternative technologies being investigated for stochastic computing [3, 6, 70], and by the more computationally efficient deterministic variants which parallel our transition to the deterministic phase computer and are now accepted as variants of stochastic computing [52]. There are also many significant theoretical

and application innovations that we never envisioned—interesting ideas take on a life of their own, nurtured by the community of research.

Acknowledgements I am grateful to John Esch for his help in verifying my commentary on the research at the University of Illinois and for providing the photograph of his RASCEL stochastic computer. I am grateful to David Hill for prompting my memory of certain dates and events and for providing the material by Thistle documenting his early stochastic computer. I would also like to thank the editors of this volume for providing the opportunity to contribute this account of the origins of stochastic computing knowing that there are very few of us still alive to do so. I hope it will be of interest to the stochastic computing research community of this era, and wish them well.

Note Early stochastic computing material is available at <http://cpsc.ucalgary.ca/~gaines/reports>.

References

1. Chusin Afuso. “Analog computation with random pulse sequences”. PhD thesis. University of Illinois, 1968.
2. A. Alaghi and J. P. Hayes. “Exploiting correlation in stochastic circuit design”. *IEEE 31st International Conference on Computer Design (ICCD)*. 2013, pp. 39–46.
3. A. Alaghi, W. Qian, and J. P. Hayes. “The promise and challenge of stochastic computing”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.8 (2018), pp. 1515–1531.
4. Armin Alaghi, Cheng Li, and John P. Hayes. “Stochastic circuits for real-time image-processing applications”. *Proceedings of the 50th Annual Design Automation Conference*. Austin, Texas: ACM, 2013, pp. 1–6.
5. John H. Andraea. “STeLLA: A scheme for a learning machine”. *Proceedings 2nd IFAC Congress: Automation & Remote Control*. Ed. by V. Broida. London: Butterworths, 1963, pp. 497–502.
6. A. Basu et al. “Low-power, adaptive neuromorphic systems: Recent progress and future directions”. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8.1 (2018), pp. 6–27.
7. T. E. Bray. “An optoelectronic-magnetic neuron component”. *Proceedings National Electronics Conference*. 1961, pp. 302–310.
8. Peter S. Burge et al. “Stochastic bit-stream neural networks”. *Pulsed Neural Networks*. Ed. by Wolfgang Maass and Christopher M. Bishop. Cambridge, MA: MIT Press, 1999, pp. 337–352.
9. Gene L. Clapper. “Machine looks, listens, learns”. *Electronics* (Oct. 30, 1967).
10. Electronics. “Dropping the guard”. *Electronics* (Dec. 12, 1966), p. 48.
11. John W. Esch. “RASCEL - A Programmable Analog Computer Based on a Regular Array of Stochastic Computing Element Logic”. PhD thesis. University of Illinois, 1969.
12. John W. Esch. “System and method for frame and unit-like symbolic access to knowledge represented by conceptual structures”. Pat. 4964063. Unisys Corporation. Sept. 15, 1988.
13. John W. Esch and Robert Levinson. “An implementation model for contexts and negation in conceptual graphs”. *Conceptual Structures: Applications, Implementation and Theory*. Springer Berlin Heidelberg, 1995.
14. Peter V. Facey and Brian R. Gaines. “Real-time system design under an emulator embedded in a high-level language”. *Proceedings DATAFAIR 73*. London: British Computer Society, 1973, pp. 285–291.
15. Brian R. Gaines. “A mixed-code approach to commercial microcomputer applications”. *Conference on Microprocessors in Automation and Communications*. London: IERE, 1978, pp. 291–301.

16. Brian R. Gaines. *A Stochastic Computer: Some Notes on the Application of Digital Circuits to the Operations of Arithmetic and Differential Calculus by Means of a Probabilistic Representation of Quantities*. Tech. rep. Standard Telecommunication Laboratories, Dec. 9, 1965.
17. Brian R. Gaines. "Adaptive control theory: the structural and behavioural properties of adaptive controllers". *Encyclopaedia of Linguistics, Information & Control*. Ed. by A.R. Meetham and R.A. Hudson. London: Pergamon Press, 1969, pp. 1–9.
18. Brian R. Gaines. "Foundations of stochastic computing systems". *Digest of IEEE International Convention*. New York: IEEE, 1968, p. 33.
19. Brian R. Gaines. "Interpretive kernels for microcomputer software". *Proceedings Symposium Microprocessors at Work*. London: Society of Electronic & Radio Technicians, 1976, 56–69.
20. Brian R. Gaines. "Linear and nonlinear models of the human controller". *International Journal of Man-Machine Studies* 1.4 (1969), pp. 333–360.
21. Brian R. Gaines. *MINIC 1 Manual*. Colchester, UK: Department of Electrical Engineering Science, 1969.
22. Brian R. Gaines. *MINSYS Manual*. Colchester, Essex: Department Electrical Engineering Science, 1974.
23. Brian R. Gaines. "R68-18 Random Pulse Machines". *IEEE Transactions on Computers* C-17.4 (1968), pp. 410–410.
24. Brian R. Gaines. "Stochastic computer thrives on noise". *Electronics* (July 10, 1967), pp. 72–79.
25. Brian R. Gaines. "Stochastic computers". *Encyclopaedia of Linguistics, Information & Control*. Ed. by A.R. Meetham and R.A. Hudson. London: Pergamon Press, 1969, pp. 66–76.
26. Brian R. Gaines. "Stochastic computing". *Spring Joint Computer Conference*. Vol. 30. Atlantic City: AFIPS, 1967, pp. 149–156.
27. Brian R. Gaines. "Stochastic Computing Arrangement". British pat. 184652. Standard Telephones & Cables Ltd. Mar. 7, 1966.
28. Brian R. Gaines. "Stochastic computing systems". *Advances in Information Systems Science*, 2. Ed. by J. Tou. New York: Plenum Press, 1969, pp. 37–172.
29. Brian R. Gaines. "Teaching machines for perceptual-motor skills". *Aspects of Educational Technology*. Ed. by D. Unwin and J. Leedham. London: Methuen, 1967, pp. 337–358.
30. Brian R. Gaines. "Techniques of identification with the stochastic computer". *Proceedings IFAC Symposium on The Problems of Identification in Automatic Control Systems*. 1967, pp. 1–10.
31. Brian R. Gaines. "Training the human adaptive controller". *Proceedings Institution Electrical Engineers* 115.8 (1968), pp. 1183–1189.
32. Brian R. Gaines. "Trends in stochastic computing". *Colloquium on Parallel Digital Computing Methods—DDA's and Stochastic Computing*. Vol. 30. London: IEE, 1976, pp. 1–2.
33. Brian R. Gaines. "Uncertainty as a foundation of computational power in neural networks". *Proceedings of IEEE First International Conference on Neural Networks*. Vol.3. Ed. by M. Caudhill and C. Butler. 1987, pp. 51–57.
34. Brian R. Gaines. "Varieties of computer — their applications and inter-relationships". *Proceedings of IFAC Symposium on Pulse Rate and Pulse Number Signals in Automatic Control*, Budapest: IFAC, 1968, pp. 1–16.
35. Brian R. Gaines and John H. Andreae. "A learning machine in the context of the general control problem". *Proceedings of the 3rd Congress of the International Federation for Automatic Control*. London: Butterworths, 1966, 342–348 (discussion, session 14, p.93).
36. Brian R. Gaines, M. Haynes, and D. Hill. "Integration of protection and procedures in a high-level minicomputer". *Proceedings IEE 1974 Computer Systems and Technology Conference*. London: IEE, 1974.
37. Brian R. Gaines and Peter L. Joyce. "Phase computers". *Proceedings of 5th Congress of International Association for Analog Computation*. 1967, pp. 48–57.
38. Brian R. Gaines and Ray A. Shemer. "Fitting Control Mathematics to Control Hardware: An Aspect of the 1968 IFAC Pulse-Symposium". *Automatica* 5 (1969), pp. 37–40.

39. Brian R. Gaines et al. "Design objectives for a descriptor-organized minicomputer". *European Computing Congress Conference Proceedings, EUROCOMP 74*. London: Online, 1974, pp. 29–45.
40. F. Jenik. "Electronic neuron models as an aid to neurophysiological research". *Ergebnisse der Biologie* 25 (1962), pp. 206–245.
41. F. Jenik. "Pulse processing by neuron models". *Neural Theory and Modeling: Proceedings of the 1962 Ojai Symposium*. Ed. by Richard F. Reiss. Stanford, CA: Stanford University Press, 1964, pp. 190–212.
42. K. S. Lashley. "The problem of cerebral organization in vision". *Visual Mechanisms*. Ed. by H. Kluüver. Oxford: Cattell, 1942.
43. Robert Legenstein and Wolfgang Maass. "Ensembles of spiking neurons with noise support optimal probabilistic inference in a dynamically changing environment". *PLoS Computational Biology* 10.10 (2014), e1003859 1–27.
44. E. R. Lewis. "The locus concept and its application to neural analogs". *IEEE Transactions on Bio-medical Electronics* 10.4 (1963), pp. 130–137.
45. Y. Liu et al. "An energy-efficient online-learning stochastic computational deep belief network". *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2018), pp. 1–1.
46. A.R. Luria. *The Role of Speech in the Regulation of Normal and Abnormal Behavior*. Oxford: Pergamon Press, 1961.
47. P. Mars and W. J. Poppelbaum. *Stochastic and Deterministic Averaging Processors*. Stevenage: IEE/Peregrinus, 1981.
48. T. B. Martin. "Analog signed processing by neural networks". *Proceedings National Electronics Conference*. 1961, pp. 317–321.
49. Robert Massen. *Stochastische Rechentchnik: Eine Einführung in die Informationsverarbeitung mit zufälligen Pulsfolgen*. Munich: Hanser, 1977.
50. E. P. McGrogan. "Improved transistor neural models". *Proceedings National Electronics Conference*. 1961, pp. 302–310.
51. Robert King Merton. *The Sociology of Science: Theoretical and Empirical Investigations*. Chicago: University of Chicago Press, 1973.
52. M. Hassan Najafi, David J. Lilja, and Marc Riedel. "Deterministic methods for stochastic computing using low-discrepancy sequences". *IEEE/ACM International Conference On Computer-Aided Design (ICCAD '18)*. New York: ACM, 2018.
53. Albert B. Novikoff. *On Convergence Proofs for Perceptrons*. Tech. rep. SRI Project 3605. Menlo Park, CA: Stanford Research Institute, 1963.
54. Joshua C. Peterson, Joshua T. Abbott, and Thomas L. Griffiths. "Evaluating (and improving) the correspondence between deep neural networks and human representations". *Cognitive Science* (2018), 42.8, 2648–2699.
55. L. S. Pontryagin et al. *The Mathematical Theory of Optimal Processes*. Oxford, England: Pergamon Press, 1962.
56. W. J. Poppelbaum. *Computer Hardware Theory*. New York, Macmillan, 1972.
57. W. J. Poppelbaum. *Record of Achievements and Plans of the Information Engineering Laboratory*. Champaign, Urbana, IL: Department of Computer Science, University of Illinois, 1973.
58. W. J. Poppelbaum. "Statistical processors". *Advances in Computers*. Ed. by Rubinoff Morris and C. Yovits Marshall. Vol. Volume 14. Elsevier, 1976, pp. 187–230.
59. W. J. Poppelbaum. "Transistor Flip-Flop Circuit". Pat. 2933621. University of Illinois Foundation. Aug. 2, 1956.
60. W. J. Poppelbaum. "What next in computer technology?" *Advances in Computers*. Ed. by L. Alt Franz and Rubinoff Morris. Vol. Volume 9. Elsevier, 1969, pp. 1–21.
61. W. J. Poppelbaum, C. Afuso, and J.W. Esch. "Stochastic computing elements and systems". *Fall Joint Computer Conference*. Vol. 31. New York: Books, Inc, 1967, pp. 635–644.
62. W. J. Poppelbaum et al. "Unary Processing". *Advances in Computers*. Ed. by C. Yovits Marshall. Vol. Volume 26. Elsevier, 1987, pp. 47–92.

63. W. Qian et al. "An architecture for fault-tolerant computation with stochastic logic". *IEEE Transactions on Computers* 60.1 (2011), pp. 93–105.
64. Werner Reichardt. "Evaluation of optical motion information by movement detectors". *Journal of Comparative Physiology A* 161 (1987), pp. 533–547.
65. S. T. Ribeiro and G. K. Ujhelyi. "Electro-Optical Modulation of Radiation Pattern Using Curved Electrodes". U.S. pat. 3433554. Secretary of the Navy. May 1, 1964.
66. Sergio Telles Ribeiro. "Comments on Pulsed-Data Hybrid Computers". *IEEE Transactions on Electronic Computers* EC-13.5 (1964), pp. 640–642.
67. Sergio Telles Ribeiro. "Phase Plane Theory of Transistor Bistable Circuits". PhD thesis. University of Illinois, 1963.
68. Sergio Telles Ribeiro. "Random pulse machines". *IEEE Trans. Electronic Computers* EC-16.6 (1967), pp. 261–276.
69. Frank Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". *Psychological Review* 65.6 (1958), pp. 386–408.
70. Sayed Ahmad Salehi et al. "Computing mathematical functions using DNA via fractional coding". *Scientific Reports* 8.1 (2018), p. 8312.
71. Hermann Schmid. "An operational hybrid computing system provides analog-type computation with digital elements". *IEEE Transactions on Electronic Computers* EC-12.6 (1963), pp. 715–732.
72. Ray A. Shemer. "A hybrid-mode modular computing system". *Proceedings of IFAC Symposium on Pulse Rate and Pulse Number Signals in Automatic Control*. Budapest, 1968.
73. Ray A. Shemer. "A Hybrid-Mode Modular Computing System". PhD thesis. 1970.
74. I. W. Smith, D. A. Hearn, and P. Williamson. "Software development for Batchmatic computer numerical control system". *Proceedings of the Fourteenth International Machine Tool Design and Research Conference*. Ed. by F. Koenigsberger and S. A. Tobias. London: Macmillan, 1974, pp. 381–389.
75. Mandyam V. Srinivasan and Gary D. Bernard. "A proposed mechanism for multiplication of neural signals". *Biological Cybernetics* 21.4 (1976), pp. 227–236.
76. William G. Thistle. *A Novel Special Purpose Computer*. Tech. rep. CADRE Technical Note 1460. Valcartier, Quebec: Canadian Armament Research and Development Establishment, 1962.
77. William G. Thistle. "Integrating Apparatus". Canadian pat. 721406. Ministry of National Defence. Nov. 30, 1962.
78. Financial Times "George Kent backs a micro-computer venture". *Financial Times* (Feb. 5, 1970), p. 11.
79. G. K. Ujhelyi and S. T. Ribeiro. "An electro-optical light intensity modulator". *Proceedings of the IEEE* 52.7 (1964), pp. 845–845.
80. Gabor K. Ujhelyi, Sergio T. Ribeiro, and Andras M. Bardos. "Data Display Device". U.S. pat. 3508821. Carson Laboratories. Aug. 11, 1966.
81. John Von Neumann. *The Computer and the Brain*. New Haven, Yale University Press, 1958.
82. Computer Weekly. "MINIC system is bought by George Kent". *Computer Weekly* (Feb. 12, 1970).
83. Wikipedia. *List of multiple discoveries*. 2018. https://en.wikipedia.org/wiki/List_of_multiple_discoveries.
84. F.K. Williamson et al. "A high-level minicomputer". *Information Processing* 74. Amsterdam: Noth-Holland, 1974, pp. 44–48.
85. Yiu Kwan Wo. "APE machine: A novel stochastic computer based on a set of autonomous processing elements". PhD thesis. 1973.
86. S. Wolff, J. Thomas, and T. Williams. "The polarity-coincidence correlator: A nonparametric detection device". *IRE Transactions on Information Theory* 8.1 (1962), pp. 5–9.

Tutorial on Stochastic Computing



Chris Winstead

Abstract This chapter offers a pedagogical introduction to stochastic computing for those who may be new to the subject or require a refresher on the basic concepts. The level of presentation is suitable for graduate students or advanced undergraduates as well as for professionals in the field.

Keywords Stochastic computing · Stochastic arithmetic · Stochastic decoding · Error correction · Low density parity-check codes · Image processing · Neural networks · Spike timing dependent plasticity · Memristors

Introduction

Stochastic computing circuits have a number of features that have attracted the attention of researchers for several decades. This chapter introduces the fundamental concepts of stochastic computation and describes their attraction for applications in approximate arithmetic, error correction, image processing and neural networks. Some disadvantages and limitations are also discussed, as well as a discussion of future circuits that utilize native non-determinism to avoid certain disadvantages. Some background on digital design, probability theory and stochastic processes is necessary. Prior knowledge of image processing or machine learning topics—particularly Bayesian networks and neural networks—is also helpful for following the application examples. For deeper study on the topic, the reader is directed to several recent key references that explore each of the sub-topics in greater detail. A good starting point is the recent survey article by Alaghi et al. [2]. This chapter does not provide a comprehensive bibliography on stochastic computing; references in the chapter bibliography are selected to provide the greatest benefit to new students of the subject.

C. Winstead (✉)

Department of Electrical and Computer Engineering, UMC 4120, Utah State University, Logan, UT, USA

e-mail: chris.winstead@usu.edu

Fundamental Concepts

Stochastic computing circuits are able to realize arithmetic functions with very few logic gates. This is achieved by encoding numerical values within the statistics of random (or pseudorandom) binary sequences. For example, the number $x = 1/3$ can be represented by the sequence $0, 0, 1, 0, 1, 0, \dots$, wherein the frequency of 1's is equal to $1/3$. A numerical value encoded in this way is called a *stochastic number*. Throughout this chapter, we will use the terms *bit sequence* or *bit stream* interchangeably to refer to the specific random bits that encode a stochastic number. Since many different sequences can encode the same value, stochastic numbers are defined by the sequence statistics rather than the particular order of bits.

Definition 1 (Stochastic Number) Given a probability p_X , $0 \leq p_X \leq 1$, the corresponding stochastic number X is a sequence of random binary numbers X_0, X_1, \dots for which any $X_j \in \{0, 1\}$ may equal 1 with probability p_X .

Throughout this chapter, we will use capital letters (X) to refer to stochastic numbers, and subscripted capital letters (X_ℓ) to refer to the individual sequence bits, where the subscript (ℓ) indicates the clock cycle index. When analyzing logic operations, we will often omit the subscript when stating combinational relationships that hold for all clock cycles. We will also use capital letters to refer to binary values, integers, and parameters, which will be clearly indicated by context. A lower-case letter (x) represents the real value associated with the stochastic number, and the sequence probability is indicated by the letter p with a subscript to indicate the stochastic number (as in p_X).

Given a sufficiently long bit sequence, the sequence's mean value is expected to converge to the probability p_X . Stochastic numbers have precision that improves with sequence length, a property called *progressive precision*. For instance, the value $3/7$ can be precisely represented by a sequence of at least length 7, however the value $5/14$ requires a sequence of at least 14 bits. Since the individual bits in the sequence are random, a much longer sequence length is required before the sequence's actual average converges within the desired precision.

In practice, most stochastic computing circuits produce non-ideal stochastic numbers in which the bits depend on the sequence history. The resulting auto-correlation can sometimes distort or reduce the efficiency of stochastic computations, posing a serious challenge that takes several forms that are discussed in later sections of this chapter. Because of the potential inaccuracies, we need to carefully distinguish between ideal and non-ideal stochastic numbers.

Definition 2 (Ideal Stochastic Number) An ideal stochastic number has the properties of a Bernoulli process, wherein the sequence bits are all statistically independent from each other.

It is sometimes said that an ideal stochastic number is *memoryless*, because each bit has no statistical dependence on the sequence history. A non-ideal stochastic number does depend on the sequence history, and can be considered as a hidden Markov model.

Fig. 1 A stochastic number generator that uses a comparator to convert k -bit binary number B to stochastic number X with $p_X = B/2^K$

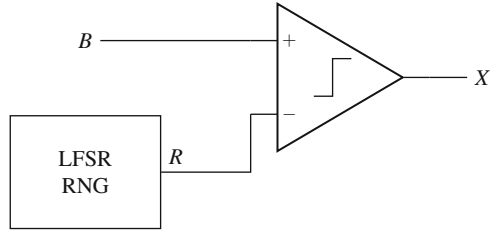


Table 1 Example stochastic number from an RNG sequence

Clock cycle	R (binary)	r (real value)	X
0	0100	0.25	0
1	0011	0.1875	1
2	0011	0.1875	1
3	0110	0.375	0
4	0111	0.4375	0
5	1101	0.8125	0
6	0111	0.4375	0

Stochastic numbers are usually produced using a uniform random number generator (RNG) based on a linear feedback shift register (LFSR), as described in our first example. In this tutorial, we examine only the most basic methods for generating stochastic bit streams. Stochastic number generation techniques are reviewed in more depth in chapter “[Generating Stochastic Bitstreams](#)”.

Example 1 (LFSR Stochastic Number Generator) The general schematic for an LFSR-based stochastic number generator is shown in Fig. 1. A K -bit unsigned binary integer B is supplied, indicating a probability $p_X = B/2^K$. At each clock cycle, the LFSR-based RNG produces a new K -bit random number R . If $R < B$ then the output for that clock cycle is $X = 1$, otherwise $X = 0$. Table 1 shows a hypothetical RNG sequence and stochastic output X for a 4-bit input probability of 0.3125. The four-bit fractional binary representation is $B = 0101$, interpreted as $\frac{0}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16}$. The output sequence in Table 1 has a mean value of $1/3$, which is close to the specified value of 0.3125.

Stochastic Numerical Formats

Binary sequences can directly represent positive real numbers between zero and one. In order to represent larger numbers, they must be mapped onto the unit interval. This can be done in a variety of different ways, resulting in distinct numerical representations or formats. Some of the more common formats are defined in this section. We begin with the most common *unipolar* format.

Definition 3 (Unipolar Stochastic Number) Given a non-negative real number x and a positive real parameter M , such that $0 \leq x \leq M$, the stochastic number X is said to be unipolar if $x = Mp_X$.

The unipolar format gives the most direct mapping from real intervals to stochastic numbers. Unipolar arithmetic is the easiest format to analyze, since there is a simple proportional mapping between the real number and the sequence probability. In some applications it is convenient to use a unit scale factor ($M = 1$), so that all values are restricted to the domain between zero and one. Throughout this chapter, unless otherwise specified we will assume unit scale factors in the examples.

A major drawback to the unipolar format is that it is restricted to non-negative real numbers. This motivates us to consider an alternative *bipolar* format, where each sequence bit with binary value 1 or 0 is understood to encode a real value of $+M$ or $-M$, respectively. The corresponding real number is encoded in the sequence average.

Definition 4 (Bipolar Stochastic Number) Given a real number x and positive real parameter M , such that $-M \leq x \leq M$, the stochastic number X is said to be bipolar if $x = M(2p_X - 1)$; equivalently $p_X = 0.5(1 + x/M)$.

As with the unipolar representation, in this chapter we will assume a unit scale factor unless otherwise specified. To give a simple example of a bipolar stochastic number, the binary sequence $X = 011000$ has probability $p_X = 2/6$. Assuming the unit scale factor ($M = 1$), the corresponding real value is $x = 2(2/6) - 1 = -1/3$. We can alternatively map the binary sequence to its corresponding bipolar sequence $-1, +1, +1, -1, -1, -1$, which has average $-2/6 = -1/3$.

We next consider formats suitable for representing very large numbers. We first consider the *Likelihood Ratio* format, defined in Definition 5. As a shorthand, we will refer to stochastic numbers encoded in this format as *ratioed* numbers.

Definition 5 (Likelihood Ratio (LR) Stochastic Number) For a real number x , $0 \leq x < \infty$, the stochastic number X encodes a likelihood ratio if $x = p_X/(1 - p_X)$; equivalently $p_X = x/(1 + x)$.

The LR format can be understood by analogy to a floating-point binary format. Precision for the mantissa is very limited, but the numerical range is expansive. LR formats are used in some algorithms for estimation, inference and error correction. A closely related format is the Log-Likelihood Ratio format, which offers better accuracy for numbers close to zero:

Definition 6 (Log Likelihood Ratio (LLR) Stochastic Number) For a real number x , $0 \leq x < \infty$, the stochastic number X encodes a log likelihood ratio if $x = \log(p_X/(1 - p_X))$; equivalently $p_X = e^x/(1 + e^x)$.

To simplify transformations between LR and LLR representations, we will sometimes use the notation ℓ_X to mean an LR corresponding to stochastic number

Table 2 A comparison of stochastic numerical formats

p_X	Unipolar x	Bipolar x	LR x	LLR x
0	0	-1	0	$-\infty$
0.25	1/4	-1/2	1/3	-1.4
0.33	1/3	-1/3	1/2	-0.7
0.5	1/2	0	1	0
0.66	2/3	1/3	2	0.7
0.75	3/4	1/2	3	1.4
1	1	1	∞	∞

X , and L_X to mean the LLR corresponding to X . This simplifies the interpretation of circuits in the LLR domain, since

$$L_X = \log \ell_X. \tag{1}$$

Among conventional algorithms, LLR representations are perhaps more commonplace than LR. The log scale provides a useful means of compressing large numerical ranges. To help the reader get a sense of these numerical formats, Table 2 relates sequence probabilities to the corresponding real value in each representation. Unit scale factors are assumed for the unipolar and bipolar formats. Other formats are possible beyond what is presented here, for instance we could define a log-probability format similar to the LLR format.

Stochastic Arithmetic Operations

We now consider a series of examples illustrating the most basic stochastic arithmetic operations. In the simplest cases, stochastic numbers are filtered through ordinary logic networks to perform standard operations like addition and multiplication. If the inputs are ideal stochastic numbers, and the logic network contains purely combinational operations with no memory or feedback paths, then the network’s outputs are also ideal stochastic numbers with the desired Bernoulli properties. The combinational examples are mostly simple applications of probability theory, but can become more complicated when interpreted in the context of different numerical formats.

Example 2 (NOT Operation (Bipolar Negation, Ratioed 1/x))

Our first and simplest example is the NOT operation, where all the bits in a stochastic bit stream are inverted. The output stream’s probability is related to that of the input stream by

$$p_Q = 1 - p_X. \tag{2}$$

A numerical example of NOT gate’s stochastic operation is shown in Fig. 2.

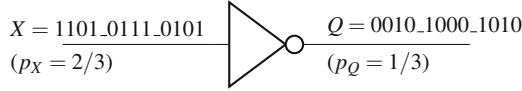


Fig. 2 The NOT gate as a stochastic $1 - p_X$ operation. For the bipolar format, $x = 1/3$ and $q = -1/3$ ($q = -x$). For the ratioed format, $x = 2$ and $q = 1/2$ ($q = 1/x$). For the LLR format, $x = 0.693$ and $q = -0.693$ ($q = -x$)

Unipolar Case For a unipolar stochastic input with non-unit scale constant M , the equivalent real-valued output can be expressed as

$$\frac{q}{M} = 1 - \frac{x}{M}$$

$$\Rightarrow q = M - x.$$
(3)

Here we assume that x and q have the same scale constant.

Bipolar Case For a bipolar input, inverting the bits of the unipolar sequence has the effect of flipping all the signs in the bipolar sequence. The NOT operation therefore negates the bipolar value.

Ratioed Case Substituting the expressions $p_X = x/(x + 1)$ and $p_Q = q/(q + 1)$, we find that

$$\frac{q}{q + 1} = 1 - \frac{x}{x + 1} = \frac{1}{x + 1}$$

$$\Rightarrow q = \frac{1}{x}.$$
(4)

LLR Case Taking the logarithm of the ratioed expression gives

$$L_Q = \log \frac{1}{\ell_X}$$

$$= -\log \ell_X$$

$$= -L_X$$
(5)

If both X and Q are LLR stochastic numbers, then the corresponding real-valued computation is

$$q = -x.$$
(6)

Example 3 (AND Operation (Unipolar Multiplier))

Given two independent ideal stochastic numbers A and B as inputs, the AND operation has output probability $p_Q = p_A p_B$. A numerical example for the case $1/2 \times 2/3 = 1/3$ is shown in Fig. 3.

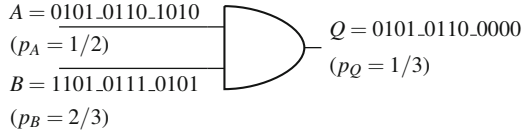


Fig. 3 The AND gate as a unipolar stochastic multiplier. For the bipolar case. $a = 0$, $b = 1/3$ and $q = -1/3$. For the ratioed case, $a = 1$, $b = 2$ and $q = 1/2$

Unipolar Case If the inputs are unipolar stochastic numbers, then the AND gate can be interpreted as a multiplier. If the unipolar inputs have scale constants M_A and M_B , then the output scale constant is $M_Q = M_A M_B$. Supposing $a = 3$ with $M_A = 6$, and $b = 2$ with $M_B = 3$, then the output is expected to be $q = 6$ with $M_Q = 18$. This is consistent with the example probabilities shown in Fig. 3.

Bipolar Case The bipolar case is more complicated, and is left as an exercise for the reader. Given bipolar inputs with unit scale constants, the reader should find that $q = (a + b + ab - 1) / 2$.

Ratioed Case The behavior is also interesting for ratioed stochastic numbers:

$$\begin{aligned}
 p_Q &= p_A p_B \\
 \Rightarrow \left(\frac{q}{q+1} \right) &= \left(\frac{a}{a+1} \right) \left(\frac{b}{b+1} \right) \\
 \Rightarrow q &= \frac{ab}{1+a+b}
 \end{aligned} \tag{7}$$

For large numbers (i.e. $a + b \gg 1$), this can be approximated as

$$q \approx \frac{ab}{a+b}, \tag{8}$$

which is useful for a variety of calculations.

LLR Case The LLR behavior has an interesting interpretation for small numbers, when $a \ll 1$ and $b \ll 1$:

$$\begin{aligned}
 L_Q &= \log \ell_Q \approx \log \ell_A \ell_B \\
 &= \log \ell_A + \log \ell_B \\
 &= L_A + L_B
 \end{aligned} \tag{9}$$

If A , B , and Q are all LLR stochastic numbers, then the corresponding computation is approximated by

$$q \approx a + b. \tag{10}$$

Example 4 (OR Operation)

The OR operation does not have a straightforward arithmetic application, but makes for an instructive exercise. Given two independent ideal stochastic inputs a and b , the OR operations has

$$p_Q = p_A + p_B - p_A p_B. \tag{11}$$

A numerical example is shown in Fig. 4, with input probabilities $p_A = 1/2$ and $p_B = 2/3$, with output probability $5/6$.

Note that if $p_A p_B \ll p_A + p_B$, then the OR gate can be considered an approximate adder for probabilities:

$$p_Q \approx p_A + p_B. \tag{12}$$

A numerical example for this approximation is shown in Fig. 5. In this case, the input probabilities are $1/6$ and $1/12$, and the output probability is their sum, $1/4$.

Unipolar Case For unipolar inputs with scale constants M_A, M_B , the output is

$$q = M_A M_B \left(\frac{a}{M_A} + \frac{b}{M_B} - \frac{ab}{M_A M_B} \right) \tag{13}$$

Repeating the same scale constants and values from Example 3, the output is expected to be $q = 18(3/6 + 2/3 - 6/18) = 15$ with $M_Q = 18$, which corresponds to $p_Q = 5/6$ as shown in Fig. 4.

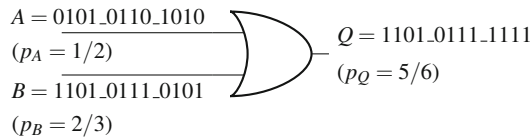


Fig. 4 Stochastic behavior of the OR gate. The unipolar input probabilities are $p_A = 1/2$ and $p_B = 2/3$, and the output probability is $p_Q = 5/6$. For the bipolar case, the corresponding values are $a = 0, b = 1/3, q = 1/3$. For the ratioed case, $a = 1, b = 2, q = 5$. For the LLR case, $a = 0, b = 0.693$, and $q = 1.609$

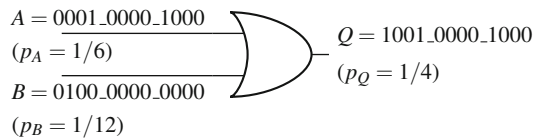


Fig. 5 The OR gate as an approximate adder for small input probabilities in the unipolar and ratioed representations. Numerical values for the unipolar case are indicated in the figure. For the ratioed case, $a = 1/5, b = 1/11$ and $q = 1/3$, which is close to $a + b$

Bipolar Case As with the bipolar AND operation from Example 3, the reader can verify that, given bipolar inputs with unit scale constants, the result is $q = (a + b - ab + 1) / 2$.

Ratioed Case If a and b are ratioed stochastic inputs, then the output is

$$\begin{aligned} \left(\frac{q}{q+1} \right) &= \left(\frac{a}{a+1} \right) + \left(\frac{b}{b+1} \right) - \left(\frac{ab}{(a+1)(b+1)} \right) \\ &= \frac{a+b+ab}{1+a+b+ab} \\ &\Rightarrow q = a + b + ab. \end{aligned} \tag{14}$$

If a and b are large numbers, $ab \gg 1$, then $q \approx ab$, so the OR gate acts as a multiplier for large numbers in the bipolar format. On the other hand, if $a \ll 1$ and $b \ll 1$ then $q \approx a + b$, so the OR gate acts as an adder for small numbers. Figure 5 shows example values for the ratioed case with small-valued inputs. In the example, the ratioed output is $q = 0.33$, which is somewhat close to the expected $a + b = 0.2909$.

LLR Case In the LLR format, the OR gate does not have a simple interpretation except in the case of large numbers. In that case, the computation is approximately

$$\begin{aligned} L_Q &\approx \log \ell_A \ell_B \\ &= \log \ell_A + \log \ell_B \end{aligned} \tag{15}$$

Then if A , B , and Q are all understood to be LLR stochastic numbers, then the OR gate acts as an LLR adder:

$$q \approx a + b. \tag{16}$$

Example 5 (MUX Operation (Unipolar/Bipolar Weighted Adder))

Multiplexing provides a simple means of mixing two stochastic numbers together. The result is a weighted adder circuit that works for both unipolar and bipolar numerical formats. The inputs a and b , along with the output q , are assumed to have the same numerical format. Regardless of chosen format, the MUX's select input, S , is a unipolar stochastic number with unit scale. The real value associated with S serves as a weight constant between 0 and 1 (most often 1/2). An example of MUX-based unipolar addition is shown in Fig. 6.

It is easy to see that the MUX's output is a weighted mixture of the two inputs, hence the output probability is

$$p_Q = (1 - p_S) p_A + p_S p_B. \tag{17}$$

Fig. 6 The MUX as a weighted stochastic adder in the unipolar format

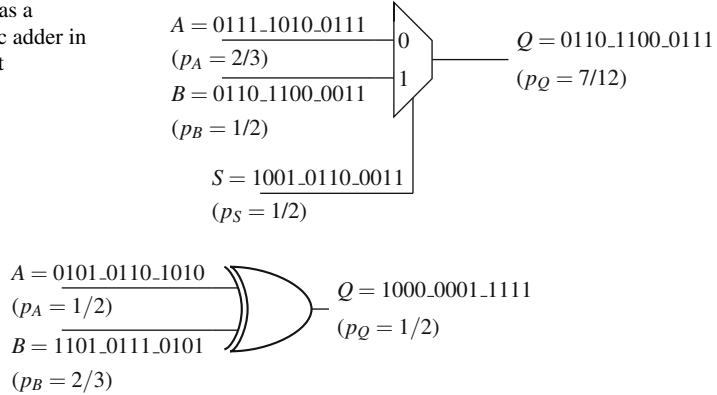


Fig. 7 Stochastic behavior of the XOR gate. The unipolar computation is indicated in the figure. In the bipolar domain, the corresponding values are $a = 0$, $b = 1/3$ and $q = ab = 0$. In the ratioed domain, the values are $a = 1$, $b = 2$ and $q = 1$

For the unipolar case, this can be viewed as either a weighted sum or an averaging of the two inputs. The reader can verify that the same result holds for the bipolar case. For ratioed and LLR stochastic numbers, the result is less convenient and is omitted from this example.

Example 6 (XOR Gate)

Another fundamental logic operation is the XOR gate. An example of its unipolar behavior is shown in Fig. 7.

Unipolar Case The output Q is 1 only if the two inputs are unequal, so the output probability is

$$\begin{aligned}
 p_Q &= p_A (1 - p_B) + (1 - p_A) p_B \\
 &= p_A + p_B - 2p_A p_B.
 \end{aligned} \tag{18}$$

Bipolar Case Substituting the bipolar mappings for a , b and q , we get

$$\begin{aligned}
 \frac{q+1}{2} &= \frac{a+1}{2} + \frac{b+1}{2} - \frac{(a+1)(b+1)}{2} \\
 \Rightarrow q &= -ab.
 \end{aligned} \tag{19}$$

The XOR gate is consequently interpreted as a negating multiplier for bipolar stochastic numbers. An ordinary (non-negating) multiplier is implemented by the complementary XNOR operation.

Ratioed Case The reader can verify that, for ratioed stochastic numbers, the behavior is expressed by

$$q = \frac{a+b}{1+ab} \tag{20}$$

which is very similar to the AND gate behavior. For small values of a and b , such that $ab \ll 1$, the XOR gate acts as an adder. For large values it implements the operation

$$q \approx \frac{a + b}{ab}. \tag{21}$$

The LLR behavior is not illuminating and is omitted from this example.

Operations with Memory

Memory has interesting consequences for stochastic computing circuits, some useful and others problematic. We begin with applications of the simplest memory element, the D flip flop or delay element. For a stochastic number X , we use the notation $D^\ell X$ to indicate a delay of ℓ clock cycles. If X is ideal, then the bit stream value of X should be statistically independent from $D^\ell X$ for any $\ell > 0$. This fact can be useful to perform normalizing operations, as demonstrated in the next few examples.

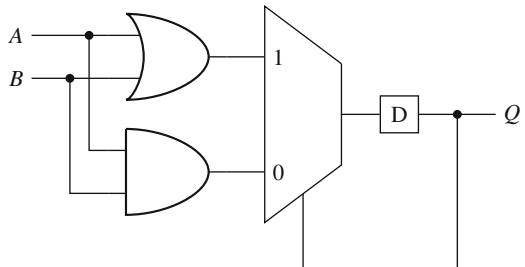
Example 7 (Ratioed Multiplier)

This example considers the sequential logic circuit shown in Fig. 8, where the inputs are assumed to be ideal stochastic numbers with a ratioed numerical format. The output Q plays two roles: as an output it is assumed to have a ratioed format, but it is also used as feedback to control the MUX's select input.

If A and B are ideal stochastic numbers, then by definition $D^1 A$ is statistically independent of A , and $D^1 B$ is statistically independent of B . Furthermore, A and B are independent of each other. During each clock cycle, the MUX selects between the independent bit streams, so the MUX output C inherits the ideal property that $D^1 C$ is independent of C . Since Q is a delayed replica of C , we conclude that $D^1 Q$ is statistically independent of Q . Since the AND, OR and MUX signals are all statistically independent from each other at each clock cycle, the unipolar probability solution is straightforward:

$$p_Q = p_Q (p_A + p_B - p_A p_B) + (1 - p_Q) p_A p_B. \tag{22}$$

Fig. 8 Ratioed stochastic multiplier circuit



After simplifying, we find that

$$p_Q = \frac{p_A p_B}{1 - p_A - p_B + 2p_A p_B}. \tag{23}$$

Lastly, we substitute the ratioed format expressions from Definition 5 and find that

$$\begin{aligned} \frac{q}{1+q} &= \frac{ab}{1+ab} \\ \Rightarrow q &= ab. \end{aligned} \tag{24}$$

Hence this circuit operates as a multiplier for ratioed stochastic numbers. In the LLR domain, it operates as an adder.

To give a precise example, we suppose the D element initially holds a zero value, and evaluate the circuit for two ratioed stochastic inputs representing $a = 4$ and $b = 1/2$. The corresponding probabilities are $p_A = 4/5$ and $p_B = 1/3$. Example sequences for these probabilities are

$$\begin{array}{r} A = 1110_1111_1011 \\ B = 0110_0000_0101 \\ \hline A \& B = 0110_0000_0001 \\ A | B = 1110_1111_1111 \\ \hline Q = 0_1110_1010_1011 \end{array}$$

Ignoring the initial state of Q , the remaining sequence has a mean of $8/12$, which corresponds to a real value of $q = 2$ in the ratioed format. This is the expected result.

Example 8 (J/K Flip-Flop)

The J/K flip-flop is a classical memory element that sets $Q_{\ell+1} := J\bar{K} + Q_{\ell}\bar{J}\bar{K} + \bar{Q}_{\ell}JK$. The typical schematic symbol and operation table are shown in Fig. 9.

Assuming that the inputs J and K are ideal stochastic numbers, the output Q is delayed by one clock cycle and therefore independent of the two inputs. Then the flip-flop’s logical definition maps directly to a probability expression:

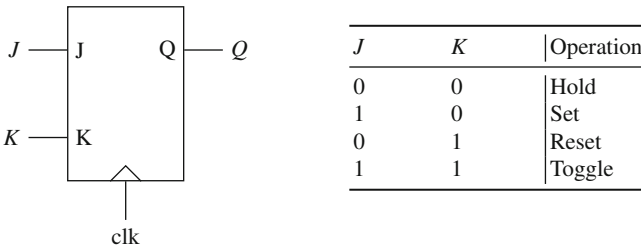


Fig. 9 Symbol and operation table for the J/K flip-flop

$$\begin{aligned}
p_Q &= p_J (1 - p_K) + p_Q (1 - p_J) (1 - p_K) \\
&\quad + (1 - p_Q) p_J p_K \\
&= p_Q + p_J - p_Q (p_J + p_K) \\
\Rightarrow p_Q &= \frac{p_J}{p_J + p_K}
\end{aligned} \tag{25}$$

This can be described as a unipolar normalizing operation, useful for computing the relative proportion of two signals.

Example 9 (Bayes' Law)

Bayesian inference is a point of intersection for many applications ranging from error correction to neural networks. For readers who may be unfamiliar with the concept, suppose we are uncertain about some fact, for instance I may worry that I've left the oven on. Let A be a binary random variable representing the event that the oven is on. Based on past experience, I can guess at a *prior probability* p_A that I left it turned on. Additionally suppose that I have a remote sensor B which is supposed to indicate whether the oven is on. The sensor is unreliable for some reason, and it is only accurate with probability p_B . According to Bayes' Law, I can combine my prior probability with my sensor evidence to obtain an improved *posterior probability* p_Q :

$$\begin{aligned}
p_Q = \Pr(A | B) &= \frac{\Pr(B | A) \Pr(A)}{\Pr(B)} \\
&= \frac{\Pr(B | A) \Pr(A)}{\sum_{a \in \{\text{on}, \text{off}\}} \Pr(B | A = a) \Pr(A = a)}
\end{aligned} \tag{26}$$

Now we substitute the probability values $p_A = \Pr(A = \text{on})$, $1 - p_A = \Pr(A = \text{off})$, $p_B = \Pr(B | A = \text{on})$, and $1 - p_B = \Pr(B | A = \text{off})$ to obtain a simplified expression for Bayes' Law:

$$p_Q = \frac{p_A p_B}{(1 - p_A)(1 - p_B) + p_A p_B} \tag{27}$$

This operation can be implemented a number of different ways. One of the common implementations uses a J/K flip flop as shown in Fig. 10. In this circuit, the J/K flip-flop sets $Q := 1$ if A and B are both 1, and resets $Q := 0$ if A and B are both 0. If $A \neq B$ in a given clock cycle, then the value of Q does not change. We assume that A and B are ideal stationary stochastic numbers, so that their statistics do not vary over time. In that case, it can be shown that Q converges in mean so that p_Q does not vary from one clock cycle to the next. Q is not ideal, but in any given clock cycle Q is statistically independent of A and B . Then the output probability can be expressed as

Fig. 10 Stochastic Bayes' Law circuit based on the J/K flip flop. This circuit also works as a ratioed multiplier

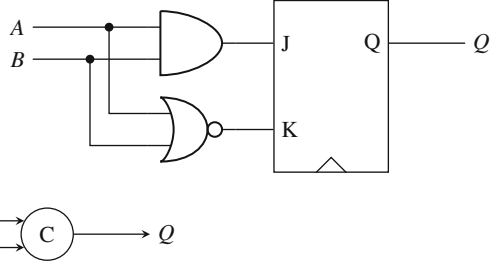


Fig. 11 Muller C-element device symbol. The C-element is a classical asynchronous gate functionally equivalent to the Bayes' Law circuit in Fig. 10

$$\begin{aligned}
 Q_{\ell+1} &= AB + \bar{A}BQ_{\ell} + A\bar{B}Q_{\ell} \\
 \Rightarrow p_Q &= p_A p_B + (1 - p_A) p_B p_Q + (1 - p_B) p_A p_Q \\
 &= \frac{p_A p_B}{1 - p_B - p_A + 2p_A p_B}.
 \end{aligned}
 \tag{28}$$

This is the expression for Bayes' Law. By an interesting coincidence, the output probability is the same as for the ratioed multiplier in Example 7, so the ratioed multiplier can just as well be used to implement Bayes' Law. Conversely, the Bayes circuit can serve as a ratioed multiplier, or as an adder in the LLR domain.

The J/K-based circuit shown in Fig. 10 is functionally equivalent to a classic logic gate known as the Muller C-element, which is widely used in asynchronous digital circuits and has several alternative implementations. In future examples, we will use the C-element symbol shown in Fig. 11 to stand in for the J/K Bayes' Law circuit.

As a concrete example of the C-element's function, we consider input streams with probabilities $p_A = 5/12$ and $p_B = 1/3$. The output probability should be $p_Q = 0.263$, which is a little more than $1/4$.

$$\begin{array}{l}
 A = 0101_1100_0100 \ (5/12) \\
 B = 0011_0001_0010 \ (1/3) \\
 \hline
 Q = 0001_1100_0000 \ (1/4)
 \end{array}$$

In the ratioed domain, the corresponding values are $a = 0.7143$, $b = 0.5$ and $q = 1/3$, which is close to the product $ab = 0.3572$. The accuracy tends to improve as the stream length increases.

Example 10 (Toggle Flip-Flop)

The toggle flip-flop (TFF) is an interesting case where the output has probability $1/2$, independent of the input probability (so long as the input is non-zero). A numerical example is shown in Fig. 12. The TFF can be used to generate a known constant stochastic number without requiring an additional RNG.

Fig. 12 Toggle flip-flop example

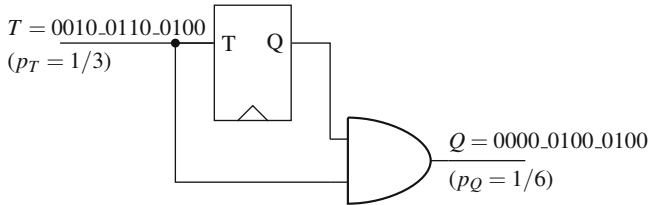
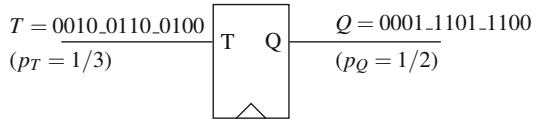


Fig. 13 TFF-based unipolar divide-by-two circuit

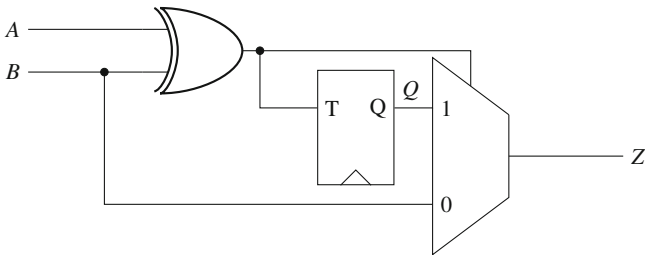


Fig. 14 TFF-based adder circuit for unipolar and bipolar formats

The main drawback to the TFF is that it introduces substantial correlation into the output sequence. For example, if the input probability is close to 1, then the TFF’s output will follow a regular 1-0-1-0 pattern, giving it a nearly deterministic behavior. This can potentially interfere with the accuracy of some circuits. Nevertheless there are some important applications as demonstrated in the following two examples.

Example 11 (Unipolar Divide-by-Two)

One immediate application of the TFF is a divide-by-two circuit shown in Fig. 13. Since the TFF generates a unipolar output with probability 1/2, this can be multiplied into the original input stream using an AND gate. If the circuit’s input is an ideal stochastic number, then the TFF’s delay ensures statistically independent inputs at the AND gate, which is required for proper function as a stochastic multiplier (consequences of statistical dependence are studied in Example 14). This circuit inherits some correlation effects from the TFF. For instance, if the input has probability 1, then the output will follow the same deterministic 1-0-1-0 pattern as the TFF.

Example 12 (Unipolar/Bipolar Adder)

A second TFF application is a non-weighted adder circuit, shown in Fig. 14, that works for unipolar and bipolar formats. In this circuit, when $A \neq B$ the TFF toggles

and its output Q is selected, so $Z := Q$. When $A = B$ the TFF is bypassed so $Z = B$. The output stream has probability

$$\begin{aligned} p_Z &= p_A p_B + (1/2)((1 - p_A) p_B + p_A (1 - p_B)) \\ &= (1/2)(p_A + p_B). \end{aligned} \tag{29}$$

As with the other TFF circuits, the output of this adder is not ideal. For instance, suppose that $p_A = 1$ and $p_B = 0$. In that circumstance, the TFF's output is always selected, and we again see the regular 1-0-1-0 pattern.

Correlation and Non-ideality

One of the greatest challenges for synthesizing scalable stochastic computing systems is that various correlations tend to emerge from the incremental operations. When stochastic numbers are non-ideal, many of the operations from section “[Stochastic Arithmetic Operations](#)” are invalidated, and the consequences can be difficult to analyze. In this situation, it may not be possible to directly feed the output from one operation into the input of another. In this section we begin with two examples to illustrate the problem, and the remaining examples present possible solutions. We will examine two major approaches, known as *isolation* and *regeneration*. We will also look at a few examples where the operations are insensitive to correlation or even benefit from certain kinds of correlation. A more detailed treatment of correlation effects is given in chapter “[Accuracy and Correlation in Stochastic Computing](#)”.

Example 13 (Autocorrelation in the Toggle Flip-Flop)

Non-ideality appears in the output of the TFF from Example 10, in the form of long-running sequence auto-correlation. If the TFF's input has a high probability, then the output tends to toggle during the majority of clock cycles. Frequent toggling causes it to follow a 1-0-1-0 pattern, occasionally disrupted by a repeated 1 or a repeated 0. On the other hand, if the TFF's input has a low probability, then toggling is rare, which causes long runs of 1's followed by long runs of 0's. Over a very large number of clock cycles, the sequence average eventually converges to 1/2, but the behavior is very different from a Bernoulli process and can give badly distorted computational results on finite time scales.

Figure 15 shows simulation results comparing the autocorrelation of an ideal stochastic number to that of a TFF output, when the input probability is 0.9. The TFF output exhibits a quasi-periodic autocorrelation due to the repeating 1-0-1-0 pattern.

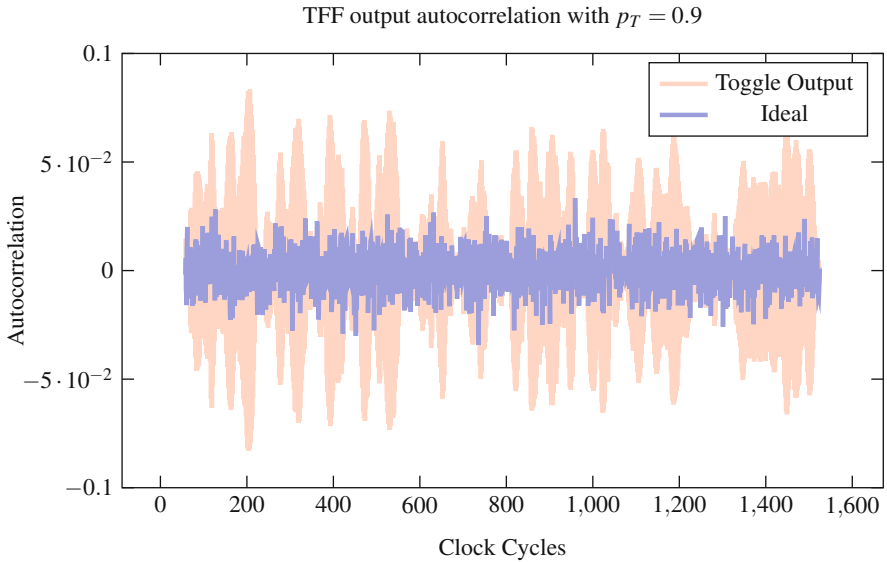


Fig. 15 Non-ideal autocorrelation in TFF output

Isolation Methods

Example 14 (Squaring Circuit)

Since an AND gate serves as a unipolar multiplier, it might seem natural to try and use it as a squaring circuit by feeding the same stochastic number into both inputs, as shown in Fig. 16.

In this instance we will not obtain the desired output statistics. Since the AND gate will always see the same value on both its inputs, the output simply follows the input. But if the input is an ideal stochastic number, then the delayed stream DX is statistically independent of X . By inserting a delay element, as shown in Fig. 17, we ensure statistical independence between the gate’s inputs so that the gate can operate properly as a multiplier.

Supposing that the D register initially contains a 0, the AND gate sees these inputs:

Fig. 16 Failed attempt at implementing the p_x^2 function

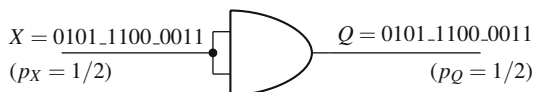
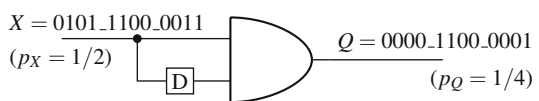


Fig. 17 Corrected squaring circuit using a delay register



$$\begin{array}{r} 0101_1100_0011 \\ \& 0010_1110_0001 \\ \hline = 0000_1100_0001 \end{array}$$

The delay method only works for ideal stochastic numbers. If successive bits are correlated, then the result cannot be trusted. Consider the case when all 1s appear at the beginning of the sequence, like this:

$$\begin{array}{r} 1111_1100_0000 \\ \& 0111_1110_0000 \\ \hline = 0111_1100_0000 \end{array}$$

Here the output sequence is nearly the same as the input sequence, and has $p_Q = 5/12$ which is approximately p_X rather than p_X^2 .

Example 15 (Sample Memory Regeneration)

In Example 14, we saw a delay-based solution to remove statistical dependence. This doesn't work if there is autocorrelation in the stochastic sequence. But in many cases the autocorrelation has a limited depth, so that it effectively vanishes after some time. Then X is approximately independent of $D^\ell X$ for some $\ell > 0$. So for non-ideal stochastic numbers we can sometimes resolve correlation by using a larger number of D flip-flops in a shift-register configuration, like the one shown in Fig. 18. It is also possible that the input stream has a periodic autocorrelation. For example, the output from a TFF tends to repeat a distinct 1-0-1-0 pattern. This can be disrupted by using a randomly addressed MUX to vary the shift-register's depth as shown in Fig. 19.

With the random-addressed delay chain, periodic autocorrelation can often be mitigated, though not fully resolved, given sufficient memory depth. For sequences with long runs of 1s and 0s (e.g. the TFF output induced by a low-probability input sequence), the memory's depth has to be much greater than the expected run length. The MUX width does not necessarily need to be large, since the MUX inputs can be tapped from widely spaced registers.

As an example, the random-addressed delay chain was simulated for regenerating the output from a toggle flip flop, with a low input probability of $p_X = 0.05$. The

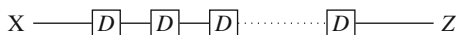


Fig. 18 Long delay chain to isolate long-lasting autocorrelation

Fig. 19 Delay chain with random-address output to isolate periodic autocorrelation

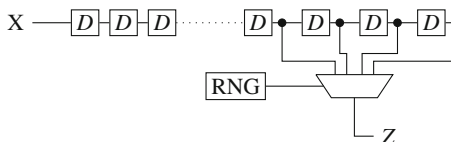


Fig. 20 Example of isolation to reduce autocorrelation in a TFF output

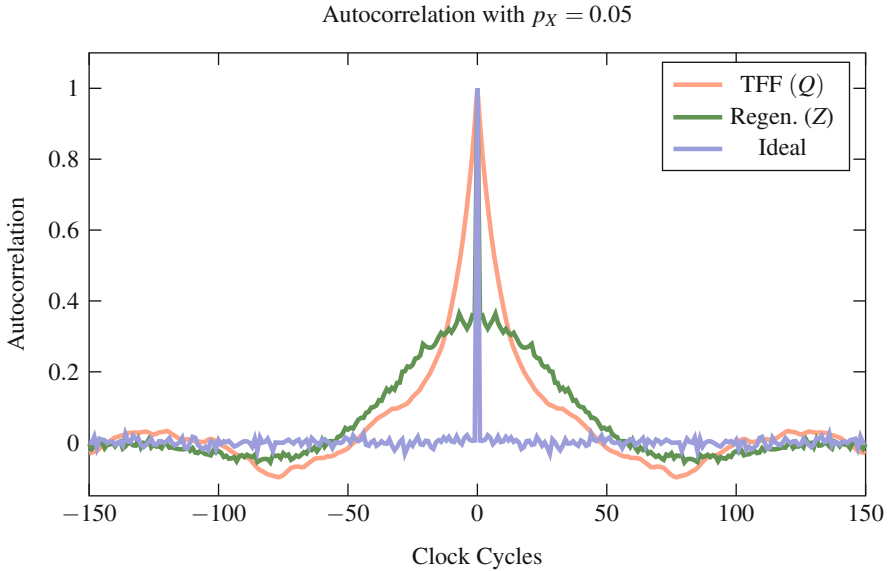
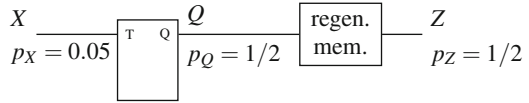


Fig. 21 Change in TFF autocorrelation after applying the isolation method shown in Fig. 20 when the input probability is close to zero

schematic for this simulation is shown in Fig. 20. For this input probability, the TFF output Q has runs of 1s and 0s with an average run length of about 20. This results in a very long-lived autocorrelation for Q , as seen in the simulation results shown in Fig. 21. After regeneration via the sample memory, the correlation effect is reduced but not completely eliminated.

This simulation used a memory depth of 64 delays, with 8 taps spaced 7 delays apart, at delay indices 64, 57, 50, \dots , 8. The simulation was repeated for a high input probability, $p_X = 0.95$, and we see from the result in Fig. 22 that the regeneration is more successful. Again, however, the autocorrelation is reduced but not entirely eliminated.

Regeneration Methods

To remove autocorrelation from a stochastic number, there are several sample-and-regenerate solutions that produce a new statistically independent bit stream with the same statistics as the original. All regeneration methods introduce some latency into

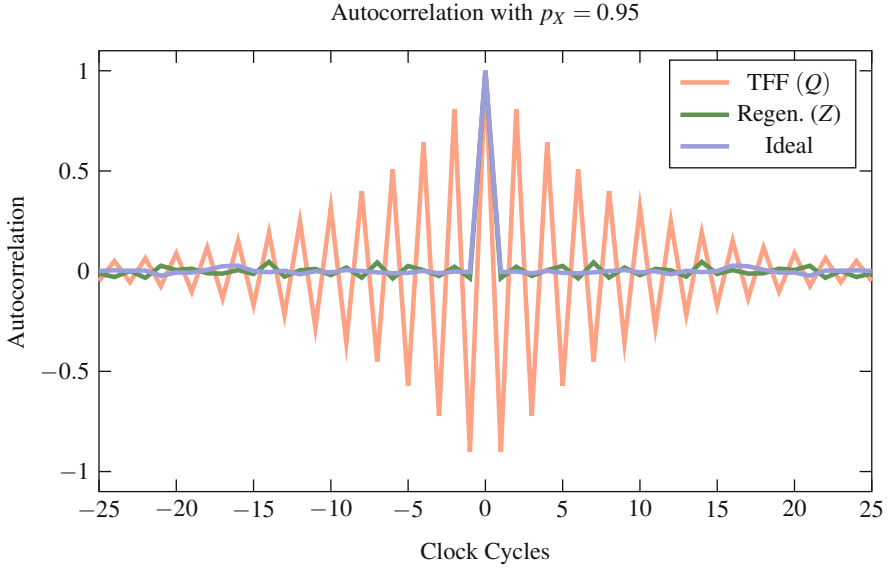


Fig. 22 Change in TFF autocorrelation after applying the isolation method shown in Fig. 20 when the input probability is close to one

the system, since it is not possible to instantaneously sample the input statistics. In some special cases, the latency effects may prove advantageous, for instance it may help stabilize iterative calculations where feedback is present. But most of the time the latency is a drawback.

Example 16 (Counter-Based Regeneration)

The most direct means of regenerating a stochastic number is to use a binary counter to estimate the input stream's probability. A generic schematic for this approach is shown in Fig. 23, where the output stream Z is generated using a uniform RNG and a comparator. This circuit is a modification of the stochastic number generator from Example 1. If the counter has K bits, then the counter may accumulate for 2^K clock cycles, yielding the unsigned integer count C representing the total number of non-zero bits that occurred in that time window. Then the probability estimate is

$$\hat{p}_X = \frac{C}{2^K} \quad (30)$$

If the RNG produces uniformly distributed random numbers in the interval $[0, 2^K - 1]$, then the output stream has $p_Z = \hat{p}_X$.

There are several tradeoffs associated with the simple counter approach in Fig. 23. Since the counter must accumulate 2^K samples, the output probability can only be updated with a period of 2^K clock cycles. Alternatively, the probability

Fig. 23 Counter-based regeneration schematic

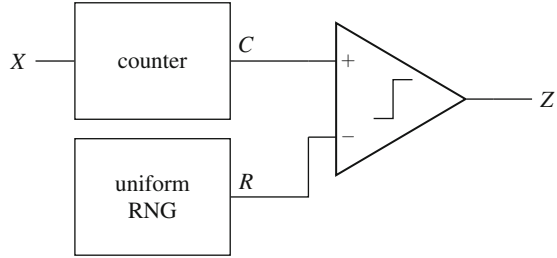
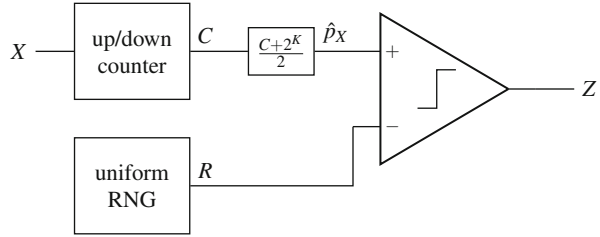


Fig. 24 Free-running regeneration based on up/down counter



estimate can be updated every clock cycle if a shift-register is used with depth 2^K , so that the oldest sample can be subtracted out at the same time as the newest sample arrives.

An alternative counter solution is use an up/down counter (UDC) as shown in Fig. 24. In this approach, the input sequence is treated as a bipolar stream. The UDC estimates the bipolar average, so the count C is treated as a signed integer. Whenever the input is $X = 1$, the counter increments; whenever $X = 0$ the counter decrements. Then the bipolar average is converted to a probability by

$$\hat{p}_X = \frac{C + 2^K}{2} \tag{31}$$

This mapping is trivial to implement. One major advantage for the UDC is that it can run continuously, providing a revised probability estimate every clock cycle, without the need for a shift register.

Feedback Methods

Example 17 (Tracking Forecast Regeneration)

A negative feedback approach, depicted in Fig. 25, can provide excellent characteristics in terms of both latency and gate complexity. Such an approach can be used to replace the counter from Example 16, instead using the estimator defined by

$$C(t + 1) = C(t) (1 - \beta) + \beta X(t), \tag{32}$$

Fig. 25 Simplified signal-flow schematic for a tracking forecast memory

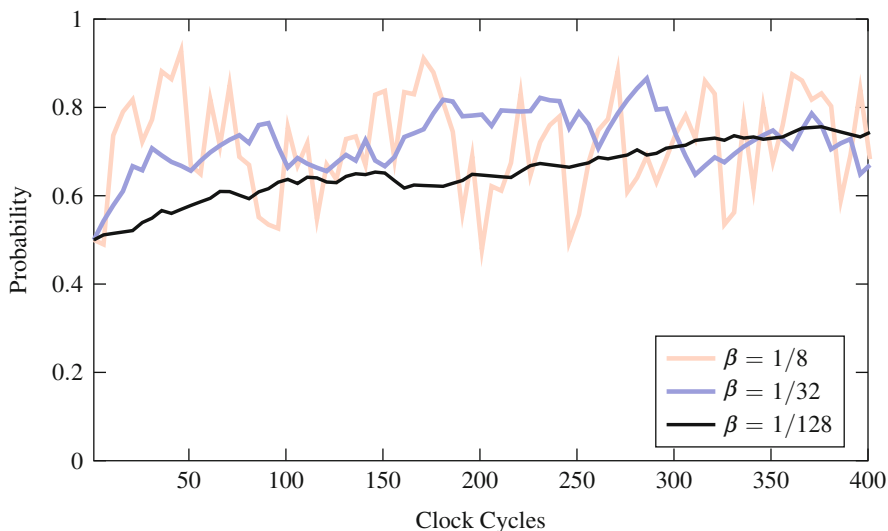
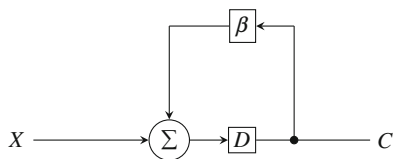


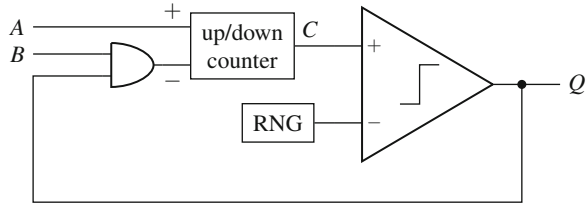
Fig. 26 Convergence behavior of tracking forecast memories

where β is a parameter, $0 < \beta < 1$, that controls the step size. In essence, this process acts as a low-pass filter to track the constant (or slow-changing) probability value from the fast-switching bit stream. A small β means slower tracking but better accuracy and stability. Feedback estimation is the basis of Tracking Forecast Memory (TFM) regeneration methods which have proved very effective in error correction decoders.

In practice, the β parameter can be chosen as a power of 2 in order to minimize implementation complexity. As a practical example, the generic TFM from Fig. 25 was simulated for three different values of β . The simulation results, shown in Fig. 26, demonstrate the time/stability tradeoff for estimating a stochastic number with $p_X = 0.75$. In this example, the counter's input is an ideal stochastic number with $p_X = 0.75$ from an initial value of 0.5. With smaller values of β , the circuit has higher latency but the estimate is more stable and accurate.

The preceding examples treat regeneration as a modular problem. It is also common to merge stochastic regeneration into the design of functional circuits. This opens the possibility of using feedback in interesting ways, as illustrated by the divider circuit in Example 18.

Fig. 27 Regenerative unipolar stochastic divider circuit



Example 18 (Regenerative Stochastic Divider)

Correlation can prove beneficial in certain feedback scenarios, such as the divider circuit shown in Fig. 27. An AND gate is used as a multiplier, and the up/down counter acts as a negative error amplifier. This circuit has one steady-state solution where $p_A = p_Q p_B$, so that the up/down counter is incremented and decremented at equal rates.

Applications

This section gives an introduction to various stochastic computing applications which also appear in later chapters of this book.

Error Correction

A particularly interesting application of stochastic computing is in the field of error correction decoding [4]. In many communication and memory systems, binary data is encoded by appending parity bits, so that the combined data word obeys prescribed structural rules, not unlike the rules of spelling in alphabetic writing systems. If errors occur in one or a small number of bits, the structural rules allow for those errors to be detected and corrected. As introductory examples, we consider the two simplest codes: the single-parity check (SPC) code and the repetition code.

Example 19 (Single Parity Check Code)

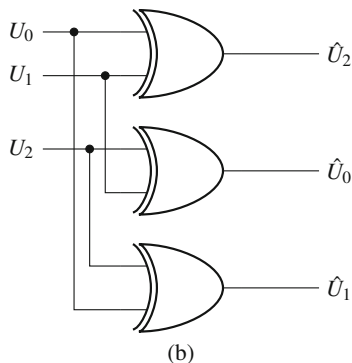
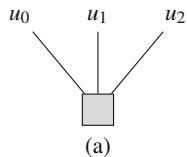
Given two bits of source data u_0 and u_1 , we wish to detect errors by appending a single parity bit u_2 to the sequence. The value of u_2 is determined by the parity rule

$$u_2 = u_0 \oplus u_1 \tag{33}$$

where \oplus is addition modulo 2. So the possible sequences are

u_0	u_1	u_2
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 28 Graph symbol (a) and stochastic implementation (b) for a single parity check code decoder



This behavior corresponds to that of an XOR gate. Now suppose that we are able to retrieve only two bits of the sequence, say u_0 and u_2 , but the remaining bit u_1 is unknown. In that situation we can infer the value of u_1 by applying the parity rule: $u_1 = u_0 \oplus u_2$.

Now let's alter the situation and say that our retrieval system is able to estimate probabilities for each of the three bits, p_0 , p_1 , and p_2 , where each p_j indicates the probability that $u_j = 1$, based on a measured signal. In this situation we can estimate the *extrinsic probability* of one bit (say, u_1) based on evidence from the other two:

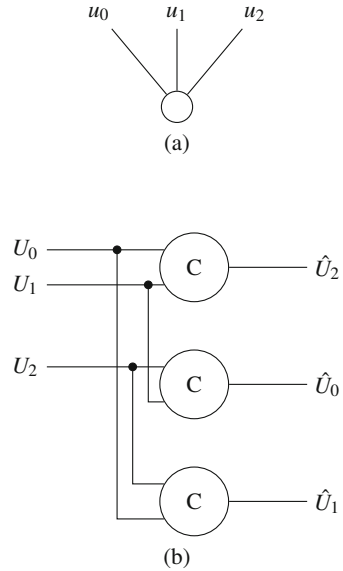
$$p_{1|0,2} = p_0(1 - p_2) + (1 - p_0)p_2. \quad (34)$$

This calculation is an instance of the widely used *belief propagation* algorithm. In the context of stochastic computing, it has the form of an XOR operation for unipolar stochastic numbers. So we can obtain the extrinsic probabilities for all three bits by using three XOR operations. In the literature on error correction, this behavior is usually packed into the simple square symbol shown in Fig. 28a; the full implementation is shown beneath in Fig. 28b. In this example the “hat” in \hat{U}_0 , \hat{U}_1 and \hat{U}_2 indicates estimation of those bits based on extrinsic information.

Example 20 (Repetition Code)

Given one bit of source data u_0 , we wish to detect and correct errors by appending two parity bits, u_1 and u_2 , to the sequence. Both u_1 and u_2 are copies of the original data bit. In this case the possible sequences are

Fig. 29 Graph symbol (a) and stochastic implementation (b) for a repetition code decoder



$$\begin{array}{ccc} u_0 & u_1 & u_2 \\ \hline 0 & 0 & 0 \\ 1 & 1 & 1 \end{array}$$

As we did in Example 19, suppose that we are able to retrieve only u_0 and u_2 , but the remaining bit u_1 is lost. We can infer the value of u_1 only if $u_0 = u_2$, then u_1 must have the same value. Now suppose the retrieval system estimates bit probabilities p_0 , p_1 , and p_2 . In the belief propagation algorithm, we use Bayes’ Law to obtain the extrinsic probabilities. For bit u_1 , the extrinsic probability is

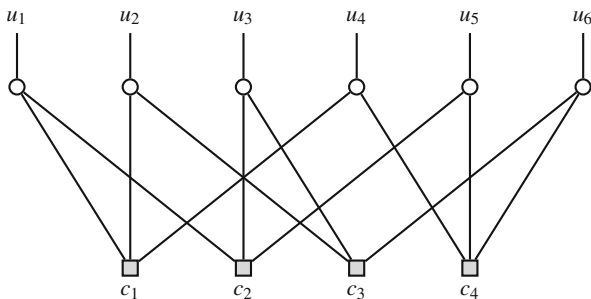
$$p_{1|0,2} = \frac{p_1 p_2}{p_0 p_2 + (1 - p_0)(1 - p_2)}. \tag{35}$$

For unipolar stochastic numbers, this calculation is implemented by the C-element circuit from Example 9. The compact symbol and its implementation are shown in Fig. 29a, b, respectively.

Example 21 (Low-Density Parity-Check (LDPC) Code)

A practical example of high-performance error correction is the LDPC code, which is adopted as an option in most modern high-capacity data communication standards. In this example we will only describe some of the basic features of stochastic LDPC decoders. For more information, the reader is referred to the substantial literature on the topic, e.g. [11, 12]. More topics on stochastic LDPC decoding are explored in chapter “[Stochastic Decoding of Error-Correcting Codes](#)”.

Fig. 30 Tanner graph, also called a factor graph, representing an LDPC code



An LDPC code can be described as an interlocking network of SPC and repetition codes. The structure is usually modeled by the Tanner graph, like the example shown in Fig. 30.

This graph has two major components: *symbol nodes* (circles) and *parity-check nodes* (squares). Each symbol node is essentially a repetition sub-code, and each parity node is an SPC sub-code. These sub-codes are woven together to create a powerful composite code. For decoding information in a retrieval system, the square and circle nodes are implemented as instances of the circuits described in Examples 19 and 20, respectively. To correct errors in the received data, the belief propagation algorithm is applied locally at every node in the graph, and is iterated until either a valid codeword is obtained (success), or an iteration limit is reached (failure).

If all the stochastic operations were ideal, then the belief propagation operations for SPC and repetition codes would scale properly to obtain a larger belief propagation decoder for the LDPC code. Unfortunately the LDPC code's structure contains cycles which setup feedback paths in the decoding circuit. These cycles are closely related to the concept of *trapping sets*, which affect all LDPC decoding algorithms but are especially difficult for stochastic decoders [5].

The concept of a trapping set is simple: suppose the entire circuit has converged to a correct state except for a few nodes that are involved in a feedback cycle. Then we can remove most of the network and focus just on the nodes and edges within the cycle, like the one¹ depicted in Fig. 31. Once we've stripped away most of the graph, the remaining parity check nodes have essentially no function in the subgraph that remains. The equivalent circuit along the subgraph can therefore be simplified as cycle of symbol nodes, with an equivalent circuit shown in Fig. 32.

If each of the C-elements in this cycle has a zero output, then they will remain permanently locked in the zero state. This is because the J/K flip-flops (see Example 9) are only able to change state if both the node's inputs agree. But in this feedback situation, they can only agree on zero.

¹The subgraph associated with a trapping set should usually contain both degree-1 and degree-2 parity check nodes. Here we have omitted the degree-1 nodes since they have no relevant effect on the circuit's behavior.

Fig. 31 A short cycle in the Tanner graph that constitutes a trapping set

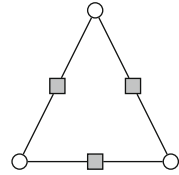
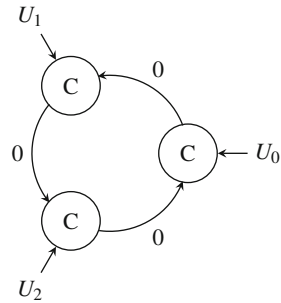


Fig. 32 A deterministic fixed-state on a trapping set in a stochastic decoder



To resolve the deterministic feedback in stochastic decoders, it is necessary to break every feedback cycle with a stochastic regeneration circuit. In practice, this usually means inserting a regeneration circuit along every edge in the Tanner graph. Various solutions are discussed in the literature which make use of counter-based regeneration, sample memory regeneration, and tracking forecast regeneration. The regeneration circuits can be merged with the Bayes’ Law circuits in interesting ways, leading to compact realizations. With regeneration circuits in place, the stochastic circuit approximates the behavior of a belief propagation decoder.

Image Processing

Image processing tasks require complicated arithmetic operations performed in parallel across every pixel in an image. Stochastic computing circuits bring the potential to implement these tasks with very low hardware cost [7]. These algorithms often require relatively low precision, which makes them attractive as a potential low-latency application for stochastic computing. In addition to the numerical formats and operations already discussed (e.g. addition, multiplication, etc), image processing algorithms may call for exponentiation, scaling, thresholding, sorting, and trigonometric functions, among others. To implement these operations, researchers have developed a number of sophisticated stochastic approaches. These implementations commonly mix different numerical formats into the same circuit, making them more advanced than previous examples in this chapter.

In this section, we will examine a few examples of advanced stochastic computing operations based on finite state machines (FSMs). The general state diagram for a stochastic FSM operation is shown in Fig. 33. In this section we consider

Fig. 33 A stochastic finite state machine

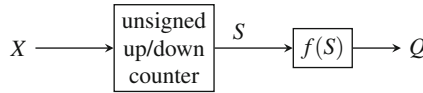
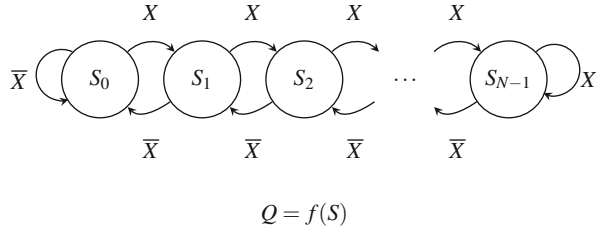


Fig. 34 Generic implementation of a stochastic finite state machine

state diagrams with a linear topology of N states labeled S_j , $j = 0, 1, \dots, N$. State transitions are determined by the stochastic input X . The system's stochastic output Q is a function of the state variable S . A variety of different functions can be achieved by adjusting the state depth N and the output mapping $f(S)$.

A possible FSM implementation is shown in Fig. 34. The circuit uses an unsigned up/down counter to realize the state-transition behavior. The counter increments whenever $X = 1$, and decrements when $X = 0$. Then the output mapping f is implemented as a combinational function of S .

This architecture proves to be configurable for a variety of interesting functions. A few empirical examples are offered below. A complete design methodology for stochastic FSMs is provided by Li et al. [8].

Example 22 (Stochastic Exponentiation)

For a bipolar stochastic input X and a unipolar stochastic output Q , a stochastic exponentiation function is obtained by the output mapping given below, which depends on an integer parameter G :

$$f(S) = \begin{cases} 1, & S < N - G - 1 \\ 0, & \text{otherwise} \end{cases} \tag{36}$$

The unipolar output has

$$p_Q = \begin{cases} \exp(-2Gx), & 0 < x \leq 1 \\ 1, & -1 \leq x \leq 0. \end{cases} \tag{37}$$

For proper function, a relatively large N is required, and it is assumed that $G \ll N$. For a concrete example, Fig. 35 shows simulation results for $N = 64$ with $G = 4$, $G = 6$ and $G = 8$. The expected output statistics (dotted) are shown alongside the measured output statistics (solid). The stochastic sequences were evaluated for ten thousand clock cycles. The circuit's accuracy is relatively good for low values of G

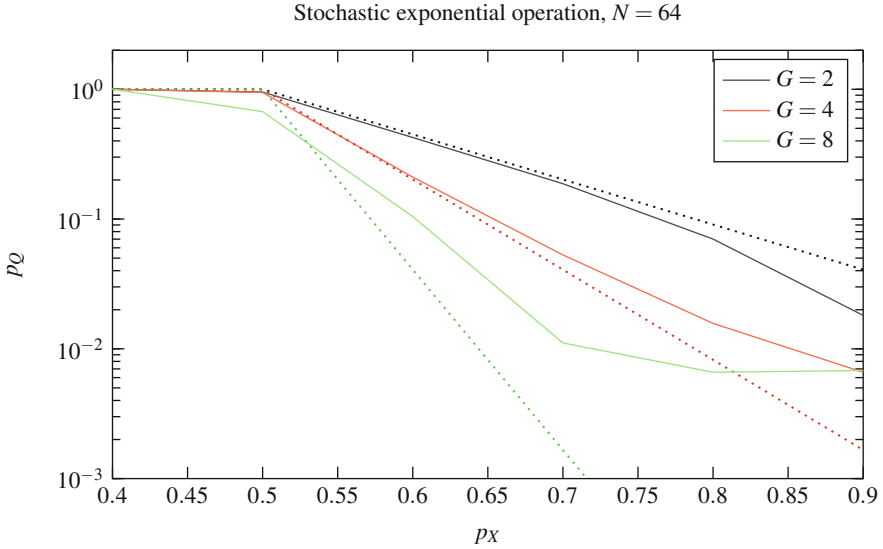


Fig. 35 Simulation results for a stochastic FSM-based exponentiation circuit. Dotted curves indicate ideal behavior, solid curves indicate measured results

and high values of p_Q , but degrades markedly when the output probability is low. The output from this circuit is also non-ideal, since the state-machine mapping tends to emit long runs of 1's and 0's, similar to the toggle flip-flop.

Example 23 (Tanh Function)

Supposing a bipolar stochastic input X and a bipolar output Q , a stochastic tanh function is achieved by this mapping:

$$f(S) = \begin{cases} 1, & S \geq N/2 \\ 0, & \text{otherwise} \end{cases} \tag{38}$$

This mapping yields the output function

$$p_Q = \tanh\left(\frac{N}{2}x\right). \tag{39}$$

This behavior was simulated for several values of N , with sequence lengths of ten thousand clock cycles. The results are shown in Fig. 36.

The simulation curves show that the tanh function becomes threshold-like for relatively small values of N . Since the input is a bipolar number, the threshold occurs close to a real value of $x = 0$, corresponding to the probability $p_X = 0.5$. This makes the tanh circuit useful as a component for stochastic comparators and sorting circuits.

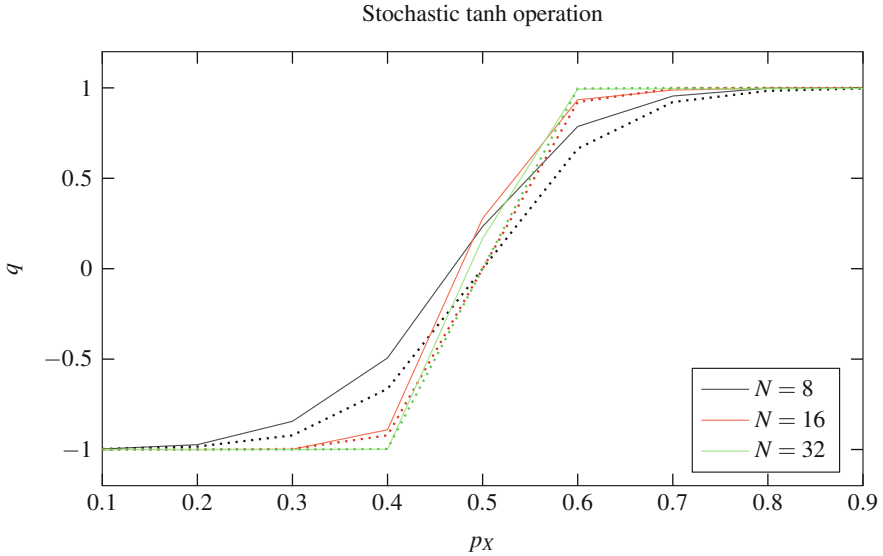
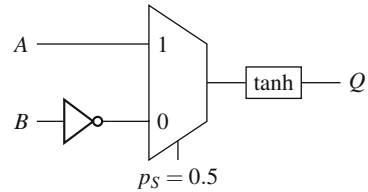


Fig. 36 Simulation results for a stochastic FSM-based tanh circuit. Dotted curves indicate ideal behavior, solid curves indicate measured results

Fig. 37 Bipolar stochastic comparator circuit



Example 24 (Stochastic Comparator and Median Filter)

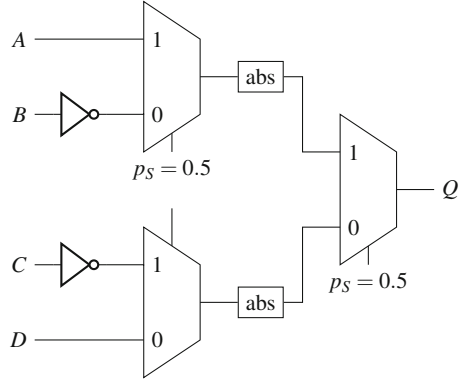
Given bipolar stochastic inputs A and B , the circuit shown in Fig. 37 implements a comparison by using the thresholding behavior of the tanh function. First, input B is negated by the inverter, and a MUX is used to produce the difference $0.5(A - B)$. The bipolar tanh output is nearly $q = 1$ when $A > B$, and nearly $q = -1$ when $A < B$. When A and B are nearly equal, the circuit produces a “soft” threshold output.

This circuit can be used to construct a sorting module, which has application in median filters for image denoising [7]. A median filter replaces each pixel’s intensity with the median intensity among the nearest neighboring pixels. The median value is detected using a tree of pairwise comparisons about each pixel neighborhood.

Example 25 (Stochastic Absolute Value and Edge Detection)

Given a bipolar stochastic input X , with a bipolar output Q , an FSM-based absolute value function is implemented by the state mapping

Fig. 38 Bipolar stochastic Roberts-cross operator



$$f(S) = \begin{cases} S \bmod 2, & S < N/2 \\ S + 1 \bmod 2, & S \geq N/2 \end{cases} \quad (40)$$

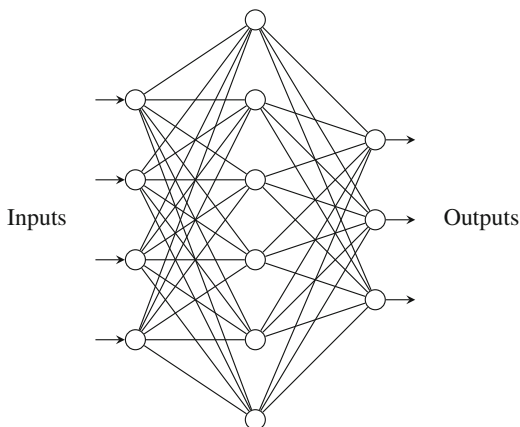
This operation is useful in computing an absolute-difference, $q = |a - b|$, which is needed for calculations like the Roberts Cross operator used in edge detection filters. The Roberts Cross operation takes four bipolar inputs, a, b, c, d , representing a neighborhood of four pixels. The operation calculates $q = 0.5 (|a - b| + |c - d|)$, hence it is comprised of two absolute-difference circuits and a bipolar adder [2].

Figure 38 shows a stochastic implementations of the Roberts Cross operation adapted from [2]. The circuit uses the same MUX-based difference operation seen in the stochastic comparator from Example 24. The MUX outputs are then fed into absolute-value operations and finally summed using a MUX-based bipolar adder (Example 5).

Neural Networks

There is an appealing analogy between stochastic computing and the stochastic on/off firing activity of biological neurons [2, 10]. A family of artificial neural networks called Spiking Neural Networks (SNNs) are designed to model this biological activity. In an SNN, each neuron emits a stochastic pulse train. An analog-like signal is encoded by the density of pulses, or by the average time between pulses. Although biological pulse trains are asynchronous, with pulses occurring in continuous time, we may think of synchronous stochastic numbers as an approximation to biological reality. Bayesian networks can also be considered as a class of neural networks, so we have already seen how stochastic computation applies in that domain. In this section we will examine some operations that are appropriate to SNNs and similar neural network families. Similar topics related to Brain-Inspired Computing are studied in chapter “Brain-Inspired Computing”.

Fig. 39 A typical neural network structure



The most common neural network applications use feed-forward topologies in which signals propagate in a single direction through one or more layers of neuron processors, as in Fig. 39. Each circle in the figure represents a “neuron,” and every edge represents a “synapse.” In stochastic neural networks, the synapse is essentially a wire that connects the stochastic output from one neuron to an input of another.

At minimum, a neuron must implement two steps: the first is a weighted summation of all synapse inputs. For a neuron with K synapse inputs, the summation y is given by:

$$y = \sum_{j=0}^{K-1} w_j x_j, \quad (41)$$

where w_j is a positive weight parameter and x_j is the real-valued signal arriving on the j th synapse.

The second fundamental operation is the *activation function* $f_A(y)$, which is a soft thresholding function that determines the probability of “firing” (i.e. emitting a non-zero binary value) on their neuron’s output. A typical neuron schematic is shown in Fig. 40.

A particular neural network design may choose from a variety of different activation functions. In most instances the *logistic function* is used:

$$f_A(y) = \frac{1}{1 + e^{-ky}}. \quad (42)$$

Other popular activation functions include tanh (which we saw in Example 23), arctan and other functions with a sigmoid shape. Also popular are non-linear or piecewise-linear functions, such as the “rectified linear unit” (ReLU), among others. The ReLU and logistic activation functions are plotted in Fig. 41.

Fig. 40 Generalized signal flow diagram for a single neuron

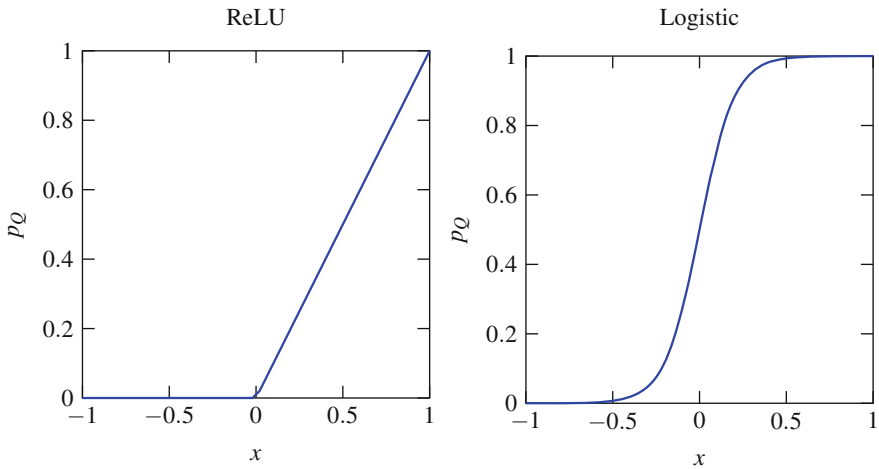
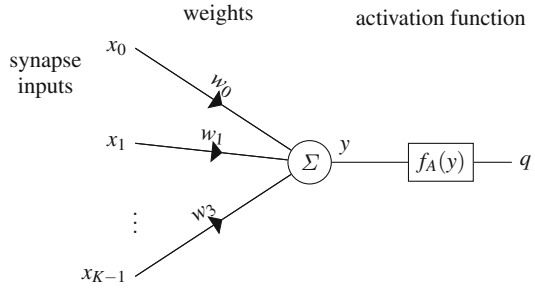


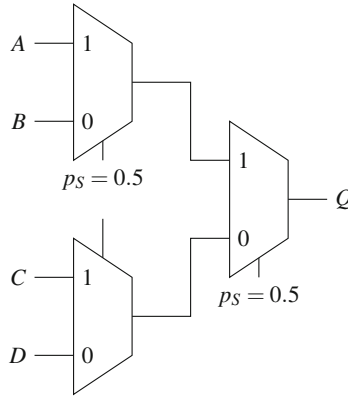
Fig. 41 Rectified linear unit (left) and logistic (right) activation functions

Example 26 (Summation of Sparse Pulses)

A MUX-based adder (Example 5) can implement weighted addition of unipolar or bipolar inputs, but is not suitable for a large number of inputs. This is because of the weighting: $p_Q = 0.5(p_A + p_B)$. Suppose there are four inputs. Then we could use a MUX tree as in Fig. 42. If there are K inputs, the summation is scaled by $1/K$. This is acceptable if the inputs have probabilities above $1/K$, but if the input probabilities are typically small, the summation result tends toward zero.

Since synapse pulses tend to be sparse, resulting in low probability values, it is better to use a direct summation without scaling. For very small unipolar inputs, an OR gate suffices, as illustrated in Fig. 43. A tree of OR gates provides an approximate sum over K inputs, provided each input conveys a probability much less than $1/K$. This is the opposite extreme from the MUX tree, and this condition is not generally guaranteed. Some stochastic neural networks address this problem by introducing new numerical formats to accommodate a wider dynamic range [2].

Fig. 42 Attenuation in a two-layer MUX-based stochastic adder



$$p_Q = 0.25(p_A + p_B + p_C + p_D) \tag{43}$$

Fig. 43 An OR gate as an approximate unipolar adder when probabilities are small

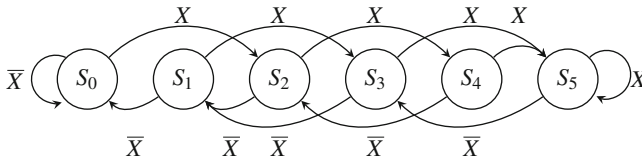
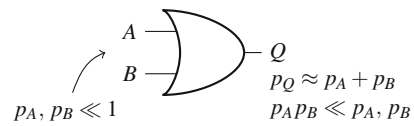


Fig. 44 Stochastic finite state machine design for the rectified linear unit function

Example 27 (Rectified Linear Unit (ReLU))

For a bipolar stochastic input and unipolar output, the ReLU function can be written as $f_A(y) = \max(y, 0)$ for $-1 \leq y \leq 1$. This function is approximated by an FSM with non-linear topology [10], as depicted in Fig. 44.

Here the output state mapping is

$$f(S) = \begin{cases} 1, & S = 5 \\ 0, & \text{otherwise} \end{cases} \tag{44}$$

Since the topology is not linear, this FSM needs a slightly different implementation from the up/down counter circuit used in section “Image Processing”. The FSM implementation was simulated for ten thousand samples per data point to obtain the results shown in Fig. 45. The FSM behavior is close to the ReLU function, but is not perfectly discontinuous. It could be said to behave more like a “soft” ReLU.

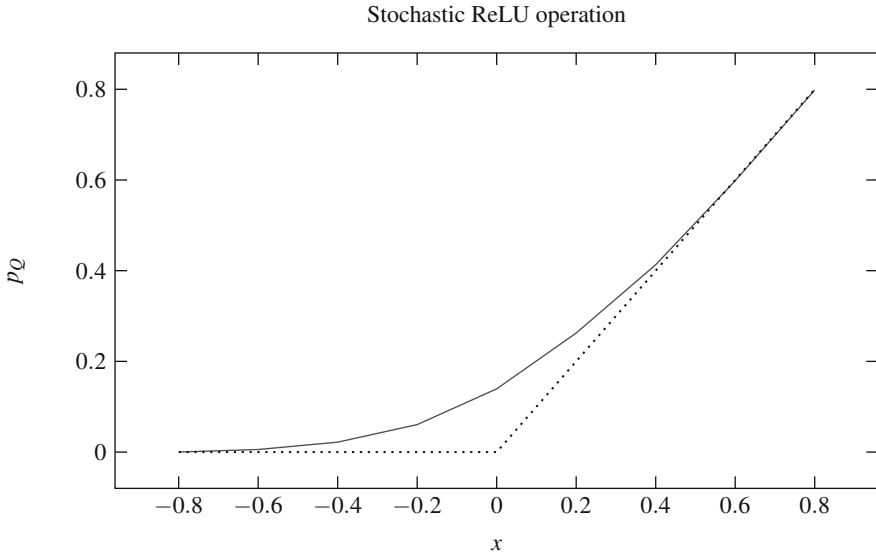


Fig. 45 Simulation results for a stochastic rectified linear unit

Nano-Technology and Post-CMOS Electronics

Many researchers have recognized the fault-tolerant attributes of stochastic computing circuits as one of their chief advantages. Conventional digital computers are highly sensitive to single-event upset (SEU) events, where a logic signal is momentarily flipped due to some internal noise or external interference. SEUs are unpredictable events that are catastrophic for conventional computers. By contrast, a small rate of SEU events has very little impact on stochastic computing circuits. Fault tolerance is particularly desirable for anticipated applications of nano-scale devices fabricated near the boundary of physical scaling limits. These devices are sensitive to the states of a small number of electrons and atoms, making them more susceptible to SEUs than historical bulk MOSFET devices.

Some nano-scale devices can be operated in a stochastic mode, where their outputs act like a Poisson process rather than a traditional logic operation. The process statistics can be predictably “tuned” by adjusting electrical parameters such as clock pulse height and width. This opens the possibility of using nano-devices as efficient stochastic number generators or for stochastic regeneration. “Native” stochastic computing has been examined for magnetic tunnel junctions [9], and for types of memristor devices [6], among others. Here we consider one example of a resistance-switching memristor device, which has been studied for application in spiking neural networks [1] and can also be applied for stochastic computation. Related topics are [RRAM Solutions for Stochastic Computing](#) and [Spintronic Solutions for Stochastic Computing](#).

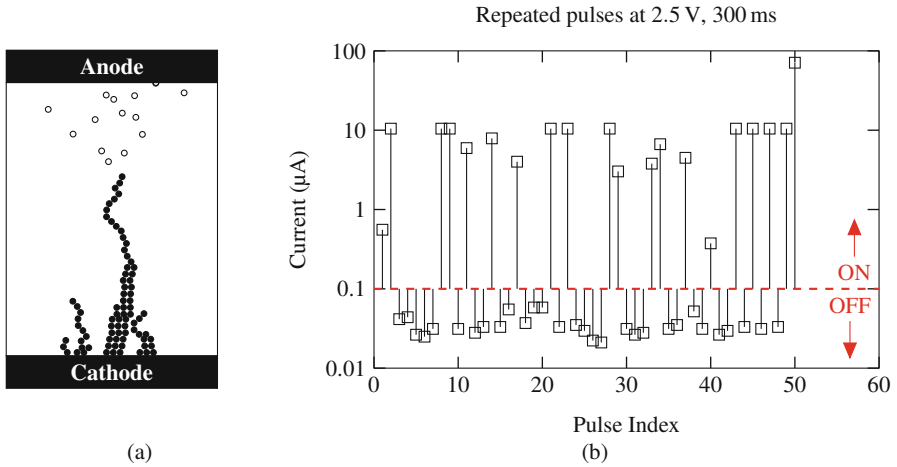


Fig. 46 (a) Formation of conductive filaments due to ion migration in a resistance switching device. (b) Experimental results showing stochastic sub-threshold switching as reported in [3]

Example 28 (Non-deterministic Memristor Devices)

A common memristor structure consists of dissimilar metal anode and cathode terminals separated by an oxide, as shown in the cross-sectional illustration in Fig. 46a. The structure can be fabricated on the order of a single nanometer, with very small oxide thickness between the terminals. When a positive voltage is applied between the anode and cathode, ion migration causes conductive filaments to aggregate within the oxide. This process eventually completes a low-resistance path between the terminals, referred to as the device’s “ON” state.

Filament formation is reversible by applying a negative voltage to the anode relative to the cathode. This causes ions to migrate away from the anode, restoring the device to a high-resistance “OFF” state. A switching memristor can be reliably switched ON and OFF by applying voltage pulses with amplitude that exceeds the device’s switching threshold voltage.

When pulses are applied with sub-threshold amplitudes, the device’s switching is random. The probability of switching is affected by the pulse amplitude, duration and by the density of pulses in time. Generally speaking, high amplitude, width and density all increase the probability of switching. Results from an example sub-threshold switching experiment are shown Fig. 46b (using data from Gaba et al. [3]). The device’s switching activity can be fit to a Poisson process, and the statistical parameters appear to be a predictable function of the pulse parameters. Statistical data from [3] are reproduced in Fig. 47, showing experimental data against a best-fit distribution.

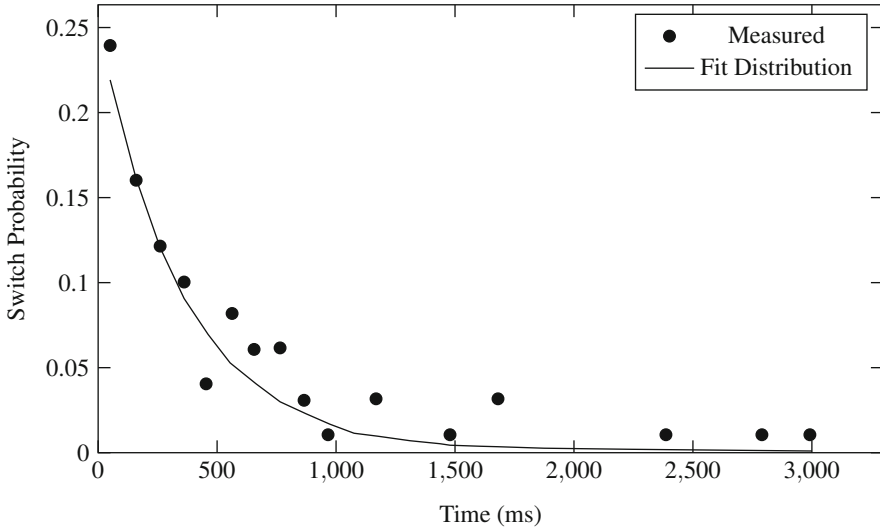


Fig. 47 Distribution of resistance switching delay due to sub-threshold pulsing, as reported in [3]

Conclusion

This chapter offered a brief introduction to the fundamentals of stochastic computing, and to topics and applications that are of current interest in the field. The remaining chapters of this book offer a more advanced look at the fundamental methods and applications of stochastic computing. While this tutorial focused mainly on functional analysis of stochastic circuits, there is still much to be learned about their precision, accuracy, efficiency, latency, and other tradeoffs that affect the competitiveness of stochastic computing against traditional methods. It is by no means a universal replacement for conventional computing architectures, but there are important niches where stochastic computing clearly shines, and this has motivated a continued and growing interest in the subject.

References

1. Al-Shedivat, M., Naous, R., Cauwenberghs, G., Salama, K.: Memristors empower spiking neurons with stochasticity. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on* **5**(2), 242–253 (2015). <https://doi.org/10.1109/JETCAS.2015.2435512>
2. Alaghi, A., Qian, W., Hayes, J.P.: The promise and challenge of stochastic computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* pp. 1–1 (2017). <https://doi.org/10.1109/TCAD.2017.2778107>

3. Gaba, S., Sheridan, P., Zhou, J., Choi, S., Lu, W.: Stochastic memristive devices for computing and neuromorphic applications. *Nanoscale* **5**(13), 5872–5878 (2013)
4. Gaudet, V.C., Rapley, A.C.: Iterative decoding using stochastic computation. *Electronics Letters* **39**(3), 299–301 (2003). <https://doi.org/10.1049/el:20030217>
5. Huang, K.L., Gaudet, V.C., Salehi, M.: Trapping sets in stochastic ldpc decoders. In: 2015 49th Asilomar Conference on Signals, Systems and Computers, pp. 1601–1605 (2015). <https://doi.org/10.1109/ACSSC.2015.7421418>
6. Knag, P., Lu, W., Zhang, Z.: A native stochastic computing architecture enabled by memristors. *IEEE Transactions on Nanotechnology* **13**(2), 283–293 (2014). <https://doi.org/10.1109/TNANO.2014.2300342>
7. Li, P., Lilja, D.J., Qian, W., Bazargan, K., Riedel, M.D.: Computation on stochastic bit streams digital image processing case studies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **22**(3), 449–462 (2014). <https://doi.org/10.1109/TVLSI.2013.2247429>
8. Li, P., Lilja, D.J., Qian, W., Riedel, M.D., Bazargan, K.: Logical computation on stochastic bit streams with linear finite-state machines. *IEEE Transactions on Computers* **63**(6), 1474–1486 (2014). <https://doi.org/10.1109/TC.2012.231>
9. Onizawa, N., Katagiri, D., Gross, W.J., Hanyu, T.: Analog-to-stochastic converter using magnetic tunnel junction devices for vision chips. *IEEE Transactions on Nanotechnology* **15**(5), 705–714 (2016). <https://doi.org/10.1109/TNANO.2015.2511151>
10. Smithson, S.C., Boga, K., Ardakani, A., Meyer, B.H., Gross, W.J.: Stochastic computing can improve upon digital spiking neural networks. In: 2016 IEEE International Workshop on Signal Processing Systems (SiPS), pp. 309–314 (2016). <https://doi.org/10.1109/SiPS.2016.61>
11. Tehrani, S.S., Mannor, S., Gross, W.J.: Fully parallel stochastic LDPC decoders. *IEEE Transactions on Signal Processing* **56**(11), 5692–5703 (2008). <https://doi.org/10.1109/TSP.2008.929671>
12. Tehrani, S.S., Naderi, A., Kamendje, G.A., Hemati, S., Mannor, S., Gross, W.J.: Majority-based tracking forecast memories for stochastic LDPC decoding. *IEEE Transactions on Signal Processing* **58**(9), 4883–4896 (2010). <https://doi.org/10.1109/TSP.2010.2051434>

Accuracy and Correlation in Stochastic Computing



Armin Alaghi, Paishun Ting, Vincent T. Lee, and John P. Hayes

Abstract This chapter begins by reviewing the sources of inaccuracy in stochastic computing, focusing on correlation, that is, dependencies among stochastic bit-streams. The measurement of correlation is considered, and the *SCC* metric is defined. The properties of correlation are then explored including some that have only been discovered recently. Correlation can be seen in two ways: either as corrupting a function f , or as changing f to a different, but potentially useful one. Therefore, to ensure that a stochastic circuit works as expected it is important to manage correlation appropriately. This can be done with correlation-controlling units, which must be used carefully to avoid unexpected functional changes and excessive hardware area or latency overhead. There are also cases where correlation has no effect at all (correlation insensitivity). Identifying such immunity to correlation can aid the design of stochastic circuits. Finally, design of stochastic number generators to provide specified levels of correlation is discussed.

Keywords Cross correlation · Autocorrelation · Random number sources · Correlation metrics · Correlation mitigation

Error Sources in Stochastic Circuits

Stochastic computing (SC) can be summarized concisely as computing with probabilities represented by pseudo-random bit-streams [7]. As discussed in the preceding chapters, its advantages include small size, low power, error tolerance, the use of standard digital components, and bio-compatibility. Its primary disadvantages are low accuracy, long computing times, and costly randomness sources. Consequently,

P. Ting · J. P. Hayes (✉)
University of Michigan, Ann Arbor, MI, USA
e-mail: paishun@umich.edu; jhayes@umich.edu

A. Alaghi · V. T. Lee
University of Washington, Seattle, WA, USA
e-mail: armin.alaghi@cs.washington.edu; vlee2@cs.washington.edu

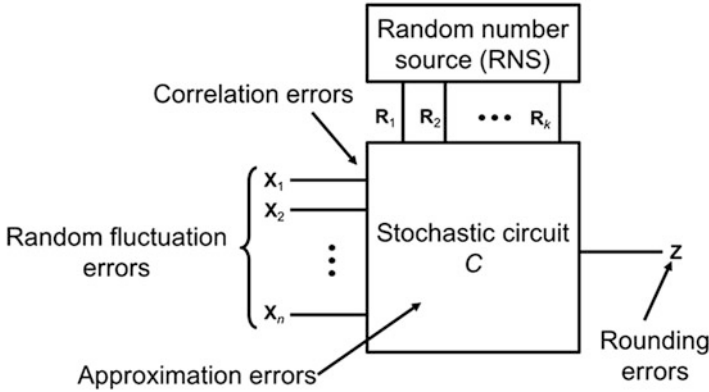


Fig. 1 Structure of a generic stochastic circuit annotated with known sources of inaccuracy

the design of a stochastic circuit involves complex trade-offs among accuracy, computing time and hardware cost. These trade-offs are vividly seen in the phenomenon of correlation, which is the focus of this chapter.

Figure 1 illustrates the general structure of a stochastic circuit. At its core is a combinational or sequential logic circuit C , which is supplied by a user with n stochastic numbers (SNs) $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$ of length N . \mathbf{X}_i is composed of N bits, where the t -th bit $\mathbf{X}(t)$ is assumed to be randomly generated in clock cycle t . These bits are, at least approximately, independent of one another, hence the name *stochastic*. \mathbf{X}_i has a numerical value denoted X_i which lies in the unit interval $[0,1]$. In SC's unipolar format, X_i is the probability p_{X_i} of a 1 appearing anywhere within \mathbf{X}_i . C also has a set of k ancillary inputs $\mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_k$, which are typically N -bit SNs of constant value $R_i = 0.5$. These constants are usually not user-supplied; rather they are an integral part of a stochastic circuit's design and enable the circuit to approximate a desired function to some desired level of accuracy. If the \mathbf{X}_i 's are derived from binary (base-2) numbers, additional randomness sources (not shown) are needed to perform binary-to-stochastic number conversion. In general, if C is combinational, it computes an arithmetic function of the form $Z(X_1, X_2, \dots, X_n, R_1, R_2, \dots, R_k)$. For example, when used as a stochastic adder, a two-way multiplexer computes the function

$$Z(X_1, X_2, R_1) = 0.5(X_1 + X_2) \quad (1)$$

where R_1 supplies the scaling factor 0.5.

An assumption made almost universally in determining the stochastic function computed by a circuit C is that its input bit-streams $X_1, X_2, \dots, X_n, R_1, R_2, \dots, R_k$ are independent. Roughly speaking, this means that the bit-streams should be as uncorrelated as possible. This requirement is assumed to be satisfied when all $n + k$ inputs are derived from $n + k$ separate Bernoulli randomness sources. *Correlation* therefore refers to stochastic signals that are related in some way, and it can take

many forms. Cross correlation, or simply correlation, occurs between two or more non-independent SNs. For example, the SN $\mathbf{X} = 10111010$ is highly correlated in a negative sense with $\mathbf{Y}_1 = 01000101$, since their 1s and 0s never overlap. The SN $\mathbf{Y}_2 = 10011000$ is also correlated with \mathbf{X} because its 1s always overlap the 1s of \mathbf{X} . The SN $\mathbf{Y}_3 = 01011101$, which is generated by rotating or shifting \mathbf{X} to the right by one bit, is not significantly cross correlated with \mathbf{X} , but the one-cycle-delayed version of \mathbf{Y}_3 is. Cross correlation may change the functionality of both combinational and sequential stochastic circuits by favoring certain input patterns. On the other hand, *temporal* correlation, or *autocorrelation*, refers to correlation between a bit-stream or part of a bit-stream and a delayed version of itself. For instance, $\mathbf{Y}_4 = 011001110$ contains some autocorrelation due to the fact that 01 is always followed by 1. Autocorrelation can severely affect the functionality of a sequential stochastic circuit by biasing it towards certain state-transition behavior.

Defining and measuring correlation is surprisingly difficult. A survey made in 2010 by Choi et al. [11] catalogs 76 different correlation metrics developed in different fields over many years, none of which is well suited to SC! Relatively easy to define is the independence or no-correlation assumption, which allows a stochastic circuit C 's SN inputs to be treated as Bernoulli processes, and the function of C to be expressed and analyzed using basic probability theory. For example, if two independent SNs \mathbf{X}_1 and \mathbf{X}_2 of value X_1 and X_2 , respectively, are applied to an AND gate, the output value Z is the arithmetic product X_1X_2 . This reflects the fact that the probability of the AND gate outputting a 1 is the probability of a 1 at the first input multiplied by the probability of a 1 at the second input, provided the inputs are not cross correlated. If \mathbf{X}_1 and \mathbf{X}_2 are correlated, Z can deviate from X_1X_2 in complex ways, as we will see shortly.

Random number sources (RNSs) play a central role in the design and operation of stochastic circuits. They provide the stochasticity needed by stochastic number generators (SNGs) to produce SNs with a sufficient level of independence, but they are a big contributor to overall hardware cost [25]. SC designers generally rely on linear feedback shift registers (LFSRs) as RNSs because of their relatively small size and low cost. An LFSR is a deterministic finite-state machine (FSM) whose behavior is pseudo-random, meaning that it only approximates a true random source [14]. An SC designer must usually optimize the use of RNSs in a way that provides sufficient randomness while meeting a cost budget.

SC is a type of approximate computing and trades off computational errors for other benefits. It has several error sources, as shown in Fig. 1. These error sources are peculiar to SC and do not include *physical* errors due to unreliable hardware or *soft* errors caused by environmental effects like cosmic radiation [8]. The errors in question are briefly summarized next.

Rounding Errors Errors caused by rounding or quantization reflect the fact that with N bits, a bit-stream can only represent exactly the $N + 1$ numbers in the set $S_N = \{0, 1/N, 2/N, \dots, (N-1)/N, 1\}$. If a desired number X is not in this set, then it must be rounded off to the nearest member of S_N . For instance, with $N = 16$ and

$X = 0.1555$, we can round X down to $2/16 = 0.1250$ or, slightly more accurately, round it up to $3/16 = 0.1785$. Rounding errors can be mitigated by increasing N to expand S_N . Note, however, that N must be doubled just to add 1 bit of precision to the numbers in S_N .

Approximation Errors These errors result from the fact that most arithmetic functions of interest cannot be implemented exactly by a stochastic circuit. As a result, they must be approximated by stochastic functions that are implementable. All stochastic function values must be scaled to lie in the unit interval $[0,1]$. Without constant \mathbf{R}_i 's as inputs, the only single-variable stochastic functions that can be combinational realized exactly are the trivial cases X and $1-X$. Hence, common arithmetic functions like X^2 , \sqrt{X} and $\sin(X)$ must be approximated by some synthesizable stochastic function of the form $Z(X, R_1, R_2, \dots, R_k)$. Only a few general methods for finding such functions are known; all are relatively complex and have particular design styles [4, 26]. For example, the ReSC synthesis method employs Bernstein polynomials with constant coefficients in the unit interval to approximate $Z(X)$ [25].

Random Fluctuations The (pseudo) random nature of the bits forming an N -bit SN \mathbf{X} as it emerges from an SNG is also a major error source. Fluctuations in \mathbf{X} 's bit-pattern cause its estimated or measured value \hat{X} to deviate from the target or exact value X . Since \mathbf{X} can have any of 2^N different bit-patterns, X and \hat{X} can differ significantly, especially when N is small. Figure 2 shows how three SNG-generated SNs fluctuate around their target value 0.5 as N changes. Such random fluctuation errors can be quantified by the mean square error (MSE) $E_{\mathbf{X}} = \mathbb{E}[(\hat{X} - X)^2]$. Like

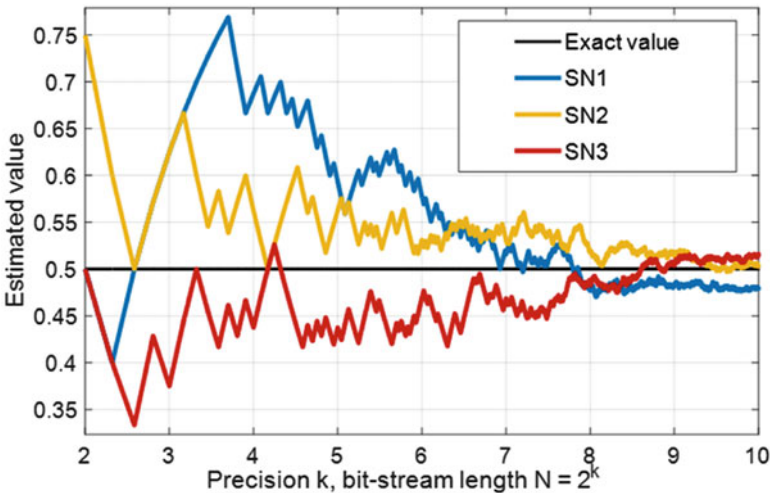


Fig. 2 Random fluctuations in three SNs with the exact value 0.5 as bit-stream length N increases [31]

rounding errors, random fluctuation errors tend to diminish with increasing N . Note, however, that when N is odd, \bar{X} must differ from $X = 0.5$ by at least one bit. Hence as N increases toward infinity, the graphs plotted in Fig. 2 continue to oscillate around 0.5 with a steadily decreasing MSE that approaches, but never reaches, zero.

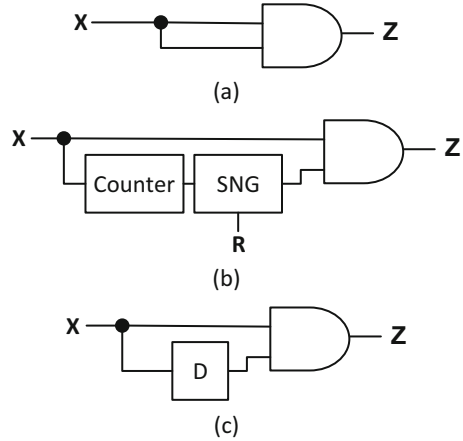
Constant-Induced Errors It was recently observed that the ancillary SNs $\mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_k$ (see Fig. 1) found in most SC designs are an unexpected and significant error source [31]. This is because their influence on the output value Z is subject to time-dependent random variations. Interestingly, constant-induced errors can be eliminated completely by removing the \mathbf{R}_i 's and transferring their function to sequential subcircuits inside C that track the behavior of the \mathbf{R}_i 's. A systematic algorithm called CEASE has been devised for efficiently removing constants and the errors they produce [31].

Correlation To maintain accuracy, it is often desirable that the bit-streams applied to a stochastic circuit retain their independence as they are being processed. This independence is reduced by correlation from several sources including: interactions among bit-streams during normal computation that introduce dependencies and similarities, poor randomness properties of individual RNSs that cause successive bits to be related, sharing of RNSs either directly or indirectly across the SNGs to reduce overall hardware costs, and temporal dependencies injected by sequential circuits. As a result, correlation errors tend to increase with circuit size and the number of layers of processing. They cannot be eliminated merely by increasing bit-stream length N .

At this point, we see that the accuracy of a stochastic circuit is impacted by many loosely related factors that are addressed by many different methods and are by no means fully understood. Correlation is amongst the most intractable of these factors. Figure 3 illustrates an example of how cross correlation can introduce errors and how to appropriately fix such errors. The problem here is to design a stochastic squarer to compute X^2 using the standard AND-gate-based multiplier described previously. To use it for squaring requires two independent, and therefore different, bit-streams with the same value X . This may be achieved by generating the bit-streams from two independent RNSs. However, the design of Fig. 3a uses a single input bit-stream \mathbf{X} that fans out into two identical, and therefore highly correlated copies that have a shared RNS and re-converge at the AND gate. Consequently, $Z = X$ instead of X^2 . This illustrates correlation due to RNS sharing and reconvergent fanout.

Figure 3b, c shows two ways to mitigate the correlation problem. The circuit in Fig. 3b converts one copy of \mathbf{X} from stochastic to binary and then back to stochastic again using a new RNS; this process is known as *regeneration*. As a result, the AND gate sees two independent SNs of value X and so computes a good approximation to X^2 . The design of Fig. 3c employs a D flip-flop called an isolator [13] to delay one copy of \mathbf{X} by a clock cycle. Instead of seeing the same bit $\mathbf{X}(t)$ twice in clock cycle t , the AND gate sees $\mathbf{X}(t)$ and $\mathbf{X}(t-1)$, which are independent by the Bernoulli

Fig. 3 Three designs for a squarer: (a) Incorrect design that ignores correlation. Decorrelated designs using (b) regeneration, and (c) isolation [29]



property. This method of decorrelation is termed *isolation* and is usually much less expensive than regeneration [29].

Some stochastic operations, notably the scaled addition of Eq. (1) implemented by a multiplexer, do not require their inputs to be independent. Such circuits are said to be *correlation insensitive* (CI) [5]. The CI property allows the two input SNs \mathbf{X}_1 and \mathbf{X}_2 of the adder to share a common RNS without producing correlation-based errors of the type illustrated by Fig. 3a. This can be explained by the fact that the adder's output bit $\mathbf{Z}(t)$ at clock cycle t is either $\mathbf{X}_1(t)$ or $\mathbf{X}_2(t)$, so there is no interaction between the two data inputs.

While correlation usually reduces the accuracy of stochastic circuits, in some cases its deliberate use can change a circuit's function to a new one that is advantageous in some way [2]. For example, an XOR (exclusive-OR) gate supplied with uncorrelated inputs \mathbf{X}_1 and \mathbf{X}_2 realizes the not-so-useful function $X_1 + X_2 - X_1X_2$. If the inputs are positively correlated by enforcing maximum overlap of 1s, the XOR realizes the absolute difference function $|X_1 - X_2|$. This has been used to design an edge-detector for image processing that contains orders of magnitude fewer gates than a comparable non-stochastic circuit [2]. Correlation is similarly used in the design a stochastic division circuit CORDIV that has accuracy advantages [10].

The design and optimization of RNSs for correlation management are also an important issue in SC [2, 23]. The problems fall into two categories: (1) designing RNSs and SNGs to generate bit-streams with desirable cross correlation and autocorrelation properties, and (2) strategically reducing the use of RNSs to decrease hardware cost while maintaining moderate independence requirements for SNs. The latter problem usually requires inexpensive re-randomization techniques and can take advantage of any CI properties for RNS sharing. Making effective use of correlation in SC is by no means well understood and is a subject of on-going research.

The rest of the chapter is organized as follows. Section “Measuring Correlation” reviews the SC correlation metric for correlation measurement and describes how

it can be used to analyze and design stochastic circuits. Section “Correlation-Controlling Units” introduces correlation-controlling circuits intended to eliminate or insert correlation. Correlation insensitivity and its role in avoiding correlation problems are discussed in section “Correlation Insensitive Stochastic Circuits”. Finally, section “Design and Optimization of RNSs” explains how careful RNS design can be exploited to engineer correlation and reduce hardware cost.

Measuring Correlation

Being able to measure and quantify correlation rigorously in the SC context is of crucial importance in the analysis and the design of stochastic circuits. We begin by examining stochastic cross correlation SCC , a correlation measure specifically designed for SC. We then discuss how the function of a stochastic circuit changes with different values of SCC .

Quantifying Correlation with SCC

An early effort to quantify correlation for SC was made by Jeavons et al. [16]. Instead of directly providing a correlation measure for SC, they define two SNs \mathbf{X} and \mathbf{Y} as independent or uncorrelated if the value of the SN \mathbf{Z} obtained from ANDing \mathbf{X} and \mathbf{Y} is XY . This definition effectively says that two SNs are independent if a stochastic multiplier can compute their product accurately. Obviously, this definition of independence assumes the computation to be otherwise error-free, i.e., it has no random fluctuation errors, rounding errors, etc. However, it is rarely the case that \mathbf{Z} 's value is exactly XY , even when \mathbf{X} and \mathbf{Y} are generated using independent RNSs. With only this definition of independence, it remains challenging to quantify the behavior of stochastic circuits under different levels of correlation.

Table 1 shows how the function of an AND-based multiplier changes under the influence of correlation. The multiplier performs as expected when the inputs \mathbf{X} and \mathbf{Y} are independent. However, it computes $Z = \min(X, Y)$ when \mathbf{X} and \mathbf{Y} are

Table 1 SC functions implemented by a two-input AND gate with different levels of input SN correlations

	X	Y	$X \wedge Y$	Function
Uncorrelated	01010101(0.5)	11110011(0.75)	01010001(0.375)	$X \times Y$
Positively correlated	11110000(0.5)	11111100(0.75)	11110000(0.5)	$\min(X, Y)$
Negatively correlated	11110000(0.5)	00111111(0.75)	00110000(0.25)	$\max(0, X + Y - 1)$

maximally correlated in the positive sense, i.e., when the 1s in \mathbf{X} and \mathbf{Y} overlap as much as possible. On the other hand, it computes $Z = \max(0, X + Y - 1)$ when the 1s in \mathbf{X} and in \mathbf{Y} overlap as little as possible. Instead of using vague terms like maximally correlated or negatively correlated, it is desirable to be able to rigorously quantify correlation for SC. Unfortunately, none of the 76 correlation measures summarized in [11] perfectly fits the needs of SC, including the Pearson correlation measure ρ which is widely used in statistical analysis. Pearson correlation presents a problem for SC, because its value depends on the actual value of the bit-streams being compared. For example, the maximum Pearson correlation value $\rho = +1$ implies that the bit-streams are identical. This means that bit-streams having different values, even if their 1s maximally overlap, fail to attain the maximum value of ρ .

A suitable correlation metric for SNs would yield a value $+1$ for maximum overlapping of 1s and 0s, a value -1 for minimum overlapping of 1s and 0s, and a value 0 for independent SNs. The metric should not be impacted by the actual value of the SN, and should also provide intuitive functional interpolation for correlation value other than $+1$, -1 or 0 .

The correlation measure called the *SC correlation coefficient* or *stochastic cross correlation (SCC)* has been proposed to fit SC's needs [2]. For a pair of SNs \mathbf{X} and \mathbf{Y} , SCC is defined as follows

$$SCC(\mathbf{X}, \mathbf{Y}) = \begin{cases} \frac{p_{\mathbf{X}\wedge\mathbf{Y}} - p_{\mathbf{X}}p_{\mathbf{Y}}}{\min(p_{\mathbf{X}}, p_{\mathbf{Y}}) - p_{\mathbf{X}}p_{\mathbf{Y}}} & \text{if } p_{\mathbf{X}\wedge\mathbf{Y}} > p_{\mathbf{X}}p_{\mathbf{Y}} \\ \frac{p_{\mathbf{X}\wedge\mathbf{Y}} - p_{\mathbf{X}}p_{\mathbf{Y}}}{p_{\mathbf{X}}p_{\mathbf{Y}} - \max(p_{\mathbf{X}} + p_{\mathbf{Y}} - 1, 0)} & \text{otherwise} \end{cases} \quad (2)$$

where $p_{\mathbf{X}} = X$, $p_{\mathbf{Y}} = Y$, and $p_{\mathbf{X}\wedge\mathbf{Y}} = p(\mathbf{X}(t) = 1, \mathbf{Y}(t) = 1)$ for all t . To measure SCC for SNs \mathbf{X} and \mathbf{Y} , $p_{\mathbf{X}\wedge\mathbf{Y}}$ is first computed which is the probability of both \mathbf{X} and \mathbf{Y} being 1. This value is then centralized by shifting by the estimate for $-p_{\mathbf{X}}p_{\mathbf{Y}}$ that corresponds to the uncorrelated value. Centralization ensures uncorrelated bit-streams yield $SCC = 0$, which is consistent with both Pearson correlation and the definition of independence in [16]. Finally, the centralized value is normalized by dividing it by the maximum attainable value. Normalization guarantees that for two maximally similar or different SNs \mathbf{X} and \mathbf{Y} , $SCC(\mathbf{X}, \mathbf{Y})$ has value $+1$ or -1 , respectively. Unlike the Pearson correlation measure $\rho(\mathbf{X}, \mathbf{Y})$, $SCC(\mathbf{X}, \mathbf{Y})$ does not vary with the value of the SNs.

Equivalently, and perhaps more intuitively, SCC can also be defined in terms of how often 1s and 0s occur in the two SNs. For two N -bit SNs \mathbf{X} and \mathbf{Y} , let N_{ij} denote the number of occurrences of the 2-bit pattern $\mathbf{X}(t)\mathbf{Y}(t) = ij$. Obviously, $N_{00} + N_{01} + N_{10} + N_{11} = N$. We can then compute SCC by replacing the estimates of $p_{\mathbf{X}}$, $p_{\mathbf{Y}}$, and $p_{\mathbf{X}\wedge\mathbf{Y}}$ in Eq. (2) by $(N_{11} + N_{10})/N$, $(N_{11} + N_{01})/N$ and N_{11}/N , respectively, to obtain

$$SCC(\mathbf{X}, \mathbf{Y}) = \begin{cases} \frac{N_{11}N_{00} - N_{10}N_{01}}{N \times \min(N_{11} + N_{10}, N_{11} + N_{01}) - (N_{11} + N_{10})(N_{11} + N_{01})} & \text{if } N_{11}N_{00} > N_{10}N_{01} \\ \frac{N_{11}N_{00} - N_{10}N_{01}}{(N_{11} + N_{10})(N_{11} + N_{01}) - N \times \max(N_{11} - N_{00}, 0)} & \text{otherwise} \end{cases}$$

where the numerator $N_{11}N_{00} - N_{10}N_{01}$ is common to many correlation measures including Pearson correlation:

$$\rho(\mathbf{X}, \mathbf{Y}) = \frac{N_{11}N_{00} - N_{10}N_{01}}{\sqrt{(N_{11} + N_{10})(N_{11} + N_{01})(N_{00} + N_{01})(N_{00} + N_{01})}}$$

The major difference between SCC and ρ lies in the denominator. SCC normalizes the measure in a way that maximally positively/negatively correlated SNs would produce a $+1/-1$ correlation value. Pearson correlation, on the other hand, is normalized by the variance of the bit-streams, which does depend on the value of the bit-streams.

Table 2 shows some examples of SN pairs and compares 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 $+1$ and -1 when the maximal overlap and minimal overlap of 1s and 0s occur, respectively.

The SCC metric of correlation provides a precise way to define a circuit’s stochastic behavior under the influence of various (cross) correlation levels. It further allows us to explore new SC designs enabled by intentionally introducing non-zero correlations. Figure 4 shows a pair of SNs \mathbf{X} and \mathbf{Y} having $SCC(\mathbf{X}, \mathbf{Y}) = +1$ applied to an XOR gate, which computes $X + Y - 2XY$ if \mathbf{X} and \mathbf{Y} are independent. The correlation between the inputs changes the circuit’s functionality to the potentially more useful absolute difference function, which leads to a highly efficient way of implementing edge detection in SC-based vision chips [6]. This illustrates the usefulness of deliberately injected correlation in designing stochastic circuits.

So far, we have only discussed cross correlation between SNs. Autocorrelation in stochastic circuits is much less well understood. Except the standard autocorrelation metric used in signal processing, an autocorrelation measure that is suitable for SC appears to be lacking. Almost all existing SC designs therefore assume the

Table 2 Example SNs with their SCC and Pearson correlation values [2]

SN pattern	SCC(X, Y)	$\rho(\mathbf{X}, \mathbf{Y})$
$\mathbf{X} = 11110000 \quad \mathbf{Y} = 11001100$	0	0
$\mathbf{X} = 11110000 \quad \mathbf{Y} = 00001111$	-1	-1
$\mathbf{X} = 11111100 \quad \mathbf{Y} = 11110000$	1	0.58
$\mathbf{X} = 11111100 \quad \mathbf{Y} = 00001111$	-1	-0.58
$\mathbf{X} = 11111100 \quad \mathbf{Y} = 11100001$	0	0
$\mathbf{X} = 11000000 \quad \mathbf{Y} = 11111100$	1	0.33



Fig. 4 XOR gate with maximal positively correlated inputs which implements the absolute-difference subtraction function $|X - Y|$ [2]

inputs to be free of autocorrelation. It is well recognized that the presence of autocorrelation, if not carefully handled, presents accuracy problems in sequential SC designs, including applications such as Bayesian inference [12], LDPC decoding [24, 27], and neural networks [9]. There are, fortunately, several ways to mitigate autocorrelation in SC, which we discuss in section ‘‘Correlation-Controlling Units’’.

Stochastic Functions with Different SCC Levels

Table 1 shows the functionality of the AND gate at SCC 0, +1, and -1 . To derive the stochastic function of the AND gate at any other SCC level, we need to calculate the linear combination of the function at $SCC = 0$ and the function at $SCC = +1$ or -1 , depending on the direction of the correlation [2]. For instance, the AND gate with $SCC = 0.5$ implements the function $Z = 0.5(\min(X, Y) + XY)$. In the general case, if we have a circuit implementing a two-input Boolean function $z = f(x, y)$ with input SNs \mathbf{X} and \mathbf{Y} having arbitrary correlation level SCC , the value of SN \mathbf{Z} at the output of the circuit will be

$$Z = \begin{cases} (1 + SCC) \cdot F_0 + SCC \cdot F_{-1} & \text{if } SCC(\mathbf{X}, \mathbf{Y}) < 0 \\ (1 - SCC) \cdot F_0 + SCC \cdot F_{+1} & \text{otherwise} \end{cases} \quad (3)$$

Here F_0 , F_{+1} and F_{-1} denote the stochastic function implemented by the same circuit at SCC levels 0, -1 and $+1$, respectively. Using probabilistic transfer matrices (PTMs), Alaghi and Hayes [2] show that for any two-input combinational circuit, we can derive F_0 , F_{-1} , and F_{+1} via the following matrix multiplication

$$[i_0 \ i_1 \ i_2 \ i_3] \cdot [t_0 \ t_1 \ t_2 \ t_3]^T$$

in which the t_k 's denote the truth table of the corresponding Boolean function and the i_k 's are obtained from Table 3. As an example, suppose we want to derive the stochastic function implemented by an XOR gate at SCC levels 0 and $+1$. The truth table PTM of the XOR gate is $[0 \ 1 \ 1 \ 0]^T$, so we will have $F_0 = (1 - X) \cdot Y + (1 - Y) \cdot X$ and $F_{+1} = \max(Y - X, 0) + \max(X - Y, 0) = |X - Y|$. To find the stochastic function of the XOR gate with $SCC = 0.25$, we simply calculate the linear combination $F_{0.25} = 0.75F_0 + 0.25F_{+1}$.

Table 3 PTM elements used to derive the stochastic function of a two-input combinational circuit at SCC levels 0, -1 and $+1$ [2]

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

Deriving the stochastic function of circuits or Boolean function with more than two inputs is not trivial, because SCC does not extend easily to multiple inputs. The most convenient method of quantifying correlation between more than two inputs is done by using PTMs, which enumerate the probability distribution of any combination of 0s and 1s among the signals. However, a systematic method of handling multi-input functions with arbitrary SCC levels is not known, except in a few special cases. One such case is when all a function's inputs are independent [1]. When all the inputs are maximally positively correlated with $SCC = +1$, we may also be able to derive the circuit's stochastic function. For instance, a k -input AND gate with maximally correlated inputs $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_k$ implements the function $\min(X_1, X_2, \dots, X_k)$.

Correlation-Controlling Units

As noted previously, managing correlation is among the more challenging design tasks in SC, as correlation plays a key role in defining a stochastic circuit's accuracy and functionality. It involves complex trade-offs between hardware cost, latency, and the desired level of accuracy. This section reviews some of the approaches proposed for managing correlation in SC using correlation-controlling units, a class of sub-circuits inserted into stochastic arithmetic units to increase or reduce correlation levels. The major design consideration in deploying correlation controllers is to achieve an appropriate level of correlation while meeting hardware and latency constraints. Although correlation can be viewed as a resource to facilitate the design of certain stochastic functions, most stochastic circuits are intended to work with independent SNs. Therefore, much prior work in correlation control was devoted to decorrelation, the process of eliminating undesired correlation completely (i.e., making SCC as close to 0 as possible). There have also been efforts in designing circuits to increase or decrease SCC , such that the SCC is pushed towards $+1$ or -1 , respectively.

Regeneration-Based Decorrelation

Perhaps the most direct way to eliminate correlation is through *regeneration*, where SNs are first converted to binary form using stochastic-to-binary converters, and then are converted back to SNs by SNGs with suitably independent RNSs. A regenerated SN has a value which is the same as, or very close to, its original value. However, the positions of its 1s are expected to be different.

An example of regeneration-based decorrelation is shown in Fig. 3b, where the goal is to produce one of the two copies of \mathbf{X} using an RNS that is independent of the original \mathbf{X} . In this example, it is sufficient to regenerate \mathbf{X} such that the two inputs of the multiplier are not cross correlated, as the multiplier is a combinational

stochastic circuit. However, regeneration-based decorrelation can also be applied to remove autocorrelation, which can be a major factor impacting the accuracy of sequential stochastic circuits. This is because the process of regeneration provides an opportunity to completely reorder the bits in the SN being regenerated. To eliminate autocorrelation in the target SNs, it is sufficient to use RNSs that have no autocorrelation.

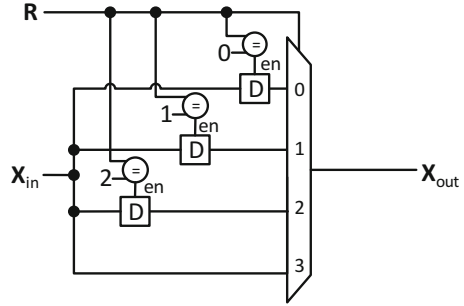
In principle, regeneration-based decorrelation can solve most types of correlation problems. However, it is also subject to several design constraints that must be taken into consideration when deploying it. First, regeneration incurs significant latency overhead and stalls computation. This is because it requires a stochastic-to-binary format conversion that is implemented using a standard binary counter, which simply counts the number of 1s in the incoming SN. This process can take as many clock cycles as the length N of the SN. The arithmetic units after the regeneration-based decorrelator cannot start computation until conversion completes. Given that many SC applications have N s in the thousands, regeneration can easily incur a latency of many thousands of clock cycles. Furthermore, the hardware overhead for a regeneration-based decorrelator is very high. In the squarer example shown in Fig. 3b, the overhead of regenerating a single SN is a counter, a stochastic number generator (SNG) which is essentially a comparator, and a new independent RNS, which is usually a precious resource in SC systems. The actual hardware cost depends on and grows with the length of the SN to be decorrelated. When compared to the main arithmetic component, which is a single AND gate in this case, the hardware overhead is extremely high. A study of a typical SC image-processing system, reported that more than 80% of its area came from circuits for SN generation [25]. Excessive use of regeneration-based decorrelators can consequently degrade the performance and increase the cost of a stochastic circuit drastically. It is therefore good practice to consider other types of decorrelation when feasible.

Lastly, it is worth noting that some variants of the regeneration-based method actively trade higher accuracy for less hardware area and latency by regenerating each bit of the SN based on a small portion of the bit-stream [28]. Since each regenerated bit is based on partial information from the original SN, the regeneration process can incur more errors, but in return has shorter latency and less hardware. For example, the single-ended counter hysteresis (SCH) decorrelator proposed in [28] keeps updating the current estimate of the input's binary form at each clock cycle through a dynamically balanced system that reaches steady state quickly. This is enabled by using a small counter for stochastic-to-binary conversion that is usually much smaller than the full counter needed to count the entire input SN.

Shuffle-Based Decorrelation

Similar to regeneration, shuffling performs decorrelation by re-positioning 1s in the target SN [19]. The difference is that the shuffling method limits the likelihood of

Fig. 5 Shuffle-based decorrelator of depth 3, where \mathbf{R} is a random number uniformly distributed among 0, 1, 2 and 3 [19]



moving a 1 to a faraway location. In other words, a 1 appearing in the t -th bit of the original SN is likely to be placed somewhere near the t -th bit of the new SN. Shuffling works by first observing and storing a subset of bits from the initial SN, and then using a new RNS to select a stored bit per cycle for outputting. Figure 5 shows a shuffler design with depth $D = 3$, meaning that it can store 3 bits for future output. The RNS here is a random number R taking a value in $\{0, 1, 2, 3\}$ uniformly at each clock cycle. Each newly received bit from the input is stored in the shuffler. At the same time, one of the stored bits (including the newly received one) is selected for output. The output SN therefore contains the same number of 1s as the input SN, since the shuffler only outputs what it received. However, the position of 1s can be very different, hence achieving the effect of decorrelation.

The advantages of the shuffle-based decorrelation include relatively low latency and low hardware cost compared to the regeneration-based method. Typical numbers for the depth D are relatively small, like 4, 6 or 8. This not only allows the received bits from the input bit-stream to be stored directly in stochastic form without inducing excessive hardware overhead, but also reduces the number of bits in the random source serving as the selection signal. The latency for the shuffling method, i.e., the number of clock cycles required before generating the first output bit, is roughly equal to the depth D , which is low compared to regeneration, especially when the input SN is very long.

Shuffling re-randomizes the position of 1s in an SN, and can thus cope with both cross correlation and autocorrelation. However, it may not completely eliminate correlation, but rather only reduce it. This is because the new SN generated by shuffling is not completely independent of the original SN. Consider an SN of value 0.5, with all 1s in the first half of the bit-stream and all 0s in the second half. In this case, the shuffler would receive and store only 1s in the beginning, and so has no choice but to output 1s initially. After the 0s start coming into the shuffler, the 1s remaining in the shuffler will quickly be released to the output, and the likelihood that a 1 is released into the output at a very late clock cycle is low. One can expect that the output SN will tend to have more 1s in the first half and more 0s in the second half. This is, however, very unlikely for a Bernoulli bit-stream. Shuffling and its variants have been successfully deployed in applications like Bayesian inference [12] and LDPC decoding [27, 28] to reduce autocorrelation.

Isolation-Based Decorrelation

Unlike the aforementioned decorrelation methods, isolation does not alter the positions of 0s and 1s in the SN. It was proposed in the 1960s [13] mainly to cope with cross correlation by adding appropriate delays to SNs. The added delays shift SNs temporally so that correlated bits from different SNs are staggered. An example of isolation-based decorrelation appears in Fig. 3c, where the isolator (a delay element implemented by a D flip-flop) is inserted into one of the two inputs of the squarer. By delaying one copy of \mathbf{X} by one clock cycle, the output $\mathbf{Z}(t) = p(\mathbf{X}(t) = 1, \mathbf{X}(t - 1) = 1) = p(\mathbf{X}(t) = 1)p(\mathbf{X}(t - 1) = 1)$, so $Z = X^2$ as expected, provided that $\mathbf{X}(t)$ and $\mathbf{X}(t - 1)$ are statistically independent for all t , as asserted by the Bernoulli property.

The major advantages of isolation are very low hardware cost and low latency, compared to regeneration. However, the application of isolators tends to be difficult. Carelessly placing isolators in a stochastic circuit can lead to several problems, such as failure to decorrelate correctly and unexpectedly changing the circuit’s function. These problems occur when the placement fails to track and delay correlated signals properly for some signal lines since isolators can inject undesired autocorrelation into the circuit and some isolators can turn out to be unnecessary. Figure 6a shows a stochastic circuit that is intended to compute X^4 by naïvely cascading two squarer circuits of the kind in Fig. 3c. While this construction appears to make sense at the first sight, the resulting circuit does not compute X^4 as expected; instead, it computes $Z = X^3$, a huge functional error! To see this, observe that at time t , the output of the first AND gate is $\mathbf{X}(t) \wedge \mathbf{X}(t - 1)$, and therefore the inputs to the second AND gate are $\mathbf{Y}_1(t) = \mathbf{X}(t) \wedge \mathbf{X}(t - 1)$ and $\mathbf{Y}_2(t) = \mathbf{X}(t - 1) \wedge \mathbf{X}(t - 2)$. By ANDing these two bit-streams, we get the final output as $\mathbf{Z}(t) = \mathbf{Y}_1(t) \wedge \mathbf{Y}_2(t) = \mathbf{X}(t) \wedge \mathbf{X}(t - 1) \wedge \mathbf{X}(t - 2)$, implying that $Z = XXX = X^3$. The cause of this error is unanticipated autocorrelation. Note that the squarer is implemented by an AND gate and an isolator, which effectively makes the circuit sequential. The adjacent bits of

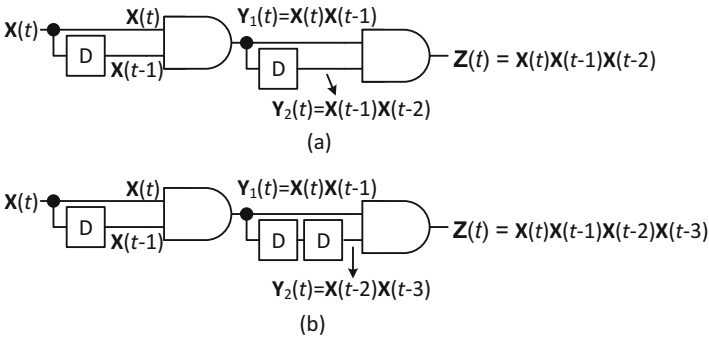


Fig. 6 Stochastic implementation of X^4 using isolator flip-flops; (a) an incorrect implementation with insufficient isolators; (b) a correct implementation with three isolators

the squarer’s output bit-stream are correlated. Therefore, delaying this bit-stream by only one clock cycle yields a cross-correlated SN. A correct implementation of X^4 is given in Fig. 6b, where the second squarer has two isolators inserted in the bottom input line.

Generally speaking, isolators must be inserted in a way that all undesired correlations between interacting SNs are eliminated. Finding a correct isolator placement while minimizing the isolator usage is a challenging problem. An isolator insertion algorithm called VAIL has been proposed for combinational stochastic circuits [29]. It formulates isolator insertion as a linear integer program, where the objective is to minimize the isolator count. A set of constraints are enforced on the number of isolators that can be placed on each line of the circuit to be decorrelated. These constraints, when satisfied, ensure that undesired correlation between interacting SNs is removed without affecting other SN interactions.

Correlation Injection by Synchronization

While almost all stochastic circuits are designed to work with uncorrelated inputs, there exist circuits implementing useful functions enabled by positively or negatively correlated inputs. For example, if the XOR gate in Fig. 4 is used to compute absolute difference, it requires its two inputs to be maximally correlated. To generate inputs with predetermined correlation for such circuits, one can resort to special types of SNGs that are capable of controlling the amount of correlation. However, regenerating SNs with specific correlation levels in the middle of an SC system is expensive, both in hardware cost and in system latency.

In error-tolerant SC applications such as many machine-learning and image-processing tasks, another way to inject correlation is to use a sequential unit called a *synchronizer*, which attempts to maximize the correlation level between a pair of SNs [19]. This approach, while providing no guarantee of attaining the desired correlation, is usually far less expensive than regeneration in terms of hardware and latency cost. Figure 7a shows the state-transition graph of a three-state synchronizer, whose key idea is to align the bits with the same value from inputs \mathbf{X} and \mathbf{Y} as much as possible. For example, when the synchronizer receives the pattern $\mathbf{X}(t)\mathbf{Y}(t) = 01$, it will output 00 and then go from state S_0 to S_2 , which remembers the 1 received from \mathbf{Y} for later release. If $\mathbf{X}(t)\mathbf{Y}(t) = 10$ is received, then the synchronizer will return to S_0 and output 11. This effectively transforms $\mathbf{X}(t)\mathbf{Y}(t) = (01, 10)$ to $(00, 11)$, which has obviously become more correlated.

Observe that the synchronizer in Fig. 7a does not guarantee that its outputs will have exactly the same value as \mathbf{X} and \mathbf{Y} . This synchronizer-induced error occurs when the computation ends at any state other than S_0 , and hence there are some remembered bits yet to be released into the outputs. Also, the synchronizer only increases the correlation level; it does not guarantee that the output will be maximally correlated. In fact, it does not provide any promises on the final correlation level of the outputs. This is because this synchronizer can only remember

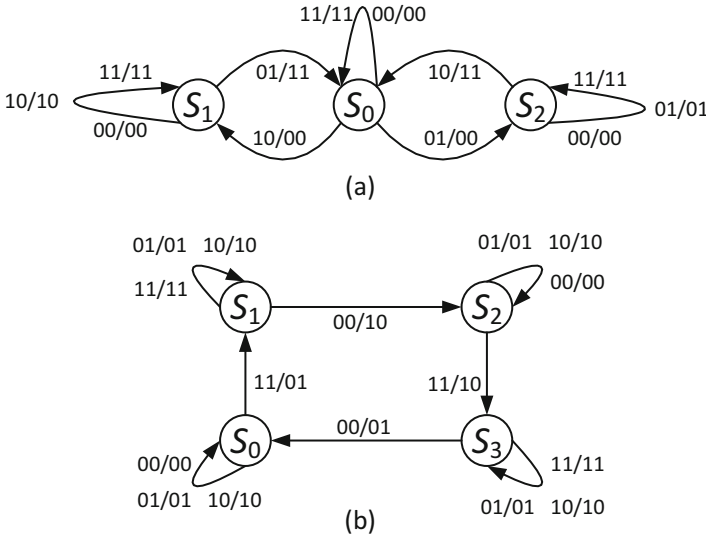


Fig. 7 State-transition graphs for correlation-controlling units that inject correlation between a pair of SN: (a) synchronizer that increases SCC; (b) desynchronizer that reduces SCC [19]

one unreleased bit from either \mathbf{X} or \mathbf{Y} . Thus, at state S_0 , if two consecutive bit patterns $\mathbf{XY} = (01, 01)$ are received, the synchronizer will have no choice but to release a 1 from \mathbf{Y} without matching it with another 1 from \mathbf{X} . In that case, the output will be $(00, 01)$, and the synchronizer will end at state S_2 . In general, increasing the number of states allows the synchronizer to remember more yet-to-be-aligned bits, and hence can produce outputs that are more correlated. But this comes at the cost of more synchronizer-induced error, because the probability of ending at a state other than the initial state is higher.

Based on the synchronizer concept, we can push the *SCC* of two SNs towards -1 using a desynchronizer. The state-transition graph of a four-state desynchronizer is depicted in Fig. 7b. Like the synchronizer, the desynchronizer takes two input SNs \mathbf{X} and \mathbf{Y} , and generates two output SNs with the same value but with stronger negative correlation or an *SCC* closer to -1 . The key idea in the desynchronizer design is to intentionally misalign bits of the same value while still preserving the encoded SN value. To do this, the desynchronizer selectively absorbs and releases bits to maximize the occurrence of the patterns $\mathbf{XY} = (10)$ and (01) , and minimize the occurrence of the patterns $\mathbf{XY} = (11)$ and (00) . If the desynchronizer receives the pattern $\mathbf{XY} = (11)$, it will pass one of the bits and save the other bit to emit later. In the desynchronizer design shown in Fig. 7b, the FSM alternates between storing \mathbf{X} and \mathbf{Y} when it receives $\mathbf{XY} = (11)$ but alternative variants are possible. When the desynchronizer receives the pattern $\mathbf{XY} = (00)$ it will emit the stored bit in the FSM to misalign the bits. If the desynchronizer receives the pattern $\mathbf{XY} = (01)$ or (10) , it will simply pass the inputs to the outputs since the bits at that SN offset

are already different. This effectively yields more negatively correlated SNs. For instance, the input pattern $\mathbf{XY} = (11, 00)$ becomes $\mathbf{XY} = (01, 10)$ after passing through the desynchronizer.

The desynchronizer has similar tradeoffs to the synchronizer. Bits that get saved in the desynchronizer may not be emitted before the end of execution which can yield a slight negative bias. Notice also that the desynchronizer FSM can only save one bit at a time. As a result, there are cases where it may be forced to pass the pattern $\mathbf{XY} = (11)$ or (00) . For instance, if the desynchronizer receives the pattern $\mathbf{XY} = (11, 11)$ it will output $(01, 11)$. In this case, the desynchronizer absorbs a bit from the first occurrence of $\mathbf{XY} = (11)$ but not from the second $\mathbf{XY} = (11)$. This forces the desynchronizer to simply pass $\mathbf{XY} = (11)$ to the output on the second occurrence. This limitation can be addressed by augmenting the desynchronizer to allow it to absorb more bits to improve its efficacy. Again, this increases the potential error due to bits that get saved in the FSM but are not released before the end of execution.

To illustrate the strengths and weaknesses of each correlation manipulation technique, consider an image processing pipeline which consists of a 3×3 Gaussian blur followed by a Roberts Cross edge detector. The Gaussian blur kernel requires input SNs for each multiply in the kernel to be uncorrelated, while the Roberts Cross edge detector requires inputs to the subtractor to be positively correlated. Figure 8 shows the resulting image along with energy efficiency and average absolute error for three different configurations: (1) no correlation correction between kernels, (2) regeneration before the edge detector, and (3) synchronizers before the edge detector. Absolute error is measured as the deviation from a floating-point baseline implementation. The resulting image without any correlation correction at all clearly suffers from significant accuracy losses. Using correlation controlling circuits like regeneration or the synchronizer, on the other hand, leads to much more accurate results. The synchronizer is more energy efficient and yields comparable accuracy to regeneration.





	Floating point	No correction	Regeneration	Synchronizer
Image result				
Energy	-	1383 nJ / frame	1971 nJ / frame	1505 nJ / frame
Absolute error	0	0.076	0.019	0.020

Fig. 8 Image processing case study results for Gaussian blur kernel followed by Roberts Cross edge detector [19]

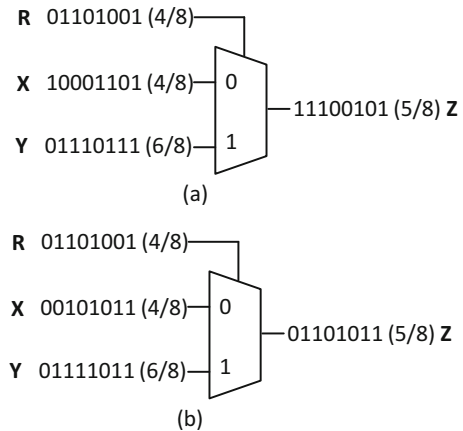
Correlation Insensitive Stochastic Circuits

Next we address a desirable SC property called correlation insensitivity, which is found in some stochastic circuits. *Correlation insensitive* (CI) circuits are immune to the effects of cross correlation or autocorrelation. In other words, a designer does not have to worry about the correlation among certain CI inputs, and can treat them as if they are uncorrelated. Our discussion will focus on cross-correlation insensitivity (cross-CI), including the detection and the characteristics of cross-CI. We will also briefly review autocorrelation insensitivity. Finally, we discuss some special classes of stochastic circuits that have distinctive autocorrelation features.

Cross-Correlation Insensitivity

Cross-correlation insensitivity (cross-CI) implies that a stochastic circuit C 's function is not impacted by cross correlation between some of its primary input SNs. Roughly speaking, cross-CI occurs if two SNs cannot simultaneously impact the circuit's outputs. This is a very useful property for stochastic circuits, especially for their input lines to which user-supplied and potentially correlated SNs are applied. For instance, Fig. 9a shows a standard multiplexer-based stochastic adder that computes $Z = 0.5(X + Y)$ with independent input SNs X and Y , while Fig. 9b shows the same adder but with highly correlated X and Y . Here R is an SN with a fixed probability 0.5, and must be independent of both of the user-supplied inputs X and Y . On the other hand, X and Y can be cross correlated in any way without affecting the adder's functionality. This is because at each clock cycle, either a bit from X or a bit from Y is selected by R to send to the output Z , i.e., Z can never depend on both X and Y at the same time.

Fig. 9 (a) Stochastic addition showing accurate results with (a) uncorrelated and (b) correlated inputs X and Y



A formal definition of cross-CI for combinational stochastic circuits is given in [5], which states that for a stochastic circuit C that implements the Boolean function $z = f(x_1, x_2, \dots, x_n)$, x_i and x_j form a cross-CI pair for C , if and only if the following holds:

$$(dz/dx_i) \wedge (dz/dx_j) = 0 \quad (4)$$

where 0 denotes the zero Boolean function and dz/dx_i denotes the Boolean difference of z with respect to x_i , i.e.,

$$\begin{aligned} dz/dx_i &= f(x_1, x_2, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) \oplus \\ &f(x_1, x_2, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \end{aligned}$$

A proof of Eq. (4) can be found in [5]; here we provide a brief intuitive explanation. The Boolean difference dz/dx_i is a Boolean function of (x_1, x_2, \dots, x_n) whose minterms correspond to the input assignments such that a change of x_i 's value will lead to a change of z 's value. Therefore, Eq. (4) simply says that if there is no input assignment such that x_i 's value change and x_j 's value change can each change z 's value, then x_i and x_j form a CI pair.

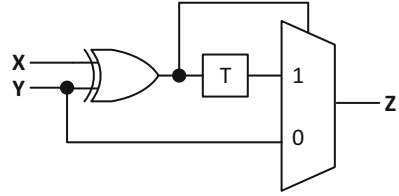
The preceding definition is useful for identifying CI pairs in a given stochastic circuit. For example, recall that the multiplexer in Fig. 9 implements the function $Z = 0.5(X + Y)$ in the stochastic domain, and the function $z = x \wedge r' \vee y \wedge r$ in the Boolean domain. Here x and y form a CI pair, because

$$\begin{aligned} dz/dx &= (y \wedge r) \oplus (r' \vee y) = r' \\ dz/dy &= (y \wedge r') \oplus (r \vee x) = r \end{aligned}$$

so $(dz/dx) \wedge (dz/dy) = r \wedge r' = 0$, which confirms that x and y are indeed a CI pair. On the other hand, x and r are not a CI pair because $dz/dr = x \oplus y$ and $(dz/dx) \wedge (dz/dr) \neq 0$. From this, we see that in the multiplexer-based adder, \mathbf{X} and \mathbf{Y} can be correlated without incurring any correlation error. However, \mathbf{R} must be independent of both \mathbf{X} and \mathbf{Y} for the adder to work accurately.

The ancillary input \mathbf{R} in the adder case is introduced to provide the scaling operation needed by this type of adder. However, it imposes a constraint of independence that must be satisfied by using an RNS that is uncorrelated to both \mathbf{X} and \mathbf{Y} . This not only incurs a large hardware overhead, it also injects unnecessary random fluctuation into the circuit, causing a constant-induced error [31]. It turns out that it is always possible to remove the need for independent RNSs and the constant-induced error by transferring the role of all ancillary inputs to sequential elements. This was first shown possible in [20] via an ancillary-input-free adder constructed in ad hoc fashion; see Fig. 10. This design therefore accurately computes the scaled sum $Z = 0.5(X + Y)$, regardless of correlation between \mathbf{X} and \mathbf{Y} . More recently, a systematic method CEASE for removing ancillary inputs by role transfer was

Fig. 10 A sequential adder computing $Z = 0.5(X + Y)$ whose inputs are CI [20]



proposed that is applicable to all combinational stochastic circuits with ancillary inputs [31]. CEASE thus removes the correlation associated with ancillary inputs.

The CI property enables inputs to take SNs generated from a shared RNS without compromising the circuit's accuracy. This is useful in statistical analysis of stochastic circuits by greatly improving the simulation quality, and in SC designs by allowing CI inputs to accept SNs generated from a common RNS.

Circuits with Autocorrelation Features

Combinational stochastic circuits are inherently immune to autocorrelation, as the change of input ordering only changes the output ordering, but not the output value. On the other hand, sequential stochastic circuits operate in response to their input sequences, and thus are generally impacted by autocorrelation. The autocorrelation among input patterns at different time steps can steer the state transition in a biased way. Further, sequential components can also inject new autocorrelation into the SNs they process. Careless use of sequential components in SC systems can thus lead to significant autocorrelation-induced errors [9]. Managing autocorrelation in SC is an open research area, since many sequential SC designs can only operate in autocorrelation-free environments, including those synthesized using the “linear finite-state machine” architecture [21]. Some classes of sequential circuits with special autocorrelation properties are known and are discussed below.

Modulo-Counting Circuits The circuits generated by CEASE have a highly desirable property: they are immune to autocorrelation-induced errors. CEASE, as mentioned previously, is an algorithm to remove ancillary inputs of a combinational circuit by introducing sequential components [31]. The resulting sequential circuit is equivalent to a modulo counter with multiple moduli that keeps accumulating (weighted) input bits for later release, which occurs when the counter overflows. For instance, the ancillary-input-free adder in Fig. 10, while constructed in ad hoc fashion, is also effectively a modulo counter as can be seen from its state-transition graph in Fig. 11. When the adder receives the input pattern 11 or 00, it directly outputs a 1 or a 0, respectively. However, when the adder receives 10 or 01, it will directly output a 0, but it accumulates a value 0.5 into its memory by going from state S_0 to state S_1 . The next time the adder receives 10 or 01, it will overflow, output a 1, and return to state S_0 . One can easily see that this modulo counting

Fig. 11 State transition graph for the sequential adder in Fig. 9 [31]

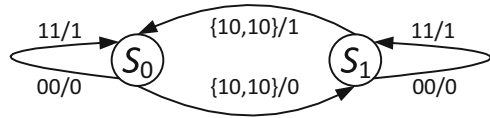
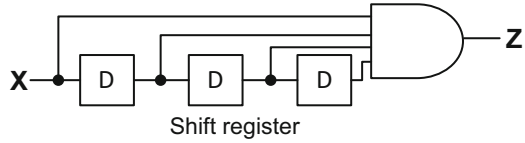


Fig. 12 Implementation of X^4 in canonical SRB form [30]



process guarantees that there will be a single 1 released into the output, whenever the adder receives two 1s from \mathbf{X} or \mathbf{Y} , thereby computing $0.5(X + Y)$. In general, CEASE-generated circuits not only avoid the potential correlation problems induced by ancillary inputs, but also are insensitive to autocorrelation. This is because the number of 1s in the output is completely determined by the number of times the modulo counter overflows, which is obviously independent of the ordering of the input pattern.

Shift-Register-Based Circuits In general, sequential stochastic circuits have strict correlation specifications on their inputs, which usually requires them to be autocorrelation-free. However, sequential stochastic circuits also inject autocorrelation into the SNs they process. This makes it difficult to cascade sequential designs, since autocorrelation introduced by an upstream circuit will degrade the accuracy of a downstream circuit. For example, sequential circuits employing the linear FSM architecture [21] require their inputs to be autocorrelation-free, but at the same time they produce output SNs with a high level of autocorrelation. It is therefore difficult to connect multiple linear circuits without sacrificing accuracy. Autocorrelation injected by a linear FSM has a diminishing but continuing effect over time. A current output bit can be correlated with all previous output bits, although the correlation level is lower with bits that are further away in time. This implies that when its input value changes, the output of a linear FSM may take a very long time to respond to the change, so the change can have an extended accuracy-reducing impact.

Thus, it is sometimes desirable to use alternative designs that have less severe autocorrelation problems. There is a class of sequential stochastic circuits called *shift-register-based (SRB)* which have a highly desirable property: their output autocorrelation is bounded in time [30]. SRB circuits realize a type of FSM termed a *definite machine* that has finite input memory [18]. They also have a canonical implementation consisting of a feed-forward shift register built around a combinational component. Many SC designs, including those generated by the STRAUSS synthesizer [4], belong to the SRB class. For example, Fig. 12 shows the canonical SRB implementation of X^4 , which has a 3-tap shift register that produces three delayed copies of the input SN \mathbf{X} . SRB circuits have their output autocorrelation bounded in time, because each output bit is completely determined by the m most recent input bits, where $m - 1$ is the number of taps of the shift

register. Therefore, output bits that are separated by m clock cycles are determined by different and independent sets of input bits, and hence must be uncorrelated. In the X^4 example, $\mathbf{Z}(4) = \mathbf{X}(4)\mathbf{X}(3)\mathbf{X}(2)\mathbf{X}(1)$, while $\mathbf{Z}(8) = \mathbf{X}(8)\mathbf{X}(7)\mathbf{X}(6)\mathbf{X}(5)$, so $\mathbf{Z}(8)$ and $\mathbf{Z}(4)$, which are four clock cycles apart, are uncorrelated. The definiteness of the SRB circuits guarantees that m clock cycles after an input value change, the output value will have fully responded to the change. Furthermore, it is possible to sample the output SN every m cycles to get a completely autocorrelation-free bit-stream, which facilitates the use of such SNs as inputs to circuits that must avoid autocorrelation.

Design and Optimization of RNSs

Random number sources provide the randomness to drive the dynamics of the stochastic signals. RNSs with insufficient randomness can result in significant accuracy loss for stochastic circuits that require independent inputs. This can occur, for example, if a shared RNS is used to drive multiple SNGs for SN generation. The quality of RNSs also plays an important role in the accuracy of SC. It has been shown that, instead of using an RNS that has good randomness property like an LFSR, using carefully designed deterministic number sequences can sometimes result in significantly improved accuracy. For specialized circuits that work with correlated inputs, SNs with any SCC level can be generated by interpolating independent SNs and maximally correlated SNs with multiple independent RNSs. The CI property also is important in SNG design, as it allows a single RNS to be shared by multiple SNGs without compromising accuracy.

Generating SNs with Predetermined Correlation

In most existing stochastic circuits, it is desirable to have SNGs that can generate high quality uncorrelated SNs, i.e., SNs that have $\text{SCC} = 0$. In SNG design, arguably the most common RNSs are obtained by tapping an LFSR of maximum period, and are essentially pseudo-random. However, it has been shown that deterministic number sources such as plain binary counters can also be used as in SN generation without compromising accuracy [17]. In fact, circuits that use such deterministic number sources are usually more accurate than the ones using LFSRs, because random fluctuation errors are eliminated and correlation control is easier.

To achieve fast convergence rates during a stochastic computation, researchers have also looked into using quasi-Monte Carlo methods and low-discrepancy sequences [3, 22]. While these methods provide good convergence when generating a few uncorrelated SNs, they are affected by the curse of dimensionality and are no better than counter-based SNGs. In many cases, the convergence rate of the SNs is not relevant, and only the accuracy at the end of computation matters. In such cases,

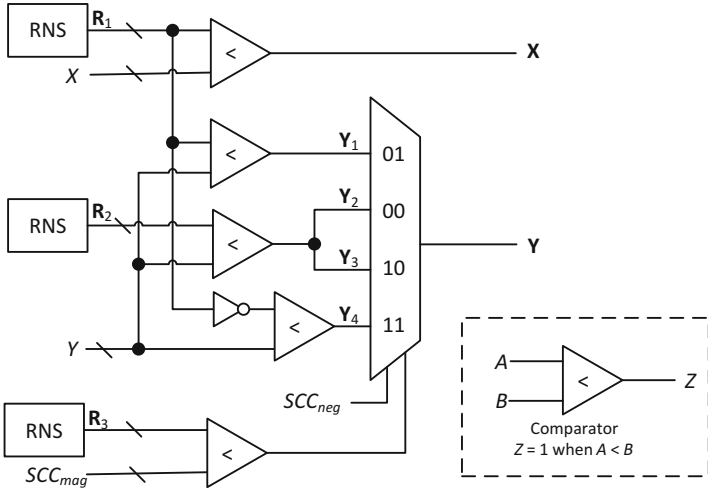


Fig. 13 SNG that generates a pair of SNs with a user-specified SCC level [2]

low-discrepancy sequences have no significant benefit over other deterministic number sources. It is worth noting that low-discrepancy number sources (and similar deterministic number sources) usually create autocorrelation within SNs, which may not be desirable.

As noted earlier, while most stochastic circuits require independent inputs, circuits exist that need a specific SCC level at their inputs. Figure 13 shows an SNG design [2] that generates a pair of SNs with a predetermined SCC level using three independent RNSs. This SNG is essentially a direct reflection of Eq. (3). Depending on the sign of SCC, the SNG selects a maximally correlated \mathbf{XY} pair and linearly combines it with the independent \mathbf{XY} pair, weighted by the magnitude of SCC. For example, to generate the bit-streams with $SCC(\mathbf{X}, \mathbf{Y}) = 0.5$, the inputs SCC_{neg} and SCC_{mag} of Fig. 13 are set to 0.0 and 0.5, respectively, so that the selection signal of the multiplexer will have 0.5 probability of choosing 01 and another 0.5 probability of choosing 00. The SN \mathbf{Y}_1 that is maximally correlated with \mathbf{X} , is applied to data input 01 of the multiplexer, while \mathbf{Y}_2 , which is completely independent with \mathbf{X} is applied to multiplexer input 00. Combining \mathbf{Y}_1 and \mathbf{Y}_2 with a 0.5 probability of outputting each result in an SN \mathbf{Y} that has correlation $SCC = 0.5$ with \mathbf{X} .

Optimizing Random Number Sources

RNSs provide the randomness required by stochastic circuits, and are a key design resource. While there are stochastic systems that use non-LFSR-based RNSs, LFSRs remain the popular choice due to their compatibility with digital logic and their relatively small hardware area, on top of the fact that they have been

intensively studied for many years. The design considerations around deploying RNSs include: providing adequate randomness, minimizing hardware overhead, and reducing unnecessary use of the RNSs.

As discussed earlier, an SN \mathbf{X} can be derived from its binary counterpart B by an SNG containing an RNS and a comparator that compares the B with the random number \mathbf{R} from the RNS at each clock cycle. The SNG outputs a 1 whenever $B > R$; otherwise the SNG outputs a 0. A common approach is to treat k taps from the LFSR as the k -bit random number \mathbf{R} . Sharing exactly the same random number \mathbf{R} with other SNGs can reduce overall hardware cost, but will result in a maximally correlated SN which is usually undesirable. Previous work attempts to squeeze out more randomness from a single LFSR by adding a re-wiring layer that shuffles the order of the bits in \mathbf{R} . In [15], the authors show that circularly shifting \mathbf{R} is a low-cost and effective way to reduce the SCC of two SNs sharing the same LFSR. Specifically, they experimentally demonstrate that by circularly shifting a k -bit random number by approximately $k/2$ bits, the SCC level can be reduced by around 75%, compared to random shuffling which achieves only 40% reduction in SCC on average. They further show that taking advantage of the CI property can reduce the need for RNSs.

Conclusions

Correlation is a pervasive phenomenon in stochastic circuits and a major source of computational inaccuracy and high hardware cost. Although it has been intensively studied in recent years, it still remains quite poorly understood. In general, correlation is difficult to quantify and expensive to mitigate. The *SCC* metric has proven useful as a measure of the cross correlation between two bit-streams but it is not readily extended to more than two bit-streams. No comparable measure of autocorrelation has emerged, even though autocorrelation has a significant impact on the accuracy of sequential stochastic circuits. The main methods of reducing or eliminating undesired correlation are regeneration and isolation, with isolation being by far the less expensive of the two.

Correlation affecting the inputs of a circuit with output function f can be viewed in two contrasting ways: as corrupting f 's output by introducing errors, or as changing f to a different function which may be useful. While a few examples of such "good" correlation-based designs are known, no general and practical way of taking advantage of correlation in SC design is known at present. In some cases, interacting bit-streams are unaffected by correlation, but again there is no easy way to identify and exploit correlation insensitivity. Among the more promising recent discoveries in this direction are several classes of highly accurate sequential machines that are insensitive to correlation on their inputs, and the development of general-purpose correlation-controlling units.

Abbreviations

CI	Correlation insensitive
FSM	Finite-state machine
LDPC	Low density parity check code
LFSR	Linear feedback shift register
MSE	Mean square error
PTM	Probabilistic transfer matrix
RNS	Random number source
SC	Stochastic computing
SCC	Stochastic correlation coefficient
SCH	Single-ended counter hysteresis
SN	Stochastic number
SNG	Stochastic number generator
SRB	Shift register based

References

1. A. Alaghi and J.P. Hayes, "A Spectral Transform Approach to Stochastic Circuits," *Proc. Intl. Conf. Computer Design (ICCD)*, pp. 315–312, 2012.
2. A. Alaghi and J.P. Hayes, "Exploiting Correlation in Stochastic Circuit Design," *Proc. Intl. Conf. on Computer Design (ICCD)*, pp. 39–46, Oct. 2013.
3. A. Alaghi and J.P. Hayes, "Fast and Accurate Computation Using Stochastic Circuits," *Proc. Design, Automation, and Test in Europe Conf. (DATE)*, pp. 1–4, 2014.
4. A. Alaghi and J.P. Hayes, "STRAUSS: Spectral Transform Use in Stochastic Circuit Synthesis," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 34, pp. 1770–1783, 2015.
5. A. Alaghi and J.P. Hayes, "Dimension Reduction in Statistical Simulation of Digital Circuits," *Proc. Symp. on Theory of Modeling & Simulation (TMS-DEVS)*, pp. 1–8, 2015.
6. A. Alaghi, C. Li and J.P. Hayes, "Stochastic Circuits for Real-time Image-Processing Applications," *Proc. Design Autom. Conf. (DAC)*, article 136, 6p, 2013.
7. A. Alaghi, W.-K. Qian and J.P. Hayes, "The Promise and Challenge of Stochastic Computing," *IEEE Trans. CAD*, vol. 37, pp.1515–1531, Aug. 2018.
8. R. Baumann, "Soft Errors in Advanced Computer Systems," *IEEE Design & Test of Computers*, vol.22, pp. 258–266, 2005.
9. B.D. Brown and H.C. Card, "Stochastic Neural Computation I: Computational Elements," *IEEE Trans. Comp.*, vol. 50, pp. 891–905, 2001.
10. T.-H. Chen and J.P. Hayes, "Design of Division Circuits for Stochastic Computing," *Proc. IEEE Symp. on VLSI (ISVLSI)*, pp. 116–121, 2016.
11. S.S. Choi, S.H. Cha and C. Tappert, "A Survey of Binary Similarity and Distance Measures," *Jour. Systemics, Cybernetics and Informatics*, vol. 8, pp. 43–48, 2010.
12. J. Friedman et al. "Approximation Enhancement for Stochastic Bayesian Inference," *Elsevier Int. Jour. of Approximate Reasoning*, 85, pp.139–158, 2017.
13. B.R. Gaines, "Stochastic Computing Systems," *Advances in Information Systems Science*, vol. 2, J.T. Tou (ed.), Springer, pp. 37–172, 1969.
14. S.W. Golomb, *Shift Register Sequences*. Revised ed., Aegean Park Press, Laguna Hills, CA, 1982.

15. H. Ichihara et al., "Compact and Accurate Digital Filters Based on Stochastic Computing," *IEEE Trans. Emerging Topics in Comp.*, 2018 (early access).
16. P. Jeavons, D.A. Cohen and J. Shawe-Taylor., "Generating Binary Sequences for Stochastic Computing," *IEEE Trans. Info. Theory*, vol. 40, pp. 716–720, 1994.
17. D. Jenson and M. Riedel, "A Deterministic Approach to Stochastic Computation," *Proc. Intl. Conf. Computer-Aided Design (ICCAD)*, pp. 1–8, 2016.
18. Z. Kohavi and N.K. Jha, *Switching and Finite Automata Theory*, 3rd ed. Cambridge Univ. Press, 2010.
19. V.T. Lee, A. Alaghi and L. Ceze, "Correlation Manipulating Circuits for Stochastic Computing," *Proc. 2018 Design, Automation & Test in Europe (DATE) Conf.*, pp. 1417–1422, 2018.
20. V.T. Lee et al., "Energy-Efficient Hybrid Stochastic-Binary Neural Networks for Near-Sensor Computing," *Proc. Design, Automation and Test in Europe Conf. (DATE)*, pp. 13–18, 2017.
21. P. Li et al., "Logical Computation on Stochastic Bit Streams with Linear Finite-State Machines," *IEEE Trans. Computers*, vol. 63, pp. 1474–1486, 2014.
22. S. Liu and J. Han, "Energy Efficient Stochastic Computing with Sobol Sequences," *Proc. Design, Automation, and Test in Europe Conf. (DATE)*, pp. 650–653, 2017.
23. Y. Liu et al., "Synthesis of Correlated Bit Streams for Stochastic Computing," *Proc. Asilomar Conf. on Signals, Systems and Computers*, pp. 167–174, 2016.
24. A. Naderi et al., "Delayed Stochastic Decoding of LDPC Codes," *IEEE Trans. Signal Processing*, vol. 59, pp. 5617–5626, 2011.
25. W. Qian et al., "An Architecture for Fault-Tolerant Computation with Stochastic Logic," *IEEE Trans. Comp.*, vol. 60, pp. 93–105, 2011.
26. W. Qian and M. D. Riedel, "The Synthesis of Robust Polynomial Arithmetic with Stochastic Logic," *Proc. Design Autom. Conf. (DAC)*, pp. 648–653, 2008.
27. S.S. Tehrani, W. J. Gross, and S. Mannor, "Stochastic Decoding of LDPC Codes," *IEEE Comm. Letters*, vol. 10, pp. 716–718, 2006.
28. S. Tehrani et al., "Relaxation Dynamics in Stochastic Iterative Decoders," *IEEE Trans. Signal Processing*, vol. 58, pp. 5955–5961, 2010.
29. P.S. Ting and J.P. Hayes, "Isolation-Based Decorrelation of Stochastic Circuits," *Proc. Intl. Conf. Computer Design (ICCD)*, pp. 88–95, 2016.
30. P.S. Ting and J.P. Hayes, "On the Role of Sequential Circuits in Stochastic Computing," *Proc. Great Lakes VLSI Symp. (GLSVLSI)*, pp. 475–478, 2017.
31. P. S. Ting and J.P. Hayes, "Eliminating a Hidden Error Source in Stochastic Circuits," *Proc. Symp. Defect & Fault Tolerance in VLSI and Nano. Systems (DFT)*, pp.44–49, Oct. 2017.

Synthesis of Polynomial Functions



Marc Riedel and Weikang Qian

Abstract This chapter addresses the fundamental question: what functions can stochastic logic compute? We show that, given stochastic inputs, any combinational circuit computes a *polynomial function*. Conversely, we show that, given any polynomial function, we can synthesize stochastic logic to compute this function. The only restriction is that we must have a function that maps the unit interval $[0, 1]$ to the unit interval $[0, 1]$, since the stochastic inputs and outputs are probabilities. Our approach is both general and efficient in terms of area. It can be used to synthesize arbitrary polynomial functions. Through polynomial approximations, it can also be used to synthesize non-polynomial functions.

Keywords Polynomials · Bernstein polynomials · Non-polynomials · Synthesis · Computability · Combinational circuits

Introduction

First introduced by Gaines [1] and Poppelbaum [2, 3] in the 1960s, the field of stochastic computing has seen widespread interest in recent years. Much of the work, both early and recent, has had more of an applied than a theoretical flavor. The work of Gaines, Poppelbaum, Brown & Card [4], as well as recent papers pertaining to image processing [5] and neural networks [6] all demonstrate how to compute specific functions for particular applications.

This chapter has a more theoretical flavor. It addresses the fundamental question: can we characterize the class of functions that stochastic logic can compute? Given a combinational circuit, that is to say a circuit with no memory elements, the answer

M. Riedel (✉)
University of Minnesota, Minneapolis, MN, USA
e-mail: mriedel@umn.edu

W. Qian
Shanghai Jiao Tong University, Shanghai, China
e-mail: qianwk@sjtu.edu.cn

is rather easy: given stochastic inputs, we show such a circuit computes a *polynomial function*. Since the stochastic inputs and outputs are probabilities, this polynomial function maps inputs from the unit interval $[0, 1]$ to outputs in the unit interval $[0, 1]$.

The converse question is much more challenging: given a target polynomial function, can we synthesize stochastic logic to compute it? The answer is yes: we prove that there exists a combinational circuit that computes *any* polynomial function that maps the unit interval to the unit interval. So the characterization of stochastic logic is complete. Our proof method is constructive: we describe a synthesis methodology for polynomial functions that is general and efficient in terms of area. Through polynomial approximations, it can also be used to synthesize non-polynomial functions.

Characterizing What Stochastic Logic Can Compute

Consider basic logic gates. Table 1 describes the functions that they implement given stochastic inputs. These are all straight-forward to derive algebraically. For instance, given a stochastic input x representing the probability of seeing a 1 in a random stream of 1s and 0s, a NOT gate implements the function

$$\text{NOT}(x) = 1 - x. \quad (1)$$

Given inputs x, y , an AND gate implements the function:

$$\text{AND}(x, y) = xy. \quad (2)$$

An OR gate implements the function:

$$\text{OR}(x, y) = x + y - xy. \quad (3)$$

An XOR gate implements the functions

$$\text{XOR}(x, y) = x + y - 2xy. \quad (4)$$

Table 1 Stochastic function implemented by basic logic gates

Gate	Inputs	Function
NOT	x	$1 - x$
AND	x, y	xy
OR	x, y	$x + y - xy$
NAND	x, y	$1 - xy$
NOR	x, y	$1 - x - y + xy$
XOR	x, y	$x + y - 2xy$
XNOR	x, y	$1 - x - y + 2xy$

It is well known that any Boolean function can be expressed in terms of AND and NOT operations (or entirely in terms of NAND operations). Accordingly, the function of any combinational circuit can be expressed as a nested sequence of multiplications and $1 - x$ type operations. It can easily be shown that this nested sequence results in a polynomial function. (Note that special treatment is needed for any reconvergent paths.)

We will make the argument based upon truth tables. Here we will consider only univariate functions, that is to say stochastic logic that receives multiple independent copies of a single variable t . (Technically, t is the Bernoulli coefficient of a random variable X_i , where $t = [\text{Pr}(X_i = 1)]$.) Please see [7] for a generalization to multivariate polynomials.

Consider a combinational circuit computing a function $f(X_1, X_2, X_3)$ with the truth table shown Table 2. Now suppose that each variable has independent probability t of being 1:

$$[\text{Pr}(X_1) = 1] = t \tag{5}$$

$$[\text{Pr}(X_2) = 1] = t \tag{6}$$

$$[\text{Pr}(X_3) = 1] = t \tag{7}$$

The probability that the function evaluates to 1 is equal to the sum probabilities of occurrence of each row that evaluates to 1. The probability of each row, in turn, is obtained from the assignments to the variables, as shown in Table 3. Summing up the rows that evaluate to 1, we obtain

Table 2 Truth table for a combinational circuit

X_1	X_2	X_3	$f(X_1, X_2, X_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 3 Probabilities of each row in Table 2, assuming each variable has independent probability t

X_1	X_2	X_3	Probability of row	$f(X_1, X_2, X_3)$
0	0	0	$(1 - t)^3$	0
0	0	1	$(1 - t)^2 t$	1
0	1	0	$(1 - t)t(1 - t)$	0
0	1	1	$(1 - t)t^2$	1
1	0	0	$t(1 - t)^2$	0
1	0	1	$t(1 - t)t$	1
1	1	0	$t^2(1 - t)$	1
1	1	1	t^3	1

$$(1 - t)^2t + (1 - t)t^2 + t(1 - t)t + t^2(1 - t) + t^3 \tag{8}$$

$$= (1 - t)^2t + 3(1 - t)t^2 + t^3 \tag{9}$$

$$= t + t^2 - t^3 \tag{10}$$

Generalizing from this example, suppose we are given any combination circuit with n inputs that each evaluate to 1 with independent probability t . We conclude that the probability that the output of the circuit evaluates to 1 is equal to the sum of terms of the form $t^i(1 - t)^j$, where $0 \leq i \leq n, 0 \leq j \leq n, i + j = n$, corresponding to rows of the truth table of the circuit that evaluate to 1. Expanding out this expression, we always obtain a polynomial in t .

We note that the analysis here was presented as early as 1975 in [8]. Algorithmic details for such analysis were first fleshed out by the testing community [9]. They have also found mainstream application for tasks such as timing and power analysis [10, 11].

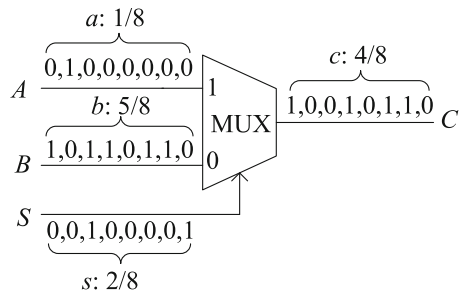
Synthesizing any Polynomial Function

In this chapter, we will explore the more challenging task of *synthesizing* logical computation on stochastic bit streams that implements the functionality that we want. Naturally, since we are mapping probabilities to probabilities, we can only implement functions that map the unit interval $[0, 1]$ onto the unit interval $[0, 1]$. Consider the behavior of a *multiplexer*, shown in Fig. 1. It implements scaled addition: with stochastic inputs a, b and a stochastic select input s , it computes a stochastic output c :

$$c = sa + (1 - s)b. \tag{11}$$

(We use the convention of upper case letters for random variables and lower case letters for the corresponding probabilities.)

Fig. 1 Scaled addition on stochastic bit streams, with a multiplexer (MUX). Here the inputs are $1/8, 5/8$, and $2/8$. The output is $2/8 \times 1/8 + (1 - 2/8) \times 5/8 = 4/8$, as expected



Based on the constructs for multiplication (an AND gate) and scaled addition (a multiplexer), we can readily implement polynomial functions of a specific form, namely polynomials with non-negative coefficients that sum up to a value no more than one:

$$g(t) = \sum_{i=0}^n a_i t^i$$

where, for all $i = 0, \dots, n$, $a_i \geq 0$ and $\sum_{i=0}^n a_i \leq 1$.

For example, suppose that we want to implement the polynomial $g(t) = 0.3t^2 + 0.3t + 0.2$. We first decompose it in terms of multiplications of the form $a \cdot b$ and scaled additions of the form $sa + (1 - s)b$, where s is a constant:

$$g(t) = 0.8(0.75(0.5t^2 + 0.5t) + 0.25 \cdot 1).$$

Then, we reconstruct it with the following sequence of multiplications and scaled additions:

$$\begin{aligned} w_1 &= t \cdot t, \\ w_2 &= 0.5w_1 + (1 - 0.5)t, \\ w_3 &= 0.75w_2 + (1 - 0.75) \cdot 1, \\ w_4 &= 0.8 \cdot w_3. \end{aligned}$$

The circuit implementing this sequence of operations is shown in Fig. 2. In the figure, the inputs are labeled with the probabilities of the bits of the corresponding stochastic streams. Some of the inputs have fixed probabilities and the others have variable probabilities t . Note that the different lines with the input t are each fed with *independent* stochastic streams with bits that have probability t .

What if the target function is a polynomial that is not decomposable this way? Suppose that it maps the unit interval onto the unit interval but it has some coefficients less than zero or some greater than one. For instance, consider the polynomial $g(t) = \frac{3}{4} - t + \frac{3}{4}t^2$. It is not apparent how to construct a network of stochastic multipliers and adders to implement it.

Fig. 2 Computation on stochastic bit streams implementing the polynomial $g(t) = 0.3t^2 + 0.3t + 0.2$

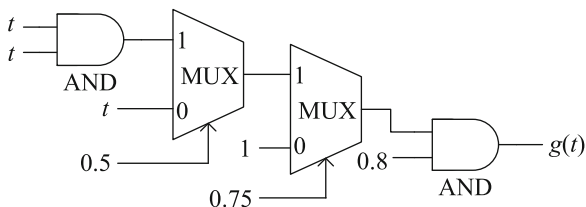
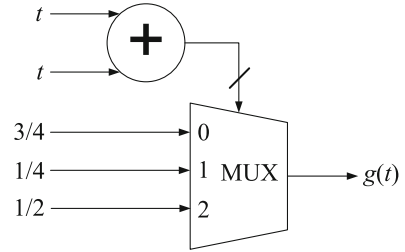


Fig. 3 A generalized multiplexing circuit implementing the polynomial $g(t) = \frac{3}{4} - t + \frac{3}{4}t^2$



We propose a general method for synthesizing arbitrary univariate polynomial functions on stochastic bit streams. A necessary condition is that the target polynomial maps the unit interval onto the unit interval. We show that this condition is also sufficient: we provide a constructive method for implementing any polynomial that satisfies this condition. Our method is based on some novel mathematics for manipulating polynomials in a special form called a Bernstein polynomial [12–15]. In [16] we showed how to convert a general power-form polynomial into a Bernstein polynomial with coefficients in the unit interval. In [17] we showed how to realize such a polynomial with a form of “generalized multiplexing.”

We illustrate the basic steps of our synthesis method with the example of $g(t) = \frac{3}{4} - t + \frac{3}{4}t^2$. (We define Bernstein polynomials in the section “[Bernstein Polynomials](#)”. We provide further details regarding the synthesis method in the section “[Synthesizing Polynomial Functions](#)”.)

1. Convert the polynomial into a Bernstein polynomial with all coefficients in the unit interval:

$$g(t) = \frac{3}{4} \cdot [(1 - t)^2] + \frac{1}{4} \cdot [2t(1 - t)] + \frac{1}{2} \cdot [t^2].$$

Note that the coefficients of the Bernstein polynomial are $\frac{3}{4}$, $\frac{1}{4}$ and $\frac{1}{2}$, all of which are in the unit interval.

2. Implement the Bernstein polynomial with a multiplexing circuit, as shown in Fig. 3. The block labeled “+” counts the number of ones among its two inputs; this is either 0, 1, or 2. The multiplexer selects one of its three inputs as its output according to this value. Note that the inputs with probability t are each fed with *independent* stochastic streams with bits that have probability t .

Bernstein Polynomials

In this section, we introduce a specific type of polynomial that we use, namely Bernstein polynomials [12, 13].

Definition 1 A **Bernstein polynomial** of degree n , denoted as $B_n(t)$, is a polynomial expressed in the following form [15]:

$$\sum_{k=0}^n \beta_{k,n} b_{k,n}(t), \tag{12}$$

where each $\beta_{k,n}$, $k = 0, 1, \dots, n$ ¹ is a real number and

$$b_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}. \tag{13}$$

The coefficients $\beta_{k,n}$ are called **Bernstein coefficients** and the polynomials $b_{0,n}(t), b_{1,n}(t), \dots, b_{n,n}(t)$ are called **Bernstein basis polynomials** of degree n . □

We list some pertinent properties of Bernstein polynomials.

1. The *positivity* property:

For all $k = 0, 1, \dots, n$ and all t in $[0, 1]$, we have

$$b_{k,n}(t) \geq 0. \tag{14}$$

2. The *partition of unity* property:

The binomial expansion of the left-hand side of the equality $(t + (1-t))^n = 1$ shows that the sum of all Bernstein basis polynomials of degree n is the constant 1, i.e.,

$$\sum_{k=0}^n b_{k,n}(t) = 1. \tag{15}$$

3. Converting power-form coefficients to Bernstein coefficients:

The set of Bernstein basis polynomials $b_{0,n}(t), b_{1,n}(t), \dots, b_{n,n}(t)$ forms a basis of the vector space of polynomials of real coefficients and degree no more than n [14]. Each power basis function t^j can be uniquely expressed as a linear combination of the $n + 1$ Bernstein basis polynomials:

$$t^j = \sum_{k=0}^n \sigma_{jk} b_{k,n}(t), \tag{16}$$

for $j = 0, 1, \dots, n$. To determine the entries of the transformation matrix σ , we write

$$t^j = t^j (t + (1-t))^{n-j}$$

¹Here $\binom{n}{k}$ denotes the binomial coefficient “ n choose k .”

and perform a binomial expansion on the right hand side. This gives

$$t^j = \sum_{k=j}^n \frac{\binom{k}{j}}{\binom{n}{j}} b_{k,n}(t),$$

for $j = 0, 1, \dots, n$. Therefore, we have

$$\sigma_{jk} = \begin{cases} \binom{k}{j} \binom{n}{j}^{-1}, & \text{for } j \leq k \\ 0, & \text{for } j > k. \end{cases} \quad (17)$$

Suppose that a power-form polynomial of degree no more than n is

$$g(t) = \sum_{k=0}^n a_{k,n} t^k \quad (18)$$

and the Bernstein polynomial of degree n of g is

$$g(t) = \sum_{k=0}^n \beta_{k,n} b_{k,n}(t). \quad (19)$$

Substituting Eqs. (16) and (17) into Eq. (18) and comparing the Bernstein coefficients, we have

$$\beta_{k,n} = \sum_{j=0}^n a_{j,n} \sigma_{jk} = \sum_{j=0}^k \binom{k}{j} \binom{n}{j}^{-1} a_{j,n}. \quad (20)$$

Equation (20) provide a means for obtaining Bernstein coefficients from power-form coefficients.

4. Degree elevation:

Based on Eq. (13), we have that for all $k = 0, 1, \dots, m$,

$$\begin{aligned} & \frac{1}{\binom{m+1}{k}} b_{k,m+1}(t) + \frac{1}{\binom{m+1}{k+1}} b_{k+1,m+1}(t) \\ &= t^k (1-t)^{m+1-k} + t^{k+1} (1-t)^{m-k} \\ &= t^k (1-t)^{m-k} = \frac{1}{\binom{m}{k}} b_{k,m}(t), \end{aligned}$$

or

$$\begin{aligned}
 b_{k,m}(t) &= \frac{\binom{m}{k}}{\binom{m+1}{k}} b_{k,m+1}(t) + \frac{\binom{m}{k}}{\binom{m+1}{k+1}} b_{k+1,m+1}(t) \\
 &= \frac{m+1-k}{m+1} b_{k,m+1}(t) + \frac{k+1}{m+1} b_{k+1,m+1}(t).
 \end{aligned}
 \tag{21}$$

Given a power-form polynomial g of degree n , for any $m \geq n$, g can be uniquely converted into a Bernstein polynomial of degree m . Suppose that the Bernstein polynomials of degree m and degree $m + 1$ of g are $\sum_{k=0}^m \beta_{k,m} b_{k,m}(t)$ and $\sum_{k=0}^{m+1} \beta_{k,m+1} b_{k,m+1}(t)$, respectively. We have

$$\sum_{k=0}^m \beta_{k,m} b_{k,m}(t) = \sum_{k=0}^{m+1} \beta_{k,m+1} b_{k,m+1}(t).
 \tag{22}$$

Substituting Eq. (21) into the left-hand side of Eq. (22) and comparing the Bernstein coefficients, we have

$$\beta_{k,m+1} = \begin{cases} \beta_{0,m}, & \text{for } k = 0 \\ \frac{k}{m+1} \beta_{k-1,m} + \left(1 - \frac{k}{m+1}\right) \beta_{k,m}, & \text{for } 1 \leq k \leq m \\ \beta_{m,m}, & \text{for } k = m + 1. \end{cases}
 \tag{23}$$

Equation (23) provides a means for obtaining the coefficients of the Bernstein polynomial of degree $m + 1$ of g from the coefficients of the Bernstein polynomial of degree m of g . We will call this procedure *degree elevation*.

Uniform Approximation and Bernstein Polynomials with Coefficients in the Unit Interval

In this section, we present two of our major mathematical findings on Bernstein polynomials. The first result pertains to uniform approximation with Bernstein polynomials. We show that, given a power-form polynomial g , we can obtain a Bernstein polynomial of degree m with coefficients that are as close as desired to the corresponding values of g evaluated at the points $0, \frac{1}{m}, \frac{2}{m}, \dots, 1$, provided that m is sufficiently large. This result is formally stated by the following theorem.

Theorem 1 *Let g be a polynomial of degree $n \geq 0$. For any $\epsilon > 0$, there exists a positive integer $M \geq n$ such that for all integers $m \geq M$ and $k = 0, 1, \dots, m$, we have*

$$\left| \beta_{k,m} - g\left(\frac{k}{m}\right) \right| < \epsilon,$$

where $\beta_{0,m}, \beta_{1,m}, \dots, \beta_{m,m}$ satisfy $g(t) = \sum_{k=0}^m \beta_{k,m} b_{k,m}(t)$. \square

Please see [7] for the proof of the above theorem.

The second result pertains to a special type of Bernstein polynomials: those with coefficients that are all in the unit interval. We are interested in this type of Bernstein polynomial since we can show that it can be implemented by logical computation on stochastic bit streams

Definition 2 Define U to be the set of Bernstein polynomials with coefficients that are all in the unit interval $[0, 1]$:

$$U = \left\{ p(t) \mid \exists n \geq 1, 0 \leq \beta_{0,n}, \beta_{1,n}, \dots, \beta_{n,n} \leq 1, \text{ such that} \right. \\ \left. p(t) = \sum_{k=0}^n \beta_{k,n} b_{k,n}(t) \right\}. \square$$

The question we are interested in is: which (power-form) polynomials can be converted into Bernstein polynomials in U ?

Definition 3 Define the set V to be the set of polynomials which are either identically equal to 0 or equal to 1, or map the open interval $(0, 1)$ into $(0, 1)$ and the points 0 and 1 into the closed interval $[0, 1]$, i.e.,

$$V = \{p(t) \mid p(t) \equiv 0, \text{ or } p(t) \equiv 1, \\ \text{or } 0 < p(t) < 1, \forall t \in (0, 1) \text{ and } 0 \leq p(0), p(1) \leq 1\}. \square$$

We prove that the set U and the set V are equivalent, thus giving a clear characterization of the set U .

Theorem 2

$$V = U. \quad \square$$

The proof of the above theorem utilizes Theorem 1. Please see [7] for the proof.

We end this section with two examples illustrating Theorem 2. In what follows, we will refer to a Bernstein polynomial of degree n converted from a polynomial g

as “the Bernstein polynomial of degree n of g ”. When we say that a polynomial is of degree n , we mean that the power-form of the polynomial is of degree n .

Example 1 Consider the polynomial $g(t) = \frac{5}{8} - \frac{15}{8}t + \frac{9}{4}t^2$. It maps the open interval $(0, 1)$ into $(0, 1)$ with $g(0) = \frac{5}{8}$ and $g(1) = 1$. Thus, g is in the set V . Based on Theorem 2, we have that g is in the set U . We verify this by considering Bernstein polynomials of increasing degree.

- The Bernstein polynomial of degree 2 of g is

$$g(t) = \frac{5}{8} \cdot b_{0,2}(t) + \left(-\frac{5}{16}\right) \cdot b_{1,2}(t) + 1 \cdot b_{2,2}(t).$$

Note that the coefficient $\beta_{1,2} = -\frac{5}{16} < 0$.

- The Bernstein polynomial of degree 3 of g is

$$g(t) = \frac{5}{8} \cdot b_{0,3}(t) + 0 \cdot b_{1,3}(t) + \frac{1}{8} \cdot b_{2,3}(t) + 1 \cdot b_{3,3}(t).$$

Note that all the coefficients are in $[0, 1]$.

Since the Bernstein polynomial of degree 3 of g satisfies Definition 2, we conclude that g is in the set U . \square

Example 2 Consider the polynomial $g(t) = \frac{1}{4} - t + t^2$. Since $g(0.5) = 0$, thus g is not in the set V . Based on Theorem 2, we have that g is not in the set U . We verify this. By contraposition, suppose that there exist $n \geq 1$ and $0 \leq \beta_{0,n}, \beta_{1,n}, \dots, \beta_{n,n} \leq 1$ such that

$$g(t) = \sum_{k=0}^n \beta_{k,n} b_{k,n}(t).$$

Since $g(0.5) = 0$, therefore, $\sum_{k=0}^n \beta_{k,n} b_{k,n}(0.5) = 0$. Note that for all $k = 0, 1, \dots, n$, $b_{k,n}(0.5) > 0$. Thus, we have that for all $k = 0, 1, \dots, n$, $\beta_{k,n} = 0$. Therefore, $g(t) \equiv 0$, which contradicts the original assumption about g . Thus, g is not in the set U . \square

Synthesizing Polynomial Functions

Computation on stochastic bit streams generally implements a multivariate polynomial $F(x_1, \dots, x_n)$ with integer coefficients. The degree of each variable is at most one, i.e., there are no terms with variables raised to the power of two, three

or higher. If we associate some of the x_i 's of the polynomial $F(x_1, \dots, x_n)$ with real constants in the unit interval and the others with a common variable t , then the function F becomes a real-coefficient *univariate* polynomial $g(t)$. With different choices of the original Boolean function f and different settings of the probabilities of the x_i 's, we get different polynomials $g(t)$.

Example 3 Consider the function implemented by a multiplexer operating on stochastic bit streams, A , B , and S . It is a multivariate polynomial, $g(a, b, s) = sa + (1 - s)b = b + sa - sb$. The polynomial has integer coefficients. The degree of each variable is at most one. If we set $s = a = t$ and $b = 0.8$ in the polynomial, then we get a univariate polynomial $g(t) = 0.8 - 0.8t + t^2$. \square

The first question that arises is: what kind of univariate polynomials can be implemented by computation on stochastic bit streams? In [16], we proved the following theorem stating a necessary condition on the polynomials. The theorem essentially says that, given inputs that are probability values—that is to say, real values in the unit interval—the polynomial must also evaluate to a probability value. There is a caveat here: if the polynomial is not identically equal to 0 or 1, then it must evaluate to a value in the open interval $(0, 1)$ when the input is also in the open interval $(0, 1)$.

Theorem 3 *If a polynomial $g(t)$ can be implemented by logical computation on stochastic bit streams, then*

1. $g(t)$ is identically equal to 0 or 1 ($g(t) \equiv 0$ or 1), or
2. $g(t)$ maps the open interval $(0, 1)$ to itself ($g(t) \in (0, 1)$, for all $t \in (0, 1)$) and $0 \leq g(0), g(1) \leq 1$. \square

For instance, as shown in Example 3, the polynomial $g(t) = 0.8 - 0.8t + t^2$ can be implemented by logical computation on stochastic bit streams. It is not hard to see that $g(t)$ satisfies the necessary condition. In fact, $g(0) = 0.8$, $g(1) = 1$ and $0 < g(t) < 1$, for all $0 < t < 1$.

The next question that arises is: can *any* polynomial satisfying the necessary condition be implemented by logical computation on stochastic bit streams? If so, how? We propose a synthesis method that solves this problem; constructively, we show that, provided that a polynomial satisfies the necessary condition, we can implement it. First, in the section “[Synthesizing Bernstein Polynomials with Coefficients in the Unit Interval](#)”, we show how to implement a Bernstein polynomial with coefficients in the unit interval. Then, in the section “[Synthesis of Power-Form Polynomials](#)”, we describe how to convert a general power-form representation into such a polynomial.

Synthesizing Bernstein Polynomials with Coefficients in the Unit Interval

If all the coefficients of a Bernstein polynomial are in the unit interval, i.e., $0 \leq b_{i,n} \leq 1$, for all $0 \leq i \leq n$, then we can implement it with the construct shown in Fig. 4.

Fig. 4 Combinational logic that implements a Bernstein polynomial $B_n(t) = \sum_{i=0}^n b_{i,n} B_{i,n}(t)$ with all coefficients in the unit interval

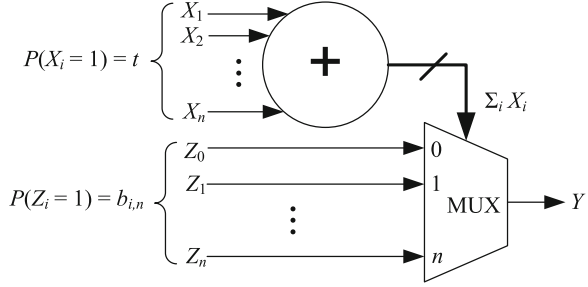
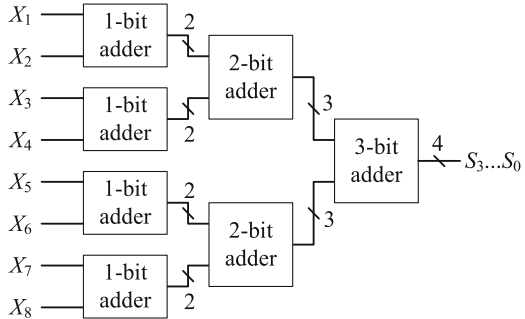


Fig. 5 The implementation of an 8-bit Boolean weight counter



The block labeled “+” in Fig. 4 has n inputs X_1, \dots, X_n and $\lceil \log_2(n + 1) \rceil$ outputs. It consists of combinational logic that computes the weight of the inputs, that is to say, it counts the number of ones in the n Boolean inputs X_1, \dots, X_n , producing a *binary radix encoding* of this count. We will call this an n -bit Boolean “weight counter.” The multiplexer (MUX) shown in the figure has “data” inputs Z_0, \dots, Z_n and the $\lceil \log_2(n + 1) \rceil$ outputs of the weight counter as the selecting inputs. If the binary radix encoding of the outputs of the weight counter is k ($0 \leq k \leq n$), then the output Y of the multiplexer is set to Z_k .

Figure 5 gives a simple design for an 8-bit Boolean weight counter based on a tree of adders. An n -bit Boolean weight counter can be implemented in a similar way.

In order to implement the Bernstein polynomial

$$B_n(t) = \sum_{i=0}^n b_{i,n} B_{i,n}(t),$$

we set the inputs X_1, \dots, X_n to be independent stochastic bit streams with probability t . Equivalently, X_1, \dots, X_n can be viewed as independent random Boolean variables that have the same probability t of being one. The probability that the count of ones among the X_i 's is k ($0 \leq k \leq n$) is given by the binomial distribution:

$$P\left(\sum_{i=1}^n X_i = k\right) = \binom{n}{k} t^k (1-t)^{n-k} = B_{k,n}(t). \tag{24}$$

We set the inputs Z_0, \dots, Z_n to be independent stochastic bit streams with probability equal to the Bernstein coefficients $b_{0,n}, \dots, b_{n,n}$, respectively. Notice that we can represent $b_{i,n}$ with stochastic bit streams because we assume that $0 \leq b_{i,n} \leq 1$. Equivalently, we can view Z_0, \dots, Z_n as $n + 1$ independent random Boolean variables that are one with probabilities $b_{0,n}, \dots, b_{n,n}$, respectively.

The probability that the output Y is one is

$$\begin{aligned} y &= P(Y = 1) \\ &= \sum_{k=0}^n \left(P \left(Y = 1 \mid \sum_{i=1}^n X_i = k \right) P \left(\sum_{i=1}^n X_i = k \right) \right). \end{aligned} \quad (25)$$

Since the multiplexer sets Y equal to Z_k , when $\sum_{i=1}^n X_i = k$, we have

$$P \left(Y = 1 \mid \sum_{i=1}^n X_i = k \right) = P(Z_k = 1) = b_{k,n}. \quad (26)$$

Thus, from Eqs. (13), (24), (25), and (26), we have

$$y = \sum_{k=0}^n b_{k,n} B_{k,n}(t) = B_n(t). \quad (27)$$

We conclude that the circuit in Fig. 4 implements the given Bernstein polynomial with all coefficients in the unit interval. We have the following theorem.

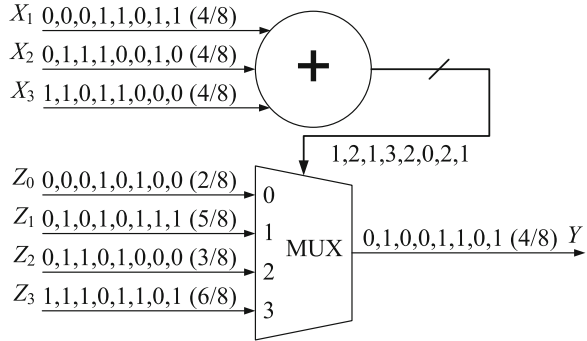
Theorem 4 *If all the coefficients of a Bernstein polynomial are in the unit interval, i.e., $0 \leq b_{i,n} \leq 1$, for $0 \leq i \leq n$, then we can synthesize logical computation on stochastic bit streams to implement it. \square*

Example 4 The polynomial $g_1(t) = \frac{1}{4} + \frac{9}{8}t - \frac{15}{8}t^2 + \frac{5}{4}t^3$ can be converted into a Bernstein polynomial of degree 3:

$$g_1(t) = \frac{2}{8}B_{0,3}(t) + \frac{5}{8}B_{1,3}(t) + \frac{3}{8}B_{2,3}(t) + \frac{6}{8}B_{3,3}(t). \quad \square$$

Figure 6 shows a circuit that implements this Bernstein polynomial. The function is evaluated at $t = 0.5$. The stochastic bit streams X_1, X_2 and X_3 are independent, each with probability $t = 0.5$. The stochastic bit streams Z_0, \dots, Z_3 have probabilities $\frac{2}{8}, \frac{5}{8}, \frac{3}{8}$, and $\frac{6}{8}$, respectively. As expected, the computation produces the correct output value: $g_1(0.5) = 0.5$. \square

Fig. 6 Computation on stochastic bit streams that implements the Bernstein polynomial $g_1(t) = \frac{2}{8}B_{0,3}(t) + \frac{5}{8}B_{1,3}(t) + \frac{3}{8}B_{2,3}(t) + \frac{6}{8}B_{3,3}(t)$ at $t = 0.5$



Synthesis of Power-Form Polynomials

In the previous section, we saw that we can implement a polynomial through logical computation on stochastic bit streams if the polynomial can be represented as a Bernstein polynomial with coefficients in the unit interval. A question that arises is: what kind of polynomials can be represented in this form? Generally, we seek to implement polynomials given to us in power form. In [16], we proved that any polynomial that satisfies Theorem 3—so essentially any polynomial that maps the unit interval onto the unit interval—can be converted into a Bernstein polynomial with all coefficients in the unit interval.² Based on this result and Theorem 4, we can see that the *necessary* condition shown in Theorem 3 is also a *sufficient* condition for a polynomial to be implemented by logical computation on stochastic bit streams.

Example 5 Consider the polynomial $g_2(t) = 3t - 8t^2 + 6t^3$ of degree 3. Since $g_2(t) \in (0, 1)$, for all $t \in (0, 1)$ and $g_2(0) = 0, g_2(1) = 1$, it satisfies the necessary condition shown in Theorem 3. Note that

$$\begin{aligned} g_2(t) &= B_{1,3}(t) - \frac{2}{3}B_{2,3}(t) + B_{3,3}(t) \\ &= \frac{3}{4}B_{1,4}(t) + \frac{1}{6}B_{2,4}(t) - \frac{1}{4}B_{3,4}(t) + B_{4,4}(t) \\ &= \frac{3}{5}B_{1,5}(t) + \frac{2}{5}B_{2,5}(t) + B_{5,5}(t). \end{aligned}$$

Thus, the polynomial $g_2(t)$ can be converted into a Bernstein polynomial with coefficients in the unit interval. The degree of such a Bernstein polynomial is 5, greater than that of the original power form polynomial. \square

²The degree of the equivalent Bernstein polynomial with coefficients in the unit interval may be greater than the degree of the original polynomial.

Given a power-form polynomial $g(t) = \sum_{i=0}^n a_{i,n} t^i$ that satisfies the condition of Theorem 3, we can synthesize it in the following steps:

1. Let $m = n$. Obtain $b_{0,m}, b_{1,m}, \dots, b_{m,m}$ from $a_{0,n}, a_{1,n}, \dots, a_{n,n}$ by Eq. (16).
2. Check to see if $0 \leq b_{i,m} \leq 1$, for all $i = 0, 1, \dots, m$. If so, go to step 4.
3. Let $m = m - 1$. Calculate $b_{0,m}, b_{1,m}, \dots, b_{m,m}$ from $b_{0,m-1}, b_{1,m-1}, \dots, b_{m-1,m-1}$ based on Eq. (13). Go to step 2.
4. Synthesize the Bernstein polynomial

$$B_m(t) = \sum_{i=0}^m b_{i,m} B_{i,m}(t)$$

with the generalized multiplexing construct in Fig. 4.

Synthesizing Non-Polynomial Functions

In real applications, we often encounter non-polynomial functions, such as trigonometric functions. In this section, we discuss the implementation of such functions; further details are given in [18]. Our strategy is to approximate them by Bernstein polynomials with coefficients in the unit interval. In the previous section, we saw how to implement such Bernstein polynomials.

We formulate the problem of implementing an arbitrary function $g(t)$ as follows. Given $g(t)$, a continuous function on the unit interval, and n , the degree of a Bernstein polynomial, find real numbers $b_{i,n}$, $i = 0, \dots, n$, that minimize

$$\int_0^1 (g(t) - \sum_{i=0}^n b_{i,n} B_{i,n}(t))^2 dt, \quad (28)$$

subject to

$$0 \leq b_{i,n} \leq 1, \text{ for all } i = 0, 1, \dots, n. \quad (29)$$

Here we try to find the optimal approximation by minimizing an objective function, Eq. (28), that measures the approximation error. This is the square of the L^2 norm on the difference between the original function $g(t)$ and the Bernstein polynomial $B_n(t) = \sum_{i=0}^n b_{i,n} B_{i,n}(t)$. The integral is on the unit interval because t , representing a probability value, is always in the unit interval. The constraints in Eq. (29) guarantee that the Bernstein coefficients are all in the unit interval. With such coefficients, the construct in Fig. 4 computes an optimal approximation of the function.

The optimization problem is a constrained quadratic programming problem [18]. Its solution can be obtained using standard techniques.

Example 6 Consider the non-polynomial function $g_3(t) = t^{0.45}$. We approximate this function by a Bernstein polynomial of degree 6. By solving the constrained quadratic optimization problem, we obtain the Bernstein coefficients:

$$b_{0,6} = 0.0955, b_{1,6} = 0.7207, b_{2,6} = 0.3476, b_{3,6} = 0.9988, \\ b_{4,6} = 0.7017, b_{5,6} = 0.9695, b_{6,6} = 0.9939. \quad \square$$

Discussion

This chapter presented a necessary and sufficient condition for synthesizing stochastic functions with combinational logic: the target function must be a polynomial that maps the unit interval $[0, 1]$ to the unit interval $[0, 1]$. The “necessary” part was easy: given stochastic inputs, any combinational circuits produces a polynomial. Since the inputs and outputs are probabilities, this polynomial maps the unit interval to the unit interval.

The “sufficient” part entailed some mathematics. First we showed that any polynomial given in power form can be transformed into a Bernstein polynomial. This was well known [13]. Next we showed that, by elevating the degree of the Bernstein polynomial, we always obtain a Bernstein polynomial with coefficients in the unit interval. This was a new result, published in [16]. Finally, we showed that any Bernstein polynomial with coefficients in the unit interval can be implemented by a form of “general multiplexing”. These results were published in [17, 18].

The synthesis method is both general and efficient. For a wide variety of applications, it produces stochastic circuits that have remarkably small area, compared to circuits that operate on a conventional binary positional encodings [18]. We note that our characterization applies only to combinational circuits, that is to say logic circuits without memory elements. Dating back to very interesting work by Brown & Card [4], researchers have explored stochastic computing with sequential circuits, that is to say logic circuits with memory elements. With sequential circuits, one can implement a much larger class of functions than polynomials. For instance, Brown & Card showed that a sequential circuit can implement the *tanh* function. A complete characterization of what sort of stochastic functions can be computed by sequential circuits has not been established. However, we point the reader to recent work on the topic: [19–22].

References

1. B. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*. Plenum Press, 1969, vol. 2, ch. 2, pp. 37–172.
2. W. J. Poppelbaum, A. Dollas, J. B. Glickman, and C. Otoole, "Statistical processors," in *Advances in Computers*, M. C. Yovits, Ed., 1976, vol. 17, pp. 187–230.
3. —, "Unary processing," in *Advances in Computers*, M. C. Yovits, Ed., 1987, vol. 26, pp. 48–89.
4. B. Brown and H. Card, "Stochastic neural computation I: Computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.
5. P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams: Digital image processing case studies," *IEEE Transactions on VLSI Systems*, vol. 22, pp. 449–462, 2014.
6. A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "Vlsi implementation of deep neural network using integral stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2688–2699, Oct 2017.
7. W. Qian, "Digital yet deliberately random: Synthesizing logical computation on stochastic bit streams," Ph.D. dissertation, University of Minnesota, 2011.
8. K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Transactions on Computers*, vol. 24, no. 6, pp. 668–670, 1975.
9. J. Savir, G. Ditlow, and P. H. Bardell, "Random pattern testability," *IEEE Transactions on Computers*, vol. 33, no. 1, pp. 79–90, 1984.
10. J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," in *Design Automation Conference*, 2001, pp. 661–666.
11. R. Marculescu, D. Marculescu, and M. Pedram, "Logic level power estimation considering spatiotemporal correlations," in *International Conference on Computer-Aided Design*, 1994, pp. 294–299.
12. S. N. Bernstein, "Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités," *Communications of the Kharkov Mathematical Society*, vol. 13, pp. 1–2, 1912.
13. G. Lorentz, *Bernstein Polynomials*. University of Toronto Press, 1953.
14. R. Farouki and V. Rajan, "On the numerical condition of polynomials in Bernstein form," *Computer Aided Geometric Design*, vol. 4, no. 3, pp. 191–216, 1987.
15. J. Berchtold and A. Bowyer, "Robust arithmetic for multivariate Bernstein-form polynomials," *Computer-Aided Design*, vol. 32, no. 11, pp. 681–689, 2000.
16. W. Qian, M. D. Riedel, and I. Rosenberg, "Uniform approximation and Bernstein polynomials with coefficients in the unit interval," *European Journal of Combinatorics*, vol. 32, no. 3, pp. 448–463, 2011.
17. W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Design Automation Conference*, 2008, pp. 648–653.
18. W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.
19. P. Li, W. Q. M. D. Riedel, K. Bazargan, and D. J. Lilja, "The synthesis of linear finite state machine-based stochastic computational elements," in *Asia and South Pacific Design Automation Conference*, 2012, pp. 757–762.
20. P. Li, D. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," in *International Conference on Computer-Aided Design*, 2012, pp. 480–487.
21. P. Li, D. J. Lilja, W. Qian, M. D. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finite state machines," *IEEE Transactions on Computers*, 2013.
22. N. Saraf and K. Bazargan, "Sequential logic to transform probabilities," in *International Conference on Computer-Aided Design*, 2013.

Deterministic Approaches to Bitstream Computing



Marc Riedel

Abstract Stochastic logic allows complex arithmetic to be performed with very simple logic, but it suffers from high latency and poor precision. Furthermore, the results are always somewhat inaccurate due to random fluctuations. The random or pseudorandom sources required to generate the representation are costly, consuming a majority of the circuit area (and diminishing the overall gains in area). This chapter reexamines the foundations of stochastic computing and comes to some surprising conclusions. It demonstrates that one can compute deterministically using the same structures that are used to compute stochastically. In doing so, the latency is reduced by an exponential factor; also, the area is reduced significantly (and this correlates with a reduction in power); and finally, one obtains completely accurate results, with no errors or uncertainty. This chapter also explores an alternate view of this deterministic approach. Instead of viewing signals as digital bit streams, we can view them as periodic signals, with the value encoded as the fraction of the time that the signal is in the high (on) state compared to the low (off) state in each cycle. Thus we have a *time-based encoding*. All of the constructs developed for stochastic computing can be used to compute on these periodic signals, so the designs are very efficient in terms of area and power. Given how precisely values can be encoded in the time, the method could produce designs that have much lower latency than conventional ones.

Keywords Deterministic computation · Unary · Pulse-width modulation · Convolution · Time-based computing

M. Riedel (✉)
University of Minnesota, Minneapolis, MN, USA
e-mail: mriedel@umn.edu

© Springer Nature Switzerland AG 2019
W. J. Gross and V. C. Gaudet (eds.), *Stochastic Computing: Techniques and Applications*, https://doi.org/10.1007/978-3-030-03730-7_6

Introduction

As detailed throughout this book, the topic of stochastic computing has been investigated from many angles, by many different researchers. In spite of the activity, it is fair to say that the practical impact of the research has been modest. In our view, interest has been sustained because of the intellectual appeal of the paradigm. It presents a completely different way of computing functions with digital logic. Complex functions can be computed with remarkably simple structures. For instance, multiplication can be performed with a single AND gate. Complex functions such as exponentiation, absolute value, square roots, and hyperbolic tangent can each be computed with a very small number of gates [1]. Although this is a claim that can only be justified through design examples, stochastic designs consistently achieve $50\times$ to $100\times$ reductions in gate count over a wide range of applications in signal, image and video processing, compared to conventional binary radix designs [1]. Savings in area correlate well with savings in power, a critical metric.

Note that while stochastic computation is digital—operating on 0s and 1s—and performed with ordinary logic gates, it has an “analog” flavor: conceptually, the computation consists of mathematical operations on real values, the probabilities of the streams. The approach is a compelling and natural fit for computing mathematical functions, for applications such as image processing and neural processing.

The intellectual appeal notwithstanding, the approach has a glaring weakness: the latency it incurs. A stochastic representation is not compact: to represent 2^M distinct numbers, it requires roughly 2^{2M} bits, whereas a conventional binary representation requires only M bits. When computing on serial bit streams, this results in an exponential, near-disastrous increase in latency. The simplicity of the logic generally translates to very short critical paths, so one could, in principle, bump up the clock to very high rates. This could mitigate the increase in latency. But there are practical limitations to increasing the clock rate [2, 3].

Another issue is the cost of generating randomness. Most implementations have used pseudo-random number generators such as linear-feedback shift registers (LFSRs). The cost of these easily overwhelms the total area cost, completely offsetting the gains made in the structures for computation [4, 5]. Researchers have explored sources of true randomness [6, 7]. Indeed, with emerging technologies such as nanomagnetic logic, exploiting true randomness from physical sources could tip the scales, making stochastic computing a winning proposition [8]. Still, the latency and the cost of interfacing random signals with deterministic signals make it a hard sell.

In this chapter, we reexamine the foundations of stochastic computing, and come to some surprising conclusions. Why is computing on probabilities so powerful, conceptually? Why can complex functions be computed with such simple structures? Intuition might suggest that somehow we are harnessing deep aspects of probability theory; perhaps we are computing approximate answers to hard

problems efficiently through “sampling” as with Monte Carlo simulations. This intuition is wrong.

The key to the efficiency of stochastic computing is much simpler: it stems from performing streaming computation on values represented by *quantity*, on a uniform representation, rather than representing values by *position*, as they are in binary radix. In this chapter, we demonstrate that, if the computation is properly structured, we can compute deterministically on bit streams using the same structures as we use when computing stochastically.

Next, this chapter explores a generalization of this deterministic approach. Instead of computing on bit *streams*, we explore computing on *periodic* signals, with the value encoded as the fraction of the time that the signal is in the high (on) state compared to the low (off) state in each cycle. Consider the examples in Figs. 1 and 2. We will call digital bit streams of this sort **uniform** bit streams. We will call analog signals of this sort **pulse-width modulated (PWM)** signals. By exploiting pulse width modulation, signals with specific values can be generated by adjusting the frequency and duty cycles of the PWM signals. Note that a PWM signal is, in fact, *digital* in the sense that the voltage level is either 0 or 1. However, it represents a real-valued (analog) signal by its duration.

But how can one compute on such periodic signals? Recall that with stochastic logic, multiplication is performed with a single AND gate. Simply connecting two periodic signals to the inputs of an AND gate will evidently not work. With the two signals lining up, multiplying $1/2$ by $1/2$ would produce an output signal equal to $1/2$,

Fig. 1 Digital signals encoded as periodic bit streams. The values represented are (a) 0.5, (b) 0.875, and (c) 0.455

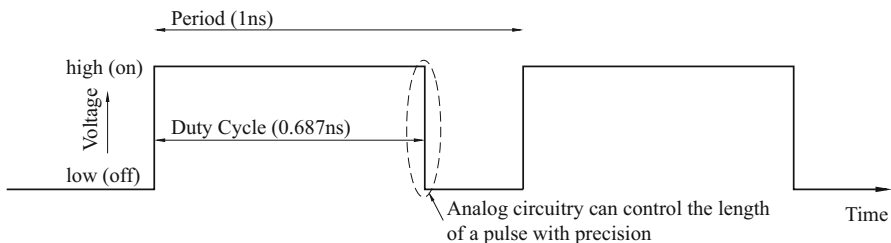
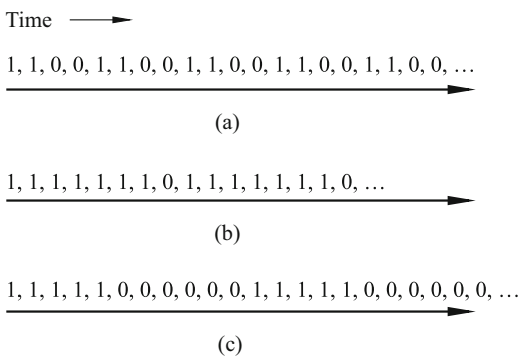


Fig. 2 A Pulse-Width Modulated (PWM) signal. The value represented is the fraction of the time that the signal is high in each cycle, in this case 0.687

Fig. 3 Multiplication with a single AND gate, operating on deterministic periodic bit streams

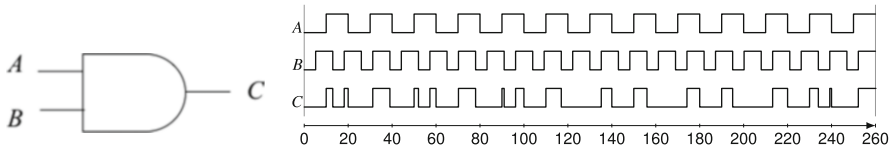
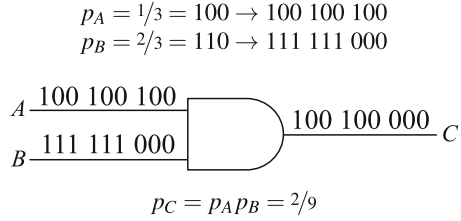


Fig. 4 Multiplication with a single AND gate: operating on deterministic periodic signals. Signal *A* represents 0.5 with a period of 20ns; Signal *B* represents 0.6 with a period of 13ns. The output signal *C* from $t=0$ ns to 260ns represents 0.30, the expected value from multiplication of the inputs

not equal to $1/4$, the value required. However, suppose that one adopts the following strategy when generating the bit streams: hold each bit of one stream, while cycling through all the bits of the other stream. Figure 3 gives an example. Here the value $1/3$ is represented by the bits 100 repeating, while the value $2/3$ is represented by the 110, clock-divided by three. The result is $2/9$, as expected. This method works in general for all stochastic constructs.

In an analogous way, we can perform operations on PWM signals. For instance, one can use periodic signals with relatively prime frequencies. Figure 4 shows an example of multiplying two values, 0.5 and 0.6, represented as PWM signals. The period of the first is 20ns and that of the second is 13ns. The figure shows that, after performing the operation for 260ns, the fraction of the total time the output signal is high equals the value expected when multiplying the two input values, namely 0.3.

The idea of computing on time-encoded signals has a long history [9–12]. We have been exploring the idea of time-based computing with constructs developed for stochastic computing [13, 14]. We note that other researchers have explored very similar ideas in the context of LDPC decoding [15].

As we will argue, compared to computing on stochastic bit streams, we can reduce the latency significantly—by an exponential factor—with deterministic approaches. Of course, compared to binary radix, uniform bit streams still incur high latency. However, with PWM signals, the precision is no longer dependent on the length of pulses, but rather on how accurately the duty cycle can be set.

As technology has scaled and device sizes have gotten smaller, the supply voltages have dropped while the device speeds have improved [16]. Control of the dynamic range in the voltage domain is limited; however, control of the length of pulses in the time domain can be precise [16, 17]. Encoding data in the time

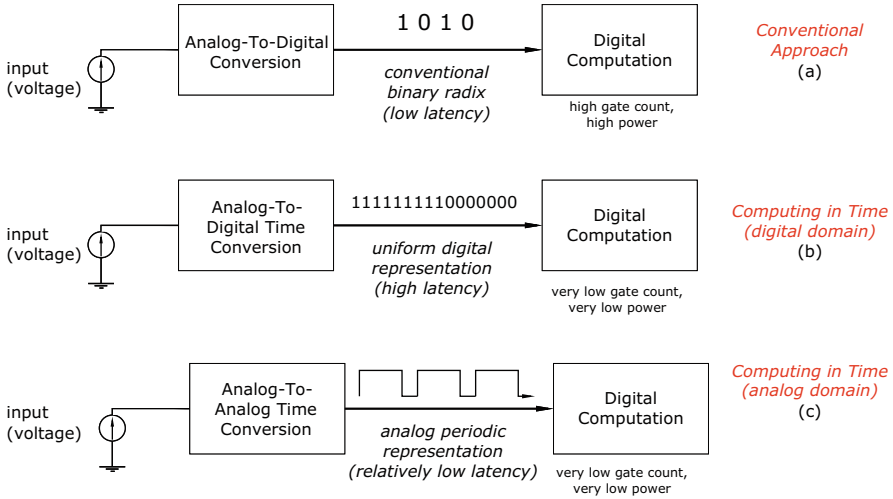


Fig. 5 Comparison of (a) the conventional approach, namely digital computation on binary radix; to (b) our methodology on uniform bit streams; and (c) our methodology on pulse-width modulated (PWM) signals

domain can be done more accurately and more efficiently than converting signals into binary radix. Given how precisely values can be encoded in time, our method could produce designs that are much faster than conventional ones—operating in the terahertz range. Figure 5 compares the conventional approach, consisting of an analog-to-digital converter (ADC) that produces binary radix, to the new methods that we are proposing here.

A Deterministic Approach

The benefit of stochastic computing is that complex operations can be performed with very simple logic. We point the reader to a subset of the work that demonstrates this: [4, 5, 18, 18–29]. This body of work includes examples of both basic and applied operations, ranging from multiplication, scaled addition, and exponentiation; to polynomial approximations of trigonometric functions; to LDPC decoders; to video processing operations such as edge detection. Across the board, the examples demonstrate a reduction in area by an order of magnitude or more compared to conventional designs.

An obvious drawback of the stochastic paradigm is the high latency that results, due to the length of the bit streams. Another is that the computation suffers from errors due to random fluctuations and correlations between the streams. These effects worsen as the circuit depth and the number of inputs increase [4]. While the logic to perform the computation is simple, generating random or pseudorandom bit

streams is costly. Indeed, in prior work, pseudorandom constructs such as linear feedback shift registers (LFSRs) accounted for as much as 90% of the area of stochastic circuit designs [19, 20]. This significantly diminishes the area benefits.

In this chapter, we argue that randomness is *not* a requirement for the paradigm. We show that the same computation can be performed on deterministically generated bit streams. The results are completely accurate, with no random fluctuations. Without the requirement of randomness, bit streams can be generated inexpensively. Most importantly, with our approach, the latency is reduced by a factor of approximately $1/2^n$, where n is the equivalent number of bits of precision. (For example, for the equivalent of 10 bits of precision, the bit stream length is reduced from 2^{20} to only 2^{10} .) As is the case with stochastic bit streams, all bits in our deterministic streams are weighted equally. Accordingly, as is the case with stochastic circuits, our deterministic circuits have a high degree of tolerance to soft errors.

Intuitive View

Conceptually, an operation such as multiplication in stochastic logic works by randomly *sampling* the inputs. This is achieved by randomizing the input bit streams and then intersecting them. This approach is easy to understand but incurs a lot of overhead: one must create the random bit streams, say with constructs such as LFSRs; this is costly. Furthermore, one must do a *lot* of sampling. Indeed, as explained in the section “[Comparing Stochastic and Deterministic Representations](#)”, in order to obtain a result that is equivalent in precision to n bits, one must sample 2^{2n} bits. Randomness requires, in effect, “oversampling” to get a statistically accurate result [5].

But is such randomly sampling necessary? *Why not simply intersect two deterministic bit streams.* Consider the mathematical operation of convolution. Intuitively, it consists of three operations: slide, multiply, and sum. Figure 6 illustrates this. If we implement this operation on uniform deterministic bit streams, the result will be equivalent to a stochastic operation.

Example 1 An example of multiplication by “clock-diving” one input stream and repeating the other was shown in Fig. 3 in the introduction. Suppose that we wish to perform scaled addition, say on inputs $p_A = 1/3$, $p_B = 2/3$, and $p_S = 2/3$. We can divide the bit stream on the select input S to a multiplexer, while the bit streams for the operands A and B repeat:

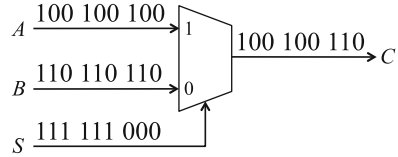
$$p_A = 1/3 = 100 \rightarrow 100100100$$

$$\sum_{i=1}^L \sum_{j=1}^L X_i Y_j \quad \begin{array}{r} X \\ Y \end{array} \begin{array}{r} 110 \\ 100 \end{array} \rightarrow$$

(a)
(b)

Fig. 6 Discrete convolution. (a) Mathematical operation on two bit streams, X and Y . (b) Intuition: convolution is equivalent to sliding one bit streams past the other

Fig. 7 Scaled addition via convolution, by clock dividing a signal



$$p_B = 2/3 = 110 \rightarrow 110110110$$

$$p_S = 2/3 = 110 \rightarrow 111111000$$

□

Figure 7 illustrates that the result is $p_C = p_S p_A + (1 - p_S) p_B = 2/9 + 2/9 = 4/9$.

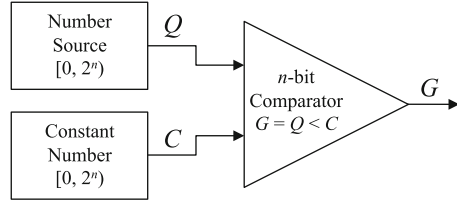
Comparing Stochastic and Deterministic Representations

A stochastic representation maintains the property that each bit of one stream meets every bit of an other stream the same number of times, but this property occurs *on average*, meaning the bit streams have to be much longer than the resolution they represent due to random fluctuations. The bit stream length N required to estimate the average proportion within an error margin ϵ is

$$N > \frac{p(1 - p)}{\epsilon^2}$$

(This is proved in [5].) To represent a value within a binary resolution $1/2^n$, the error margin ϵ must equal $1/2^{n+1}$. Therefore, the bit stream must be greater than 2^{2n} uniform bits long, as the $p(1 - p)$ term is at most 2^{-2} . This means that the length of a stochastic bit stream increases *exponentially* with the desired resolution. This results in enormously long bit streams. For example, if we want to find the proportion of a random bit stream with 10-bit resolution ($1/2^{10}$), we will have to observe at least 2^{20} bits. This is over a thousand times longer than the bit stream required by a deterministic uniform representation.

The computations also suffer from some level of correlation between bit streams. This can cause the results to bias away from the correct answer. For these reasons, stochastic logic has only been used to perform approximate computations. Another related issue is that the LFSRs must be at least as long as the desired resolution in order to produce bit streams that are sufficiently random. A “Randomizer Unit”, described in [22], uses a comparator and LFSR to convert a binary encoded number into a random bit stream. Each independent random bit stream requires its own generator. Therefore, circuits requiring i independent inputs with n -bit resolution

Fig. 8 Converter module

need i LFSRs with length L approximately equal to $2n$. This results in the LFSRs dominating a majority of the circuit area.

By using deterministic bit streams, we avoid all problems associated with randomness while retaining all the computational benefits associated with a stochastic representation. However, we can use *much* shorter bit streams to achieve the same precision: to represent a value with resolution $1/2^n$ in a deterministic representation, the bit stream must be 2^n bits long. The computations are also completely accurate; they do not suffer from correlation. The next section discusses three methods for generating independent deterministic bit streams and gives their circuit implementations. Without the requirement of randomness, the hardware cost of the bit stream generators is reduced, so it is a win in every respect.

Deterministic Methods

We present three alternative approaches to deterministic computation on uniform bit streams. These differ in how the uniform bit streams are generated. We note that the computational structures themselves are identical to those developed for stochastic computing. Accordingly, existing designs for arithmetic, signal processing and video processing can be used. We illustrate the approach with the simplest example: multiplication with an AND gate.

The three methods for generating the uniform bit streams are: (1) using relatively prime lengths; (2) rotation; and (3) clock division. For each method, the hardware complexity of the circuit implementation is given. The computation time of each method is the same. Each method is implemented using a bit stream generated from “converter” modules, illustrated in Fig. 8. The modules are similar to the “Randomizer Unit” [19]; the difference is that the LFSR is replaced by a deterministic number source.

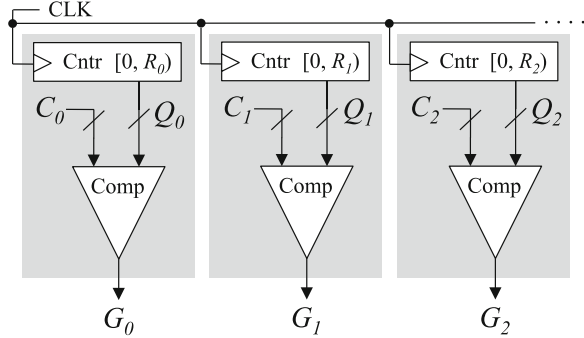
Relatively Prime Bit Lengths

The “relatively prime” method maintains independence by using bit streams that have relatively prime lengths. Here the ranges $[0, R_i)$ between converter modules

$a_0 a_1 a_2 a_3 a_0 a_1 a_2 a_3 a_0 a_1 a_2 a_3$
 $b_0 b_1 b_2 b_0 b_1 b_2 b_0 b_1 b_2 b_0 b_1 b_2$

Fig. 9 Two bit streams generated by the “relatively prime” method

Fig. 10 Circuit implementation of the “relatively prime” method



are relatively prime. Figure 9 demonstrates the method with two bit streams *A* and *B*, one with operand length four and the other with operand length three. The bit streams are shown in array notation to show the position of each bit in time.

Independence between bit streams is maintained because the remainder, equal to the overlap between bit streams, always results in a new rotation (or initial phase) of stream. Intuitively, this occurs because the bit lengths share no common factors. This results in every bit of each operand seeing every bit of the other operand. For example, *a*₀ sees *b*₀, *b*₁, and *b*₂; *b*₀ sees *a*₀, *a*₃, *a*₂, and *a*₁; and so on. Using two bit streams with relatively prime bit lengths *j* and *k*, the output of a logic gate repeats with period *jk*. This means that, with multi-level circuits, the output of the logic gates will also be relatively prime. This allows for the same arithmetic logic as a stochastic representation.

A circuit implementation of the “relatively-prime” method is shown in Fig. 10. Each converter module uses a counter as a number source for iterating through each bit of the stream. The state of the counter *Q*_{*i*} is compared with the stream constant *C*_{*i*}. The relatively prime counter ranges *R*_{*i*} between modules maintain independence. In terms of general circuit components, the circuit uses *i* counters and *i* comparators, where *i* is the number of generated independent bit streams. Assuming the max range is a binary resolution 2^{*n*} and all modules are close to this value (i.e., 256, 255, 253, 251...), the circuit contains approximately *i* *n*-bit counters and *i* *n*-bit comparators.

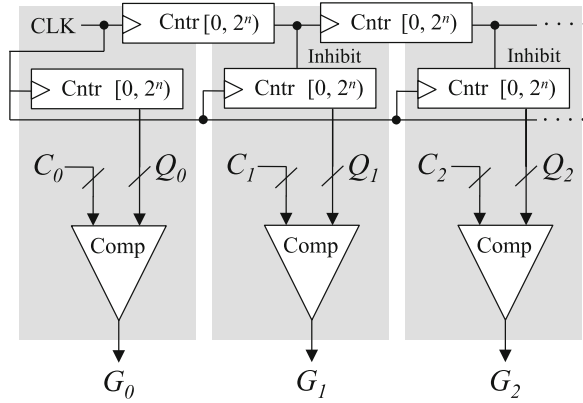
Rotation

In contrast to the previous method, the “rotation” method allows bit streams of arbitrary length to be used. Instead of relying on relatively prime lengths, the bit

Fig. 11 Two bit streams generated by the “rotation” method

$a_0 a_1 a_2 a_3 a_0 a_1 a_2 a_3 a_0 a_1 a_2 a_3 a_0 a_1 a_2 a_3$
 $b_0 b_1 b_2 b_3 b_3 b_0 b_1 b_2 b_2 b_3 b_0 b_1 b_1 b_2 b_3 b_0$

Fig. 12 Circuit implementation of the “rotation” method



streams are explicitly rotated. This requires the sequence generated by the number source to change after it iterates through its entire range. For example, a simple way to generate a bit stream where the stream lengths rotates in time is to inhibit or stall a counter every 2^n clock cycles (where n is the length of the counter). Figure 11 demonstrates this method with two bit streams, both of length four.

By rotating bit stream B 's length, it is straightforward to see that each bit of one bit stream sees every bit in the other stream. Assuming all streams have the same length, we can extend the example with two bit streams to examples with multiple bit streams; here we would be inhibiting counters at powers of the operand length. This allows the operands to rotate relative to longer bit streams.

A circuit implementation, shown in Fig. 12, follows from the previous example. We can generate any number of independent bit streams as long as the counter of every i th converter module is inhibited every 2^{ni} clock cycles. This can be managed by adding additional counters between each module. These counters control the phase of each converter module and maintain the property that each converter module rotates relative to the other modules. Using n -bit binary counters and comparators, the circuit requires i n -bit comparators and $2i - 1$ n -bit counters. The advantage of using rotation as a method for generating independent bit streams is that we can use operands with the same resolution, but this requires slightly more circuitry than the “relatively-prime” method.

Fig. 13 Two bit streams generated by the “clock division” method

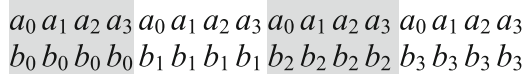
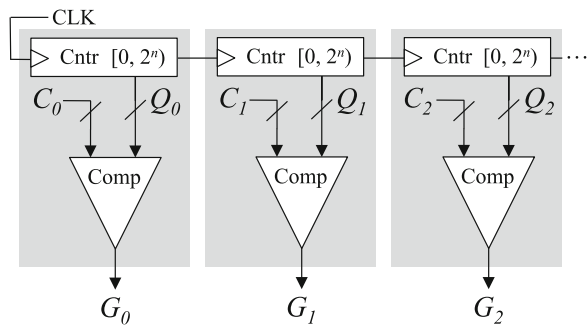


Fig. 14 Circuit implementation of the “clock division” method



Clock Division

The “clock division” method works by clock dividing operands. Similar to the “rotation” method, it operates on streams of arbitrary lengths. (This method was first seen in Examples 1 and 2 in the section “Intuitive View”.) Figure 13 demonstrates this method with two bit streams, both with bit streams of length four. Bit stream *B* is clock divided by the length of bit stream *A*’s value.

Assuming all operands have the same length, we can generate an arbitrary number of independent bit streams as long as the counter of every *i*th converter module increments every 2^{ni} clock cycles. This can be implemented in circuit form by simply chaining the converter module counters together, as shown in Fig. 14. Using *n*-bit binary counters and comparators, the circuit requires *i* *n*-bit comparators and *i* *n*-bit counters. This means the “clock division” method allows operands of the same length to be used with approximately the same hardware complexity as the “relatively-prime” method.

Comparing the Three Deterministic Methods to Stochastic Methods

Here we compare the hardware complexity and latency of the deterministic methods with conventional stochastic methods. Perfectly precise computations require the output resolution to be at least equal to the product of the independent input resolutions. For example, with input bit stream lengths of *n* and *m*, the precise output contains *nm* bits.

Consider a stochastic representation implemented with LFSRs. As discussed in the section “Comparing Stochastic and Deterministic Representations”, a stochastic representation requires bit streams that are 2^{2n} bits long to represent a value with $1/2^n$ precision. In order to ensure that the generated bit streams are sufficiently random

Table 1 Gate count for basic deterministic components, where n is the resolution and i is the number of inputs

Component	Gate count
Comparator	$3n$
Counter	$6n$
LFSR	$12ni$

Table 2 Gate count for stochastic and deterministic bit stream generators, where n is resolution and i is the number of inputs. Latency for each method

Representation	Method	Gate count	Latency
Stochastic	Randomizer	$12ni^2 + 3ni$	2^{2ni}
Deterministic	Rel. prime	$9ni$	2^{ni}
	Rotation	$15ni - 6n$	
	Clock div.	$9ni$	

and independent, each LFSR must have at least as many states as the required output bit stream. Therefore, to compute with perfect precision each LFSR must have at least length $2ni$.

With our deterministic methods, the resolution n of each of the i inputs is determined by the length of its converter module counter. The output resolution is simply the product of the counter ranges. For example, with the “clock division” method, each converter module counter is connected in series. The series connection forms a large counter with 2^{ni} states. This shows that output resolution is not determined by the length of each individual number source, but by their concatenation. This allows for a large reduction in circuit area compared to stochastic designs.

To compare the area of the circuits, we assume three gates for every cell of a comparator and six gates for each flip-flop of a counter or LFSR (this is similar to the hardware complexity used in [29] in terms of fanin-two NAND gates). For i inputs with n -bit binary resolution, the gate count for each basic component is given by Table 1. Table 2 gives the total gate count and bit stream length for precise computations in terms of independent inputs i with resolution n for prior stochastic methods as well as the deterministic methods that we propose here. The basic component totals for each deterministic method were discussed in section “[Deterministic Methods](#)”. For stochastic methods, we assume that each “Randomizer Unit” needs one comparator and one LFSR per input.

The equations of Table 2 show that our deterministic methods use less area and compute to the same precision, in exponentially less time. It is a win on both metrics, but the reduction in latency is especially compelling. Consider a reduction in latency from $1/2^{20} = 1,048,576$, to just $1/2^{10} = 1,024$!

An Analog Approach

Building on the insight that stochastic computation can be implemented deterministically, we explore computation on “Pulse-Width Modulated” (PWM) signals. We encode values as the fraction of the time that the signal is in the high (on)

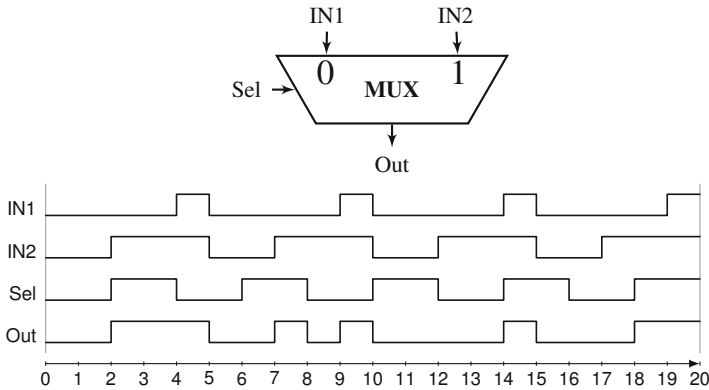


Fig. 15 An example of the scaled addition of two PWM signals using a MUX. Here IN1 and IN2 represent 0.2 and 0.6 with a period of 5ns. Sel represents 0.5 with a period of 4ns. The output signal from $t=0\text{ns}$ to 20ns represents 0.40 ($8\text{ns}/20\text{ns}=4/10$), the expected value from the scaled addition of the inputs

compared to the low (off) state in each cycle. An example was shown in Fig. 2 in the introduction.

As we will show, the key is choosing different periods for the PWM signals, and letting the system run over multiple cycles. If we choose relatively prime periods and run the signals to their common multiple, we achieve the effect of “convolving” the signals. This is analogous to the approach that we took with deterministic digital bit streams in the section “[Relatively Prime Bit Lengths](#)”, where we used relatively prime bit stream lengths.

Figure 4 in the introduction showed an example of multiplication on PWM signals. Here we show an example of addition. Recall that with stochastic logic, scaled addition can be performed with a multiplexer (MUX). The performance of a MUX as a stochastic scaled adder/subtractor is insensitive to the correlation between its inputs. This is because only one input is connected to the output at a time [24]. Thus, highly overlapped inputs like PWM signals with the same frequency can be connected to the inputs of a MUX. The important point when performing scaled addition and subtraction with a MUX on PWM signals is that the period of the select signal should be relatively prime to the period of the input signals.

Figure 15 shows an example of scaled addition on two numbers, 0.2 and 0.6, represented by two PWM signals. Both have periods of 5ns. A PWM signal with a duty cycle of 50% and period of 4ns is connected to the select input of the MUX. As shown, after performing the operation for 20ns, the fraction of the total time the output signal is high equals the expected value, 0.40.

Conclusion

While it is easy conceptually to understand how stochastic computation works, randomness is costly. This chapter argues that randomness is not necessary. Instead of relying upon statistical sampling to operate on bit streams, we can explicitly “convolve” them: we slide one operand past the other, performing bitwise operations. We argued that the logic to perform this convolution is less costly than that to generate pseudorandom bit streams. More importantly, we can use much shorter bit streams to achieve the same accuracy as with statistical sampling through randomness. Indeed, the results of our computation are predictable and completely accurate for all input values.

Of course, compared to a binary radix representation, our deterministic representation is still not very compact. With M bits, a binary radix representation can represent 2^M distinct numbers. To represent real numbers with a resolution of 2^{-M} , i.e., numbers of the form $\frac{a}{2^M}$ for integers a between 0 and 2^M , we require a stream of 2^M bits. However, contrast this with a stochastic representation that requires 2^{2M} bits to achieve the same precision!

We conclude that there is no clear reason to compute on stochastic bit streams. Even when randomness is free, say harvested from thermal noise or some other physical source, stochastic computing entails very high latency. In contrast, computation on deterministic uniform bit streams is less costly, has much lower latency, and is completely accurate.

We do note that there is one drawback to the approach: bit stream lengths grow with each level of logic. This is, in fact, a mathematical requirement. Consider the multiplication of two numbers, each encoded with a precision of n binary bits. Regardless of the encoding, the precision of the result must be greater than the precision of the two operands: up to n^2 bits are required. Stochastic encodings have the same requirement. However, with randomness it is easy to approximate the result, by simply truncating the length of the streams. Accordingly, most stochastic circuits keep constant bit stream lengths regardless of the levels of logic. We concede that there is no straight-forward way to optimally truncate the results of our deterministic computation. See [30] for a discussion of this topic.

This chapter also presented an alternated view of deterministic computation. Instead of streams of digital bits, we can encode data as periodic pulses, with the value represented by the fraction of the duty cycle of each pulse. We are still representing data digitally, though not by an encoding in space, but rather through an encoding in time. With data represented this way, we can use the same theory and all the same constructs developed for stochastic computing on these deterministic, periodic signals.

This time-based approach is motivated by the observation that, as technology has scaled and device sizes have gotten smaller, the supply voltages have dropped while the device speeds have improved. Control of the dynamic range in the voltage domain is limited; however, control of the length of pulses in the time domain can be precise. Given how precisely values can be encoded in the time, the method

could produce designs that are much faster than conventional ones—operating in the terahertz range. This remains a work in progress. Potentially, this paradigm could deliver circuits that are as efficient in terms of area and power as stochastic circuits, with considerably lower latency.

References

1. M. H. Najafi, P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, “A reconfigurable architecture with sequential logic-based stochastic computing,” *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, pp. 57:1–57:28, June 2017. [Online]. Available: <http://doi.acm.org/10.1145/3060537>
2. E. Friedman, “Clock distribution networks in synchronous digital integrated circuits,” *Proceedings of the IEEE*, vol. 89, no. 5, pp. 665–692, May 2001.
3. Y. Jiang, H. Zhang, H. Zhang, H. Liu, X. Song, M. Gu, and J. Sun, “Design of mixed synchronous/asynchronous systems with multiple clocks,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.
4. A. Alaghi and J. P. Hayes, “Survey of stochastic computing,” *ACM Transaction on Embedded Computing*, vol. 12, 2013.
5. W. Qian, “Digital yet deliberately random: Synthesizing logical computation on stochastic bit streams,” Ph.D. dissertation, University of Minnesota, 2011.
6. N. C. Laurenciu and S. D. Cotozana, “Low cost and energy, thermal noise driven, probability modulated random number generator,” in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 2724–2727.
7. S. Balatti, S. Ambrogio, R. Carboni, V. Milo, Z. Wang, A. Calderoni, N. Ramaswamy, and D. Ielmini, “Physical unbiased generation of random numbers with coupled resistive switching devices,” *IEEE Transactions on Electron Devices*, vol. 63, no. 5, pp. 2029–2035, May 2016.
8. N. Rangarajan, A. Parthasarthy, N. Kani, and S. Rakheja, “Voltage-tunable stochastic computing with magnetic bits,” in *2017 IEEE International Magnetism Conference (INTERMAG)*, April 2017, pp. 1–2.
9. R. D’Angelo and S. Sonkusale, “Analogue multiplier using passive circuits and digital primitives with time-mode signal representation,” *Electronics letters*, vol. 51, pp. 1754–1755, 2015.
10. V. Ravinuthula, V. Garg, J. G. Harris, and J. Fortes, “Time-mode circuits for analog computation,” *International Journal of Circuit Theory and Applications*, vol. 37, pp. 631–659, 2009.
11. K. M. Mhaidat, M. A. Jabri, and D. W. Hammerstrom, “Representation, methods, and circuits for time-based conversion and computation,” *International Journal of Circuit Theory and Applications*, vol. 39, pp. 299–311, 2011.
12. Y. P. Tsividis and J. O. Voorman, Eds., *Integrated Continuous-Time Filters: Principles, Design and Applications*. IEEE Press, 1993.
13. M. H. Najafi and D. J. Lilja, “High-speed stochastic circuits using synchronous analog pulses,” in *ASP-DAC 2017, 22nd Asia and South Pacific Design Automation Conference*, 2017.
14. M. H. Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani, “Time-encoded values for highly efficient stochastic circuits,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1644–1657, May 2017.
15. K. Cushon, C. Leroux, S. Hemati, S. Mannor, and W. J. Gross, “A min-sum iterative decoder based on pulsewidth message encoding,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 11, pp. 893–897, Nov 2010.
16. I. T. R. for Semiconductors (ITRS), “Itrs 2.0,” 2015.
17. G. W. Roberts and M. Ali-Bakhshian, “A brief introduction to time-to-digital and digital-to-time converters,” *IEEE Transactions on Circuits and System-II*, vol. 57, no. 3, pp. 153–157, January 2010.

18. P. Li, D. Lilja, W. Qian, K. Bazaragan, and M. D. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," in *International Conference on Computer-Aided Design*, 2012, pp. 480–487.
19. W. Qian, X. Li, M. D. Riedel, K. Bazaragan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.
20. W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Design Automation Conference*, 2008, pp. 648–653.
21. W. Qian, M. D. Riedel, K. Bazaragan, and D. Lilja, "The synthesis of combinational logic to generate probabilities," in *International Conference on Computer-Aided Design*, 2009, pp. 367–374.
22. W. Qian, M. D. Riedel, H. Zhou, and J. Bruck, "Transforming probabilities with combinational logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (to appear)*, 2011.
23. W. Qian, C. Wang, P. Li, D. Lilja, K. Bazaragan, and M. D. Riedel, "An efficient implementation of numerical integration using logical computation on stochastic bit streams," in *International Conference on Computer-Aided Design*, 2012, pp. 156–162.
24. A. Alaghi and J. P. Hayes, "On the functions realized by stochastic computing circuits," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '15. New York, NY, USA: ACM, 2015, pp. 331–336. [Online]. Available: <http://doi.acm.org/10.1145/2742060.2743758>
25. S. S. Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority-based tracking forecast memories for stochastic ldpc decoding," *IEEE Transactions on Signal Processing*, vol. 58, pp. 4883–4896, 2010.
26. B. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*. Plenum Press, 1969, vol. 2, ch. 2, pp. 37–172.
27. B. Brown and H. Card, "Stochastic neural computation I: Computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.
28. P. Li, D. J. Lilja, W. Qian, K. Bazaragan, and M. D. Riedel, "Case studies of logical computation on stochastic bit streams," in *Lecture Notes in Computer Science: Proceedings of Power and Timing Modeling, Optimization and Simulation Workshop*, G. Goos, J. Hartmanis, and J. Leeuwen, Eds. Springer, 2012.
29. P. Li, W. Qian, and D. J. Lilja, "A stochastic reconfigurable architecture for fault-tolerant computation with sequential logic," *IEEE 30th International Conference on Computer Design (ICCD)*, 2012.
30. M. H. Najafi and D. Lilja, "High quality down-sampling for deterministic approaches to stochastic computing," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018.

Generating Stochastic Bitstreams



Hsuan Hsiao, Jason Anderson, and Yuko Hara-Azumi

Abstract Stochastic computing (SC) hinges on the generation and use of stochastic bitstreams—streams of randomly generated 1s and 0s, with the probabilities of p and $1 - p$, respectively. We consider approaches for stochastic bitstream generation, considering randomness, circuit area/performance/cost, and the impact of the various approaches on SC accuracy. We first review the widely used Linear-Feedback Shift Register (LFSR)-based approach and variants. Alternative low-discrepancy sequences are then discussed, followed by techniques that leverage post-CMOS technologies and metastability of devices as sources of randomness. We conclude with a discussion on correlations between bitstreams, and how (1) correlations can be reduced/eliminated, and (2) correlations may actually be leveraged to positive effect in certain circumstances.

Introduction

Stochastic computing (SC) requires the generation of stochastic bitstreams: streams of randomly generated 1s and 0s, where the probability of an individual bit being 1 is p and the probability of a bit being 0 is $1 - p$. For example, a length-8 bitstream having the sequence 0, 1, 0, 0, 1, 0, 0, 1 represents the value $p = 3/8$ in the unipolar representation. Inputs, intermediate values, and outputs of stochastic circuits are represented with such bitstreams. Using this representation, certain types of computations become “cheap” to implement, with multiplication being emblematic of the phenomenon. With a second bitstream, 1, 1, 0, 1, 0, 1, 1, 1, representing $q = 6/8$, one can approximate $z = p \times q$ by AND’ing the two

H. Hsiao · J. Anderson (✉)

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering,
University of Toronto, Toronto, ON, Canada
e-mail: julie.hsiao@mail.utoronto.ca; janders@eecg.toronto.edu

Y. Hara-Azumi

School of Engineering, Tokyo Institute of Technology, Tokyo, Japan
e-mail: hara@cad.ict.e.titech.ac.jp

bitstreams, to produce: 0, 1, 0, 0, 0, 0, 0, 1, representing $z = 2/8$. $2/8 = 0.25 \approx p \times q = 18/64 = 0.281$. The approximation accuracy hinges on the input streams being sufficiently long and statistically independent.

Naturally, it is desirable if such stochastic bitstreams are generated by hardware that has low area, power, and delay. However, apart from such circuit objectives, many approaches to stochastic computing require that input and intermediate-value bitstreams be free of correlations from one another, and that streams are truly random, to the extent possible. This chapter discusses stochastic bitstream generation, the associated hardware costs, as well as touching upon issues of correlations and accuracy.

A stochastic bitstream of length L may contain as few as zero 1s, or as many as L ones. As such, the probability values representable in such a bitstream lie in a discrete set: $0, 1/L, 2/L, \dots, L/L$; that is, the set contains all values in the $[0:1]$ range in steps of $1/L$. By increasing the bitstream length, the step size is reduced, permitting a higher accuracy approximation of probability p . Broadly speaking, longer bitstreams imply higher accuracy, and it is precisely this concept that underpins the inherent *progressive precision* of SC: the notion that accuracy generally improves as longer bitstreams are used, permitting early termination of computation if sufficient accuracy is achieved.

Perhaps surprisingly, a significant hurdle for the broader uptake of SC is tied to challenges associated with the seemingly straightforward task of generating input bitstreams to feed into an SC circuit. While relatively simple logic circuits may suffice to perform sophisticated computations on stochastic bitstreams (e.g. a single AND gate for multiplication), larger and more power hungry circuits are needed for the input bitstream generation. Thus, generating stochastic bitstreams is frequently more costly than computing on them! In fact, prior work has reported that over 80% of the area of several SC designs is consumed by circuitry for generating the stochastic bitstreams and circuitry for conversion back to standard binary [1]. Consequently, the design of bitstream generation circuitry remains an active and important research vector for SC.

A further challenge with regard to stochastic bitstreams involves correlations (discussed in detail in chapter “Accuracy and Correlation in Stochastic Computing”). To illustrate this, returning to the example above, if instead of computing $z = p \times q$, the objective was to compute $z = (p \times q)^2$, observe that one cannot simply AND the bitstream $p \times q$ with itself, as this would simply reproduce $p \times q$ instead of the desired $(p \times q)^2$. Rather, the correct approach is to AND the bitstream corresponding to $p \times q$ with a second uncorrelated bitstream also representing $p \times q$. There is a need, therefore, to be able to decorrelate bitstreams as part of SC.

The chapter is organized as follows: Section “[Overview](#)” provides an overview of stochastic bitstream generation and converting from/to standard binary representation. Section “[Sequence Generation](#)” describes varied approaches proposed for stochastic sequence generation, including an overview of sources of randomness in CMOS and post-CMOS technologies. Sections “[Correlated Stochastic Bitstreams](#)” and “[Decorrelating Stochastic Bitstreams](#)” describe techniques around creating

correlated bitstreams (shown to be acceptable in certain circumstances) and decorrelating bitstreams, respectively. Section “[Summary](#)” concludes the chapter.

Overview

Figure 1 depicts a block diagram of a widely used circuit to generate a stochastic bitstream. A random number generator (RNG) produces a sequence of N -bit binary values: one such value per clock cycle. The random values are then fed into a comparator and compared with an N -bit binary number, B . Based on the comparison, a 0 or a 1 is generated in the stochastic bitstream. The RNG is typically designed so that across 2^N clock cycles, all possible N -bit binary numbers are output (possibly excluding the case of all zeros). With B held constant across the 2^N clock cycles, the stochastic bitstream generated will contain B ones, and $2^N - B$ zeros, thereby representing the value $B/2^N$. The subsequent sections highlight different approaches to random sequence generation.

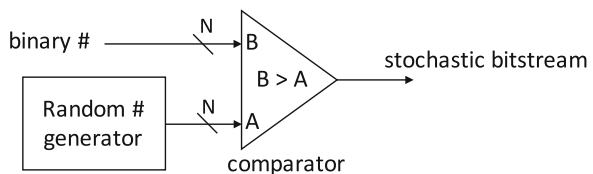
To convert a stochastic bitstream back to a normal binary representation, a counter is typically used: on each clock cycle, when the stochastic stream contains a 1, the counter is incremented. With a length- L bitstream, after L clock cycles, if the counter holds value C , the value represented by the bitstream is C/L , which as expected lies in the $[0 : 1]$ range.

Sequence Generation

Linear-Feedback Shift Registers (LFSRs)

The most widely used implementation for the random-number generator in Fig. 1 is a linear-feedback shift register (LFSR). An N -bit LFSR comprises N flip-flops, as well as XOR gates in a feedback configuration. Figure 2 shows an example 3-bit LFSR. With N bits, there are 2^N possible binary numbers, however, the all-zeros possibility is generally not used, leaving $2^N - 1$ numbers. A *maximal* LFSR walks through all such $2^N - 1$ numbers in a deterministic, yet pseudo-random order. For example, the LFSR in Fig. 2 walks through the sequence (output bits are $Q_2 Q_1 Q_0$):

Fig. 1 Stochastic bitstream generation



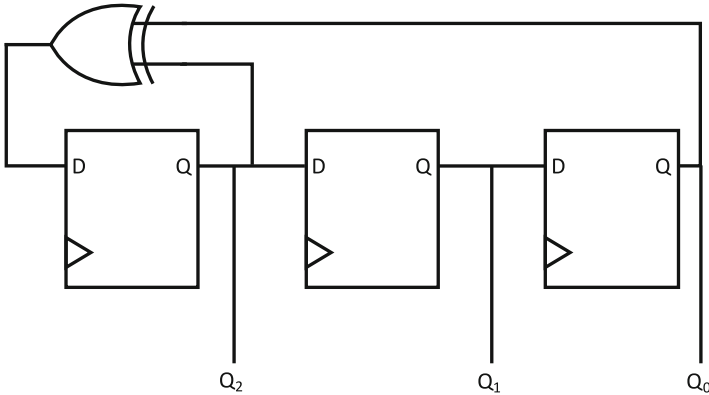


Fig. 2 3-bit linear-feedback shift register (LFSR)

1, 4, 6, 7, 3, 5, 2, 1, In general, with an n -bit maximal LFSR, an $\approx 2^n - 1$ -length stochastic bitstream can be generated.

As mentioned above, in many cases, SC relies on stochastic bitstreams being uncorrelated from one another. With the LFSR-based approach, this is conventionally achieved by using instances of the circuit in Fig. 1, wherein each RNG LFSR is seeded at a different starting number. Thus, the multiple LFSRs each move through the same number sequence, in the same order, but yet are at different positions within the sequence. The starting point of each LFSR is referred to as its *seed*.

In [2], the authors considered a variety of variations on the LFSR-based approach in a bid to raise the accuracy of SC. First, the authors considered randomly scrambling the LFSR outputs, as fed into the comparator in Fig. 1. Scrambling causes the LFSRs to appear to walk through numbers in a different order. Second, Anderson et al. [2] considers the choice of seed for each LFSR, and seed selection implications on SC accuracy. Third, the authors considered using LFSRs with different “feedback polynomials”—i.e. different tap-point positions for the inputs to the XOR gate.¹ Similar to the scrambling approach, the use of different feedback polynomials causes the LFSRs to traverse the number sequence in a different order.

The results in [2] demonstrate that scrambling LFSR outputs and the use of multiple feedback polynomials had a modest effect on SC accuracy. However, judicious selection of LFSR seeds had a significant impact on SC accuracy. Careful seeding of the LFSRs provided an accuracy improvement beyond that achievable through the use of longer bitstreams in certain cases. Further study of LFSR seeding in the SC context is a direction worthy of future research.

¹There are typically multiple ways to design an N -bit LFSR.

Low-Discrepancy Sequences

Low discrepancy (LD) sequences are also known as quasi-random sequences or sub-random sequences. When compared to pseudo-random sequences generated by LFSRs, LD sequences exhibit the property of being more equidistributed. This property provides the advantage of a faster rate of convergence to the desired probability, $O(\frac{1}{L})$ instead of $O(\frac{1}{\sqrt{L}})$, where L is the length of the SC bitstream. This implies that a shorter SC bitstream can be used to achieve the same accuracy since it is less susceptible to errors due to random fluctuations. Of many LD sequences in literature, the van der Corput/Halton sequence [3, 4] and the Sobol’ sequence [5] have been used in the context of stochastic bitstream generation. Sections “[van der Corput and Halton Sequences](#)” and “[Sobol’ Sequences](#)” discuss the computations involved in generating the LD sequence and the hardware circuits proposed for the van der Corput/Halton sequence and the Sobol’ sequence, respectively. Section “[Comparison of LD Sequences](#)” presents a comparison between the LD sequences and their LFSR counterparts.

van der Corput and Halton Sequences

The van der Corput sequence [3] $\{X_1, X_2, X_3 \dots\}$, where $0 < X_i < 1$, is an infinite-length sequence with distinct values for each term. It is “seeded” by picking a base b , and constructed by reversing the base- b representation of natural numbers $i \geq 1$. Every natural number i can be expressed in base- b as

$$i = \sum_{j=0}^{\infty} a_j(i)b^j$$

where $a_j(i) \in \{0, 1, \dots, b - 1\}$ and $a_j(i) = 0$ for all sufficiently large j . The i -th van der Corput sequence is therefore expressed as

$$X_i = \sum_{j=0}^{\infty} a_j(i)b^{-j-1}$$

Example The natural numbers $i \geq 1$ represented with $b = 3$ are

001 002 010 011 012 020 021 022 100 101 ...

where the individual digits in each number are the $a_j(i)$ coefficients. The van der Corput sequence flips the digits of i at the radix point, giving

0.100 0.200 0.010 0.110 0.210 0.020 0.120 0.220 0.001 0.101 ...

which corresponds to the following van der Corput sequence X :

$$\frac{1}{3} \quad \frac{2}{3} \quad \frac{1}{9} \quad \frac{4}{9} \quad \frac{7}{9} \quad \frac{2}{9} \quad \frac{5}{9} \quad \frac{8}{9} \quad \frac{1}{27} \quad \frac{10}{27} \quad \dots$$

The van der Corput sequence is a one-dimensional LD sequence, and the Halton sequence generalizes the van der Corput sequence to higher dimensions. The Halton sequence uses co-prime numbers as its bases for each dimension. For a circuit that requires k uncorrelated inputs, a k -dimensional Halton sequence generator will be needed, each using a different base b .

Hardware Implementation Figure 3 shows the structure of the Halton sequence generator proposed by Alaghi and Hayes [6]. It consists of a binary-coded base- b counter, where b is a prime number. The order of the output digits from the counter is reversed and the resulting number is converted to a binary number with the base- b -to-binary converters and the adder. When $b = 2$, Fig. 3 reduces to a simple binary counter. For k inputs, k copies of the circuit with different prime bases are needed.

Sobol’ Sequences

The Sobol’ sequence is a base-2 (t, s) -sequence [3, 5] that is infinite in length, with distinct values for each term. We describe the steps required to generate the Sobol’ sequence based on the algorithms proposed by Bratley and Fox [7].

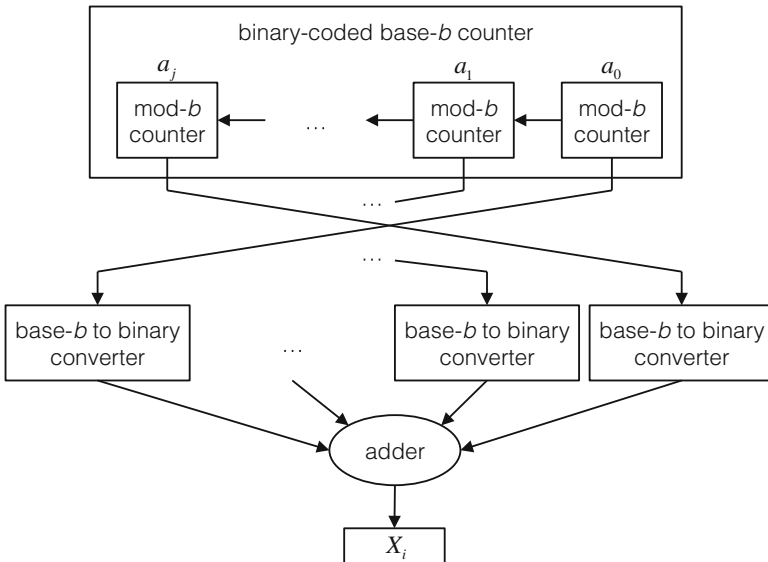


Fig. 3 Halton sequence generator proposed by Alaghi and Hayes [6]

To construct the Sobol' sequence $\{X_1, X_2, X_3, \dots\}$ where $0 < X_i < 1$, a set of direction vectors v_j (discussed later) needs to be precomputed to "seed" the sequence. With the precomputed direction vectors v_j , the Sobol' sequence can be written as:

$$X_i = b_1 v_1 \oplus b_2 v_2 \oplus \dots \tag{1}$$

where $\dots b_3 b_2 b_1$ is the binary representation of i and \oplus stands for the bitwise-XOR operation. Antonov and Saleev [8] showed that using the Gray code representation of i ($\dots g_3 g_2 g_1$) does not affect the asymptotic discrepancy of the sequence. The sequence in Eq. (1) can then be written as

$$X_i = g_1 v_1 \oplus g_2 v_2 \oplus \dots \tag{2}$$

Since the Gray code of i can be obtained with $g_k = b_k \oplus b_{k+1}$, the expression in Eq. (2) can be rewritten as

$$X_{i+1} = X_i \oplus v_c \tag{3}$$

where c is the position of the rightmost zero in the binary representation of i .

Each direction vector v_j is a binary fraction expressed as

$$v_j = 0.v_{j1}v_{j2}v_{j3} \dots v_{jk} \dots \quad \text{or} \quad v_j = \frac{m_j}{2^j}$$

where $m_j < 2^j$ and m_j is an odd number. To compute the direction vectors v_j , a primitive polynomial P with degree d in the field Z_2 is first selected. The primitive polynomial is in the form

$$P = x^d + a_1 x^{d-1} + a_2 x^{d-2} + \dots + a_{d-1} x + 1$$

where coefficients $a_i \in \{0, 1\}$. The coefficients a_i are then used to construct a set of direction vectors v_j , where

$$v_j = a_1 v_{j-1} \oplus a_2 v_{j-2} \oplus \dots \oplus a_{d-1} v_{j-d+1} \oplus [v_{j-d} \gg d]$$

In this recurrence relation, $j > d$ and the last term is v_{j-d} shifted right by d . The recurrence relation for the direction vectors can alternatively be expressed as

$$m_j = 2^1 a_1 m_{j-1} \oplus 2^2 a_2 m_{j-2} \oplus \dots \oplus 2^{d-1} a_{d-1} m_{j-d+1} \oplus 2^d m_{j-d} \oplus m_{j-d}$$

Example Suppose we pick a degree 3 primitive polynomial:

$$x^3 + x + 1$$

The coefficients here are $a_1 = 0$ and $a_2 = 1$, giving the recurrence relation:

$$m_j = 4m_{j-2} \oplus 8m_{j-3} \oplus m_{j-3}$$

To use the recurrence equation, an initial value for the first d m_j 's needs to be assigned. The initial values can be chosen freely as long as they are odd and $m_j < 2^j$. Assuming we assign $m_1 = 1$, $m_2 = 3$, and $m_3 = 7$, then

$$m_4 = 4(3) \oplus 8(1) \oplus 1 = 5$$

$$m_5 = 4(7) \oplus 8(3) \oplus 3 = 7$$

$$m_6 = 4(5) \oplus 8(7) \oplus 7 = 43$$

The first 6 direction vectors are shown in Table 1. Using these precomputed direction vectors and Eq. (2), the Sobol' sequence can be computed, as shown in Table 2. Here, we are assuming that $X_0 = 0$.

The i -th value in the sequence can also be obtained directly using Eq. (2). For example, for $i = 23$, the Gray code representation is $i = 11100$. X_{23} can therefore be computed as:

$$\begin{aligned} X_{23} &= v_3 \oplus v_4 \oplus v_5 \\ &= 0.11100 \oplus 0.01010 \oplus 0.00111 \\ &= 0.10001 \\ &= \frac{17}{32} \end{aligned}$$

Table 1 Example of generating v_j with $P = x^3 + x + 1$, $m_1 = 1$, $m_2 = 3$, and $m_3 = 7$

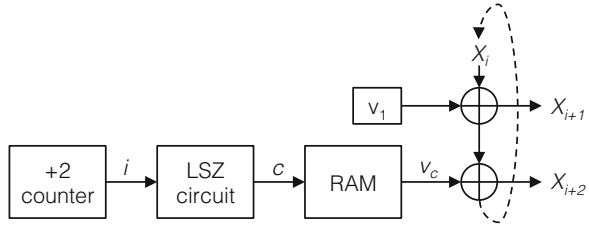
j	1	2	3	4	5	6
m_j	1	3	7	5	7	43
v_j	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{7}{8}$	$\frac{5}{16}$	$\frac{7}{32}$	$\frac{43}{64}$
v_j (binary)	0.1	0.11	0.111	0.0101	0.00111	0.101011

Table 2 Example of generating Sobol' sequence using the recurrence equation $X_{i+1} = X_i \oplus v_c$

i (binary)	X_i	c
0	$X_0 = 0^a$	1
01	$X_1 = X_0 \oplus v_1 = 0 \oplus 0.1 = 0.1$	2
10	$X_2 = X_1 \oplus v_2 = 0.1 \oplus 0.11 = 0.01$	1
011	$X_3 = X_2 \oplus v_1 = 0.1 \oplus 0.10 = 0.11$	3

^a X_0 is not part of the sequence, but an initial value to start the recurrence equation

Fig. 4 Hardware circuit for the Sobol’ sequence generator, producing 2 values in parallel, proposed by Dalal et al. [9]



Hardware Implementation Dalal et al. proposed a hardware design for a Sobol’ sequence generator based on the above algorithm, intended for Quasi-Monte Carlo simulation [9]. In their implementation, the direction vectors are pre-computed and stored in RAM. They construct the sequence using the recurrence equation in Eq. (3), which involves a least-significant zero (LSZ) computation, a memory lookup, and an XOR operation. Figure 4 shows the circuit, which produces two values in the sequence in parallel. In this circuit, i is an even number and v_1 is a copy of the v_1 value from the RAM. Since every even-numbered i will have its least-significant zero in position $c = 1$, every odd-numbered $i + 1$ will end up with the expression $X_{i+1} = X_i \oplus v_1$. The circuit in Fig. 4 produces the Sobol’ sequence in one dimension. A copy of the circuit is needed for each uncorrelated input bitstream required by the SC circuit.

Comparison of LD Sequences

Liu and Han [10] compared the energy efficiency of stochastic number generators (SNGs) based on LFSRs, Halton sequences, and Sobol’ sequences. They used the root-mean-square error (RMSE) as a measure of accuracy, then compared the performance of different circuit implementations for a given accuracy. Figure 5 shows the accuracy results achieved by the stochastic computing implementations using LFSRs, the Halton sequence, and the Sobol’ sequence, for a 2-input multiplier circuit and a 3rd-order Bernstein polynomial [11] circuit. On the same graph, the functionally equivalent circuit implemented with the conventional binary number system is plotted, providing a comparison with the stochastic computing implementations. For a simple 2-input multiplier circuit, the RMSE of the LD sequences decreases at a similar rate as binary numbers, as the length of the LD sequence increases at the rate of 2^N and the bitwidth of the binary number increases at the rate of N . The LFSR’s RMSE decreases at a much slower rate as the length increases. For a 3rd-order Bernstein polynomial circuit [11], the gap between how accuracy scales for binary and LD sequences widens slightly. However, the LD sequences still provide better accuracy than an LFSR in terms of how RMSE scales as length increases. Overall, Liu and Han [10] reports that the LD sequences provide a higher accuracy than an LFSR for the same sequence length in these two applications. Similar observations were made by Seva et al. [12] when they compared the use of an LD sequence versus an LFSR for edge detection applications.

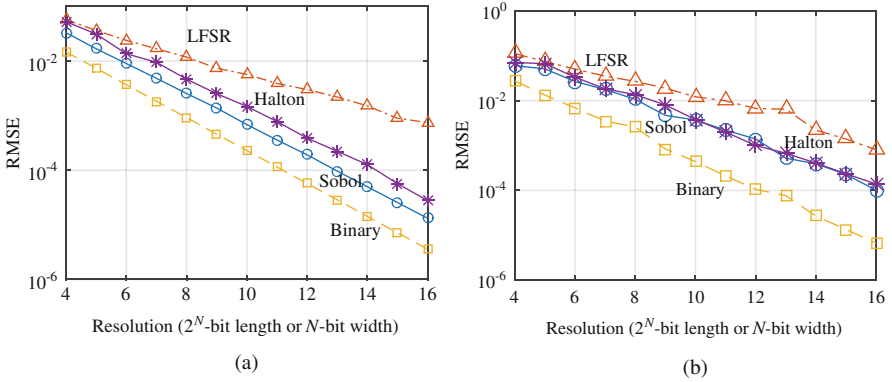


Fig. 5 Accuracy comparison between different random number generators with different sequence length for a (a) 2-input multiplier circuit and a (b) 3rd-order Bernstein polynomial circuit [10]

Liu and Han quantified the performance of the circuit using energy per operation (EPO) and throughput per area (TPA). $EPO = Power \times T_{clk} \times L$, where L is the sequence length, T_{clk} is the clock period, and $Power$ is the measured power at T_{clk} . $TPA = (\# \text{ of effective bits}) / (t_c \times L) / area$, where t_c is the critical path delay and the $(\# \text{ of effective bits})$ is the bitwidth for the corresponding binary representation ($\lfloor \log_2(L) \rfloor$). For LD sequences requiring low dimensions (i.e. a low number of uncorrelated inputs), the EPO is lower than the LFSR-based circuits and the TPA is greater than the LFSR-based circuits. With increasing dimension, the area overhead required to generate uncorrelated sequences increases faster for LD sequences than for LFSRs. This results in comparable EPO and TPA between the LD sequences and the LFSR-based circuit. Detailed performance results can be referred to in the work by Liu and Han [10].

Alternative Sources of Randomness

In digital systems, metastability (sometimes referred to as glitch) is an uncertain state between logic ‘0’ and ‘1’ due to variation or propagation delay. Under metastability, a circuit’s behavior is unpredictable. Some existing works exploit such uncertainty as source of entropy to generate true random numbers on FPGAs [13–15]. In many of such works, each true random number generator (TRNG) is built by several oscillators and a binary XOR-tree [13]. For example, as illustrated in Fig. 6, a TRNG has r distinct ring oscillators, R_1, R_2, \dots, R_r , where the i -th ring oscillator R_i is composed of n_i inverters. Due to differing numbers of inverters, and process variations, the r ring oscillators oscillate at different frequencies from one another. Then, r oscillating outputs $\Psi_1, \Psi_2, \dots, \Psi_r$ are XOR’ed to generate one output Ψ . Finally, this output Ψ is sampled at a clock frequency f . In addition to

Fig. 6 Ring oscillators-based TRNG on FPGA [13]

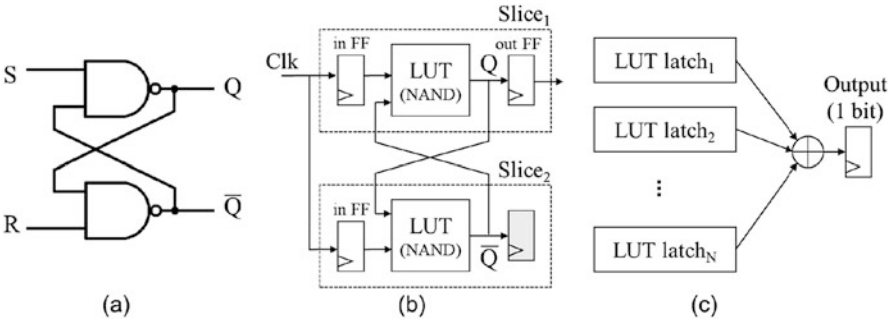
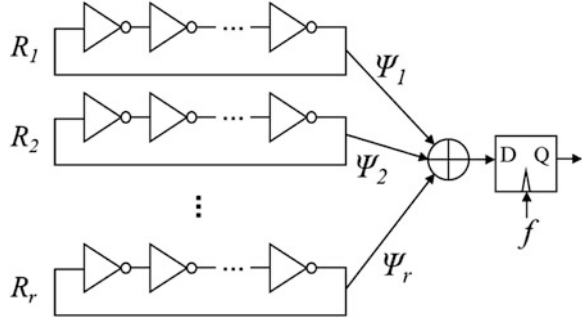
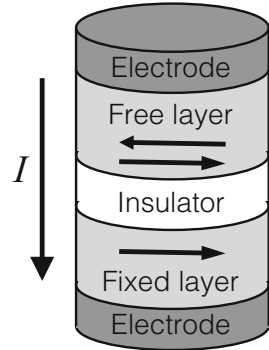


Fig. 7 RS latches-based TRNG on FPGA [14]; (a) RS latch; (b) FPGA RS latch implementation; (c) employing multiple RS latches for higher entropy

ring oscillators, some recent works such as [15] further utilize physically unclonable functions (PUF), which arise due to physical manufacturing variations within the integrated circuit, as a source of variability/randomness.

Because ring oscillator-based TRNGs are power-hungry, other implementations, such as latch-based TRNGs, have been also studied [14, 16]. For example, for an RS latch in Fig. 7a, activating both R and S inputs simultaneously is prohibited as the latch enters a metastable state having (Q, \bar{Q}) as either of $(0, 1)$ or $(1, 0)$. A TRNG can be constructed by feeding the same Clk to R and S of the RS latch—when $Clk = 1$, metastability is realized. In [14], this concept is implemented to generate entropy using two look-up tables (LUTs) in an FPGA, as shown in Fig. 7b. Although theoretically one RS latch may realize a TRNG, it is not possible to generate sufficient entropy through a single RS latch only. Therefore, multiple copies of this RS latch are needed with XOR’ing among the latch outputs (see Fig. 7c). Furthermore, using distant LUTs to realize the RS latch-based TRNG offers the opportunity to “capture” additional noise, such as thermal noise, (i.e., larger entropy) through longer wires, leading to higher-quality randomness. However, this will affect the throughput of the TRNG. The authors of [14] have studied the suitable number of latches in terms of the quality and throughput (leaving the place-and-route task to the vendor tool), and revealed 64-256 latches achieve a reasonable trade-off of quality versus cost.

Fig. 8 MTJ structure

In addition to generating random numbers in the digital domain, prior work has demonstrated the use of the probabilistic nature of device characteristics in generating sequences directly in the analog domain. Knag et al. [17] demonstrated the use of memristor memory to convert analog voltages to a stochastic bitstream. The probability that a write to a cell is successful $P(t)$ is dependent on the programming voltage V and pulse width t , given by expressions $P(t) = 1 - e^{-t/\tau}$. τ is the characteristic switching time of the memristor and depends on the programming voltage V , given by the expression $\tau(V) = \tau_0 e^{-V/V_0}$ (τ_0 and V_0 are fitting parameters). The pulse width that gives the desired probability is used in *group write*, which applies the pulse to a group of memristor cells and the number of ones successfully written represents the stochastic number. The authors further illustrated the feasibility of using a pulse train of shorter widths to approximate a longer pulse.

The feasibility of using magnetic-tunnel junction (MTJ) devices as a natural way to generate stochastic bitstreams has been investigated by various researchers, including the use of spin-transfer-torque (STT) MTJs [18–21], voltage controlled (VC) MTJs [22], and all-spin logic (ASL) [23]. An MTJ consists of two ferromagnetic layers separated by a tunneling barrier, illustrated in Fig. 8. One of the ferromagnetic layers has a fixed direction of magnetization, whereas the other can be switched and is referred to as the free layer. The direction of the free layer can be parallel or anti-parallel to the fixed layer, which corresponds to the low- and high-resistance states, respectively. The switching between parallel and anti-parallel states of the MTJ device is achieved by applying a voltage pulse, and the MTJ will probabilistically transition. The probability that the MTJ device switches is dependent on the voltage and the duration of the pulse, thereby creating the possibility for random-number generation.

Correlated Stochastic Bitstreams

As there is considerable hardware cost associated with stochastic bitstream generation, a natural question that arises is: to what extent is it possible to *share*

hardware when generating multiple bitstreams? This question has been considered in a number of recent works, e.g. [24–26], which proposed sharing portions of the bitstream-generation circuitry. With sharing of circuitry, however, comes the possibility of correlations among bitstreams.

Ichihara et al. [25] proposed that the same random number generator (LFSR) be used to generate multiple stochastic bitstreams, with a unique comparator for each stream. To reduce correlations, the LFSR output bits are rotated by a fixed and unique number of bits for each bitstream, there making the random numbers fed to each comparator appear different. The idea is depicted in Fig. 9 for the two-bitstream case. Rotating the LFSR output bits by a fixed amount is “free” in hardware, as it can be done solely with wiring.

Ichihara et al. [25] also noted that for certain functions, such as stochastic scaled addition using a 2-to-1 multiplexer, correlations between the MUX data inputs are acceptable in that they do not impact output accuracy. Ding et al. [27] leverages this property in the generation of stochastic bitstreams by using a single LFSR, along with a logic circuit, to generate multiple correlated bitstreams to feed MUX data inputs in the MUX-based stochastic computing architecture proposed in [1]. Specifically, the authors in [27] note that logic functions applied to stochastic bitstreams having a probability of 0.5 can be used to generate bitstreams with specifically chosen probabilities. The individual LFSR bits are fed into an optimized multi-output logic circuit to produce multiple output bitstreams with desired probabilities in limited area cost.

More recently, Neugebauer et al. [26] proposed an alternative hardware sharing approach, comprising feeding LFSR output bits into an *S-box*—an n-input-to-n-output Boolean function the authors borrowed from cryptography. As in [25], the work in [26] uses different bit-wise rotation amounts for different stochastic bitstreams, however, the presence of the S-box provides improved statistical independence between generated streams. Finally, Li and Lilja [24] proposes techniques for generating a lengthy stochastic bitstream, where overlapping portions of the stream are used as shorter bitstreams having specific desired probabilities.

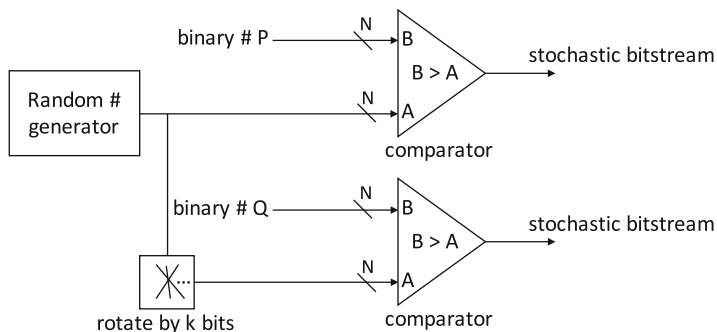


Fig. 9 Sharing the LFSR to reduce the hardware cost of stochastic bitstream generation [25]

Decorrelating Stochastic Bitstreams

Correlations among stochastic bitstreams arise from reconvergent paths in SC circuits. Excessive errors in accuracy may arise from the use of such correlated streams. There appear to be two approaches for decorrelating bitstreams in the literature. The first, and most widely used approach is generate a bitstream from an independent random source. This involves converting the original stochastic stream back to a binary representation, and then regenerating a new stochastic stream using the circuit in Fig. 1. This first approach, while straightforward, imposes a high area and delay cost.

A second approach, proposed recently, involves the introduction of *isolators* [28]. With this approach, a stochastic bitstream is interpreted as a Bernoulli process, where each successive 0/1 value is independent from one another. With this interpretation, it is straightforward to see two stochastic bitstreams can be decorrelated from one another by delaying one of the bitstreams by one or more clock cycles. The delaying effect is realized by inserting flip-flops (isolators) on carefully chosen paths within an SC circuit. The authors of [28] study the isolator concept and propose methodologies for selecting the connections within an SC circuit on which isolators should be inserted. The isolator approach holds promise to decorrelate streams with considerably less area and delay cost than from-scratch bitstream generation.

Summary

Stochastic computing (SC) relies on randomly generated streams of 1s and 0s as its number representation. While this representation is beneficial to power, computational area, reliability and fault tolerance, it is associated with a number of challenges, with a key obstacle being how to generate such bitstreams in a low area footprint and how to reduce the latency required for computations (bitstream length); i.e. to achieve reasonable accuracy in reasonable time. In this chapter, we reviewed published techniques for stochastic bitstream generation, including the commonly used LFSR-based approach, as well as the use of low-discrepancy sequences. The latter approach shows promise as a means to improve SC accuracy. We then outlined other sources of randomness that may be leveraged for random-number generation, including the use of storage-element metastability and probabilistic effects in post-CMOS technologies. Finally, we discussed correlations in stochastic bitstreams, typically undesirable, yet useful in certain circumstances.

References

1. W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.

2. J. H. Anderson, Y. Hara-Azumi, and S. Yamashita, "Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, 2016, pp. 1550–1555.
3. H. Niederreiter, *Random Number Generation and quasi-Monte Carlo Methods*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.
4. J. H. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals," *Numer. Math.*, vol. 2, no. 1, pp. 84–90, Dec. 1960.
5. I. M. Sobol', "Uniformly distributed sequences with an additional uniform property," *USSR Computational Mathematics and Mathematical Physics*, vol. 16, no. 5, pp. 236–242, 1976.
6. A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *Proceedings of the 2014 Conference on Design, Automation & Test in Europe*, March 2014, pp. 1–4.
7. P. Bratley and B. L. Fox, "Algorithm 659: Implementing Sobol's quasirandom sequence generator," *ACM Trans. Math. Softw.*, vol. 14, no. 1, pp. 88–100, Mar. 1988.
8. I. A. Antonov and V. M. Saleev, "An economic method of computing $LP\tau$ -sequences," *USSR Computational Mathematics and Mathematical Physics*, vol. 19, no. 1, pp. 252–256, 1979.
9. I. L. Dalal, D. Stefan, and J. Harwayne-Gidansky, "Low discrepancy sequences for Monte Carlo simulations on reconfigurable platforms," in *2008 International Conference on Application-Specific Systems, Architectures and Processors*, July 2008, pp. 108–113.
10. S. Liu and J. Han, "Energy efficient stochastic computing with Sobol sequences," in *Proceedings of the 2017 Conference on Design, Automation & Test in Europe*, March 2017, pp. 650–653.
11. G. Lorentz, *Bernstein Polynomials*, ser. AMS Chelsea Publishing Series. Chelsea Publishing Company, 1986.
12. R. Seva, P. Metku, and M. Choi, "Energy-efficient fpga-based parallel quasi-stochastic computing," *Journal of Low Power Electronics and Applications*, vol. 7, no. 4, 2017.
13. B. Sunar, W. J. Martin, and D. R. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Transactions on Computers*, vol. 56, no. 1, pp. 109–119, Jan 2007.
14. H. Hata and S. Ichikawa, "FPGA implementation of metastability-based true random number generator," *IEICE Trans. on Information and Systems*, vol. E95.D, no. 2, pp. 426–436, 2012.
15. S. Buchovecká, R. Lórencz, F. Kodýtek, and J. Buček, "True random number generator based on ring oscillator PUF circuit," *Microprocessors and Microsystems*, vol. 53, pp. 33–41, 2017.
16. P. Z. Wiczorek, "Dual-metastability FPGA-based true random number generator," *Electronics Letters*, vol. 49, no. 12, pp. 744–745, June 2013.
17. P. Knag, W. Lu, and Z. Zhang, "A native stochastic computing architecture enabled by memristors," *IEEE Transactions on Nanotechnology*, vol. 13, no. 2, pp. 283–293, March 2014.
18. N. Onizawa, D. Katagiri, W. J. Gross, and T. Hanyu, "Analog-to-stochastic converter using magnetic-tunnel junction devices," in *2014 IEEE/ACM International Symposium on Nanoscale Architectures*, July 2014, pp. 59–64.
19. X. Jia, J. Yang, Z. Wang, Y. Chen, H. H. Li, and W. Zhao, "Spintronics based stochastic computing for efficient bayesian inference system," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, 2018, pp. 580–585.
20. L. A. de Barros Naviner, H. Cai, Y. Wang, W. Zhao, and A. B. Dhia, "Stochastic computation with spin torque transfer magnetic tunnel junction," in *2015 IEEE 13th International New Circuits and Systems Conference*, June 2015, pp. 1–4.
21. Y. Wang, H. Cai, L. A. B. Naviner, J. O. Klein, J. Yang, and W. Zhao, "A novel circuit design of true random number generator using magnetic tunnel junction," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures*, July 2016, pp. 123–128.
22. S. Wang, S. Pal, T. Li, A. Pan, C. Grezes, P. Khalili-Amiri, K. L. Wang, and P. Gupta, "Hybrid VC-MTJ/CMOS non-volatile stochastic logic for efficient computing," in *Proceedings of the 2017 Conference on Design, Automation & Test in Europe*, March 2017, pp. 1438–1443.
23. R. Venkatesan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan, "Spintastic: Spin-based stochastic logic for energy-efficient computing," in *Proceedings of the 2015 Conference on Design, Automation & Test in Europe*, March 2015, pp. 1575–1578.

24. P. Li and D. J. Lilja, "Accelerating the performance of stochastic encoding-based computations by sharing bits in consecutive bit streams," in *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, June 2013, pp. 257–260.
25. H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue, "Compact and accurate stochastic circuits with shared random number sources," in *2014 IEEE 32nd International Conference on Computer Design*, Oct 2014, pp. 361–366.
26. F. Neugebauer, I. Polian, and J. P. Hayes, "Building a better random number generator for stochastic computing," in *Euromicro Conference on Digital System Design*, 2017, pp. 1–8.
27. Y. Ding, Y. Wu, and W. Qian, "Generating multiple correlated probabilities for MUX-based stochastic computing architecture," in *2014 IEEE/ACM International Conference on Computer-Aided Design*, 2014, pp. 519–526.
28. P. S. Ting and J. P. Hayes, "Isolation-based decorrelation of stochastic circuits," in *2016 IEEE 34th International Conference on Computer Design*, 2016, pp. 88–95.

RRAM Solutions for Stochastic Computing



Phil Knag, Siddharth Gaba, Wei Lu, and Zhengya Zhang

Abstract Stochastic computing is a low-cost form of computing. To perform stochastic computing, inputs need to be converted to stochastic bit streams using stochastic number generators (SNGs). The random number generation presents a significant overhead, which partially defeats the benefits of stochastic computing. In this work, we show that stochastic computing can be implemented in memory, through temporally varying resistive RAM (RRAM) devices. We can take advantage of the temporal variations in the resistance switching of RRAM devices for stochastic computing. By adopting a group write approach, an RRAM memory can be used to generate random bit streams and reshuffle bit streams, accomplishing two essential functions for stochastic computing without expensive SNGs. We demonstrate in-memory stochastic computing in two test applications, a gradient descent solver and a k-means clustering processor.

Introduction

Stochastic computing is a low-cost computing paradigm using probabilistic bit streams [1–3]. For example, the number 0.5 can be represented by a stream of 8 bits {0, 1, 1, 0, 1, 0, 0, 1} in which the probability of having 1 in the stream is 0.5. Similarly, the number 0.25 can be represented by {0, 1, 0, 0, 0, 1, 0, 0}. Compared to the binary system, a bit stream representation is not unique. A higher numerical precision requires a much longer bit stream, but as a benefit, a bit stream is more error-tolerant than the binary system: a bit flip introduces a least significant bit (LSB) error. Typically binary numbers are converted to bit streams for stochastic computing, and the output bit stream of stochastic computing is converted to binary.

Stochastic computing offers the key advantage of much simplified arithmetic operations. For example, the multiplication of a and b can be implemented using

P. Knag · S. Gaba · W. Lu · Z. Zhang (✉)
University of Michigan, Ann Arbor, MI, USA
e-mail: zhengya@umich.edu

an AND logic gate. An important assumption of stochastic computing is that input bit streams of an operation are independent. Any correlation between input streams reduces the accuracy of stochastic computing. For example, if we multiply two identical, i.e., 100% correlated, bit streams representing a using an AND gate, the output of the gate does not represent a^2 . An in-depth discussion of correlation in stochastic computing is covered in chapter “Accuracy and Correlation in Stochastic Computing”.

To ensure the independence assumption holds, stochastic number generators (SNG) are used everywhere in stochastic computing to randomize bit streams before a stochastic operation [4]. The randomization presents a significant cost, which partially defeats the simplicity of stochastic computing. A detailed discussion of SNG is covered in chapter “Generating Stochastic Bitstreams”.

The precision limitation of stochastic computing and the relative high cost of binary conversion are recognized as the showstoppers of stochastic computing. It is not surprising to see that the recent applications of stochastic computing have largely been limited to error-tolerant neural networks [8–10], image processing [5, 11], and decoding of low-density parity-check codes [12, 13].

In this chapter, we propose using the nondeterministic switching of emerging resistive RAM (RRAM) devices as the SNG. Data is randomized every time it is written to RRAM, which removes the need of a standalone SNG. In addition to serving as the SNG, RRAM devices can be used in an end-to-end stochastic computing system that takes analog input, computes in stochastic bit streams, and produces analog output. The end-to-end system eliminates the need of any binary number conversion, making it much more efficient than a stochastic computing module being used as a part of a binary computing system.

Resistive RAM and Stochastic Switching

Memristive devices, also known as RRAM, are a promising non-volatile memory technology thanks to high scalability [14], high endurance [15] and low power consumption [16, 17]. RRAM technology offers relatively simple fabrication, multi-bit capability [18] and ease of 3D stacking [19, 20], allowing the per-bit cost to be reduced significantly. However, a significant drawback of RRAM technology is its spatiotemporal switching variations [21, 22]. The variations span space, i.e., from device to device, and time, i.e., cycle to cycle in the same device. Spatial variations are due to problems including line edge roughness and film thickness irregularity. Such problems have been studied in-depth in CMOS designs, and variation-aware techniques in CMOS designs apply equally to RRAM designs.

The more damaging problem with RRAM devices is the temporal variations. The temporal variations are due to the probabilistic filament formation process. When a single filament growth dominates, the randomness in the switching processes is shown to follow a Poissonian distribution. Controlling the random switching requires margining and costs power and performance. Instead of suppressing randomness, a much better way is to make use of the temporal variations of RRAM devices for stochastic computing.

Stochastic Switching

The temporal variations in resistive switching devices have been studied recently [23]. A binary memristive device is usually engineered to have a relatively high threshold voltage of approximately 5 V. Bias voltages lower than the threshold voltage can be applied to uncover a device’s temporal variations. During such an experimentation, current through the device under test is continuously monitored until a sharp jump in the current is observed, indicating the device is turned ON. The wait time leading to the switch is recorded. The device is then reset to the OFF state and the experimentation is repeated.

As Fig. 1 indicates, the wait time is not constant every time and it varies from cycle to cycle even for the same device. The memristive filament formation associated with the OFF to ON transition is driven by thermodynamics and involves oxidation and ion transport. Since all these physical processes are thermally

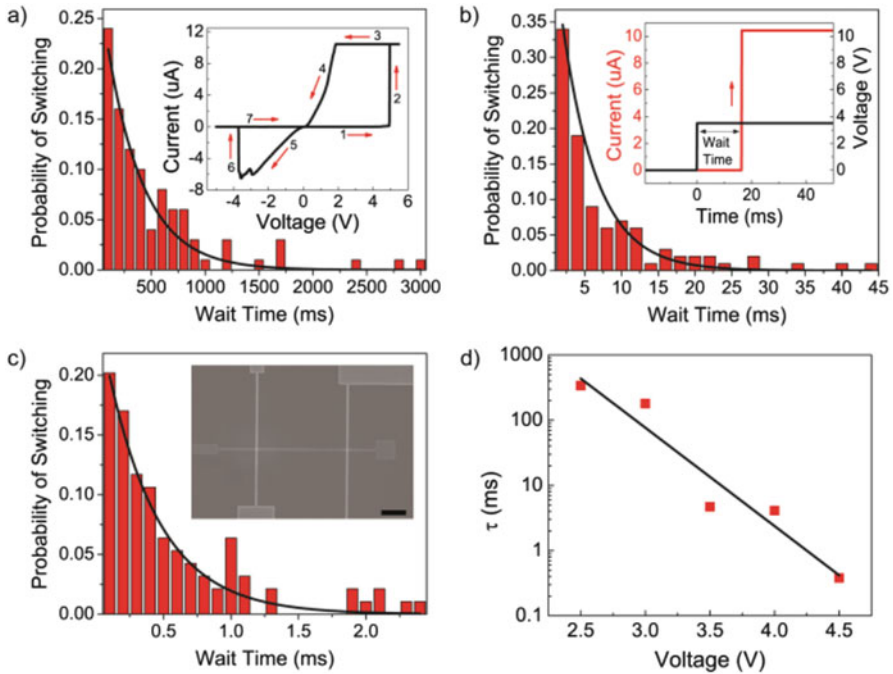


Fig. 1 Random distribution of the wait time prior to switching. (a–c) Distributions of wait times for applied voltages of 2.5 V (a), 3.5 V (b) and 4.5 V (c). Solid lines: fitting to the Poisson distribution (1) using τ as the only fitting parameter. $\tau = 340$ ms, 4.7 ms and 0.38 ms for (a)–(c), respectively. Insets: (a) DC switching curve, (b) example of a wait time measurement and (c) scanning electron micrograph of a typical device (scale bar: 2.5 μ m). (d) Dependence of τ on the programming voltage. Solid squares were obtained from fitting of the wait time distributions while the solid line is an exponential fit. Reproduced from [23]

activated and thermal activation over the dominant energy barrier is probabilistic in nature if only a dominant filament is involved [24]. Therefore, in theory, the wait time should follow a Poisson distribution, and the probability of a switching event occurring within Δt at time t is given by

$$P(t) = \frac{\Delta t}{\tau} e^{-\frac{t}{\tau}}, \quad (1)$$

where τ is the characteristic wait time.

The switching of a memristive device is determined by the characteristic wait time. The characteristic wait time decreases exponentially with the applied voltage. This is consistent with the dominant filament model since both oxidation and ion transport are dependent on the electric field. As the applied voltage is reduced, the effective activation barrier is reduced, resulting in an exponential speed up in switching time [24–26].

By integrating the distribution from (1), the switching probability by a certain time t since the external voltage is applied, $C(t)$, can be obtained.

$$C(t) = 1 - e^{-\frac{t}{\tau}}, \quad (2)$$

where τ is the characteristic wait time.

Using applied voltages of 2.5 V, 3.5 V and 4.5 V, it has been shown that the predicted switching probability from (2) matches the experimental values as shown in Fig. 1 [23]. The experimentation results confirm the hypothesis of stochastic switching and the Poisson model.

The model can be further tested by single pulse experimentation [23], where a 2.5 V single pulse of different widths from 300 ms to 1000 ms is applied to a device. After a pulse, the device state is checked, and then reset to the OFF state. For trials with a 300 ms pulse, the device switched to the ON state 20 times out of 50 attempts; and for trials with a 1000 ms pulse, the device switched to the ON state 38 out of 50 attempts. The experimental results closely follow (2).

If we quantize the current measured in each trial to binary, 1 for a high current and 0 for a low current, a memristive device can be used to generate a stochastic bit stream. For example, if the current through a device is larger (smaller) than $0.1 \mu\text{A}$, it will be treated as 1 (0). Thus if we perform single pulse experimentation n times, the memristive device will generate a n -bit long stochastic bit stream. The equivalent value of the stochastic bit stream is completely determined by the voltage and width of the pulse applied to the device.

Generating Stochastic Numbers

Implementing SNG in digital CMOS circuits is expensive because it relies on a pseudo-random number generator being implemented in CMOS circuits, such as a linear-feedback shift register (LFSR). Given that a single RRAM device is capable

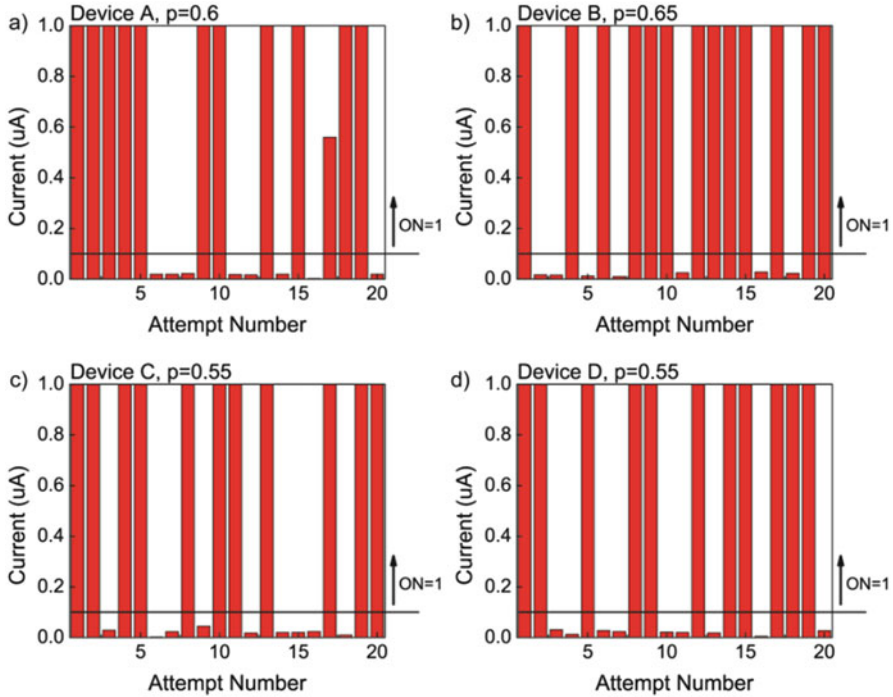


Fig. 2 Generation of non-correlated bit streams. Different devices (A–D) when programmed with identical bias conditions give non-correlated bit streams but with very similar bias (~ 0.6). Reproduced from [23]

of generating a stochastic bit stream, it can be used as an inexpensive SNG. RRAM’s high density and low standby power make it a highly attractive SNG in terms of cost and power.

Using RRAM devices to produce uncorrelated random bit streams has been experimentally demonstrated in [23]. In this work, identical voltage is applied to four devices to obtain four 20-bit long random bit streams as shown in Fig. 2. The four bit streams are uncorrelated, but the number of 1s in the four bit streams are nearly the same (0.6 in this case). In other words, four independent bit streams representing the number 0.6 have been generated without using expensive SNGs.

In-Memory Stochastic Computing

An in-memory stochastic computing system can be designed by employing RRAM for stochastic number generation. The in-memory stochastic computing is “natively” stochastic, as a read from memory generates stochastic bit streams without any separate SNGs. In doing so, we also eliminate the extra overhead

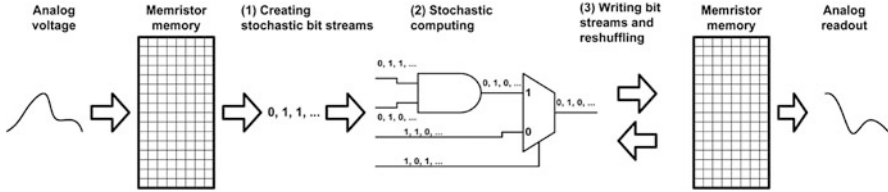


Fig. 3 Native stochastic computing system using RRAM-based stochastic memory. Reproduced from [29]

required for the deterministic use of RRAM. The approach combines stochastic computing with temporally varying RRAM devices to enable an efficient computing system that is not possible with either stochastic computing or RRAM alone.

The in-memory stochastic computing system is illustrated in Fig. 3. The system consists of RRAM integrated with CMOS periphery and logic circuits. The system directly accepts analog input. RRAM converts the analog input to a stochastic bit stream. Computing is entirely done in the bit stream domain. The output bit stream is written to RRAM. A write to RRAM allows the input to be converted to a new bit stream, which serves the purpose of reshuffling needed in stochastic computing to prevent reconvergent fanout.

The in-memory stochastic computing system accepts analog inputs directly. Binary to bit stream conversions are eliminated, but amplifiers and sample and hold circuitry may be needed. In comparison, a classic stochastic computing is entirely digital and requires analog-to-digital conversion to accept analog inputs.

The in-memory stochastic computing system takes advantage of the randomness inherent in RRAM devices that is only present when the operating voltage is relatively low, which naturally leads to a good energy efficiency. The in-memory stochastic computing also inherits all the benefits of a classic stochastic computing system: if a high performance is needed, simple stochastic arithmetic circuits can be parallelized in a flat topology; the independence between bits in a bit stream cuts the critical path delay and simplifies routing; and stochastic computing is error-resilient, and it tolerates noise and soft errors.

In the following subsections, we describe the aspects of the in-memory stochastic computing system.

Stochastic Programming

In an RRAM device, the high-resistance state represents OFF or 0, and the low-resistance state represents ON or 1. As explained previously, the switching of a RRAM device from 0 to 1 is a stochastic process. We can use voltage and pulse width to adjust the switching probability. To save energy, short pulses and low voltage are preferred. A low voltage also prevents device wear-out and prolongs device's lifetime.

Suppose we write 1 to a RRAM device with a pulse width of τ , the switching probability is $P(\tau) = 0.632$ based on (2). If we apply the same voltage pulse to a group of 16 devices, the expected number of devices that will switch to 1 is $0.632 \times 16 = 10.112$. In writing to a group of RRAM devices, the number 0.632 is converted to a stream of 16 bits, the expected number of 1s in which approximates the given number. We call the write to a group of RRAM devices *group write*.

Write Compensation

In an in-memory stochastic computing system, the output bit streams of stochastic computing are written back to RRAM. Instead of storing a bit in a RRAM device in a deterministic write, we apply a stochastic group write to apply the bit stream, bit by bit, to a group of RRAM devices. The important difference is that a deterministic write produces an exact copy, but a stochastic group write reshuffles the bit stream to introduce randomness.

Suppose we apply stochastic group write to apply a pulse train representing a bit stream to a column of RRAM cells as shown in Fig. 4a. The 8-bit stream contains two 1s, i.e., two pulses in the train, to represent 0.25. Suppose we pick an appropriate write voltage such that after the first pulse, an RRAM cell’s switching probability is 0.125. Following the first pulse to the column of 8 RRAM cells, we expect to have an average of 1 out of 8 RRAM cells to switch ON. If the write voltage is kept constant, after the second pulse, the cumulative switching probability of an RRAM

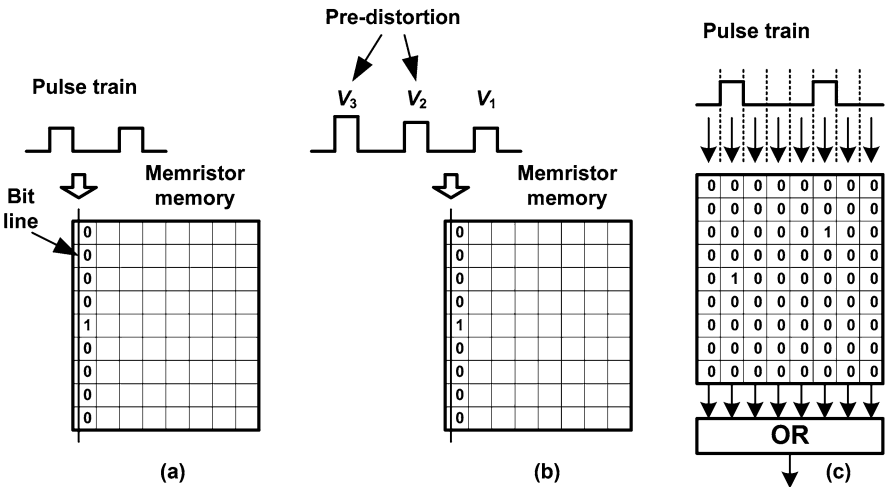


Fig. 4 (a) Stochastic group write to memristor using pulse train, (b) voltage pre-distortion, and (c) parallel single-pulse write. Reproduced from [29]

cell is less than $0.125 \times 2 = 0.25$ due to the nonlinear relationship between switching probably and the number of pulses, or pulse width. The nonlinear write process introduces inaccuracy, and a compensation scheme is needed.

Voltage Predistortion The approach is illustrated in Fig. 4b. The write voltage is increased for each subsequent pulse to undo the nonlinearity of the write process. The approach is expensive since it requires many voltage levels to be provided. A piecewise approximation can be applied to reduce the number of voltage levels needed to reduce cost. In the above example, a three-piece approximation, i.e., three voltages, reduces the relative error to 2.5%.

Downscaled Write and Upscaled Read A downscaled write scales a value to a lower range. Within a lower range, the nonlinearity error is reduced even without applying any compensation. To recover the downscaled value, an upscaled read through a scalar gain function, such as [24], can be applied in the readout. The downscaled write and upscaled read approach avoids using multiple voltages, but small nonlinearity errors remain.

Parallel Single-Pulse Write The approach is illustrated in Fig. 4c, where single pulses in a pulse train are applied to multiple columns of RRAM cells in parallel. Using this approach, only one pulse is applied to a column of cells, avoiding the nonlinearity issue experienced in successive writes. The parallel single-pulse write requires an extra merge step to compress a 2D array of bits to a 1D bit stream by OR'ing the bits in every row. An error can be introduced when there are two or more 1s in a row, where an OR produces only a single 1 in the output. Such an inaccuracy can be compensated through a simple offset correction. The parallel single-pulse write approach is relatively simple to implement, but it uses more memory.

Test Applications

We demonstrate in-memory stochastic computing using two test applications: a gradient descent solver and a k-means clustering processor. The test applications are simulated using ideal writes and the write compensation techniques: voltage predistortion and downscaled write and upscaled read. Voltage noise is intentionally introduced in the second test application to check the robustness of the design.

Gradient Descent Solver

Gradient descent is a first-order optimization algorithm [27]. The algorithm iterates between two steps: (1) compute the gradient of a cost function at the current position; (2) move in the negative direction of the gradient by a step proportional to the magnitude of the gradient. If the cost function is well conditioned, the minimum can be obtained by this algorithm.

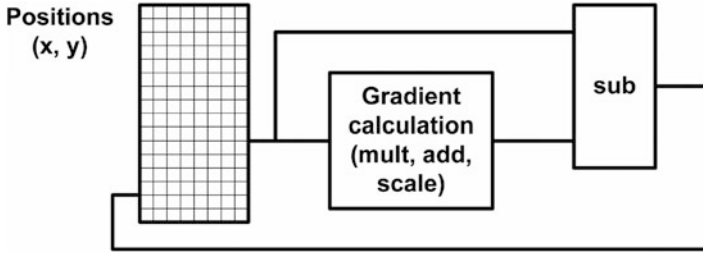


Fig. 5 Stochastic implementation of a gradient descent solver. Reproduced from [29]

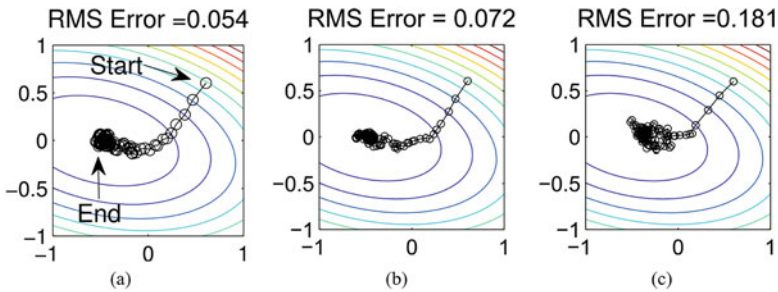


Fig. 6 Stochastic gradient descent algorithm using (a) 32-Kbit stochastic bit stream with ideal write, (b) 32-Kbit stochastic bit stream with voltage predistortion, (c) 256-Kbit stochastic bit stream with downscaled write and upscaled read. Reproduced from [29]

The hardware design of a gradient descent solver is illustrated in Fig. 5. The design can be implemented in stochastic computing using RRAM. First, the input position is stored in RRAM and the readout is in bit stream. Second, the gradient calculation is implemented using stochastic multiply and add, and the step size is obtained by scalar multiply. Finally, the position is updated and stored in RRAM for the next iteration. Known stochastic computing circuits are available to perform the necessary arithmetic functions including add, multiply, and subtract [1–3, 6, 7].

The stochastic design of a gradient descent solver is simulated using 32- and 256-Kbit stochastic bit streams following bipolar stochastic number representation. In the simulation, we used the cost function $f(x, y) = \frac{1}{24}((x + 0.5)^2 + (x + 0.5)y + 3y^2)$. Ideal write and two write compensation techniques: voltage predistortion and downscaled write and upscaled read are used in the simulations. The results demonstrate satisfactory results as shown in Fig. 6.

K-Means Clustering Processor

k-means is a popular clustering algorithm [28] for placing a set of data points into different clusters whose members are similar. The k-means algorithm involves three

steps: (1) select k cluster centers (centroids); (2) place a data point in one of the clusters to minimize the distance between the data point and the cluster centroid; (3) update the centroid of each cluster based on all the data points placed in the cluster. Steps (2) and (3) are iterated until convergence.

The hardware design of a k -means clustering processor is illustrated in Fig. 7, assuming L_1 distance metric. In a stochastic computing implementation, input data points and centroids are stored in RRAMs and the readouts are in bit streams; and L_1 distances and comparisons are calculated by stochastic arithmetic. Once an iteration of k -means clustering is done, stochastic averaging is done to update the cluster centroids.

The stochastic design of a k -means clustering processor is simulated using 4-Kbit stochastic bit streams following bipolar stochastic number representation. Sets of 256 data points are placed in three clusters based on the L_1 distance metric. Ideal write and voltage-predistortion compensation technique are used in the simulations. The results demonstrate satisfactory results shown in Fig. 8.

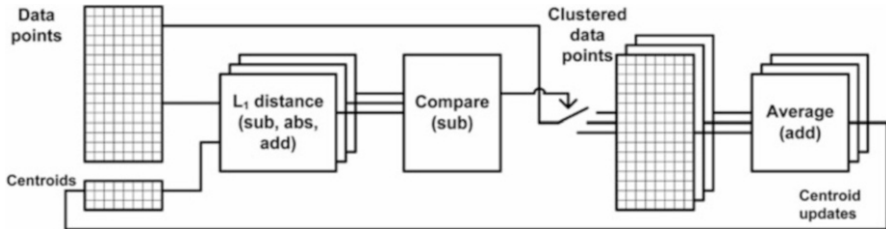


Fig. 7 Stochastic implementation of a k -means clustering processor. Reproduced from [29]

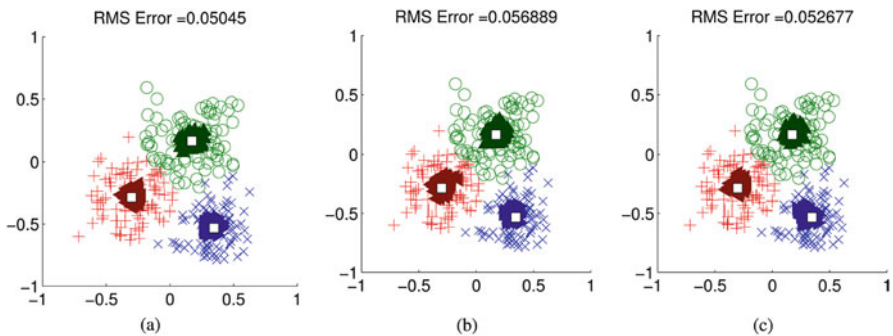


Fig. 8 256-point k -means clustering with 4-Kbit stochastic bit stream using (a) ideal write, (b) voltage pre-distortion with number of voltage levels chosen to meet 0.1% error bound, (c) voltage pre-distortion with number of voltage levels chosen to meet 0.001% error bound. Reproduced from [29]

Concluding Remarks

As a probabilistic computing method, stochastic computing relies on compute elements to generate and ensure randomness. One can argue that CMOS devices, which are designed to be deterministic, are not the perfect substrate to implement stochastic computing. In this chapter, we present in-memory stochastic computing to make use of RRAM devices to provide the randomness required by stochastic computing. Due to the significant simplification of random number generation and reshuffling, we expect that an RRAM-enabled in-memory stochastic computing will be fundamentally more efficient than a CMOS-based stochastic computing. The approach also paves a way towards adopting seemingly non-ideal post-CMOS devices in stochastic computing to achieve significant gains in performance and efficiency.

Acknowledgements This work was supported in part by NSF CCF-1217972. The work of W. Lu was supported by the NSF ECCS-0954621 and in part by the AFOSR under MURI grant FA9550-12-1-0038.

References

1. Gaines, Brian R. "Stochastic computing systems." In *Advances in information systems science*, pp. 37–172. Springer, Boston, MA, 1969.
2. Poppelbaum, W. J., C. Afuso, and J. W. Esch. "Stochastic computing elements and systems." In *Proceedings of the November 14–16, 1967, fall joint computer conference*, pp. 635–644. ACM, 1967.
3. Ribeiro, Sergio T. "Random-pulse machines." *IEEE Transactions on Electronic Computers* 3 (1967): 261–276.
4. Li, Xin, Weikang Qian, Marc D. Riedel, Kia Bazargan, and David J. Lilja. "A reconfigurable stochastic architecture for highly reliable computing." In *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, pp. 315–320. ACM, 2009.
5. Qian, Weikang, Xin Li, Marc D. Riedel, Kia Bazargan, and David J. Lilja. "An architecture for fault-tolerant computation with stochastic logic." *IEEE Transactions on Computers* 60, no. 1 (2011): 93–105.
6. Qian, Weikang, and Marc D. Riedel. "The synthesis of robust polynomial arithmetic with stochastic logic." In *Proceedings of the 45th annual Design Automation Conference*, pp. 648–653. ACM, 2008.
7. Toral, S. L., J. M. Quero, and L. G. Franquelo. "Stochastic pulse coded arithmetic." In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, vol. 1, pp. 599–602. IEEE, 2000.
8. Dickson, Jeffery A., Robert D. McLeod, and H. C. Card. "Stochastic arithmetic implementations of neural networks with in situ learning." In *Neural Networks, 1993., IEEE International Conference on*, pp. 711–716. IEEE, 1993.
9. Kim, Young-Chul, and Michael A. Shanblatt. "Architecture and statistical model of a pulse-mode digital multilayer neural network." *IEEE transactions on neural networks* 6, no. 5 (1995): 1109–1118.
10. Brown, Bradley D., and Howard C. Card. "Stochastic neural computation. I. Computational elements." *IEEE Transactions on computers* 50, no. 9 (2001): 891–905.

11. Hammadou, Tarik, Magnus Nilson, Amine Bermak, and Philip Ogunbona. "A 96/spl times/64 intelligent digital pixel array with extended binary stochastic arithmetic." In *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*, vol. 4, pp. IV-IV. IEEE, 2003.
12. Gaudet, Vincent C., and Anthony C. Rapley. "Iterative decoding using stochastic computation." *Electronics Letters* 39, no. 3 (2003): 1.
13. Tehrani, S. Sharifi, Warren J. Gross, and Shie Mannor. "Stochastic decoding of LDPC codes." *IEEE Communications Letters* 10, no. 10 (2006): 716–718.
14. Govoreanu, B., G. S. Kar, Y. Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. P. Radu et al. "10 10nm 2 Hf/HfO x crossbar resistive RAM with excellent performance, reliability and low-energy operation." In *Electron Devices Meeting (IEDM), 2011 IEEE International*, pp. 31–6. IEEE, 2011.
15. Lee, Myoung-Jae, Chang Bum Lee, Dongsoo Lee, Seung Ryul Lee, Man Chang, Ji Hyun Hur, Young-Bae Kim et al. "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta 2 O 5? x/TaO 2? x bilayer structures." *Nature materials* 10, no. 8 (2011): 625.
16. Chin, Albert, C. H. Cheng, Y. C. Chiu, Z. W. Zheng, and Ming Liu. "Ultra-low switching power RRAM using hopping conduction mechanism." *ECS Transactions* 50, no. 4 (2013): 3–8.
17. Strachan, John Paul, Antonio C. Torrezan, Gilberto Medeiros-Ribeiro, and R. Stanley Williams. "Measuring the switching dynamics and energy efficiency of tantalum oxide memristors." *Nanotechnology* 22, no. 50 (2011): 505402.
18. Park, Jubong, K. P. Biju, Seungjae Jung, Wootae Lee, Joonmyoung Lee, Seonghyun Kim, Sangsu Park, Jungho Shin, and Hyunsang Hwang. "Multibit Operation of TiO_x-Based ReRAM by Schottky Barrier Height Engineering." *IEEE Electron Device Letters* 32, no. 4 (2011): 476–478.
19. Baek, I. G., D. C. Kim, M. J. Lee, H-J. Kim, E. K. Yim, M. S. Lee, J. E. Lee et al. "Multi-layer cross-point binary oxide resistive memory (OxRRAM) for post-NAND storage application." In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pp. 750–753. IEEE, 2005.
20. Baek, I. G., C. J. Park, H. Ju, D. J. Seong, H. S. Ahn, J. H. Kim, M. K. Yang et al. "Realization of vertical resistive memory (VRRAM) using cost effective 3D process." In *Electron Devices Meeting (IEDM), 2011 IEEE International*, pp. 31–8. IEEE, 2011.
21. Yu, Shimeng, Ximeng Guan, and H-S. Philip Wong. "On the switching parameter variation of metal oxide RRAM?Part II: Model corroboration and device design strategy." *IEEE Transactions on Electron Devices* 59, no. 4 (2012): 1183–1188.
22. Chen, An, and Ming-Ren Lin. "Variability of resistive switching memories and its impact on crossbar array performance." In *Reliability Physics Symposium (IRPS), 2011 IEEE International*, pp. MY-7. IEEE, 2011.
23. Gaba, Siddharth, Patrick Sheridan, Jiantao Zhou, Shinhyun Choi, and Wei Lu. "Stochastic memristive devices for computing and neuromorphic applications." *Nanoscale* 5, no. 13 (2013): 5872–5878.
24. Jo, Sung Hyun, Kuk-Hwan Kim, and Wei Lu. "Programmable resistance switching in nanoscale two-terminal devices." *Nano letters* 9, no. 1 (2008): 496–500.
25. Strukov, Dmitri B., and R. Stanley Williams. "Exponential ionic drift: fast switching and low volatility of thin-film memristors." *Applied Physics A* 94, no. 3 (2009): 515–519.
26. Schroeder, Herbert, Victor V. Zhirnov, Ralph K. Cavin, and Rainer Waser. "Voltage-time dilemma of pure electronic mechanisms in resistive switching memory cells." *Journal of applied physics* 107, no. 5 (2010): 054517.
27. Nocedal, Jorge, and Stephen J. Wright. "Numerical optimization 2nd." (2006).
28. MacQueen, James. "Some methods for classification and analysis of multivariate observations." In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14, pp. 281–297. 1967.
29. Knag, Phil, Wei Lu, and Zhengya Zhang. "A native stochastic computing architecture enabled by memristors." *IEEE Transactions on Nanotechnology* 13, no. 2 (2014): 283–293.

Spintronic Solutions for Stochastic Computing



Xiaotao Jia, You Wang, Zhe Huang, Yue Zhang, Jianlei Yang, Yuanzhuo Qu, Bruce F. Cockburn, Jie Han, and Weisheng Zhao

Abstract With the rising requirements of computation efficiencies for artificial intelligence applications, the conventional deterministic computation approach has shown many bottlenecks in developing large scale deep learning algorithms. Especially for Bayesian inference problems with uncertainty and incompleteness, it usually requires many sampling operations which largely degrade the inference efficiencies. In this chapter, a spintronic devices based stochastic computing method is presented for efficient Bayesian inference. Stochastic computing is regarded as a promising approach to improve the area and energy efficiencies with simplified arithmetic operations. Spintronic devices are utilized to realize efficient sampling operations to overcome the inference efficiencies in terms of power, area and speed. The intrinsic randomness existing in switching process of spintronic device is exploited to realize stochastic number generator, which is the critical block for efficient circuit design of Bayesian inference. A device-to-architecture level framework is proposed to evaluate the promised performance of spintronic device based Bayesian inference system.

Introduction

Stochastic computing is an emerging computation manner which is considered to be promising for efficient probability computing [2]. Compared with the conventional

X. Jia · Y. Wang · Z. Huang · Y. Zhang (✉) · W. Zhao
Fert Beijing Institute, BDBC and School of Microelectronics, Beihang University, Beijing, China
e-mail: yz@buaa.edu.cn

J. Yang (✉)
Fert Beijing Institute, BDBC and School of Computer Science and Engineering,
Beihang University, Beijing, China
e-mail: jianlei@buaa.edu.cn

Y. Qu · B.F. Cockburn · J. Han
The Department of Electrical and Computer Engineering, University of Alberta Edmonton,
Alberta, Canada

computation approach with deterministic manner, stochastic computing has smaller hardware footprint and higher energy efficiency because the basic multiplication and accumulation operations can be executed by AND gate and multiplexer (MUX), respectively. As a result, very large scale integration on one chip is much easier to realize using stochastic computing than conventional fixed-point hardware. Benefiting from this merit, stochastic computing has been exploited in deep learning to reduce energy consumption and save area where tremendous arithmetic operations are normally essential.

Stochastic computing is usually implemented by bit-wise operations with stochastic bitstreams which are generated by random number generator (RNG) and comparator. Physical randomness is widely used as entropy source in conventional RNG, such as thermal noise, metastability, and oscillator jitter [30]. However, these RNGs all require extensive post-processing to guarantee a high level of random output, which degrades the performance in terms of speed, power, and area [15]. Consequently, it is urgent to explore new methods of generating random numbers for lower design complexity, area overhead and higher randomness.

The stochastic switching of spintronic devices, such as magnetic tunnel junction (MTJ) provides a high-quality entropy source for RNG [28]. Based on intrinsically unpredictable physical phenomenon, it can supply real random bitstreams by special circuit designs. Moreover, MTJ is regarded as one of the most outstanding candidates for next generation of computing memories. It can overcome the power dissipation issue which is the bottleneck for further scaling down of complementary metal oxide semiconductor (CMOS) technology node. For instance, MTJ features non-volatility and 3D integration, which can turn off the standby power and reduce drastically the power dissipated in data transfer between memory and logic chips [16, 32]. Moreover, MTJ is promising to operate normally in sub 0.1 V, which is another bottleneck of semiconductor devices. Compared with the conventional CMOS based RNGs, the MTJ based circuit design can effectively achieve simplified structure, more compact area, higher speed and better energy-efficiency.

In this chapter, spintronic device based stochastic computing is presented for efficient Bayesian inference system. Bayesian inference provides a powerful approach for information fusion, reasoning and decision making, rendering it as a key tool for data-efficient learning, uncertainty quantification and robust model composition. Meanwhile, energy efficiency can be drastically improved by integrating spintronic device in Bayesian inference which combines processing units and memory. The basics of MTJ device and circuit design of RNG using MTJ are firstly presented, and a device-to-architecture level framework is proposed for MTJ based Bayesian inference.

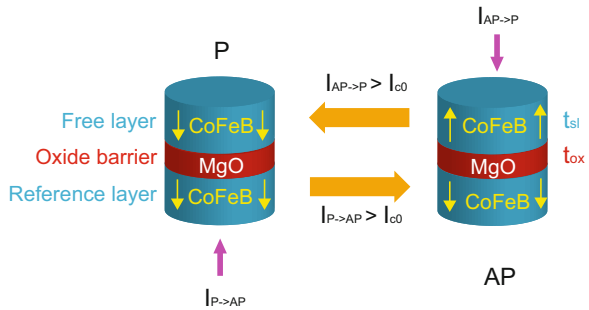
Stochastic Switching Behavior of Magnetic Tunnel Junction

MTJ consists of one oxide barrier sandwiched by two ferromagnetic (FM) layers in which the Tunnel MagnetoResistance (TMR) effect was discovered by [12] in 1975.

The resistance of MTJ depends on the relative magnetization orientation of the two FM layers (R_P at parallel (P) state and R_{AP} at antiparallel (AP) state). As the MTJ resistance can be configured comparable with CMOS transistors, it can be integrated in the memories and logic circuits to represent logic ‘0’ or ‘1’. Its characteristic is quantified by TMR ratio ($(R_{AP} - R_P)/R_P$). The development of MTJ has been quickly prompted by the improvement of TMR ratio and energy consumption reduction of switching approaches (between R_P and R_{AP}) since the first experimental demonstration of TMR effect. The switching method has evolved from field induced magnetic switching (FIMS ~ 10 mA), thermally assisted switching (TAS ~ 1 mA) to the currently widely used spin transfer torque switching (STT ~ 100 μ A). Without the need of magnetic field, STT makes it possible to achieve high density and low power magnetoresistive random access memory (MRAM). MTJ with interfacial perpendicular magnetic anisotropy (PMA-MTJ) was discovered by Shoji Ikeda [10] which features low switching current (49 μ A), and high thermal stability. Recently, the atom-thick tungsten layers have been integrated in PMA-MTJ instead of the conventional tantalium layers to obtain larger TMR ratio and higher thermal stability [18, 26].

Figure 1 shows the typical structure of STT-PMA-MTJ which mainly consists of three layers: two FM layers separated by an insulating oxide barrier. With STT mechanism, MTJ changes between two states when a bidirectional current I is higher than the critical current I_{c0} . The switching of MTJ state is not immediate after the injection of current, resulting an incubation delay. The dynamics of MTJ is mainly characterized by the average switching delay τ_{sw} (with 50% of switching probability). Depending on the magnitude of switching current, the dynamic behavior of MTJ can be divided into two regimes [14]: Sun model ($I > I_{c0}$) and Neel-brown model ($I < 0.8I_{c0}$). The former is also called precessional switching which addresses fast switching (until sub 3 ns) but consumes more energy with high current density [29]. Reversely, the latter consumes less energy with low current density but leads to a slower switching which is called thermally-assisted switching [8]. The two regimes are derived from the Landau-Lifshitz-Gilbert equation. τ_{sw} can be calculated as Eqs. (1) and (2):

Fig. 1 Typical structure of PMA-MTJ [27]



$$\tau_{sw} = \tau_0 \cdot \exp \left[\frac{\Phi_b}{k_B T} \left(1 - \frac{I}{I_{c0}} \right) \right], \quad \text{when } I < 0.8I_{c0} \quad (1)$$

$$\frac{1}{\tau_{sw}} = \left[\frac{2}{C + \ln\left(\frac{\pi^2 \zeta}{4}\right)} \right] \frac{\mu_B P_{ref}(I - I_{c0})}{em_m(1 + P_{ref} P_{free})}, \quad \text{when } I > I_{c0} \quad (2)$$

where τ_0 is the attempt period, T is the temperature, k_B is the Boltzmann constant, C is the Euler's constant, ζ is the thermal stability factor, m_m is the magnetization moment, P is the tunneling spin polarizations. Usually a high current ($I > I_{c0}$) is applied to guarantee fast writing in memory. Meanwhile, MTJ can also be switched erroneously by relatively low current ($I < 0.8I_{c0}$) during a long period of reading operation, which determines the data retention time.

STT switching method has been demonstrated intrinsically stochastic for the first time by [6]. The reversal duration of STT writing mechanism may vary significantly from one event to the next, with a standard deviation almost as large as the average switching duration and sigmoidal distributions with exponential tails [5]. The switching success probability is a function of current flowing through MTJ and pulse duration. This is very different from the traditional electronic devices such as transistors and resistors. The stochastic behavior originates from the unavoidable thermal fluctuations of magnetization which randomly interfere to activate or slow down magnetization reversal. After this observation, many other researchers have theoretically or experimentally verified this phenomenon [3, 22, 23, 31]. It has been well confirmed, both theoretically and experimentally that a spin-polarized current will deposit its spin-angular momentum into the magnetic system when passing through a small magnetic conductor. Consequently, it causes the magnetic moment to precess or even switch when the spin-current is sufficient [21]. Figure 2 illustrates the precession of magnetization under the influence of a spin current. Due to the thermal fluctuation of magnetization, the initial state of free layer magnetic moment (represented by θ) is different at each measurement. This leads to the stochastic reversal of free layer magnetization, which is revealed by the random duration for the resistance reversal [6]). For $I > I_{c0}$, the switching probability can be described as follows [24, 27]:

$$P_{sw} = \exp \left\{ -4 \cdot f \cdot \zeta \cdot \exp \left[-\frac{2 \cdot (t_{pulse} - delay)}{\tau_{sw}} \right] \right\} \quad (3)$$

$$f = \left(\frac{2}{1 - \frac{I_{c0}}{I}} \right)^{\left(\frac{-2}{1 + \frac{I}{I_{c0}}} \right)} \quad (4)$$

where t_{pulse} is the voltage pulse width, $delay$ is a fitting parameter. Figure 3 demonstrates the switching probability as function of stress voltage and pulse width.

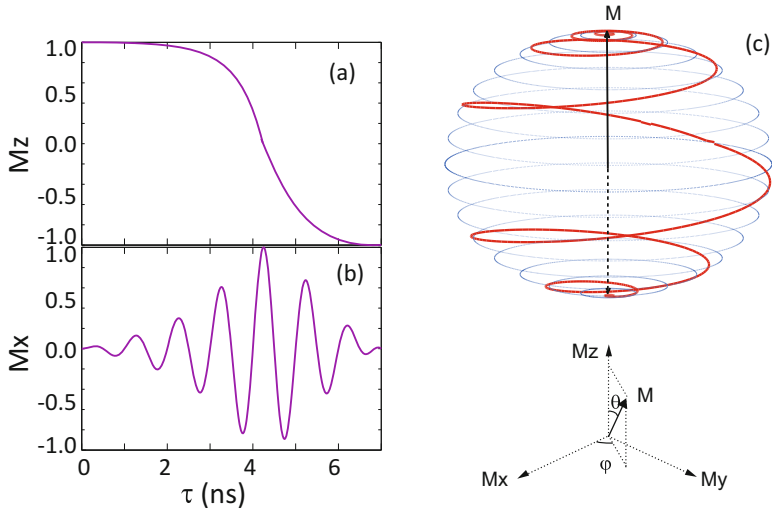


Fig. 2 The precession of magnetization under the influence of a spin current [28]: Time dependence of (a) M_z and (b) M_x , (c) The reversal process of magnetic moment

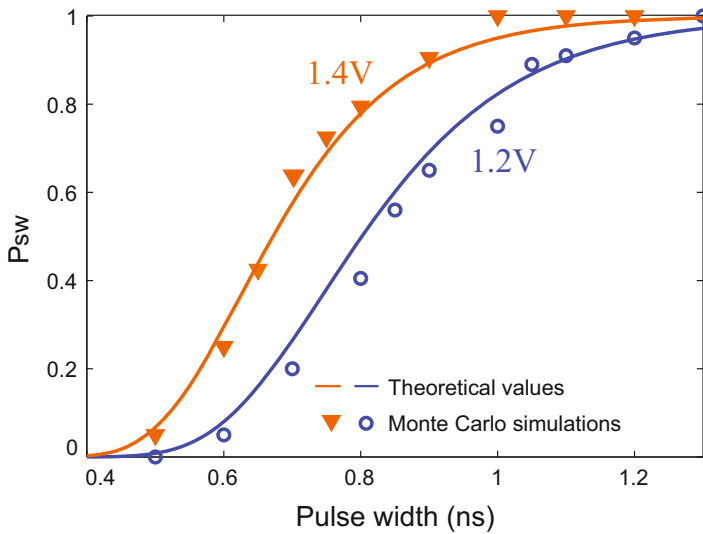


Fig. 3 Switching probability P_{sw} as a function of pulse width [27]: the lines are theoretical values plotted from Eq. (3) and the markers are statistical results from 1000 times of Monte Carlo simulation under Cadence

A tunable switching current I_{sw} can be applied to control the switching probability and further obtain random bitstream during circuit design phase.

MTJ-Based True Random Number Generator

RNG is an essential part in a stochastic computing system. Two categories of RNGs are used: pseudo-random number generators (PRNGs) and true random number generators (TRNGs) [13]. Tausworthe generators and a specific implementation, linear-feedback shift registers (LFSRs), are typical examples of PRNGs [7]. The sequences generated from PRNGs are fully deterministic but their statistical properties make them look random. The generation algorithms make the sequences fully predictable and periodic, and the same sequence will be generated from the same random seed [2]. Thus, there are interests in replacing PRNGs in cryptographic applications because of the predictability.

In contrast with PRNGs, TRNGs generate numbers with true randomness that originates from nondeterministic physical phenomena [9]. Some types of random physical events, such as the chaotic behavior in semiconductor lasers [17, 25], can produce random bitstreams extremely fast with high quality (e.g., 480 Gbit/s is reported in [17]). However, on-chip applications require schemes that are scalable and compatible with CMOS technology. Moreover, energy consumption and the generation speed are important implementation criteria for mobile devices in the IoT era. Therefore, we seek TRNGs that can produce random sequences for cryptographic applications with CMOS compatibility, high statistical quality, low area cost and high energy-efficiency.

Based on the intrinsic stochastic behavior of MTJ device, a TRNG can be easily accomplished and used for stochastic number generator (SNG). A pair of transistors is sufficient to control the process of writing and reading.

However, due to the parameter variations of MTJs, the resistance of any MTJ is slightly different from the nominal value. This difference indirectly results in varied current injected into MTJ and corresponding switching probability with fixed bias voltage. Therefore, the variation of switching probability will lead to deterioration of the randomness of final generated stochastic bitstream.

Several methods have been proposed, including Von Neumann correction, XOR gate and other designs with complicated feedback circuit. These methods always suffer from deficiency such as complicated structure, high power consumption, etc. To compensate for the poor randomness without complicated feedback circuits, we proposed a parallel structure of TRNGs [19].

This design chooses a structure of parallel MTJs considering stochastic computing as application scenario. Because the standard deviation of the average of N independent Gaussian-distributed random variables is

$$\frac{\sigma_{X_1+\dots+X_N}}{N} = \frac{\sqrt{\sigma_1^2 + \dots + \sigma_N^2}}{N} (= \frac{\sigma_N}{\sqrt{N}}, \text{ if } X_1 = \dots = X_N) \quad (5)$$

the stochastic bitstream generated by parallel MTJs will have smaller standard deviations in the probability. In other words, the biased probabilities of each single MTJ will be averaged so that the overall probability gets closer to 50%.

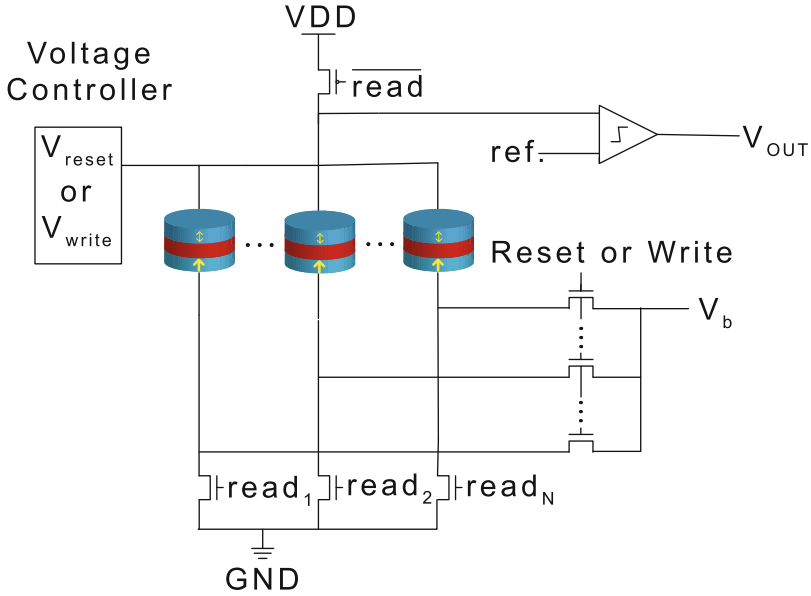


Fig. 4 Proposed TRNG with multiple parallel MTJs

The schematic of this parallel MTJ TRNG design is shown in Fig. 4. According to the different precision requirements of stochastic bitstream, the actual number of parallel MTJs can be adjusted. To generate N bits stochastic number, the circuit needs $N + 2$ phases: a reset phase, a write phase and N read phases, with each phase set to 5 ns. During each phase, the corresponding control signal is driven high while the others are held low. All MTJs work simultaneously during the first two phases while one MTJ is sensed each time in the read phases. Here the $N + 2$ phases are explained in detail:

(1) Reset Phase

After the output of the previous cycle is completed, it is necessary to reset the state of all MTJs back to initial state before the start of the write phase of the next cycle. In this phase, the control signal *Reset* will be high while others are low. The voltage controller provides V_{reset} and current flows from the free layer to the pinned layer until all MTJs are switched to the P state. The voltage difference between V_{reset} and V_b is high enough to ensure deterministic switching.

(2) Write Phase

In this phase, the control signal *Write* is high while others are low. The V_{write} should be lower than V_b to generate switching current from the pinned layer to the free layer. In testing process, the voltage is selected for 50% switching probability in 5 ns for each MTJ. In actual application scenario for stochastic computing, the switching probability can be set to any value required to generate stochastic bitstream directly without external circuit. Because all MTJs are connected in

parallel, the voltages across each MTJ and the corresponding transistors are the same. All MTJs are written under same bias voltage simultaneously but each MTJ switches independently. At the end of write phase, parts of MTJs will switch to the AP state while others remain in the P state.

(3) Read Phase

In the read phases, only one of the N $Read_n$ is high, from $Read_1$ to $Read_N$, while others are low. The current flows from VDD to GND passing through only one MTJ. Depending on the resistance variation of MTJ, the output voltages are different. The comparator will judge the state of this MTJ by comparing the output voltage with the reference value and generate single bit stochastic number. After N read phases, the RNG finally outputs N bits stochastic bitstream.

Compared to other RNG designs, the significant advantage of parallel structure is that the switching probability is controllable. In actual stochastic computing, RNG needs to generate stochastic bitstream with different proportions of ‘0’ and ‘1’. Digital comparator and other external circuit are always necessary to achieve this target. However, through variations of V_{write} , TRNG of parallel structure can output stochastic bitstream directly in any proportions of ‘0’ and ‘1’.

Moreover, all MTJs work simultaneously in reset and write phase, which requires less time compared to the structure of single MTJ. Suppose that each phase needs 5 ns, the generation speed is Eq. (6).

$$\frac{N}{N+2} \times 200 \quad (6)$$

In the parallel design, the accuracy of the switching probability is subject to the actual voltage and duration of the pulse applied to the MTJs, and variety of circuit parameters. In order to keep the precise probability, the pulses applied to the MTJs should be well controlled and the variations of the transistors should be less significant compared to that of MTJs.

The quality of the random sequences needs to be evaluated in aspects other than frequency to demonstrate the effectiveness of our approaches. Therefore, we applied the widely used statistical test suite National Institute of Standards and Technology (NIST) [20].

For the given value N , the proposed generation procedure was repeated $256/N$ times, and each MTJ was used $256/N$ times to generate random bits, where N is the number of MTJs in the array. After one sequence of 256 bits is generated, a new set of N MTJs is used to generate the next sequence.

The four curves at the left side of Fig. 5 show the pass rate trends for different categories of tests, and illustrate the quality improvement of the generators with increasing number of MTJs used. The horizontal line is the threshold of 0.981 for passing the tests. When using at least 16 MTJs, the pass rates for all tests are no less than 0.981. Therefore, it was shown by the statistical test suite that high-quality 256-bit random sequences can be generated by utilizing at least 16 MTJs in the proposed TRNG.

This TRNG of parallel structure is suitable for stochastic computing. Firstly, TRNG has better randomness compared to PRNG. Furthermore, the parallel

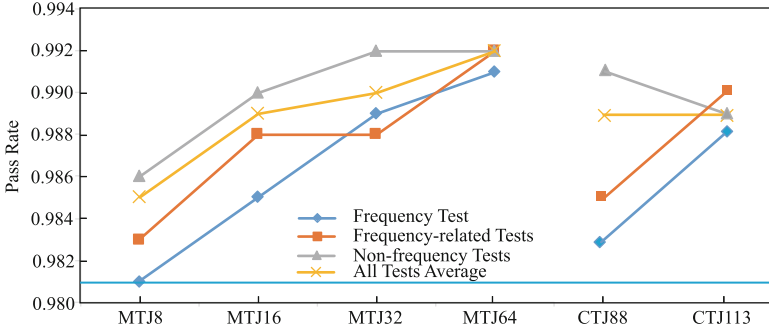


Fig. 5 Statistical quality pass rates of four MTJ-based TRNGs and two combined Tausworthe generators

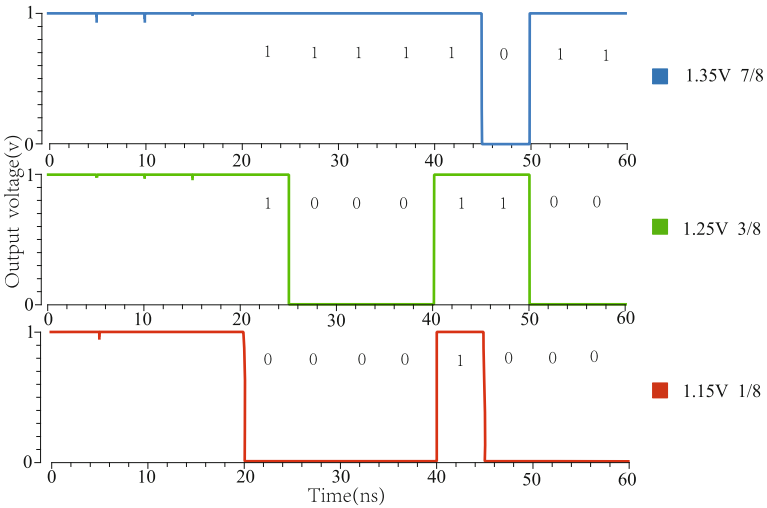


Fig. 6 The waveforms of direct output under different write voltages

structure could directly generate stochastic bitstream with different proportions of ‘0’ and ‘1’. Considering different requirements in various the application scenarios of stochastic computing, we can flexibly choose the number of MTJs in this structure.

In order to evaluate the performance of the circuit design, a simulation is processed by Cadence Virtuoso with 45 nm CMOS and 40 nm MTJ technologies. In the simulation, a behavioral model of MTJ considering the stochastic switching feature is described by Verilog-A language [27]. The waveforms of direct output are shown in Fig. 6. Through controlling the write voltage, the proportions of ‘0’ and ‘1’ in output stochastic bitstream can be adjusted to the targeted value. The write voltages applied are 1.35 V, 1.25 V and 1.15 V from top to bottom, respectively.

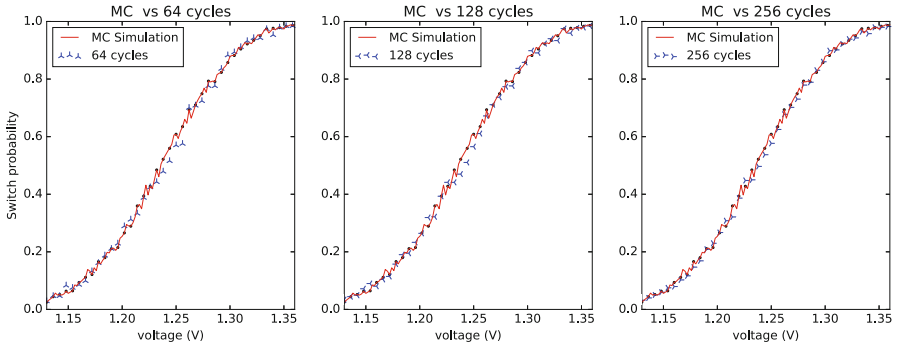


Fig. 7 Applied voltage vs. switching probability

With accurately tunable write voltage, consecutive proportion values between 0 and 1 can be obtained by this circuit.

For each bias voltage ranging from 1.13 V to 1.36 V, 1000 Monte-Carlo (MC) simulations are performed [11]. The simulated P-V relationship is illustrated in Fig. 7 by the red line. It is demonstrated in the figure that the switching probability increases monotonously as the increasing of voltage. It means that voltages and probability values are almost corresponding one by one. In order to evaluate the performance of the proposed SNG circuit, bitstreams are generated with length of 64, 128 and 256. As shown in Fig. 7, results of all the three classes bitstreams are well coincident with Monte-Carlo simulation results. Compared with Monte-Carlo simulation results, the average errors are only 1.6%, 1.3% and 1.1% for length of 64, 128 and 256, respectively. It is obvious that the longer the bitstream, the smaller the error.

Bayesian Inference System

Different applications are usually solved using different Bayesian inference mechanisms. Thus, structures of Bayesian inference system are also different. In this section, A device-to-architecture level evaluation framework is illustrated and then two different types of applications with different inference mechanisms are considered. Using the aforementioned MTJ based SNG and stochastic computing theory, we build two Bayesian inference systems for two different applications.

Evaluation Framework

Bayesian inference system is implemented by hybrid CMOS/MTJ technologies with three design hierarchies: device, circuit and architecture levels as shown in Fig 8. The hybrid CMOS/MTJ circuits are simulated by Cadence Spectre simulator. With

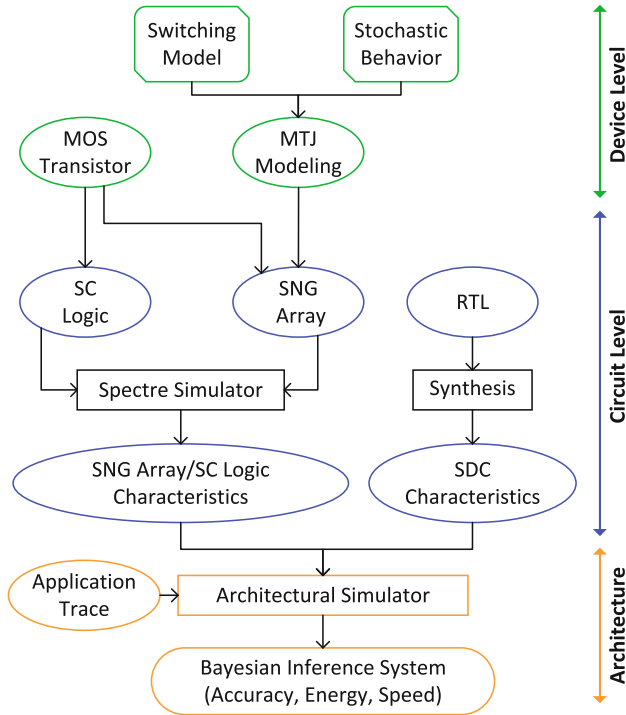


Fig. 8 Evaluation framework of Bayesian inference system

the circuit simulation results, the SNG array and stochastic computing logics are abstracted as behavioral blocks by performing characterizations. Meanwhile, the RTL implementation of stochastic to digital converter (SDC) is synthesized by Synopsys Design Compiler with 45 nm FreePDK library. After performing the characterization of SDC, an architectural level simulation is carried out according to the specified application trace. Finally, the evaluation results of Bayesian inference system are obtained in terms of inference accuracy, energy efficiency and inference speed.

Data Fusion for Target Location

Data fusion is the process of integrating multiple data sources to produce more consistent, accurate, and useful information than that provided by any individual data source. In this section, a simple data fusion example and corresponding Bayesian inference system are studied.

Problem Definition and Bayesian Inference Algorithm

Sensor fusion aims to determine a target location by multiple sensors [4]. Assuming that there are three sensors on a $2D$ plane while the width and length of $2D$ plane is 64 and three sensors are located at $(0, 0)$, $(0, 32)$, and $(32, 0)$, respectively. Each sensor has two data channels: distance (d) and bearing (b). The measured data $(d_1, b_1, d_2, b_2, d_3, b_3)$ from three sensors with two channels are utilized to inference the target location (x^*, y^*) . In this application, the probability that target object locates at one position of the plane is calculated based on the sensor data. The position with the largest probability is considered to be the position where the object target is located at.

Based on the observed data $(d_1, b_1, d_2, b_2, d_3, b_3)$, the probability of target object located on (x, y) is denoted as $p(x, y|d_1, b_1, d_2, b_2, d_3, b_3)$ and could be calculated based on Bayes' theory:

$$p(x, y|d_1, b_1, d_2, b_2, d_3, b_3) \propto p(x, y) * \prod_i p(d_i|x, y)p(b_i|x, y) \quad (7)$$

where $p(x, y)$ is denoted as prior probability, and $p(d_i|x, y)$, $p(b_i|x, y)$ are known as evidence or likelihood information. Since the target may locate at any position, the prior probability $p(x, y)$ has the same value for any position. Hence, $p(x, y)$ is ignored in the following Bayesian inference system. $p(d_i|x, y)$ means the probability that sensor i return the distance value of d_i if the target object is located at position (x, y) . The meaning of $p(b_i|x, y)$ is similar to that of $p(d_i|x, y)$. The value of $p(d_i|x, y)$ and $p(b_i|x, y)$ is calculated by Eqs. (8) and (9).

$$p(d_i|x, y) = \frac{1}{\sqrt{2\pi}\sigma_i^d} \cdot e^{-\frac{(d(x,y)-\mu_i^d)^2}{2(\sigma_i^d)^2}} \quad (8)$$

$$p(b_i|x, y) = \frac{1}{\sqrt{2\pi}\sigma_i^b} \cdot e^{-\frac{(b(x,y)-\mu_i^b)^2}{2(\sigma_i^b)^2}} \quad (9)$$

where $d(x, y)$ is the Euclidian distance between position (x, y) and the i -th sensor, μ_i^d is the distance data provided by the i -th sensor, $\sigma_i^d = 5 + \mu_i^d/10$. $b(x, y)$ is the viewing angle from the i -th sensor to position (x, y) , μ_i^b is the bear data provided by the i -th sensor, σ_i^b is set as 14.0626 degree.

Bayesian Inference System

It can be seen from Bayesian inference mechanism (Eq. 7) that the distribution of object location is calculated by the product of a series of conditional probabilities. In stochastic computing, it could be realized by AND gates. In addition, we

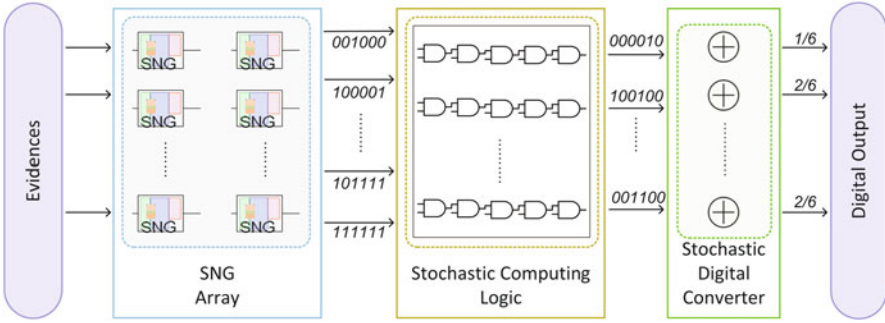


Fig. 9 Bayesian inference system for object location problem

could find that the calculation of probability value that the object locates at one position is independent of each other. Based on the analysis, the Bayesian inference architecture of solving data fusion problem is illustrated in Fig. 9 as a matrix structure. For each position, six SNGs are deployed to yield stochastic bitstreams and 5 AND gates are deployed to realize multiplication. Thus, for a 64×64 grid, 24576 SNGs and 20480 AND gates are needed. In Fig. 9, the output of each row is the posterior probability value that the object locates at this position. In our simulation, 64×64 counters are employed to decode the outputs from stochastic bitstreams to binary numbers by calculating the proportion of '1'. Utilizing the independent of inference algorithm (i.e. Eq. 7), all rows of the system could perform stochastic computing at the same time. The proposed architecture makes the best use of high parallel attribute of Bayesian inference and stochastic computing.

Simulation Results

Cadence Virtuoso is used to analyze the accuracy and efficiency of the proposed Bayesian inference system. In the simulation, 64×64 , 32×32 and 16×16 grids are utilized to evaluate our Bayesian inference system. The finer the grid, the more accurate the target position. For every grid scale, stochastic bitstreams (BSs) with length of 64, 128 and 256 are generated to perform stochastic computing. In Fig. 10, the fusion results on 64×64 grid are shown as heat maps. Figure 10a is the exact inference result using exact arithmetic computing in float-point arithmetic computer. Figure 10b, c and d are the inference results by the proposed Bayesian inference system with stochastic bitstreams length of 64, 128 and 256, respectively. The simulation results indicate that the proposed system could achieve the Bayesian inference results correctly. Compared with exact inference results, the longer the stochastic bitstream, the smaller the error. To quantify the precision of the inference system, the Kullback-Leibler divergence (KL divergence) between stochastic inference distribution and the exact reference distribution is calculated. As shown in Table 1, the first column shows the grid scale. The following 3 columns are the KL divergence value for different bitstream lengths. Taking 32×32 grid for example,

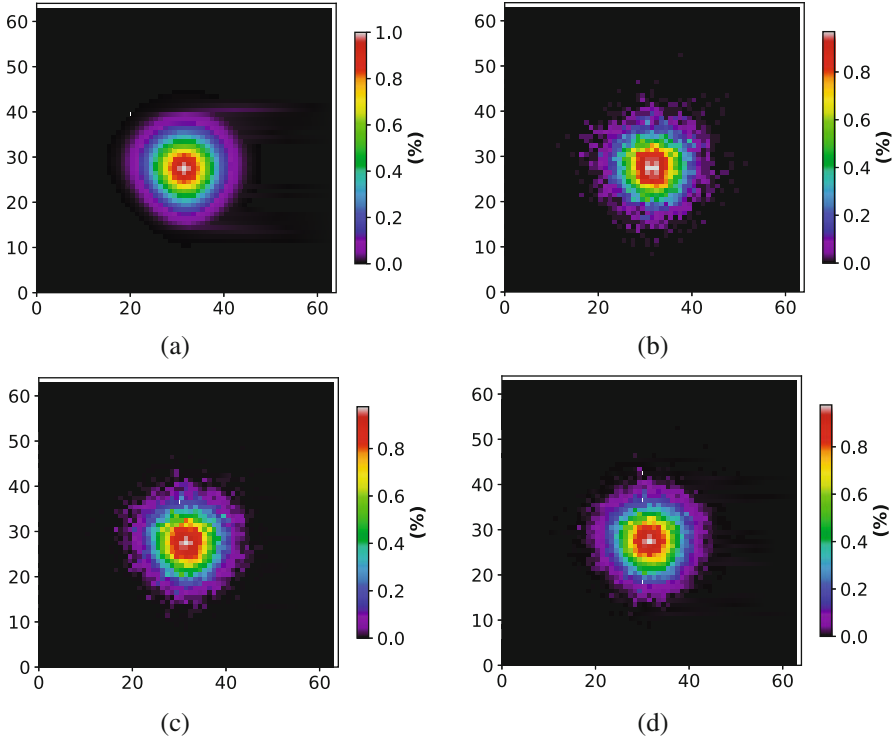


Fig. 10 Data fusion result of target location problem on 64×64 grid. (a) Exact inference results. (b)–(d) Stochastic computing results with length of 64, 128, 256

Table 1 KL divergence analysis of target location problem

Grid size	Bitstream length		
	64	128	256
64×64	0.0090	0.0043	0.0018
32×32	0.0086	0.0041	0.0019
16×16	0.0080	0.0035	0.0011

10^{-3} KL divergence requires length of 256. But for the same precision, the work in [4] requires length of 10^5 . The outstanding results benefit from the high accuracy and low correlation bitstreams generated by the MTJ based SNG. As reported in [4], for an instance with 32×32 grid, the software version on a typical laptop takes 919 mJ, and the FPGA based Bayesian machine only takes 0.23 mJ with stochastic bitstream length of 1000. Benefiting from the low power consumption of MTJs and high quality of SNG, the proposed Bayesian inference system only spends less than 0.01 mJ to achieve the same accuracy with the 32×32 grid. Speed of the proposed Bayesian inference system depends on the bitstream length.

Bayesian Belief Network

Bayesian belief network (BBN) is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph. In this section, a Bayesian belief network for heart disaster is studied.

Problem Definition and Bayesian Inference Algorithm

Figure 11 is a BBN example for heart disaster prediction. In this network, the parent nodes of heart disaster (H) are factors that cause heart disaster, including exercise (E) and diet (D). The child nodes are clinical manifestations of HD, including blood pressure (B) and chest pain (C). In addition to the graph structure, Conditional probability tables (CPT) are also given. For example, the second value 0.45 in the CPT of node HD means that if a person takes regular exercise but unhealthy diet, the risk of HD is 0.45. In this problem, we pay more attention to inference based on given evidences. For the sake of convenience, X^1 is used to indicate that the value of random variable X is *TRUE* and X^0 is used to indicate that the value of random variable X is *FALSE*. If the value of random variable X is not determined, there is no superscript. The inference mechanism could be classed as two groups based on the junction tree algorithm. The first case is considering E, D and H as a group and calculating $p(HD)$ as Eq. (10):

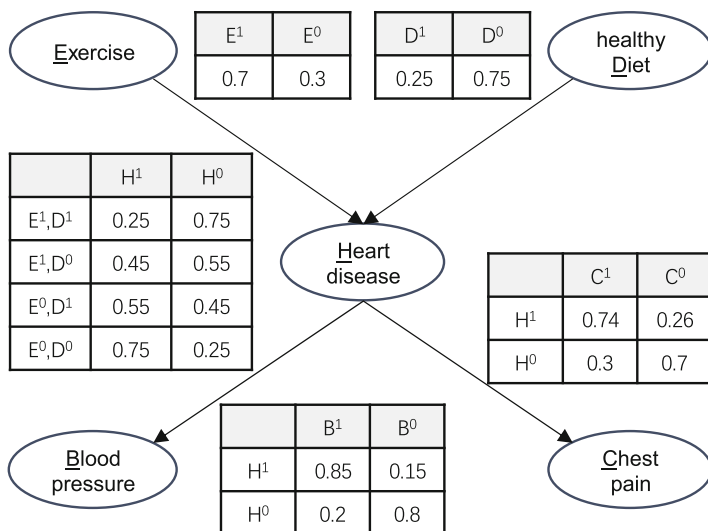


Fig. 11 Bayesian belief network for heart disaster

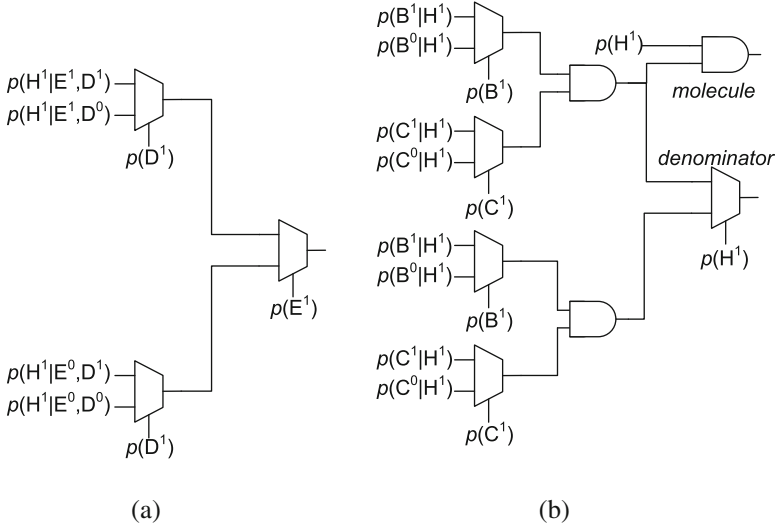


Fig. 12 (a) Bayesian inference circuit for BBN that realizes Eq. (10). (b) Bayesian Inference circuit for BBN that realizes Eq. (11)

$$p(H^1) = [p(H^1|E^1, D^1)p(D^1) + p(H^1|E^1, D^0)p(D^0)]p(E^1) + [p(H^1|E^0, D^1)p(D^1) + p(H^1|E^0, D^0)p(D^0)]p(E^0) \quad (10)$$

$p(E^1) \leftarrow 1$, if the patient often does exercise; otherwise, $p(E^1) \leftarrow 0$. If the exercise information is unavailable, $p(E^1)$ is the value in CPT i.e. $p(E^1) \leftarrow 0.7$. $p(E^0) = 1 - p(E^1)$. Based on the similar method, the value of $p(D^1)$ could be obtained. The second case is considering H, B and C as a group and calculating $p(H^1|B, C)$ as Eq. (11):

$$p(H^1|B, C) = \frac{p(B|H^1)p(C|H^1)p(H^1)}{p(B, C)} \quad (11)$$

The denominator of Eq. (11) can be calculated with the formula of full probability as Eq. (12):

$$p(B|H^1)p(C|H^1)p(H^1) + p(B|H^0)p(C|H^0)p(H^0) \quad (12)$$

Here, $p(H^1)$ is calculated by Eq. (10). In Eq. (11), the value of B and C is not labeled explicitly. Their value is determined based on the diagnostic results.

Based on the inference algorithm, the inference system could be easily constructed. Equation (10) could be calculated by three MUXs as shown in Fig. 12a. Equation (11) could be calculated by three AND gates and five MUXs as shown in Fig. 12b. Based on the evidence, the Bayesian inference is performed by different combination of MUX control signal.

Table 2 Example of BBN inference setting

Probability	$(p(D^1), p(E^1), p(B^1), p(C^1))$	[1]	Ours
$p(H^1 B^1)$	(0.25, 0.75, 1.00, 0.00)	0.803	0.805
$p(H^1 D^1, E^1, B^1)$	(1.00, 1.00, 1.00, 0.00)	0.586	0.592
$p(H^1 E^1, B^1)$	(0.25, 1.00, 1.00, 0.00)	0.687	0.694
$p(H^1 D^1, E^1, B^1, C^1)$	(1.00, 1.00, 1.00, 1.00)	0.777	0.742
$p(H^1 C^1)$	(0.25, 0.75, 0.00, 1.00)	0.703	0.700

Simulation Results

The simulation of Bayesian inference system for BBN is also used Cadence Virtuoso and the simulation results are shown in Table 2. The first column of the table lists some of the possible posterior probability. The second column gives the corresponding settings of control signal for each MUX. Column 3 shows the exact results calculated by [1]. Column 4 is the results calculated by the proposed Bayesian inference system using stochastic computing. The comparison between column 6 and column 7 indicates that the proposed Bayesian inference system for BBN could achieve reasonable results.

References

1. Pythonic bayesian belief network framework. <https://github.com/eBay/bayesian-belief-networks> (2017)
2. Alaghi, A., Hayes, J.P.: Survey of stochastic computing. *ACM Transactions on Embedded computing systems* **12**(2s), 92 (2013)
3. Marins de Castro, M., Sousa, R., Bandiera, S., et al.: Precessional spin-transfer switching in a magnetic tunnel junction with a synthetic antiferromagnetic perpendicular polarizer. *Journal of Applied Physics* **111**(7), 07C912 (2012)
4. Coninx, A., Bessi re, P., Mazer, E., Droulez, J., Laurent, R., Aslam, M.A., Lobo, J.: Bayesian sensor fusion with fast and low power stochastic circuits. In: *International Conference on Rebooting Computing*, pp. 1–8 (2016)
5. Devolder, T., Chappert, C., Ito, K.: Subnanosecond spin-transfer switching: Comparing the benefits of free-layer or pinned-layer biasing. *Phys. Rev. B* **75**, 224,430 (2007)
6. Devolder, T., Hayakawa, J., Ito, K., Takahashi, H., Ikeda, S., Crozat, P., Zerounian, N., Kim, J.V., Chappert, C., Ohno, H.: Single-shot time-resolved measurements of nanosecond-scale spin-transfer induced switching: Stochastic versus deterministic aspects. *Physical review letters* **100**(5), 057,206 (2008)
7. Han, J., Chen, H., Liang, J., Zhu, P., Yang, Z., Lombardi, F.: A stochastic computational approach for accurate and efficient reliability evaluation. *IEEE Transactions on Computers* **63**(6), 1336–1350 (2014)
8. Heindl, R., Rippard, W.H., Russek, S.E., Pufall, M.R., Kos, A.B.: Validity of the thermal activation model for spin-transfer torque switching in magnetic tunnel junctions. *Journal of Applied Physics* **109**(7), 073,910 (2011)
9. Hellekalek, P.: Good random number generators are (not so) easy to find. *Mathematics and Computers in Simulation* **46**(5–6), 485–505 (1998)

10. Ikeda, S., Miura, K., Yamamoto, H., Mizunuma, K., Gan, H.D., Endo, M., Kanai, S., Hayakawa, J., Matsukura, F., Ohno, H.: A perpendicular-anisotropy CoFeB-MgO magnetic tunnel junction. *Nature Materials* **9**, 721–724 (2010)
11. Jia, X., Yang, J., Wang, Z., Chen, Y., Zhao, W.: Spintronics based stochastic computing for efficient bayesian inference system. In: *Asia and South Pacific Design Automation Conference*, pp. 580–585 (2018)
12. Julliere, M.: Tunneling between ferromagnetic films. *Physics Letters A* **54**(3), 225–226 (1975)
13. Katz, J., Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: *Handbook of applied cryptography*. CRC press (1996)
14. Koch, R.H., Katine, J.A., Sun, J.Z.: Time-resolved reversal of spin-transfer switching in a nanomagnet. *Phys. Rev. Lett.* **92**, 088,302 (2004)
15. Liu, N., Pinckney, N., Hanson, S., Sylvester, D., Blaauw, D.: A true random number generator using time-dependent dielectric breakdown. In: *Symposium on VLSI Circuits*, pp. 216–217 (2011)
16. Matsunaga, S., Hayakawa, J., Ikeda, S., Miura, K., Endoh, T., Ohno, H., Hanyu, T.: MTJ-based nonvolatile logic-in-memory circuit, future prospects and issues. In: *Design, Automation & Test in Europe Conference & Exhibition*, pp. 433–435 (2009)
17. Oliver, N., Soriano, M.C., Sukow, D.W., Fischer, I.: Fast random bit generation using a chaotic laser: approaching the information theoretic limit. *IEEE Journal of Quantum Electronics* **49**(11), 910–918 (2013)
18. Peng, S., Zhao, W., Qiao, J., Su, L., Zhou, J., Yang, H., Zhang, Q., Zhang, Y., Grezes, C., Amiri, P.K., Wang, K.L.: Giant interfacial perpendicular magnetic anisotropy in mgo/cofe/capping layer structures. *Applied Physics Letters* **110**(7), 072,403 (2017)
19. Qu, Y., Han, J., Cockburn, B.F., Pedrycz, W., Zhang, Y., Zhao, W.: A true random number generator based on parallel STT-MTJs. In: *Design, Automation & Test in Europe Conference & Exhibition*, pp. 606–609 (2017)
20. Soto, J.: *The NIST statistical test suite*. National Institute Of Standards and Technology (2010)
21. Sun, J.Z.: Spin-current interaction with a monodomain magnetic body: A model study. *Phys. Rev. B* **62**, 570–578 (2000)
22. Sun, J.Z., Robertazzi, R.P., Nowak, J., Trouilloud, P.L., Hu, G., Abraham, D.W., Gaidis, M.C., Brown, S.L., O’Sullivan, E.J., Gallagher, W.J., Worledge, D.C.: Effect of subvolume excitation and spin-torque efficiency on magnetic switching. *Phys. Rev. B* **84**, 064,413 (2011)
23. Tomita, H., Miwa, S., Nozaki, T., Yamashita, S., Nagase, T., Nishiyama, K., Kitagawa, E., Yoshikawa, M., Daibou, T., Nagamine, M., Kishi, T., Ikegawa, S., Shimomura, N., Yoda, H., Suzuki, Y.: Unified understanding of both thermally assisted and precessional spin-transfer switching in perpendicularly magnetized giant magnetoresistive nanopillars. *Applied Physics Letters* **102**(4) (2013)
24. Tomita, H., Nozaki, T., Seki, T., Nagase, T., Nishiyama, K., Kitagawa, E., Yoshikawa, M., Daibou, T., Nagamine, M., Kishi, T., Ikegawa, S., Shimomura, N., Yoda, H., Suzuki, Y.: High-speed spin-transfer switching in GMR nano-pillars with perpendicular anisotropy. *IEEE Transactions on Magnetics* **47**(6), 1599–1602 (2011)
25. Uchida, A., Amano, K., Inoue, M., Hirano, K., Naito, S., Someya, H., Oowada, I., Kurashige, T., Shiki, M., Yoshimori, S., et al.: Fast physical random bit generation with chaotic semiconductor lasers. *Nature Photonics* **2**(12), 728 (2008)
26. Wang, M., Cai, W., Cao, K., Zhou, J., Wrona, J., Peng, S., Yang, H., Wei, J., Kang, W., Zhang, Y., Langer, J., Ocker, B., Fert, A., Zhao, W.: Current-induced magnetization switching in atom-thick tungsten engineered perpendicular magnetic tunnel junctions with large tunnel magnetoresistance. *Nature Communications* **9**(671), 1–7 (2018)
27. Wang, Y., Cai, H., d. B. Naviner, L.A., Zhang, Y., Zhao, X., Deng, E., Klein, J.O., Zhao, W.: Compact model of dielectric breakdown in spin-transfer torque magnetic tunnel junction. *IEEE Transactions on Electron Devices* **63**(4), 1762–1767 (2016)
28. Wang, Y., Cai, H., Naviner, L.A.B., Klein, J.O., Yang, J., Zhao, W.: A novel circuit design of true random number generator using magnetic tunnel junction. In: *IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 123–128 (2016)

29. Worledge, D., Hu, G., Abraham, D.W., Sun, J., Trouilloud, P., Nowak, J., Brown, S., Gaidis, M., Osullivan, E., Robertazzi, R.: Spin torque switching of perpendicular ta cofeb MgO-based magnetic tunnel junctions. *Applied Physics Letters* **98**(2), 022,501 (2011)
30. Yang, K., Fick, D., Henry, M.B., Lee, Y., Blaauw, D., Sylvester, D.: 16.3 a 23Mb/s 23pJ/b fully synthesized true-random-number generator in 28nm and 65nm CMOS. In: *Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014 IEEE International, pp. 280–281 (2014)
31. Zhao, H., Zhang, Y., Amiri, P.K., Katine, J.A., Langer, J., Jiang, H., Krivorotov, I.N., Wang, K.L., Wang, J.P.: Spin-torque driven switching probability density function asymmetry. *IEEE Transactions on Magnetics* **48**(11), 3818–3820 (2012)
32. Zhao, W., Moreau, M., Deng, E., Zhang, Y., Portal, J.M., Klein, J.O., Bocquet, M., Aziza, H., Deleruyelle, D., Muller, C., Querlioz, D., Ben Romdhane, N., Ravelosona, D., Chappert, C.: Synchronous non-volatile logic gate design based on resistive switching memories. *Circuits and Systems I: Regular Papers, IEEE Transactions on* **61**(2), 443–454 (2014)

Brain-Inspired Computing



Naoya Onizawa, Warren J. Gross, and Takahiro Hanyu

Abstract This chapter summarizes applications of stochastic computing for brain-inspired computing, which we refer to as Brainware Large-Scale Integration (BLSI). Stochastic computing exploits random bit streams, realizing area-efficient hardware for complicated functions such as multiplication and tanh, as compared with more traditional binary approaches. Using stochastic computing, we have implemented hardware for several physiological models of the primary visual cortex of brains, where these models require such complicated functions. In addition, a deep neural network using stochastic computing has been designed for area/energy-efficient hardware. In order to design BLSIs, we have introduced extended arithmetic functions, such as circular functions. As a design example, our BLSIs are implemented using Taiwan Semiconductor Manufacturing Company (TSMC) 65-nm Complementary Metal Oxide Semiconductor (CMOS) and discussed with traditional fixed-point implementations in terms of hardware performance and computation accuracy.

Keywords Neuromorphic computing · Deep neural networks · Integrated circuits

Introduction

Recently, brain-inspired computing (e.g., spiking neural networks [1] and deep learning [2, 3]) has been studied for highly accurate recognition and classification capabilities, as found in human brains. Several hardware implementations of brain-inspired computing have been presented in [4, 5], but the energy efficiency of the current hardware approaches is significantly lower than that of human brains.

N. Onizawa (✉) · T. Hanyu
Tohoku University, Sendai, Miyagi, Japan
e-mail: naoya.onizawa.a7@tohoku.ac.jp

W. J. Gross
McGill University, Montréal, QC, Canada

In this chapter, we introduce applications of stochastic computing for energy-efficient brainware large-scale integration (BLSI), such as physiological models and deep neural networks.¹ Brainware computing requires complicated functions that can be area-efficiently realized using stochastic computing. However, there were two main issues. The first issue is that several important functions, such as circular functions, were missing. The second issue is that stochastic computing based BLSIs that just applied stochastic computing for brainware computing tended to dissipate more energy than traditional binary implementations. We have addressed these issues for energy-efficient BLSIs summarized in this chapter.

The rest of this chapter is organized as follows. In section “[Overview of Brainware Large-Scale Integration \(BLSI\)](#)”, the basics of stochastic computing are briefly explained and an overview of BLSI is provided. In section “[Extended Arithmetic Functions](#)”, key arithmetic functions for BLSIs are presented. In section “[BLSI Design](#)”, two BLSIs designed using TSMC 65-nm CMOS are introduced as a simple cell model of primary visual cortex in brains [6, 7] and deep neural networks [8]. Section “[Conclusion](#)” concludes this chapter.

Overview of Brainware Large-Scale Integration (BLSI)

Basics of Stochastic Computing

Stochastic computing [9] represents information by a random sequence of bits, called a Bernoulli sequence. It has been exploited for area-efficient hardware implementation, such as low-density parity-check (LDPC) decoders [10–13], image processors [14–16], digital filters [17–19] and Multi-Input Multi-Output (MIMO) decoders [20]. Stochastic computation performs in probabilistic domain that the probabilities are represented by random sequences of bits. The probabilities are calculated by the frequency of ones in the sequence that can be mapped based on unipolar or bipolar coding. For a sequence of bits, $X(t)$, denotes the probability of observing a 1 to be $P_x = P_r(X(t) = 1)$. In unipolar coding, the represented value, x , is $x = P_x$, ($0 \leq x \leq 1$). In bipolar coding, the represented value, x , is $x = (2 \cdot P_x - 1)$, ($-1 \leq x \leq 1$). Input and output probabilities are represented using N_{sto} -bit length streams. In general, stochastic circuits take N_{sto} clock cycles to complete a one-cycle binary computation and hence the computation accuracy depends on N_{sto} .

Stochastic basic arithmetic circuits are multiplier, adder, exponential, and tanh functions briefly summarized in this subsection. A stochastic multiplier in unipolar

¹Since 2014 in BLSI project of Ministry of Education, Culture, Sports, Science and Technology (MEXT) in Japan, we have implemented several BLSIs based on stochastic computing for brain-inspired physiological models and deep neural networks.

coding is designed using a two-input AND gate and that in bipolar coding is designed using a two-input XNOR gate. An addition is realized as a scaled adder designed using a two-input multiplexer, where the selector signal is a random bit sequence.

In addition, hyperbolic tangent and exponential functions are designed using finite state machines (FSMs). At each cycle, the state transits to the right, if the input stochastic bit, $X(t)$, is “1” and the state transits to the left, otherwise. After the transition, the output stochastic bit, $Y(t)$, is determined by the current state. By changing the output condition, different functions can be designed. The stochastic tanh function, Stanh, in bipolar coding is defined as follows:

$$\tanh((N_T/2)x) \approx \text{Stanh}(N_T, x), \quad (1)$$

where N_T is the total number of states. The stochastic exponential function, Sexp, is defined in unipolar coding as follows:

$$\exp(-2Gx) \approx \text{Sexp}(N_E, G, x), \quad (2)$$

where N_E is the total number of states and G determines the number of states generating outputs of “1”. The detailed explanation is summarized in [9].

Application to BLSI

For energy-efficient brain-inspired computing, we have exploited stochastic computing to design BLSIs. The reason to choose stochastic computing is that human brains can perform well under severe noise and errors. Actually, a large-scale neuromorphic chip based on stochastic computing has been reported and performs well with noise [21].

Our stochastic BLSIs are summarized in Fig. 1. This figure shows flows of visual information in human brains from retinas. First, electrical signals (information) from

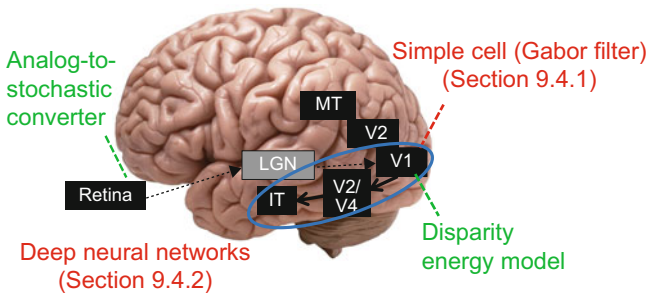


Fig. 1 Brainware LSI (BLSI) design based on stochastic computing

retinas are sent to the primary visual cortex (V1) through the lateral geniculate nucleus (LGN) and then the information are extracted in V1. The extracted information are distributed to two pathways: dorsal pathway to the middle temporal (MT) and ventral pathway to the inferior temporal (IT).

Using stochastic computing, we have designed several BLSIs such as analog-to-stochastic converters [22], 2D Gabor filters [6, 7] and a disparity energy model [23] for brainware visual information processing. The 2D Gabor filters show similar responses of simple cells of V1 and the disparity energy model exhibits the relative depth estimations using two images. Stochastic deep neural networks have been also designed [8] that shows similar responses from V1 to IT. In addition to the visual information processing, stochastic gammatone filters are designed for auditory signal processing [24], where the gammatone filters well express the performance of human auditory peripheral mechanism. Among them, two examples of BLSIs are introduced in section “**BLSI Design**”.

For designing the BLSIs, we have proposed extended arithmetic functions, such as circular functions. These arithmetic functions are summarized in section “**Extended Arithmetic Functions**”.

Extended Arithmetic Functions

Circular Functions

Sine and cosine functions used for Gabor filters were not previously presented in stochastic computing. To realize stochastic Gabor filters, the circular functions have been proposed using several Stanh functions [6]. The stochastic *sin* function, $Ssin(\omega, \lambda, x) (\approx \sin(\omega x))$, in bipolar coding is defined as follows:

$$\begin{aligned} Ssin(\omega, \lambda, x) &= \sum_{k=\lceil -\frac{\omega'}{\pi} \rceil}^{\lfloor \frac{\omega'}{\pi} \rfloor} (-1)^k \text{Stanh}\left(4\omega', \frac{1}{2}\left(\lambda x + \frac{\pi k}{\omega'}\right)\right), \end{aligned} \quad (3)$$

where ω' is a constant angular frequency and λ is ω/ω' . ω' determines the maximum angular frequency supported.

Figure 2 shows a graphical representation of $Ssin$ function using five Stanh functions, where $\omega'=2\pi$ and $\omega=\pi$ are used. In the top figure, five different Stanh functions are geometrically connected to design the approximated *sin* function. Then, λ can be tuned to set ω desired. In addition, the stochastic *cos* function, $Scos(\omega, \lambda, x) (\approx \cos(\omega x))$, is defined as the same way as $Ssin$ as follows:

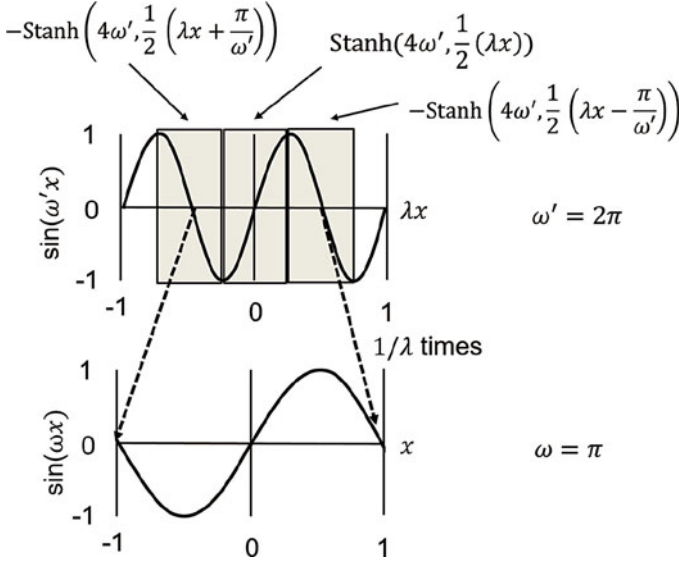


Fig. 2 Graphical representation of Ssin function using five Stanh functions, where $\omega' = 2\pi$ and $\omega = \pi$ are used

$$\begin{aligned}
 & \text{Scos}(\omega, \lambda, x) \\
 &= \sum_{k=\lceil -\frac{\omega'}{\pi} - \frac{1}{2} \rceil}^{\lfloor \frac{\omega'}{\pi} - \frac{1}{2} \rfloor} (-1)^k \text{Stanh}\left(4\omega', \frac{1}{2}\left(\lambda x + \frac{\pi(k + \frac{1}{2})}{\omega'}\right)\right), \tag{4}
 \end{aligned}$$

Improved Exponential-Scale Convolution

Convolution is often used for image filtering. In stochastic computing, a simple implementation of convolution in unipolar coding is illustrated in Fig. 3a, where a_i are the coefficients, x_i are the inputs and z is the output. As a stochastic adder implemented using a multiplier scales down the output by N (the number of inputs), the equation corresponding to Fig. 3a is described as follows:

$$z \simeq \frac{\sum_i a_i x_i}{N}. \tag{5}$$

The computation accuracy of the stochastic adder is lowered when the number of inputs is increased [25].

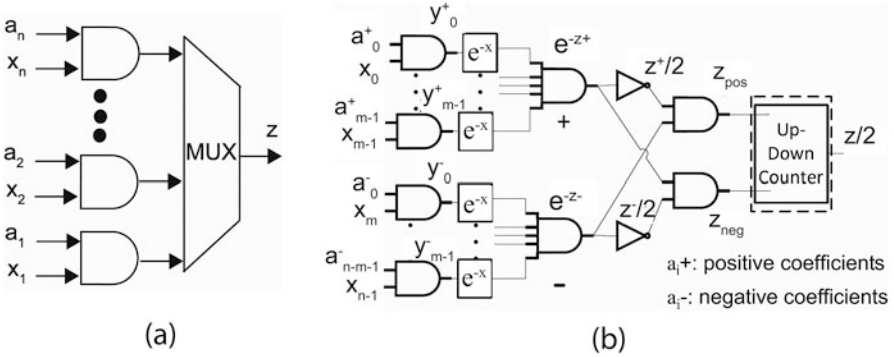


Fig. 3 Convolution architecture: (a) conventional (b) proposed exponential scale

To realize a high accuracy stochastic adder, an exponential-scale stochastic adder is presented [23] as shown in Fig. 3b. First, the input signals are converted to signals in exponential domain. Second, multiplications perform in exponential domain that is equivalent to additions in normal domain. Third, the output signals are converted back to signals in normal domain. This circuit is designed based on [26] by modifying the output block for a higher accuracy. The equation corresponding to Fig. 3b is defined as follows:

$$z \simeq \frac{\sum_i a_i^+ x_i - \sum_i a_i^- x_i}{2}, \quad (6)$$

where a_i^+ are the positive coefficients and a_i^- are the negative coefficients. The scaling factor of the output in the proposed circuit is 2 that is independent of N , leading to a higher computation accuracy than that of the conventional circuit.

Integral Stochastic Computing

Integral stochastic computing [8] is the extension of original stochastic computation, utilizing several bit streams in parallel instead of a single bit stream. It can represent a larger range than $[-1, 1]$ of the original stochastic computing in bipolar coding, leading to a higher computation accuracy with area overhead. In integral stochastic computing, a sequence of integer numbers, $S(t)$, is used to represent a real value, s . $S(t)$ is generated by combining several stochastic bit streams defined as follows:

$$S(t) = \sum_i^m X_i(t), \quad (7)$$

where $X_i(t)$ is a stochastic bit stream and m is the number of bit streams. In unipolar coding, a real value, s , is defined as follows:

$$s = \sum_i^m x_i, \tag{8}$$

where x_i ($0 \leq x_i \leq 1$) is equal to $P_{x_i} = Pr(X_i(t) = 1)$. In bipolar coding, s is defined as follows:

$$s = 2 \sum_i^m x_i - m. \tag{9}$$

s_i can be represented by a range of $[-m, m]$.

Figure 4a shows a two-input integral stochastic adder. The integral stochastic adder is realized using a binary adder that sums up two stochastic bit streams. Unlike the conventional stochastic adder, the output is not scaled down with a high computation accuracy. Figure 4b shows a two-input integral stochastic multiplier. The integral stochastic multiplier requires a binary adder that multiplies two integer values, causing a larger hardware area than that of the original stochastic multiplier. When one of the inputs to the multiplier is a stochastic bit stream, the circuit can be simplified to a bit-wise AND as shown in Fig. 4c.

FSM-based functions in integral stochastic computing are also designed by extending Stanh and Sexp functions described in Eqs. (1) and (2). A tanh function is approximated using an integral stochastic tanh function, NStanh, as follows:

$$\tanh((N_T/2m)x) \simeq NStanh(N_T, m, s). \tag{10}$$

In addition, an exponential function is approximated using an integral stochastic exponential function, NSexp, as follows:

$$\exp(-2(G/m)x) \simeq NSexp(N_E, G, m, s). \tag{11}$$

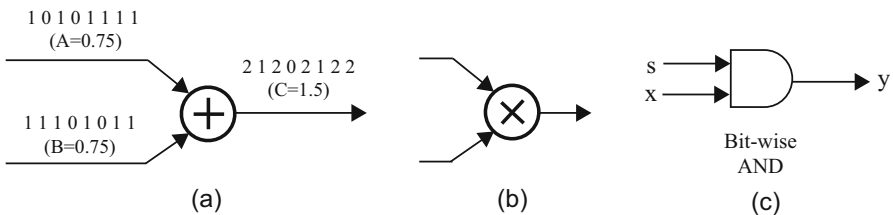


Fig. 4 Integral stochastic circuit components: (a) adder, (b) multiplier, and (c) simplified multiplier when one of two inputs is a stochastic bit stream

BLSI Design

Simple Cell of Primary Visual Cortex (2D Gabor Filter)

2D Gabor filters exhibit similar responses of simple cells in primary visual cortex (V1) of brains as shown in Fig. 5. Many simple cells activated with different spatial frequencies and angles of images are placed as the hypercolumn structure. Based on the hypercolumn structures, brains can extract many different features, such as edges and lines of images used for object recognitions and classifications in the latter part of brains.

Using stochastic computing, an energy-efficient configurable 2D Gabor-filter chip is implemented. 2D Gabor function (odd phase) [27] is defined as follows:

$$g_{\omega,\sigma,\gamma,\theta}(x, y) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \sin(2\omega x'), \tag{12}$$

where $x' = x\cos\theta + y\sin\theta$ and $y' = -x\sin\theta + y\cos\theta$. ω represents the spatial angular frequency of the sinusoidal factor, θ represents the orientation of the normal to the parallel stripes of a Gabor function, σ is the sigma of the Gaussian envelope and γ is the spatial aspect ratio of the Gabor function.

Using Eqs. (2), (3), and (4), the stochastic 2D Gabor function is defined as follows:

$$\begin{aligned} & \text{SGabor}(\omega, \gamma, \lambda, G, \theta, x, y) \\ &= \frac{\text{Sexp}\left(N_E, G, \frac{1}{2}(x'^2 + \gamma^2 y'^2)\right) + 1}{2} \text{Ssin}(\omega, \lambda, x'), \end{aligned} \tag{13}$$

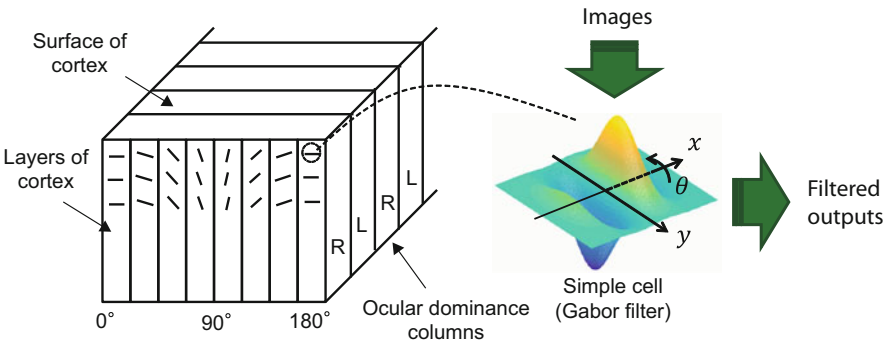


Fig. 5 Hypercolumn structure of primary visual cortex (V1) including many simple cells, where Gabor filters exhibit a similar response to a simple cell

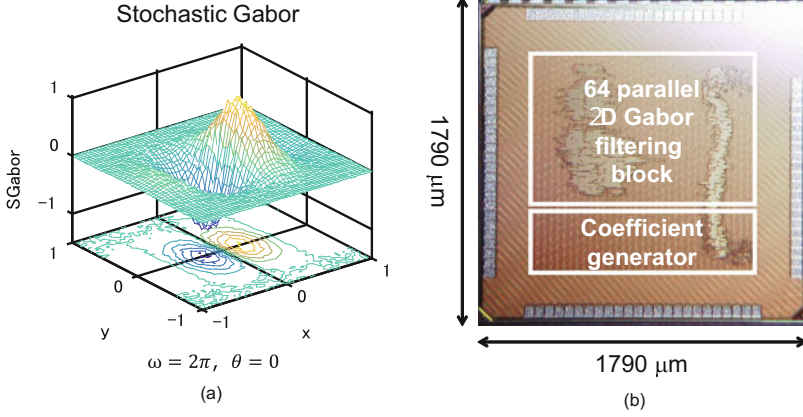


Fig. 6 Stochastic Gabor filter: (a) 51×51 coefficients and (b) chip microphotograph

$$\begin{aligned} x' &= x \text{Scos}(\pi, \lambda_\pi, \theta/\pi) + y \text{Ssin}(\pi, \lambda_\pi, \theta/\pi), \\ y' &= -x \text{Ssin}(\pi, \lambda_\pi, \theta/\pi) + y \text{Scos}(\pi, \lambda_\pi, \theta/\pi), \end{aligned}$$

where λ_π is constant. The original Gabor function on Eq. (12) is approximated as follows:

$$\alpha g_{\omega, \sigma, \gamma, \theta}(x, y) \approx \text{SGabor}(\omega, \gamma, \lambda, G, \theta, x, y), \quad (14)$$

where α is a constant value for fitting SGabor with the original Gabor function. Using Eq. (13), flexible coefficients with flexible kernel sizes are generated in hardware.

Figure 6a shows SGabor results (coefficients) for a kernel size of 51×51 with $\omega = 2\pi$ and $\theta = 0^\circ$. The number of stochastic bits (N_{sto}) is 2^{18} . In this simulation, $N_E = 256$, $G = 8$, $\omega' = 14$, $\lambda_\pi = 0.6614$, and $\gamma = 1$ are selected. $\omega' = 14$ is selected that supports the maximum angular frequency of 4π . $\gamma = 1$ is selected based on [28] that uses the same Gaussian envelope along with x and y .

Figure 6b shows a photomicrograph of the proposed stochastic Gabor filter chip using TSMC 65-nm CMOS technology. The proposed chip includes 64-parallel Gabor-filtering blocks and a coefficient generator based on Eq. (13). The filtering block is designed using a stochastic convolution circuit in unipolar coding. As the coefficients are generated in hardware if necessary, memory blocks are not required, leading to a power-gating capability. The supply voltage is 1.0 V and the area is $1.79 \text{ mm} \times 1.79 \text{ mm}$ including I/Os. The proposed circuit is designed using Verilog HDL (Hardware Description Language) and the test chip is realized using Synopsys Design Compiler and Cadence SoC Encounter.

Table 1 shows performance comparisons with related works. It is hard to compare the performance directly because they are designed with different functionalities

Table 1 Performance comparisons of 2D Gabor filters

	[29]	[30]	[28]	This work
Computation	Analog/Digital	Digital	Digital	Stochastic
Technology	0.35 μm CMOS	(FPGA)	0.13 μm CMOS	65 nm CMOS
Kernel size	(Only nearest neighbor)	3×3	$3 \times 3, 5 \times 5, 7 \times 7$ $9 \times 9, 11 \times 11$	$N \times N$ (flexible)
Kernel parameter	Fixed	Fixed	Flexible	Flexible
Power-gating capability	No	No	No	Yes
# of processing elements	61×72	1	1	64
Throughput (MP/s)	–	124.4 (3×3)	10 (5×5) 2.1 (11×11)	200 (5×5) 40 (11×11)
Frequency (MHz)	1	148.5	250	200
Power dissipation (mW)	800	–	–	102.3
Area (equivalent gate count)	–	–	63.8 k	644 k

and configurations. The memory-based methods [29, 30] use fixed coefficients with fixed kernel sizes, lacking the flexibility. In the conventional configurable Gabor filter [28], COordinate Rotation DIGital Computer (CORDIC) is exploited to dynamically generate the coefficients related to sinusoidal function for flexible Gabor filtering. However, other coefficients are stored in memory, losing the power-gating capability. In contrast, the proposed circuits achieve a higher throughput/area and a more flexible filtering than the conventional configurable Gabor filter with the power-gating capability, leading to zero standby power.

Deep Neural Networks

Recently, deep neural networks based on stochastic computing have been reported for area-efficient hardware [31, 32]. However, the energy dissipation is significantly larger than that of a fixed-point design because a large number of bit streams is required. In order to reduce the energy dissipation, integral stochastic computing has been proposed [8]. Integral stochastic computing can reduce the number of bit streams and hence the energy dissipation while maintaining the computation accuracy with the area overhead.

As a design example of deep neural networks based on stochastic computing, the deep belief network (DBN) is selected as shown in Fig. 7a. The DBN contains 784 input nodes and 10 output nodes with two different configurations of the hidden

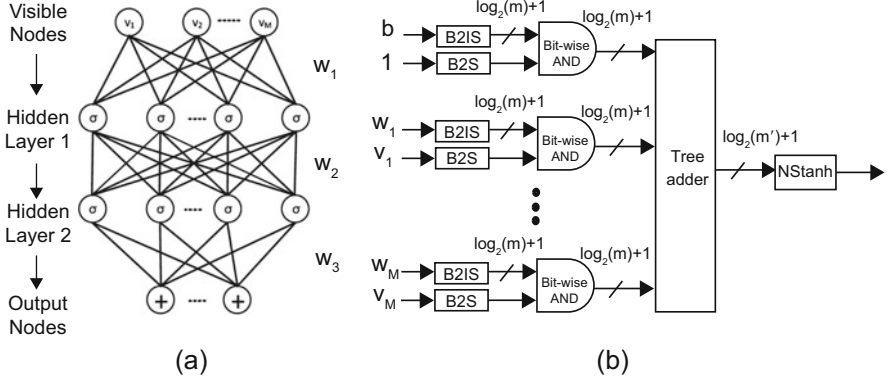


Fig. 7 Stochastic deep neural networks: (a) two-layer deep belief network (DBN) and (b) stochastic neuron

layers. The 1st hidden layer has 100 or 300 neurons and the 2nd hidden layer has 200 or 600 neurons. The function at each neuron is defined as follows:

$$z_j = \sum_{i=1}^M w_{ij} v_i + b_j, \tag{15}$$

$$h_j = \frac{1}{1 + \exp(-z_j)} = \sigma(z_j), \tag{16}$$

where M is the number of inputs, w_{ij} are the weights, v_i are the inputs, b_j is the bias, z_j is the intermediate value and h_j is the output. j is the index of the neuron. The sigmoid function can be replaced by tahn function as follows:

$$\sigma(z_j) = \frac{1 + \tanh(z_j/2)}{2} \tag{17}$$

Based on Eqs. (15), (16), and (17), neurons are designed based on integral stochastic computing as shown in Fig. 7b. First, binary signals are converted to stochastic bit streams using binary-to-stochastic (B2S) converters or integral stochastic bit streams using binary-to-integral stochastic (B2IS) converters. Second, the bit streams are multiplied using the integral stochastic multipliers and then added using the tree adder. Third, the output bit stream of the adder corresponding to z_j are the input of NStanh function based on Eq. (10) in order to determine h_j .

Table 2 shows misclassification rates of floating-point simulations and stochastic computing in the Mixed National Institute of Standards and Technology (MNIST) data set [34]. For training, floating-point simulations are used to obtain w_{ij} in both cases. For inference, 10,000 handwritten digits are tested. In stochastic computing, w_{ij} are represented by 10-bit fixed points that are converted to stochastic bit streams

Table 2 Comparisons of misclassification rates in MNIST

	Misclassification error (%)	
	Floating-point [33]	Stochastic
N_{sto}	—	256
784-100-200-10	2.29	2.33
784-300-600-10	1.82	1.90

Table 3 Hardware evaluation of two-layer DBN using TSMC 65-nm CMOS process

	Fixed-point (10-bit)	Stochastic		
	784-100-200-10	784-100-200-10	784-300-600-10	784-300-600-10
Network configuration	784-100-200-10	784-100-200-10	784-300-600-10	784-300-600-10
Supply voltage (V)	1.0	1.0	1.0	0.8
N_{sto}	—	256	16	22
Misclassification error (%)	2.3	2.33	2.27	2.30
Energy (nJ)	0.380	2.96	0.299	0.256
Gate count (M Gates (NAND2))	23.6	4.2	15.6	15.6
Latency (ns)	30	650	50	65

using B2S. As a result, the stochastic DBN with $N_{sto} = 256$ achieves similar misclassification rates to that of the floating-point simulations.

Table 3 shows the performance comparisons between 10-bit fixed-point and stochastic two-layer DBNs. Both DBNs are designed using Verilog-HDL and synthesized using Cadence RC Compiler. The power dissipation is obtained using Synopsys Power Prime. The technology is TSMC 65-nm CMOS with the frequency of 400 MHz and the supply voltage of 1 V.

In case of the same size of network (784-100-200-10), the hardware area of the stochastic implementation reduces 82.3% in comparison with that of the fixed-point design. However, the energy dissipation is 7.6 times larger because a large $N_{sto} = 256$ is required to achieve the similar misclassification rate. By utilizing the area efficiency, the large size of network (784-300-600-10) is designed using stochastic computing. In this case, the stochastic implementation reduces N_{sto} to 16 while achieving the similar misclassification rate. As a result, the proposed hardware reduces the energy dissipation and the area by 21% and 34%, respectively, in comparison with the fixed-point design.

In order to further reduce the energy dissipation, the supply voltage is dropped to 0.8 V in the stochastic implementation. Lowering the supply voltage generally induces soft errors because of timing errors, however, stochastic computing is robust against soft errors. In case of the supply voltage of 0.8 V, the stochastic implementation achieves the similar misclassification rate by slightly increasing N_{sto} from 16 to 22. As a result, a 33% energy reduction is achieved in total.

Conclusion

In this chapter, we have reported many applications of stochastic computing for BLSI. Several key arithmetic functions for BLSI are presented, such as circular functions and integral stochastic computing. The physiological models in V1 of the human brains and the deep neural networks are implemented in a TSMC 65-nm CMOS. The hardware performance is compared and discussed with that of the fixed-point design with the computation accuracy.

Future prospects include the applications of stochastic computing for models of the higher order visual cortex, larger deep neural networks, and learning algorithms.

Acknowledgements This work was supported by Brainware LSI Project of MEXT and JSPS KAKENHI Grant Number JP16K12494. This work is supported by VLSI Design and Education Center (VDEC), The University of Tokyo with the collaboration with Synopsys Corporation and Cadence Corporation.

References

1. P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, Aug 2014.
2. R. Dechter. Learning while searching in constraint satisfaction problems. In *Proc. AAAI-86, Fifth Nat. Conf. Artif. Intell.*, pages 178–183, 1986.
3. G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.
4. S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H. J. Yoo. 1.93TOPS/W scalable deep learning/inference processor with tetra-parallel mimd architecture for big-data applications. In *2015 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 1–3, Feb 2015.
5. S. Park, I. Hong, J. Park, and H. J. Yoo. An energy-efficient embedded deep neural network processor for high speed visual attention in mobile vision recognition soc. *IEEE Journal of Solid-State Circuits*, 51(10):2380–2388, Oct 2016.
6. N. Onizawa, D. Katagiri, K. Matsumiya, W. J. Gross, and T. Hanyu. Gabor filter based on stochastic computation. *IEEE Signal Processing Letters*, 22(9):1224–1228, Sept. 2015.
7. N. Onizawa, D. Katagiri, K. Matsumiya, W. J. Gross, and T. Hanyu. An accuracy/energy-flexible configurable Gabor-filter chip based on stochastic computation with dynamic voltage-frequency-length scaling. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pages 1–1, 2018 (to appear).
8. A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross. VLSI implementation of deep neural network using integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2588–2599, Oct. 2017.
9. B. D. Brown and H. C. Card. Stochastic neural computation. I. computational elements. *IEEE Transactions on Computers*, 50(9):891–905, Sep 2001.
10. V. C. Gaudet and A. C. Rapley. Iterative decoding using stochastic computation. *Electronics Letters*, 39(3):299–301, Feb. 2003.
11. S. S. Tehrani, W. J. Gross, and S. Mannor. Stochastic decoding of LDPC codes. *IEEE Communications Letters*, 10(10):716–718, Oct. 2006.

12. S. S. Tehrani, S. Mannor, and W. J. Gross. Fully parallel stochastic LDPC decoders. *IEEE Transactions on Signal Processing*, 56(11):5692–5703, Nov. 2008.
13. S. S. Tehrani, A. Naderi, G. A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross. Majority-based tracking forecast memories for stochastic LDPC decoding. *IEEE Transactions on Signal Processing*, 58(9):4883–4896, Sep. 2010.
14. L. Peng and D. J. Lilja. Using stochastic computing to implement digital image processing algorithms. In *29th ICCD*, pages 154–161, Oct 2011.
15. P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel. Computation on stochastic bit streams digital image processing case studies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(3):449–462, Mar. 2014.
16. A. Alaghi, C. Li, and J. P. Hayes. Stochastic circuits for real-time image-processing applications. In *50th DAC*, pages 1–6, May 2013.
17. K. K. Parhi and Y. Liu. Architectures for IIR digital filters using stochastic computing. In *2014 ISCAS*, pages 373–376, June 2014.
18. N. Saraf, K. Bazargan, D. J. Lilja, and M. D. Riedel. IIR filters using stochastic arithmetic. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.
19. Y. Liu and K. K. Parhi. Architectures for recursive digital filters using stochastic computing. *IEEE Transactions on Signal Processing*, 64(14):3705–3718, July 2016.
20. J. Chen, J. Hu, and J. Zhou. Hardware and energy-efficient stochastic LU decomposition scheme for MIMO receivers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(4):1391–1401, April 2016.
21. S. Sato, K. Nemoto, S. Akimoto, M. Kinjo, and K. Nakajima. Implementation of a new neurochip using stochastic logic. *IEEE Transactions on Neural Networks*, 14(5):1122–1127, Sept 2003.
22. N. Onizawa, D. Katagiri, W. J. Gross, and T. Hanyu. Analog-to-stochastic converter using magnetic tunnel junction devices for vision chips. *IEEE Transactions on Nanotechnology*, 15(5):705–714, 2016.
23. K. Boga, F. Leduc-Primeau, N. Onizawa, K. Matsumiya, T. Hanyu, and W. J. Gross. A generalized stochastic implementation of the disparity energy model for depth perception. *Journal of Signal Processing Systems*, 90(5):709–725, May 2018.
24. N. Onizawa, S. Koshita, S. Sakamoto, M. Abe, M. Kawamata, and T. Hanyu. Area/energy-efficient gammatone filters based on stochastic computation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2724–2735, Oct 2017.
25. B. Moons and M. Verhelst. Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 4(4):475–486, Dec 2014.
26. C. L. Janer, J. M. Quero, J. G. Ortega, and L. G. Franquelo. Fully parallel stochastic computation architecture. *IEEE Transactions on Signal Processing*, 44(8):2110–2117, Aug 1996.
27. D. Gabor. Theory of communications. *Journal of Inst. Elect. Eng. - Part III: Radio and Communication Engineering*, 93(26):429–441, Nov. 1946.
28. J.-B. Liu, S. Wang, Y. Li, J. Han, and X.-Y. Zeng. Configurable pipelined Gabor filter implementation for fingerprint image enhancement. In *2010 10th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pages 584–586, Nov 2010.
29. T. Morie, J. Umezawa, and A. Iwata. A pixel-parallel image processor for Gabor filtering based on merged analog/digital architecture. In *Digest of Technical Papers in 2004 Symposium on VLSI Circuits*, pages 212–213, June 2004.
30. E. Cesur, N. Yildiz, and V. Tavsanoğlu. On an improved FPGA implementation of CNN-based Gabor-type filters. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(11):815–819, Nov. 2012.
31. B. Li, M. H. Najafi, and D. J. Lilja. An FPGA implementation of a restricted boltzmann machine classifier using stochastic bit streams. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 68–69, July 2015.

32. K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2016.
33. M. Tanaka and M. Okutomi. A novel inference of a restricted boltzmann machine. In *2014 22nd International Conference on Pattern Recognition*, pages 1526–1531, Aug 2014.
34. Y. Lecun and C. Cortes. The MNIST database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>.

Stochastic Decoding of Error-Correcting Codes



François Leduc-Primeau, Saied Hemati, Vincent C. Gaudet,
and Warren J. Gross

Abstract Error-correction codes (ECC) or *channel* codes allow correcting errors in a data sequence without the need to refer to the original source of information. By allowing reliable data transmission and storage despite the inevitable noise that perturbs physical signals, they dramatically reduce the cost of computer systems, and as such play a central role in enabling the digital revolution that we witness today. However, the decoding of channel codes often represents a significant part of the latency and energy consumption of a receiver. In this chapter, we review methods that can reduce the implementation complexity of ECC decoders by relying on stochastic computing. We focus on an important type of code called low-density parity-check (LDPC) codes, which are capacity-approaching codes that can be decoded using message-passing algorithms. These algorithms are inherently parallel, making LDPC codes attractive for simultaneously achieving a large coding gain and a high data throughput. In addition, we show that the use of stochastic computing is not limited to binary LDPC codes by describing a stochastic-computing decoder for non-binary LDPC codes, and another for Turbo codes.

F. Leduc-Primeau (✉)
École Polytechnique de Montréal, Montréal, QC, Canada
e-mail: francois.leduc-primeau@polymtl.ca

S. Hemati
Department of Electrical and Computer Engineering, University
of Idaho, Moscow, ID, USA
e-mail: shemati@uidaho.edu

V. C. Gaudet
University of Waterloo, Waterloo, ON, Canada
e-mail: vcgaudet@uwaterloo.ca

W. J. Gross
McGill University, Montréal, QC, Canada
e-mail: warren.gross@mcgill.ca

Introduction

Error-correction codes (ECC) or *channel* codes are widely used to improve the efficiency of digital communication and storage systems. They allow to significantly reduce the signal power required to transmit information in a communication system or to increase the amount of stored information in a storage system. Low-density parity-check (LDPC) codes have now been established as one of the leading channel codes for approaching the channel capacity in data storage and communication systems. Notably, they have recently been selected as the channel code for the data channel in the fifth generation of cellular systems. Compared to other codes, they stand out by their ability of being decoded with a message-passing decoding algorithm that offers a large degree of parallelism, which make it possible to simultaneously achieve a large channel coding gain and a high data throughput.

Exploiting all the available parallelism in message-passing decoding is difficult because of the logic area required for replicating processing circuits, but also because of the large number of wires required for exchanging messages. The use of stochastic computing was thus proposed as a way of achieving highly parallel LDPC decoders with a smaller logic and wiring complexity. Furthermore, because the energy efficiency per operation in integrated circuits is now improving much more slowly than in the past, many researchers are looking into approaches that allow trading off the reliability of computations in return for tolerating an increase in manufacturing variability and ultimately obtaining large improvements in energy efficiency [1]. The stochastic nature of the value representation in stochastic computing makes it an interesting paradigm in which to perform such reliability versus energy optimizations. Interestingly, LDPC decoding algorithms are naturally robust to hardware faults, and it was shown that the energy usage of a decoder can be reduced with no performance degradation by operating the circuit in a regime where timing violations can occur [2].

Current decoding algorithms based on stochastic computing do not outperform standard algorithms on all fronts, but they generally offer a significant advantage in average processing throughput normalized to circuit area. Potentially, they could also offer further improvements in robustness to circuit faults. Finally, as discussed in section “[Asynchronous Decoders](#)”, their simplicity combined with the robustness of LDPC decoders makes it possible to envision asynchronous implementations with no signaling overhead, which offers another avenue for tolerating propagation delay variations occurring within the circuit.

We start this chapter by describing several LDPC decoding algorithms that perform computations using the stochastic representation in section “[Stochastic Decoding of LDPC Codes](#)”. One exciting aspect of stochastic computing is its ability to enable new circuit implementation styles that can achieve improved energy efficiency. Section “[Asynchronous Decoders](#)” reviews some work on digital asynchronous implementations of LDPC decoders. Finally, the use of stochastic computing is not limited to the decoding of binary LDPC codes. Section “[Stochastic Decoders for Non-Binary LDPC Codes](#)” presents a stochastic approach for decoding

non-binary LDPC codes, and section “[The Stochastic Turbo Decoder](#)” presents a decoder for Turbo codes based entirely on a stochastic number representation.

Stochastic Decoding of LDPC Codes

LDPC codes are part of the family of linear block codes, which are commonly defined using a parity-check matrix H of size $m \times n$. The codewords corresponding to H are the column vectors \mathbf{x} of length n for which $H \cdot \mathbf{x} = \mathbf{0}$, where $\mathbf{0}$ is the zero vector. LDPC codes can be binary or non-binary. For a binary code, the elements of H and \mathbf{x} are from the Galois Field of order 2, or equivalently $H \in \{0, 1\}^{m \times n}$ and $\mathbf{x} \in \{0, 1\}^n$. Non-binary LDPC codes are defined similarly, but the elements of H and \mathbf{x} are taken from higher order Galois Fields. The rate r of a code expresses the number of information bits contained in the codeword divided by the code length. Assuming H is full rank, we have $r = 1 - \frac{m}{n}$.

A block code can also be equivalently represented as a bipartite graph. We call a node of the first type a *variable* node (VN), and a node of the second type a *check* node (CN). Every row i of H corresponds to a check node c_i , and every column j of H corresponds to a variable node v_j . An edge exists between c_i and v_j if $H(i, j)$ is non-zero.

The key property that distinguishes LDPC codes from other linear block codes is that their parity-check matrix is sparse (or “low density”), in the sense that each row and each column contains a small number of non-zero elements. Furthermore this number does not depend on n . In other words, increasing the code size n also increases the sparsity of H . The number of non-zero elements in a row of H is equal to the number of edges incident on the corresponding check node and is called the check node degree, denoted d_c . Similarly the number of non-zero elements in a column is called the variable node degree and denoted d_v .

LDPC codes can be decoded using a variety of message-passing algorithms that operate by passing messages on the edges of the code graph. These algorithms are interesting because they have a low complexity per codeword bit while also offering a high level of parallelism. If the graph contains no cycles, there exists a message-passing algorithm that yields the maximum-likelihood estimate of each transmitted bit, called the Sum-Product algorithm (SPA) [3]. In practice, all good LDPC codes contain cycles, and in that case the SPA is not guaranteed to generate the optimal estimate of each symbol. Despite this fact, the SPA usually performs very well on graphs with cycles, and experiments have shown that an LDPC code decoded with the SPA can still be used to approach the channel capacity [4]. The SPA can be defined in terms of various likelihood metrics, but when decoding binary codes, the log likelihood ratio (LLR) is preferred because it is better suited to a fixed-point representation and removes the need to perform multiplications. Suppose that p is the probability that the transmitted bit is a 1 (and $1 - p$ the probability that it is a 0). The LLR metric Λ_j is defined as

$$\Lambda_i = \ln \left(\frac{1-p}{p} \right).$$

Algorithm 1 describes the SPA for binary codes (the SPA for non-binary codes is described in section “[Stochastic Decoders for Non-Binary LDPC Codes](#)”). The algorithm takes LLR priors as inputs and outputs an estimate of each codeword bit. If the modulated bits are represented as $x_i \in \{-1, 1\}$ and transmitted over the additive white Gaussian noise channel, the LLR priors Λ_i corresponding to each codeword bit $i \in \{1, 2, \dots, n\}$ are obtained from the channel output y_i using

$$\Lambda_i = \frac{-2y_i}{\sigma^2},$$

where σ^2 is the noise variance. The algorithm operates by passing messages on the code graph. We denote a message passed from a variable node i to a check node j as $\eta_{i,j}$, and from a check node j to a variable node i as $\theta_{j,i}$. Furthermore, for each variable node v_i we define a set V_i that contains all the check node neighbors of v_i , and similarly for each check node c_j , we define a set C_j that contains the variable node neighbors of c_j . The computations can be described by two functions: a variable node function $\text{VAR}(S)$ and a check node function $\text{CHK}(S)$, where S is a set containing the function’s inputs. If we let $S = \{\Lambda_1, \Lambda_2, \dots, \Lambda_d\}$, the functions are defined as follows:

$$\text{VAR}(S) = \sum_{i=1}^d \Lambda_i \tag{1}$$

```

input :  $\{\Lambda_1, \Lambda_2, \dots, \Lambda_n\}$ 
output:  $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n]$ 
begin
   $\theta_{j,i} \leftarrow 0, \forall i, j$ 
  for  $t \leftarrow 1$  to  $L$  do
    for  $i \leftarrow 1$  to  $n$  do // VN to CN messages
      foreach  $j \in V_i$  do
         $\eta_{i,j} \leftarrow \text{VAR}(\{\Lambda_i\} \cup \{\theta_{a,i} : a \in V_i\} \setminus \{\theta_{j,i}\})$ 
      for  $j \leftarrow 1$  to  $m$  do // CN to VN messages
        foreach  $i \in C_j$  do
           $\theta_{j,i} \leftarrow \text{CHK}(\{\eta_{a,j} : a \in C_j\} \setminus \{\eta_{i,j}\})$ 
        for  $i \leftarrow 1$  to  $n$  do // Compute the decision vector
           $\Lambda'_i \leftarrow \text{VAR}(\{\Lambda_i\} \cup \{\theta_{a,i} : a \in V_i\})$ 
          if  $\Lambda'_i \geq 0$  then  $\hat{x}_i \leftarrow 0$ 
          else  $\hat{x}_i \leftarrow 1$ 
        Terminate if  $\hat{\mathbf{x}}$  is a valid codeword
      Declare a decoding failure

```

Algorithm 1: Sum-Product decoding of an LDPC code using LLR messages

$$\text{CHK}(S) = \text{arctanh} \left(\prod_{i=1}^d \tanh(\Lambda_i) \right). \quad (2)$$

The algorithm performs up to L iterations, and stops as soon as the bit estimate vector $\hat{\mathbf{x}}$ forms a valid codeword, that is $H \cdot \hat{\mathbf{x}} = \mathbf{0}$.

A Simple Stochastic Decoder

LDPC decoders have the potential to achieve a high throughput because each of the n codeword bits can be decoded in parallel. However, the length of the codes used in practice is on the order of 10^3 , going up to 10^5 or more. This makes it difficult to make use of all the available parallelism while still respecting circuit area constraints. One factor influencing area utilization is of course the complexity of the VAR and CHK functions to be implemented, but because of the nature of the message-passing algorithm, the wires that carry messages between processing nodes also have a large influence on the area, as was identified early on in one of the first circuit implementations of an SPA LDPC decoder [5].

The need to reduce both logic and wiring complexity suggests that stochastic computing could be a good approach. The use of stochastic computation for the message-passing decoding of block codes was first proposed by Gaudet and Rapley [6]. The idea was prompted by the realization that the two SPA functions VAR and CHK had very simple stochastic implementations when performed in the probability domain. Let us first consider the CHK function. In the LLR domain, the function is given by (2), which in the probability domain becomes

$$\text{CHK}(p_1, p_2, \dots, p_d) = \frac{1 - \prod_{i=1}^d (1 - 2p_i)}{2}. \quad (3)$$

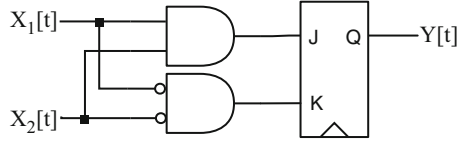
The implementation of this function in the stochastic domain is simply an exclusive-OR (XOR) gate. That is, if we have independent binary random variables X_1, X_2, \dots, X_d , each distributed such that $\Pr(X_i = 1) = p_i$, then taking

$$Y = X_1 + X_2 + \dots + X_d \pmod{2} \quad (4)$$

yields $\Pr(Y = 1) = \text{CHK}(p_1, p_2, \dots, p_d)$. This result is not as surprising as it might seem. Indeed, the modulo-2 sum is exactly the constraint that must be satisfied by the codeword bits involved in this check node operation. Using stochastic streams instead of codeword bits is akin to performing a Monte-Carlo simulation to find the probability associated with an unknown bit connected to this check node.

A circuit for computing the VAR function with 2 inputs was also presented in [6]. In the probability domain, the LLR function of (1) with $d = 2$ is given by

Fig. 1 Stochastic computation circuit for the 2-input VAR function in the probability domain



$$\text{VAR}(p_1, p_2) = \frac{p_1 p_2}{p_1 p_2 + (1 - p_1)(1 - p_2)}. \quad (5)$$

In the stochastic domain, this function can be computed approximately using the circuit shown in Fig. 1. The JK flip-flop becomes 1 if its J input is 1, and 0 if its K input is 1. Otherwise, it retains its previous value. This implementation is different in nature from the one used for the CHK function, since it contains a memory. The behavior of the circuit can be analyzed by modeling the output Y as a Markov chain with states $Y = 0$ and $Y = 1$. Suppose that the stochastic streams $X_1[t]$ and $X_2[t]$ are generated according to the expectation sequences $p_1[t]$ and $p_2[t]$, respectively, and let the initial state be $Y[0] = s_o$. Then, at time $t = 1$, we have

$$\mathbb{E}[Y[1]] = \Pr(Y[1] = 1) = \begin{cases} p_1[1]p_2[1] & \text{if } s_o = 0, \\ p_1[1] + p_2[1] - p_1[1]p_2[1] & \text{if } s_o = 1. \end{cases}$$

None of the expressions above are equal to $\text{VAR}(p_1[1], p_2[1])$, and therefore the expected value of the first output of the circuit is incorrect, irrespective of the starting state. However, if we assume that the input streams are independent and identically distributed (i.i.d.) with $p_1[t] = p_1$ and $p_2[t] = p_2$, it is easy to show that the Markov chain converges to a steady-state such that

$$\lim_{t \rightarrow \infty} \mathbb{E}[Y[t]] = \text{VAR}(p_1, p_2). \quad (6)$$

To build a circuit that will compute the VAR function for more than 2 inputs, we can make use of the fact that the VAR function can be distributed arbitrarily, which can easily be seen by considering the equivalent LLR-domain formulation in (1). For example we have $\text{VAR}(p_1, p_2, p_3) = \text{VAR}(\text{VAR}(p_1, p_2), p_3)$.

Stochastic decoders built using these circuits were demonstrated for very small codes, but they are unable to decode realistic LDPC codes. The reason is that (6) is not sufficient to guarantee the accuracy of the variable node computation, since we do not know that the input streams are stationary or close to stationary. In graphs with cycles, low precision messages can create many fixed points in the decoder's iterative dynamics that would not be there otherwise. This was noted in [7], and the authors proposed to resolve the precision issue by adding an element called a *supernode*, which takes one stochastic stream as input and outputs another stochastic stream. This approach interrupts the feedback path by using a constant expectation parameter to generate the output stochastic stream. Simultaneously, it estimates the mean of the incoming stochastic stream. The decoding is performed

in several iterations, and an iteration is completed once a stochastic stream of length ℓ has been transmitted on every edge of the graph. Once the iteration completes, the expectation parameter in each supernode is updated with the output of the mean estimator. Because the expectation parameter is kept constant while the stochastic streams are being generated, the precision can be increased by increasing ℓ .

While the supernode approach works, it requires large values of ℓ to achieve sufficient precision, and therefore a lot of time for transmitting the ℓ bits in each iteration. However, it is not necessary for the expectation parameter of a stochastic stream to be constant. Any method that can control the rate of change of the expectation sequences will allow avoiding fixed points in the decoding algorithm created by insufficient precision. In particular, this can be achieved by using low-pass filters, some of which are described in the next section.

Stochastic Decoders Using Successive Relaxation

Now that we have explained the basic concepts used to build stochastic decoders, we are ready to present stochastic decoding algorithms that are able to decode practical LDPC codes. Most such algorithms make use of a smoothing mechanism called Successive Relaxation. Message-passing LDPC decoders are iterative algorithms. We can express their iterative progress by defining a vector \mathbf{x}_o of length n containing the information received from the channel, and a second vector $\mathbf{x}[t]$ of length n_e containing all the messages sent from variable nodes to check nodes at iteration t , where n_e is the number of edges in the graph. The standard SPA decoder for an LDPC code is an iterative algorithm that is memoryless, by which we mean that the messages sent on the graph edges at one iteration only depend on the initial condition, and on the messages sent at the previous iteration. As a result, the decoder's progress can be represented as follows:

$$\mathbf{x}[t] = h(\mathbf{x}[t - 1], \mathbf{x}_o),$$

where $h()$ is a function that performs the check node and variable node message updates, as described in Algorithm 1.

In the past, analog circuit implementations of SPA decoders have been considered for essentially the same reasons that motivated the research into stochastic decoders. Since these decoders operate in continuous time, a different approach was needed to simulate their decoding performance. The authors of [8] proposed to simulate continuous-time SPA by using a method called *successive relaxation* (SR). Under SR, the iterative progress of the algorithm becomes

$$\mathbf{x}[t] = (1 - \beta) \cdot \mathbf{x}[t - 1] + \beta \cdot h(\mathbf{x}[t - 1], \mathbf{x}_o), \quad (7)$$

where $0 < \beta \leq 1$ is known as the relaxation factor. As $\beta \rightarrow 0$, the simulated progress of the decoder approaches a continuous-time (analog) decoder. However,

the most interesting aspect of this method is that it can be used not only as a simulator, but also as a decoding algorithm in its own right, usually referred to as Relaxed SPA. Under certain conditions, Relaxed SPA can provide significantly better decoding performance than the standard SPA.

In stochastic decoders, SR cannot be applied directly because the vector of messages $\mathbf{x}[t]$ is a binary vector, while $\mathbf{x}[t]$ obtained using (7) is not if $\beta < 1$. However, if we want to add low-pass filters to a stochastic decoder, we must add memories that can represent the expectation domain of the stochastic streams. Suppose that we associate a state memory with each edge, and group these memories in a vector $\mathbf{s}[t]$ of length n_e . Since the expectation domain is the probability domain, the elements of $\mathbf{s}[t]$ are in the interval $[0, 1]$. Stochastic messages can be generated from the edge states by comparing each edge state to a random threshold. We can then rewrite (7) as a mean tracking filter, where $\mathbf{s}[t]$ is the vector of estimated means after iteration t , and $\mathbf{x}[t]$, $\mathbf{x}_o[t]$ are vectors of stochastic bits:

$$\mathbf{s}[t] = (1 - \beta) \cdot \mathbf{s}[t - 1] + \beta \cdot h(\mathbf{x}[t - 1], \mathbf{x}_o[t - 1]). \quad (8)$$

The value of β controls the rate at which the decoder state can change, and since $\mathbb{E}[\mathbf{x}[t]] = \mathbf{s}[t]$, it also controls the precision of the stochastic representation.

Circuit Implementations of the VN Function

We will first consider stochastic variable node circuits with two inputs $X_1[t]$ and $X_2[t]$. As previously, we denote by $p_1[t]$ and $p_2[t]$ the expectation sequences associated with each input stream. Let E denote the event $X_1[t] = X_2[t]$. We have that

$$\mathbb{E}[X_1[t] | E] = \mathbb{E}[X_2[t] | E] = \text{VAR}(p_1[t], p_2[t]),$$

where $\text{VAR}()$ is defined in (5). Therefore, one way to implement the variable node function for stochastic streams is to track the mean of the streams at the time instants when they are equal. As long as $\Pr(E) > 0$, a mean tracker can be as close as desired to $\text{VAR}(p_1[t], p_2[t])$ if the rate of change of $p_1[t]$, $p_2[t]$ is appropriately limited. If the mean tracker takes the form of (8), this corresponds to choosing a sufficiently small β .

The first use of relaxation in the form of (8) was proposed in [9], where the relaxation (or mean tracking) step is performed in the variable node, by using a variable node circuit that is an extension of the original simple circuit shown in Fig. 1. In the original VN circuit, each graph edge had a corresponding 1-bit flip-flop. This flip-flop can be extended to an ℓ -bit shift-register, in which a ‘1’ is shifted in if both inputs $X_1[t]$ and $X_2[t]$ are equal to 1, and a ‘0’ is shifted in if both inputs are equal to 0. When a new bit is shifted in, the oldest bit is discarded.

Let us denote the number of ‘1’ bits in the shift-register by $w[t]$, and define the current mean estimate as $s[t] = w[t]/\ell$. If we make the simplifying assumptions

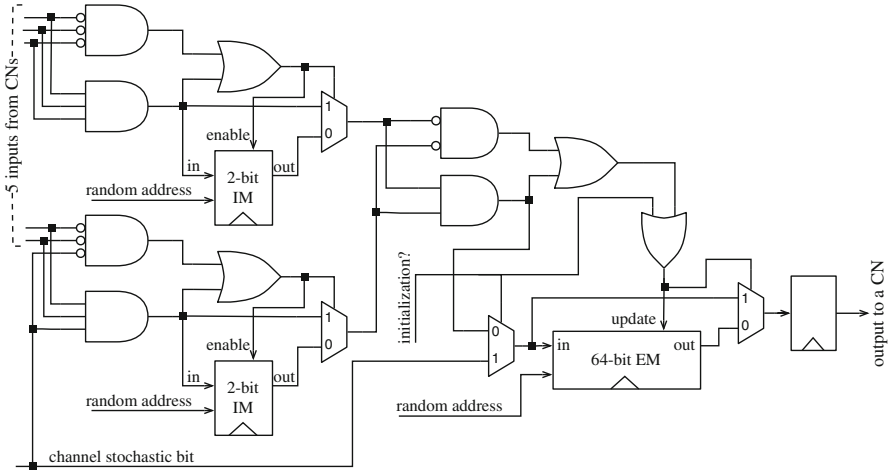


Fig. 2 Degree-6 stochastic VN circuit corresponding to one output [10]

that the bits in the shift register are independent from $X_1[t]$ and $X_2[t]$, and that when a bit is added to the shift register, the bit to be discarded is chosen at random, then it is easy to show that the shift-register implements the successive relaxation rule of (7) in distribution, with $\beta = \text{Pr}(E)/\ell$, in the sense that

$$\mathbb{E}[s[t]] = \left(1 - \frac{\text{Pr}(E)}{\ell}\right) \cdot s[t - 1] + \frac{\text{Pr}(E)}{\ell} \cdot \text{VAR}(p_1[t - 1], p_2[t - 1]).$$

When the variable node degree is large, it was suggested in [10] to implement the variable node function using a computation tree with two levels. Let us denote the computation performed by the first level circuit as VARST_1 and by the second level circuit as VARST_2 . For example, the circuit for a degree-6 VN can be implemented as

$$\text{VARST}(x_o, x_1, \dots, x_5) = \text{VARST}_2(\text{VARST}_1(x_1, x_2, x_3), \text{VARST}_1(x_o, x_4, x_5)),$$

where x_o is the current stochastic bit corresponding to the channel information, and x_1, x_2, \dots, x_5 are the stochastic bits received from the neighboring check nodes. The corresponding circuit is shown in Fig. 2. When using such a two-level implementation, it is proposed in [10] to use small shift-registers for the first level, and a large one for the second level.

Using a shift register is interesting from a pedagogical point of view because it uses a stochastic representation for the mean estimate. However, there are several reasons that discourage its use in decoders. First, the choice of relaxation factor β is tied to the precision of the mean estimate, which prevents from freely optimizing the value of β . Second, because the mean is represented stochastically, storing a high precision estimate requires a lot of register bits, which are costly circuit

components. Lastly, the relaxation factor β is not constant, since $\beta = \Pr(E)/\ell$. This can complicate the analysis and design of the decoder.

For these reasons, a better approach to performing the mean tracking, proposed in [11], is to directly represent the mean estimate $s[t]$ as a fixed-point number. The VN computation with inputs X_1, X_2 can now be implemented as

$$s[t + 1] = \begin{cases} (1 - \beta) \cdot s[t] + \beta & \text{if } X_1 = X_2 = 1, \\ (1 - \beta) \cdot s[t] & \text{if } X_1 = X_2 = 0, \\ s[t] & \text{otherwise.} \end{cases}$$

If β is chosen as a negative power of 2, the update can be implemented with only bit shifts and additions.

Asynchronous Decoders

The vast majority of modern digital circuits designs use a synchronous design approach. In a synchronous circuit, all inputs of logic circuits are only allowed to change at pre-determined moments dictated by a clock signal. This is used to ensure that the inputs are always logically valid, and furthermore prevents the occurrence of race conditions between signals in the presence of feedback paths. The synchronous design approach provides a highly desirable design abstraction that allows to manage the huge complexity of modern systems. However, it is not without costs. First, a clock signal must be distributed throughout the circuit with high accuracy. Second, the processing speed of the system is dictated by the longest possible propagation delay between any two connected clocked memory elements in the systems. This worst-case delay, known as the critical path, can be significantly longer than the average processing delay, especially when possible process, voltage, and temperature variations are taken into account.

Asynchronous circuits have the potential of decreasing the average time required for a computation by using local signals to indicate valid outputs instead of relying on a global clock. According to measurements reported in [12], the delays required to propagate messages between variable and check nodes in a fully parallel stochastic LDPC decoder represent the majority of the delay required to complete a decoding iteration, and this delay varies significantly from one wire to another. The authors of [12] thus propose to use asynchronous signaling to conduct the exchange of messages, which leads to significant speedup in total decoding time.

Because the basic circuits required to build a stochastic LDPC decoder are very simple, it is also worth considering whether their simplicity might allow constructing a decoder circuit that operates without any signaling. In this scheme, it is up to the designer to examine the circuit at the gate and even transistor level to ensure that it is free of harmful glitches or race conditions. This approach was investigated in [13], in which the authors have implemented a clockless version

Fig. 3 A degree-six clockless stochastic check node [13]

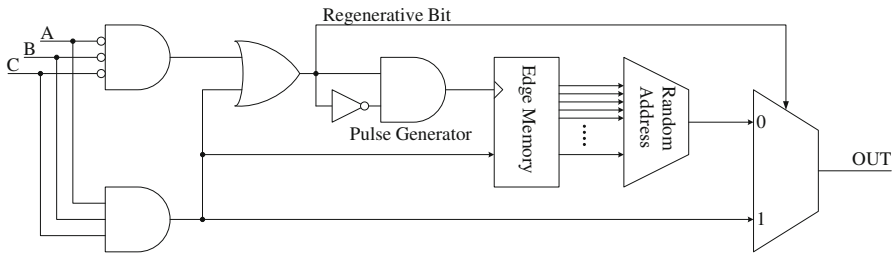
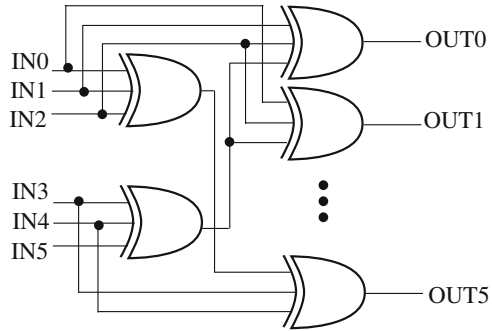


Fig. 4 A 3-input segment of a clockless stochastic variable node [13]

of the fully stochastic decoder presented in section “[Stochastic Decoders Using Successive Relaxation](#)”. A modified check node circuit, shown in Fig. 3, is presented to reduce the difference in propagation delays between the various paths. However, it is easy to see that if multiple inputs experience transitions simultaneously, the circuit has the potential to introduce logically invalid outputs. The same holds for the variable node circuit, shown in Fig. 4. Such a decoder circuit might therefore introduce random errors in the messages. There could however be a beneficial aspect to the introduction of random errors, since it is known that a randomization of the decoding algorithm can help alleviate error floors [14]. Indeed, an improvement of the error floor with respect to synchronous stochastic decoders is reported in [13]. Asynchronous decoders without signaling appear like a promising way to achieve extreme energy efficiency, but more work is required to develop robust design and analysis methodologies that can guarantee predictable results.

Stochastic Decoders for Non-Binary LDPC Codes

Non-binary LDPC codes were shown to outperform their binary counterpart at an equivalent bit length [15], and furthermore are particularly interesting for channels that exhibit bursty error patterns, which are prominent in applications such as data storage and wireless communication. Unfortunately, they are also difficult to

decode. Stochastic computation is one approach that has been explored to reduce the complexity of the decoding algorithm.

In a non-binary code, codeword symbols can take any value from the Galois Field (GF) of order q . The field order is usually chosen as a power of 2, and in that case we denote the power as p , that is $2^p = q$. The information received about a symbol can be expressed as a probability mass function (PMF) that, for each of the q possible values of this symbol, indicates the probability that it was transmitted, given the channel output. For a PMF U , we denote by $U[\gamma]$ the probability corresponding to symbol value $\gamma \in \text{GF}(q)$. Decoding is achieved by passing messages representing PMFs on the graph representation of the code, as in message-passing decoding of binary codes. However, when describing the algorithm, it is convenient to add a third node type called a *permutation* node (PN), which handles part of the computation associated with the parity-check constraint. The permutation nodes are inserted on every edge in the graph, such that any message sent from a VN to a CN or from a CN to a VN passes through a permutation node, resulting in a tripartite graph.

At every decoding iteration, a variable node v receives d_v PMF messages from neighboring permutation nodes. A PMF message sent from v to a permutation node p at iteration t , denoted by $U_{vp}^{(t)}$, is given by

$$U_{vp}^{(t)} = \text{NORM} \left(L_v \times \prod_{p' \neq p} U_{p'v}^{(t-1)} \right), \quad (9)$$

where L_v is the channel PMF, and $\text{NORM}()$ is a function that normalizes the PMF so that all its probabilities sum to 1. A PN p receives a message $U_{vp}^{(t)}$ from a VN and generates a message to a CN c by performing

$$U_{pc}^{(t)}[\gamma h_p] = U_{vp}^{(t)}[\gamma], \quad \forall \gamma \in \text{GF}(q),$$

where h_p is the element of matrix H that corresponds to VN v (which specifies the column) and CN c (which specifies the row). A CN c receives d_c messages from permutation nodes, and generates messages by performing

$$U_{cp}^{(t)} = \underset{p' \neq p}{*} U_{p'c}^{(t)}, \quad (10)$$

where $*$ is the convolution operator. Finally, a message sent by a CN also passes through a PN, but this time the PN performs the inverse operation, given by

$$U_{pv}^{(t)}[\gamma h_p^{-1}] = U_{cp}^{(t)}[\gamma], \quad \forall \gamma \in \text{GF}(q),$$

where h_p^{-1} is such that $h_p \times h_p^{-1} = 1$.

Among the computations described above, the multiplications required in (9) are costly to implement, but (10) has the highest complexity, since the number of operations required scales exponentially in the field order q and in the CN degree d_c .

Several ways of applying stochastic computing to the decoding of non-binary LDPC codes are proposed in [16]. Among these, the only algorithm that can decode any code, without restrictions on q or d_v , is the non-binary version of the Relaxed Half Stochastic (RHS) algorithm. Just like in the binary version presented in [17], the non-binary RHS decoder uses the stochastic representation for the check-node computation only.

A stochastic message sent to a check node consists of a single symbol $X \in \text{GF}(q)$, which can be represented using p bits. In comparison, messages exchanged in the SPA consist of complete PMF vectors requiring $qQ = 2^p Q$ bits, where Q is the number of bits used to represent one probability value in the PMF. Therefore, stochastic algorithms significantly reduce the routing complexity of a circuit implementation.

For binary codes, the stochastic check node function is an addition over $\text{GF}(2)$, which requires $d_c(d_c - 1)$ XOR gates for generating all d_c outputs. For non-binary codes, it is an addition over $\text{GF}(q)$. Assuming that q is a power of two, the addition can still be implemented using only XOR gates, requiring $pd_c(d_c - 1)$ gates to generate all outputs. The stochastic check node function is therefore much less complex than the SPA one. Furthermore, its complexity scales only logarithmically in q .

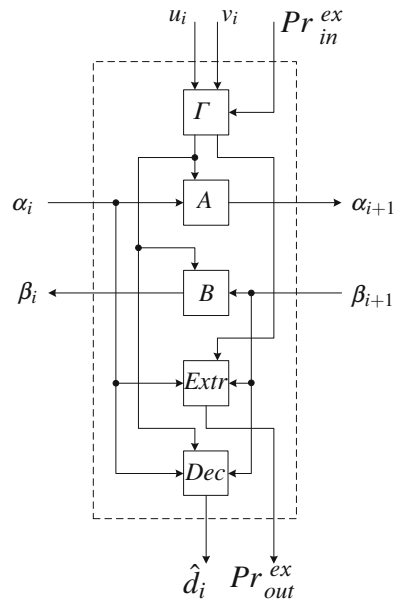
The Stochastic Turbo Decoder

Stochastic decoding was also extended to convolutional and Turbo codes in [18]. Compared to stochastic decoders for LDPC codes, the challenge in implementing a Turbo code decoder using stochastic computing resides in the need to perform additions of probability values, which occur in the *a posteriori* probability (APP) operation performed by each soft-input soft-output (SISO) component decoder. Addition cannot be implemented directly using the stochastic representation, since the stream's expected value must lie in $[0, 1]$. The sum of N streams normalized by a factor of $1/N$ can be implemented by feeding the streams into a N -input multiplexer that randomly selects one of its input with equal probability. However, many useful bits are discarded in the process, which translates into high processing latency.

To improve the precision of the addition, the implementation of [18] uses an addition technique introduced in [19], where the addition is replaced by an exponential transformation, followed by a multiplication, followed by the inverse transformation. According to the results presented, approximating the $\exp(-x)$ and the $-\ln(x)$ functions using the first two terms of their Taylor series, is sufficient to reduce the number of decoding cycles by almost one order of magnitude.

Figure 5 shows a section of a stochastic tail-biting APP decoder in [18] that has multiple inputs and outputs that facilitate exchange of information among sections. The number of sections is equal to the number of symbols to decode and each section consists of four modules. A Γ module receives the channel outputs u_i and v_i , which correspond to the i -th transmitted symbol d_i and its parity bit y_i , respectively. The

Fig. 5 A section of a stochastic tail-biting APP decoder [18]



Γ module converts the received channel outputs into *a priori* probabilities, which are represented by two stochastic sequences to stochastically compute the branch metrics, the forward metrics in the A module, and the backward metrics in the B module. These modules are involved in an iterative process since they use the forward and backward metrics α_i and β_{i+1} from their neighbors and provide them α_{i+1} and β_i . A decision-making module “ Dec ” determines the final value of each binary symbol, \hat{d}_i for the transmitted symbol d_i . In the turbo decoder, the “ $Extr$ ” module computes the output extrinsic probability Pr_{out}^{ex} , which is then used by a Γ module of the second APP decoder as the input Pr_{in}^{ex} . Simulation results showed the performance of stochastic turbo decoder was close to the floating-point Max-Log-MAP decoding algorithm for a few turbo codes [18].

References

1. Rahimi, A., Benini, L., Gupta, R.K.: Variability mitigation in nanometer CMOS integrated systems: A survey of techniques from circuits to software. Proceedings of the IEEE **104**(7), 1410–1448 (2016). <https://doi.org/10.1109/JPROC.2016.2518864>
2. Leduc-Primeau, F., Kschischang, F.R., Gross, W.J.: Modeling and energy optimization of LDPC decoder circuits with timing violations. IEEE Transactions on Communications **66**(3), 932–946 (2018). <https://doi.org/10.1109/TCOMM.2017.2778247>
3. Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Trans. on Information Theory **47**(2), 498–519 (2001)

4. Chung, S.Y., David Forney, J., Richardson, T.J., Urbanke, R.: On the design of low-density parity-check codes within 0.0045dB of the Shannon limit. *IEEE Commun. Lett.* **5**(2), 58–60 (2001)
5. Blanksby, A.J., Howland, C.J.: A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder. *IEEE Journal of Solid-State Circuits* **37**(3) (2002)
6. Gaudet, V., Rapley, A.: Iterative decoding using stochastic computation. *Electronics Letters* **39**(3), 299–301 (2003). <https://doi.org/10.1049/el:20030217>
7. Winstead, C., Gaudet, V.C., Rapley, A., Schlegel, C.B.: Stochastic iterative decoders. In: *International Symposium on Information Theory*, pp. 1116–1120 (2005)
8. Hemati, S., Banihashemi, A.: Dynamics and performance analysis of analog iterative decoding for low-density parity-check (LDPC) codes. *IEEE Trans. on Communications* **54**(1), 61–70 (Jan. 2006). <https://doi.org/10.1109/TCOMM.2005.861668>
9. Sharifi Tehrani, S., Gross, W., Mannor, S.: Stochastic decoding of LDPC codes. *IEEE Communications Letters* **10**(10), 716–718 (2006)
10. Sharifi Tehrani, S., Mannor, S., Gross, W.: Fully parallel stochastic LDPC decoders. *IEEE Trans. on Signal Processing* **56**(11), 5692–5703 (2008). <https://doi.org/10.1109/TSP.2008.929671>
11. Sharifi Tehrani, S., Naderi, A., Kamendje, G.A., Mannor, S., Gross, W.J.: Tracking forecast memories in stochastic decoders. In: *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (2009)
12. Onizawa, N., Gaudet, V.C., Hanyu, T., Gross, W.J.: Asynchronous stochastic decoding of low-density parity-check codes. In: *2012 IEEE 42nd International Symposium on Multiple-Valued Logic*, pp. 92–97 (2012). <https://doi.org/10.1109/ISMVL.2012.35>
13. Onizawa, N., Gross, W.J., Hanyu, T., Gaudet, V.C.: Clockless stochastic decoding of low-density parity-check codes. In: *2012 IEEE Workshop on Signal Processing Systems*, pp. 143–148 (2012). <https://doi.org/10.1109/SiPS.2012.53>
14. Leduc-Primeau, F., Hemati, S., Mannor, S., Gross, W.J.: Dithered belief propagation decoding. *IEEE Trans. on Communications* **60**(8), 2042–2047 (2012). <https://doi.org/10.1109/TCOMM.2012.050812.110115A>
15. Davey, M., MacKay, D.: Low-density parity check codes over $gf(q)$. *Communications Letters, IEEE* **2**(6), 165–167 (1998). <https://doi.org/10.1109/4234.681360>
16. Sarkis, G., Hemati, S., Mannor, S., Gross, W.: Stochastic decoding of LDPC codes over $GF(q)$. *IEEE Trans. on Communications* **61**(3), 939–950 (2013). <https://doi.org/10.1109/TCOMM.2013.012913.110340>
17. Leduc-Primeau, F., Hemati, S., Mannor, S., Gross, W.J.: Relaxed half-stochastic belief propagation. *IEEE Trans. on Communications* **61**(5), 1648–1659 (2013). <https://doi.org/10.1109/TCOMM.2013.021913.120149>
18. Dong, Q.T., Arzel, M., Jego, C., Gross, W.J.: Stochastic decoding of turbo codes. *IEEE Transactions on Signal Processing* **58**(12), 6421–6425 (2010). <https://doi.org/10.1109/TSP.2010.2072924>
19. Janer, C., Quero, J., Ortega, J., Franquelo, L.: Fully parallel stochastic computation architecture. *Signal Processing, IEEE Transactions on* **44**(8), 2110–2117 (1996). <https://doi.org/10.1109/78.533736>