

# **Toward Energy-Efficient, Dynamic Stochastic Computing**

by

Siting Liu

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

Doctor of Philosophy

in

Integrated Circuits and Systems

Department of Electrical and Computer Engineering

University of Alberta

© Siting Liu, 2019

# Abstract

Stochastic computing (SC) is an alternative computing paradigm originally proposed to reduce the size of digital arithmetic circuits. In SC, a number is encoded and represented by a stream of random bits or stochastic sequence (e.g., a Bernoulli sequence). Computations can be performed by bit-wise operations on the bit streams. Due to the long sequences required for accurate computation, however, the long latency and low energy efficiency present significant challenges for SC.

In this work, quasirandom numbers are used to generate the sequences for energy-efficient implementation of SC circuits. Specifically, the quasirandom numbers that lead to the Sobol sequence are introduced for the first time for use in SC. Compared to the use of pseudorandom numbers generated by a linear-feedback shift register (LFSR), using Sobol sequences improves the computation accuracy of a stochastic circuit with a reduced sequence length. A hardware Sobol sequence generator is proposed; its parallelization is implemented with a few extra XOR gates by exploiting the inherent parallelism of the Sobol sequence generation algorithm. In terms of energy consumption, throughput per area and computation time, the circuits that use parallel Sobol sequence generators outperform conventional SC circuits using LFSRs.

To further improve the energy efficiency, dynamic stochastic computing (DSC) is proposed. In DSC, a digital signal is encoded by a stochastic sequence with consistently varying probabilities. This sequence is referred to as a dynamic stochastic sequence (DSS). DSC is then used in digital signal processing (DSP), hardware ordinary differential

equation (ODE) and partial differential equation (PDE) solvers and stochastic computing-based gradient descent circuits (SC-GDCs).

In these applications, each bit in the DSS is used to encode one sample from a signal or function. For DSP, the definition of a DSS is provided; the generation and the reconstruction of a DSS are explicitly discussed and analyzed to obtain the optimal signal generation and reconstruction parameters. Experimental results show that an oversampling is required for DSP applications as in a  $\Delta - \Sigma$  modulator to produce relatively high-quality results. Different DSC circuits are then devised to implement frequency mixing, function estimation, an infinite impulse response (IIR) filter and numerical integration. The simulation results show that up to 60% time and energy savings are achieved using DSC compared to a fixed-point binary implementation with a similar accuracy for function estimation when processing the same oversampled signals. However, a fixed-width binary circuit still has a higher energy efficiency and speed when processing signals sampled at the Nyquist rate compared to a DSC-based frequency mixer using oversampled signals at a similar accuracy. Using Sobol sequences to generate a DSS encoding a continuous signal improves the accuracy of the computed result compared to the use of conventional LFSR-generated sequences.

In the proposed stochastic ODE/PDE solvers using DSS's, we showed that a stochastic integrator produces an unbiased estimate of the Euler solution. Different stochastic ODE solvers are designed for a nonhomogeneous ODE, a set of ODEs and a second-order ODE. Moreover, an array of stochastic Laplacian circuits is proposed to solve a larger-scale problem, i.e., a steady-state heat equation. Each stochastic Laplacian circuit is used to produce the numerical solution for one discretized point in a squared area, as described by one differential equation in the heat equation. Additionally, three error reduction schemes are considered to reduce the variation in the computed result, including the use of Sobol sequences. The simulation results show that these designs achieve a

higher energy and hardware efficiency than their fixed-point binary counterparts with a limited loss of accuracy.

In the SC-GDC, the DSS is used to encode the gradient information of a cost function in a machine learning model. A gradient descent algorithm is then performed by using stochastic integrators to accumulate the gradients. Optimal weights or parameters are then obtained for a particular machine learning model. An array of SC-GDCs is used to update the weights for an adaptive filter, softmax regression and to train a fully connected neural network. The gradient information is encoded by the DSS's, which are then accumulated and converted to a fixed-point number by a stochastic integrator. The SC-GDC achieves a higher or similar accuracy and a significant improvement in hardware efficiency over a conventional SC design and a 16-bit fixed-point implementation.

# Preface

This dissertation presents original work in the field of stochastic computing (SC) by Siting Liu.

In Chapter 3, a parallel Sobol sequence generator is designed and applied to stochastic circuits. I developed all of the VHSIC Hardware Description Language (VHDL) codes for the generator and the parallel stochastic circuits working with the parallel generator. To simulate the circuit functions, I also developed MATLAB codes that implement the exact behaviors of the circuits. The accuracies of the results are obtained by the MATLAB codes and the hardware measurements are obtained from the synthesized results of the VHDL codes. This work appears in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 26 (7): 1326-1339, July 2018, titled “Toward energy-efficient stochastic circuits using parallel Sobol sequences,” by S. Liu and J. Han. Dr. Han provided the original idea of the usage of low-discrepancy sequences in SC and revised the manuscript.

I proposed a new type of stochastic sequence, the dynamic stochastic sequence (DSS), which can encode a consistently changing signal. This work is introduced in Chapter 4. I developed the MATLAB codes for simulating the use of DSS in several digital signal processing applications and numerically analyzed the optimal parameters for achieving a high calculation accuracy. This work has been drafted as S. Liu and J. Han, “Dynamic Stochastic Computing for Digital Signal Processing Applications.” Dr. Han revised the manuscript and provided suggestions to improve the experiments.

Original work on a hardware ordinary differential equation (ODE)/partial differential equation (PDE) solver is presented in Chapter 5. I developed all the VHDL codes for the

proposed ODE solvers, simulated the circuits using MATLAB codes and synthesized the circuits using Synopsis Design Compiler. Part of this work is published as S. Liu and J. Han, “Hardware ODE solvers using stochastic circuits,” in the proceedings of 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC). Dr. Han revised the manuscript and provided suggestions to improve the designs.

Chapter 6 introduces original work on a stochastic computing-based gradient descent circuit (SC-GDC). It performs the gradient descent algorithm and can be used for the efficient training of learning machines. It is published in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(11): 2530-2541, Nov. 2018, as S. Liu, H. Jiang, L. Liu and J. Han, “Gradient Descent Using Stochastic Circuits for Efficient Training of Learning Machines.” I developed the SC-GDC, simulated it in several training algorithms, synthesized it and compared the performance to the other SC designs as well as the fixed-point implementations. Dr. Jiang provided the VHDL code for the adaptive filter circuits using conventional SC and helped develop the signed stochastic number generator. Dr. Duncan Elliott suggested to use saturating counters for both the hardware and software simulations in the thesis. Dr. Han and Dr. L. Liu contributed in the discussions and provided suggestions for improving the manuscript. Dr. Han provided suggestions to improve the design and revised the manuscript.

*To my beloved family*

# Acknowledgements

First, I would like to acknowledge the China Scholarship Council (CSC) and the Natural Sciences and Engineering Research Council (NSERC) of Canada. Without their funding, none of this research work would have been possible. I also appreciate CMC Microsystems for providing electronic design automation (EDA) tools.

I owe special thanks to my supervisor Dr. Jie Han. His consistent pursuit of solid research and academic integrity always inspired me to be true and serious to my research. I also benefitted from the numerous discussions that we had, both academically and personally. His insights and advice helped me a lot during my PhD study and will certainly assist my future career.

I am also very grateful to my supervisory committee members, Dr. Bruce Cockburn and Dr. Duncan Elliott, and my candidacy committee members, Dr. Jie Chen and Dr. Masum Hossain. Thanks for your valuable feedback and comments to my research. I also appreciate Dr. Jason Anderson from the University of Toronto for being my external examiner and providing suggestions to improve my thesis. I would like to thank my co-authors, Dr. Leibo Liu, Dr. Yanzhi Wang and Dr. Fabrizio Lombardi for your valuable suggestions to my research and great collaborations.

I also appreciate very much the supports from my colleagues, Ran Wang, Cong Liu, Peican Zhu, Yidong Liu, Xiaogang Song, Mohammad Saeed Ansari, Anqi Jing, Yuanzhuo Qu, and Francisco Javier Hernandez Santiago. I would also like to thank departmental IT support technicians, Zhigang Liu and Francois Brochu, and all the administrative support staff for their assistance during my study at the University of Alberta.



Last but not least, I would like to give my warmest thanks to my parents for their unconditioned love and support. My greatest thanks to my wife, Dr. Honglan Jiang, for her continued love, company and encouragement.

# Table of Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b>  |
| 1.1      | Motivation . . . . .  | 1         |
| 1.2      | Objective . . . . .   | 6         |
| 1.3      | Dissertation outline . . . . .  | 7         |
| <b>2</b> | <b>Stochastic Computing Systems</b>   | <b>8</b>  |
| 2.1      | Stochastic number representations . . . . .                                       | 8         |
| 2.1.1    | Linear mapping . . . . .  | 9         |
| 2.1.2    | Nonlinear mapping . . . . .   | 9         |
| 2.2      | Stochastic sequence generation . . . . .  | 10        |
| 2.3      | Stochastic circuits . . . . .   | 11        |
| 2.3.1    | Combinational circuits . . . . .  | 11        |
| 2.3.2    | Sequential circuits . . . . .   | 13        |
| 2.4      | Applications of stochastic computing . . . . .                                    | 15        |
| 2.5      | Limitations of stochastic computing . . . . .                                     | 16        |
| <b>3</b> | <b>Toward Energy-Efficient Stochastic Circuits Using Parallel Sobol Sequences</b> | <b>19</b> |
| 3.1      | Introduction . . . . .  | 20        |
| 3.2      | Review . . . . .  | 21        |
| 3.2.1    | Low-discrepancy sequences . . . . .   | 21        |
| 3.2.2    | Sobol sequence generation . . . . .   | 22        |
| 3.3      | Parallel Sobol sequence generator . . . . .                                       | 23        |
| 3.3.1    | Formulation . . . . .   | 24        |
| 3.3.2    | Parallelized Sobol SNG and probability estimator . . . . .                        | 26        |

|          |   |           |
|----------|---|-----------|
| 3.4      | Parallel stochastic circuits . . . . .  | 29        |
| 3.4.1    | Basic computing elements . . . . .  | 29        |
| 3.4.2    | Parallel computing elements . . . . .   | 30        |
| 3.5      | Experiments and results . . . . .   | 35        |
| 3.5.1    | Metrics . . . . .   | 35        |
| 3.5.2    | Evaluation of the proposed Sobol SNG . . . . .  | 36        |
| 3.5.3    | Stochastic combinational circuits . . . . .   | 37        |
| 3.5.4    | Stochastic sequential circuits . . . . .  | 42        |
| 3.6      | Application . . . . .   | 45        |
| 3.6.1    | A sorting network and median filter design . . . . .  | 45        |
| 3.6.2    | Experimental results of the median filter . . . . .   | 47        |
| 3.7      | Summary . . . . .   | 52        |
| <b>4</b> | <b>Dynamic Stochastic Computing for Digital Signal Processing Applications</b>                          | <b>53</b> |
| 4.1      | Introduction . . . . .  | 53        |
| 4.2      | Digital signal processing systems . . . . .   | 54        |
| 4.3      | Proposed dynamic stochastic computing systems . . . . .   | 55        |
| 4.3.1    | Generation of the dynamic stochastic sequence . . . . .   | 56        |
| 4.3.2    | Reconstruction of the DSS . . . . .   | 57        |
| 4.3.3    | Dynamic stochastic computing circuits . . . . .   | 60        |
| 4.4      | DSC-based DSP . . . . .   | 60        |
| 4.4.1    | Frequency mixer . . . . .   | 61        |
| 4.4.2    | Approximation of functions . . . . .  | 61        |
| 4.4.3    | Numerical integration . . . . .   | 63        |
| 4.4.4    | Low-pass IIR filter . . . . .   | 64        |
| 4.5      | Optimization of sequence generation and signal reconstruction . . . . .                                 | 65        |
| 4.5.1    | Optimal sampling rates and reconstruction parameters . . . . .  | 65        |
| 4.5.2    | Pseudorandom sequences vs. quasirandom sequences vs. $\Delta - \Sigma$<br>modulated sequences . . . . . | 68        |
| 4.6      | Hardware efficiency assessment . . . . .  | 69        |
| 4.6.1    | Frequency mixer . . . . .   | 70        |

|          |   |           |
|----------|---|-----------|
| 4.6.2    | Function estimator using multiplexing circuits implementing Bernstein polynomials . . . . .   | 72        |
| 4.7      | Discussion . . . . .  | 75        |
| 4.8      | Summary . . . . .   | 76        |
| <b>5</b> | <b>ODE Solvers Using Stochastic Circuits</b>  | <b>78</b> |
| 5.1      | Introduction . . . . .  | 78        |
| 5.2      | Proposed stochastic ODE solvers . . . . .   | 79        |
| 5.2.1    | Formulation . . . . .   | 79        |
| 5.2.2    | Stochastic ODE solver designs . . . . .   | 82        |
| 5.3      | Error reduction schemes . . . . .   | 84        |
| 5.4      | Experiments and results . . . . .   | 86        |
| 5.4.1    | Validation of the proposed designs . . . . .  | 86        |
| 5.4.2    | Validation of the error reduction schemes . . . . .   | 87        |
| 5.4.3    | Performance evaluation . . . . .  | 88        |
| 5.5      | Accuracy vs. sequence length . . . . .  | 91        |
| 5.6      | Stochastic Laplacian circuit for Laplace's equation . . . . .                                 | 91        |
| 5.6.1    | Proposed stochastic Laplacian circuit design . . . . .  | 94        |
| 5.6.2    | Experiments and results . . . . .   | 95        |
| 5.7      | Summary . . . . .   | 96        |
| <b>6</b> | <b>Gradient Descent Using Stochastic Circuits for Efficient Training of Learning Machines</b> | <b>98</b> |
| 6.1      | Introduction . . . . .  | 99        |
| 6.2      | Background . . . . .  | 100       |
| 6.2.1    | Gradient descent . . . . .  | 100       |
| 6.2.2    | Stochastic integrator . . . . .   | 101       |
| 6.3      | Proposed SC-GDC design . . . . .  | 102       |
| 6.3.1    | SC-GDC circuit design . . . . .   | 102       |
| 6.3.2    | Formulation of SC-GDC . . . . .   | 102       |
| 6.4      | Error analysis . . . . .  | 104       |
| 6.4.1    | Single-step variance for SC-GDC . . . . .   | 104       |

|          |  |            |
|----------|--|------------|
| 6.4.2    | Multiple-step variance bound . . . . .                               | 105        |
| 6.5      | Applications . . . . .   | 105        |
| 6.5.1    | System identification using least-mean-square adaptive filters . . . | 105        |
| 6.5.2    | Handwritten-digit recognition using softmax regression . . . . .     | 107        |
| 6.6      | Experiments and results . . . . .                                    | 109        |
| 6.6.1    | Accuracy evaluation . . . . .  | 109        |
| 6.6.2    | Hardware evaluation . . . . .  | 111        |
| 6.6.3    | Discussion . . . . .   | 114        |
| 6.7      | Signed SC-GDC units training complex NNs . . . . .                   | 115        |
| 6.7.1    | Background for back-propagation . . . . .                            | 115        |
| 6.7.2    | Design of signed SC-GDCs . . . . .                                   | 117        |
| 6.7.3    | Handwritten-digit recognition . . . . .                              | 119        |
| 6.7.4    | Experiments and results . . . . .                                    | 122        |
| 6.7.5    | Related work and discussion . . . . .                                | 123        |
| 6.8      | Summary . . . . .  | 125        |
| <b>7</b> | <b>Conclusion and future work</b>                                    | <b>127</b> |
| 7.1      | Summary . . . . .  | 127        |
| 7.2      | Future work . . . . .  | 128        |
|          | <b>Bibliography</b>  | <b>130</b> |
|          | <b>Appendix A</b>  | <b>147</b> |
|          | <b>Appendix B</b>  | <b>148</b> |
|          | <b>Appendix C</b>  | <b>149</b> |

# List of Tables

|     |   |     |
|-----|---|-----|
| 1.1 | Similarities between the stochastic computing and neuron models. . . . .  | 5   |
| 2.1 | Examples of different SC representations . . . . .  | 10  |
| 3.1 | Truth table of a 4-to-2 priority encoder for LSZ detection . . . . .  | 23  |
| 3.2 | LSZ positions of continuous non-negative integers . . . . .   | 25  |
| 3.3 | Probability distribution of $\sum P_1$ in a parallel stochastic divider. . . . .  | 34  |
| 3.4 | Hardware cost for a $2^m \times$ parallel Sobol SNG . . . . .   | 37  |
| 4.1 | Hardware efficiency evaluation of dynamic and conventional stochastic<br>frequency mixer producing one result . . . . .                               | 71  |
| 4.2 | Hardware efficiency evaluation of DSC and fixed-width frequency mixer<br>processing the same period of signal with different sampling rates . . . . . | 72  |
| 4.3 | Hardware efficiency evaluation of dynamic and conventional stochastic<br>function estimator producing one result . . . . .                            | 74  |
| 5.1 | Probability distribution of $a_i - b_i$ when the same RNG is used to generate<br>$A$ and $B$ . . . . .  | 85  |
| 5.2 | Hardware performance comparison of the ODE solvers . . . . .  | 89  |
| 5.3 | Accuracy comparison of the ODE solvers (RMSE in $10^{-3}$ ) . . . . .   | 90  |
| 6.1 | Probability distribution of $w_{i+1}$ when $y_s$ , $x_s$ and $t_s$ are independently<br>generated. . . . .  | 104 |
| 6.2 | Hardware evaluation of the LMS weight update units. . . . .   | 113 |
| 6.3 | Hardware evaluation of the SR units. . . . .  | 114 |
| 6.4 | The logic of signed stochastic integrator. . . . .  | 118 |

|     |  |     |
|-----|--|-----|
| 6.5 | Hardware evaluation of the signed SC-GDC array training a 784-128-128-10 neural network. . . . . | 124 |
| B.1 | Synthesis results of an SC-GDC training a neural network. . . . .                                | 148 |
| B.2 | Synthesis results of a fixed-point implementation training a neural network.                     | 148 |

# List of Figures

|     |   |    |
|-----|---|----|
| 1.1 | Transistor counts of stochastic circuits without considering SNGs vs. 8-bit binary circuits computing Bernstein polynomials with different orders. The binary circuits are simplified to use the least number of multipliers. . . . .   | 3  |
| 1.2 | Fault tolerance against bit-flips: (a) stochastic circuits vs. (b) conventional binary circuits. Both circuits compute $x_1x_2s + x_3(1 - s)$ . The numbers in the dotted rectangles are the faulty results (adapted from [1]). The error can be larger in the result produced by the binary circuit than the one produced by the SC circuit, especially when there is a bit-flip error on its higher bit. Note that inaccuracy can be resulted from limited sequence length in SC circuits, and it is not considered here. . . . . | 4  |
| 2.1 | An SC system. . . . .   | 8  |
| 2.2 | An SNG. . . . .   | 10 |
| 2.3 | Three stochastic combinational arithmetic circuits. . . . .   | 11 |
| 2.4 | A multiplexing circuit that implements Bernstein polynomials [1]. . . . .   | 12 |
| 2.5 | State transition graph for an FSM-based stochastic circuit. . . . .   | 13 |
| 2.6 | Stochastic integrator and its applications: (a) a stochastic integrator; (b) the symbol of a stochastic integrator; (c) an ADDIE-based stochastic divider; (d) an ADDIE-based adaptive stochastic converter. . . . .  | 14 |
| 2.7 | Components of a stochastic LDPC decoder: (a) a JKFF for information-update; (b) an XOR gate for parity-check. . . . .   | 15 |
| 2.8 | One type of deterministic sequence: (a) an SNG; (b) an AND gate produces the minimum number instead of the product when using two deterministic sequences. . . . .  | 17 |



|      |  |    |
|------|--|----|
| 3.1  | Examples of 2-dimensional (a) Sobol sequence; (b) Halton sequence; and (c) LFSR-based pseudorandom sequences. The LD sequences are more evenly distributed than the pseudorandom sequences. . . . .  | 22 |
| 3.2  | Sobol sequence generation algorithm. . . . .   | 22 |
| 3.3  | (a) A Sobol sequence generator, adapted from [2]. (b) An example is given by using (c) the designated DVA. The example reflects actual hardware operation instead of a mathematical model. . . . .   | 24 |
| 3.4  | LSZ position for the residue class modulo $2^m$ : (a) case 1: the lower $m$ bits are all '1's; (b) case 2: the lower $m$ bits are not all '1's. . . . .  | 25 |
| 3.5  | An example of Lemma 3.3.1. . . . .   | 25 |
| 3.6  | (a) Proposed $2 \times$ parallel Sobol sequence generator. The D-FFs at the final stage are used for recursively generating the Sobol sequence. (b) A working example shows how the generator works. . . . .   | 27 |
| 3.7  | Proposed $4 \times$ parallel Sobol sequence generator. . . . .   | 28 |
| 3.8  | Two uncorrelated Sobol sequences are generated by the same LSZ detection and index generation component, but with different DVAs. . . . .  | 28 |
| 3.9  | A $2^m \times$ parallel Sobol SNG, $i = 0, 1, \dots$ . . . . .   | 29 |
| 3.10 | An APC adapted from [3]. . . . .   | 29 |
| 3.11 | A $4 \times$ unipolar stochastic multiplier. The SNG generates parallel stochastic sequences for the multiplier and multiplicand, opA and opB, respectively. PE stands for a probability estimator. . . . .  | 30 |
| 3.12 | Using Sobol sequences in the FSM-based Stanh circuit produces inaccurate results: (a) the state transition graph of Stanh; (b) the results of $\tanh(3x)$ computed by using Sobol and LFSR-generated sequences. Note that $x$ is encoded by the stochastic sequence X, while $y$ is encoded by the sequence Y. . . . . | 31 |
| 3.13 | A $2 \times$ parallel stochastic divider. . . . .  | 32 |
| 3.14 | Accuracy of the Sobol SNG and LFSR-based SNG. . . . .  | 36 |
| 3.15 | Hardware measurements of parallel Sobol SNGs. . . . .  | 38 |
| 3.16 | Accuracy of combinational stochastic circuits using Sobol, Halton and LFSR-generated sequences. . . . .  | 39 |

|      |  |    |
|------|--|----|
| 3.17 | EPO of stochastic multipliers using $2^{12}$ -bit, $2^{10}$ -bit, $2^8$ -bit, $2^6$ -bit and $2^4$ -bit length from left to the right. . . . .   | 39 |
| 3.18 | TPA of stochastic multipliers using $2^{12}$ -bit, $2^{10}$ -bit, $2^8$ -bit, $2^6$ -bit and $2^4$ -bit length from left to the right. . . . .   | 40 |
| 3.19 | Runtime of stochastic multipliers using $2^{12}$ -bit, $2^{10}$ -bit, $2^8$ -bit, $2^6$ -bit and $2^4$ -bit length from left to the right. . . . .   | 40 |
| 3.20 | Accuracy, EPO, TPA and runtime of an $(n-1)$ th-order Bernstein polynomial circuit using different random sequences with the same sequence length. An $n$ -dimensional random sequence is required to implement an $(n-1)$ th-order Bernstein polynomial circuit. . . . .          | 41 |
| 3.21 | Convergence processes of stochastic dividers using Sobol sequences with $1\times$ , $2\times$ and $4\times$ parallelization, computing $0.8 \div 0.9$ . $y(t)$ is initialized with $y_0 = 0.5$ . The LFSR-based design is also considered for comparison. . . . .                  | 42 |
| 3.22 | Accuracy of stochastic dividers using Sobol sequences with $1\times$ , $2\times$ , $4\times$ and $8\times$ parallelization with different sizes of the up/down counter. The RMSEs for using Halton and LFSR-generated sequences without parallelization are also compared. . . . . | 43 |
| 3.23 | EPO of stochastic dividers using Sobol sequences with $1\times$ , $2\times$ , $4\times$ and $8\times$ parallelization with 12-bit, 10-bit, 8-bit, 6-bit and 4-bit up/down counter from left to the right. The LFSR-based design without parallelization is also shown. . . . .     | 44 |
| 3.24 | TPA of stochastic dividers using Sobol sequences with $1\times$ , $2\times$ , $4\times$ and $8\times$ parallelization with 12-bit, 10-bit, 8-bit, 6-bit and 4-bit up/down counter from left to the right. The LFSR-based design without parallelization is also shown. . . . .     | 44 |
| 3.25 | Runtime of stochastic dividers using Sobol sequences with $1\times$ , $2\times$ , $4\times$ and $8\times$ parallelization with 12-bit, 10-bit, 8-bit, 6-bit and 4-bit up/down counter from left to the right. The LFSR-based design without parallelization is also shown. . . . . | 45 |
| 3.26 | Stochastic sorter circuit. The stochastic sequence encoding a larger value is moved downward. . . . .  | 46 |

|      |  |    |
|------|--|----|
| 3.27 | The basic unit for implementing an MF. . . . .   | 47 |
| 3.28 | Stochastic MF based on the sorting network in Fig. 3.27(c). Since the output stochastic sequences from gates 1, 2, 3 and 4 are not used, they can be removed to save hardware. Each sorter unit is marked by a dotted rectangle. . . . . | 48 |
| 3.29 | Original and corrupted “cameraman” images. . . . .   | 48 |
| 3.30 | Filtering results using the stochastic and binary implementations of an MF. .  | 49 |
| 3.31 | PSNR comparison of different median filter implementations. . . . .  | 50 |
| 3.32 | Distribution of the errors produced by the stochastic sorter. . . . .  | 50 |
| 3.33 | EPO comparison of different median filter implementations. . . . .   | 51 |
| 3.34 | TPA comparison of different median filter implementations. . . . .   | 51 |
| 3.35 | Runtime comparison of different median filter implementations. . . . .   | 52 |
| 4.1  | (a) A typical DSP system; (b) a model of quantization error. . . . .   | 55 |
| 4.2  | (a) Block diagram of a DSM; (b) Z-domain model of the DSM. . . . .   | 56 |
| 4.3  | A DSC system. . . . .  | 56 |
| 4.4  | A DSNG. . . . .  | 57 |
| 4.5  | An MA circuit. . . . .   | 58 |
| 4.6  | An ADDIE. . . . .  | 59 |
| 4.7  | Original and the reconstructed signals by using an ADDIE. . . . .  | 60 |
| 4.8  | A frequency mixer by using a stochastic multiplier with DSS’s as inputs, (a) circuit and (b) mixed output. . . . .   | 62 |
| 4.9  | A summation of several exponential functions computed by (a) a multiplexing circuit and (b) the output results. . . . .  | 63 |
| 4.10 | A single-input bipolar stochastic integrator for numerical integration. . . .  | 63 |
| 4.11 | The input sequence (not shown) encodes the original signal $\cos(\pi t/2)$ . The results produced by the stochastic integrator are close to the analytical results. . . .  | 64 |
| 4.12 | The filtering results in (a) time domain and (b) frequency domain. . . . .   | 65 |

|      |   |    |
|------|---|----|
| 4.13 | Original signal (blue line) and the reconstructed signal (red line) by an MA with different parameters under different sampling rates. Sampling rates from top to bottom rows: $2^4, 2^6, 2^8, 2^{10}$ Hz. MA window lengths from left to right columns: $2^3, 2^4, 2^5, 2^6$ . The SNRs are measured in dB. . . . .  | 66 |
| 4.14 | Original signal (blue) and the reconstructed signal (red) by ADDIEs with different parameters under different sampling rates. Sampling rates from top to bottom: $2^4, 2^6, 2^8, 2^{10}$ Hz. Bit widths of the ADDIE from left to right: 2, 3, 4, 5, including one sign bit. The SNRs are measured in dB. . . .   | 67 |
| 4.15 | DSS's generated and reconstructed using different methods. The first line of the sub-caption indicates how the sequences are generated and the second line indicates how they are reconstructed. The blue line shows the original signal while the red line shows the reconstructed signals. The results in the same row use the same generation method and the results in the same column share the same reconstruction method. The MAs all have a window length of $2^6$ and the ADDIEs all have a width of 5. The SNRs are measured in dB. . . . . | 69 |
| 4.16 | Frequency mixer: DSC vs. CSC vs. fixed-width binary implementation. . .   | 71 |
| 4.17 | A DSC-based frequency mixer using oversampled signals vs. a fixed-width design using signals sampled at the Nyquist rate. . . . .   | 73 |
| 4.18 | Function estimator: DSC vs. CSC vs. fixed-width binary implementation. .  | 74 |
| 4.19 | Spectrum: $\Delta - \Sigma$ modulated vs. dynamic stochastic sequence. . . . .  | 75 |
| 4.20 | $\Delta - \Sigma$ modulated vs. dynamic stochastic sequence for frequency mixer. . . .  | 76 |
| 5.1  | Designs of stochastic ODE solvers. . . . .  | 84 |
| 5.2  | Simulation results of stochastic ODE solvers. Analytical solution vs. hardware results produced by the stochastic ODE solvers for (a) (5.14), (5.15) and (5.16); (b) (5.17) and (c) (5.18). . . . .   | 87 |
| 5.3  | RMSE of $y_1(t)$ and $y_2(t)$ for the stochastic ODE solver in Fig. 5.1(b) under different configurations. . . . .  | 88 |
| 5.4  | Comparison of stochastic and binary ODE solvers with different bit widths.  | 90 |

|      |   |     |
|------|---|-----|
| 5.5  | Stochastic ODE solvers with different sequence lengths that solve (a) (5.16) using Sobol sequences; (b) (5.16) using pseudorandom numbers; (c) (5.17) using Sobol sequences; (d) (5.17) using pseudorandom numbers; (e) (5.18) using Sobol sequences and (f) (5.18) using pseudorandom numbers. . . . . | 92  |
| 5.6  | The 2-dimensional space is divided into grids and the value at each point is solved numerically. Each point is in the center of the grid. . . . .   | 93  |
| 5.7  | Stochastic Laplacian circuits: (a) the circuit diagram of a single element; (b) a symbol and (c) an array of Laplacian circuits solving Laplace's equation (adapted from [4]). . . . .  | 94  |
| 5.8  | Modified stochastic Laplacian circuit using APC. . . . .  | 95  |
| 5.9  | Simulation results for a steady-state heat equation produced by (a) the stochastic Laplacian circuits and (b) MATLAB program using double precision. . . . .  | 96  |
| 6.1  | Proposed unipolar SC-GDC. . . . .   | 102 |
| 6.2  | Circuit design of a stochastic integrator. . . . .  | 103 |
| 6.3  | An AF. . . . .  | 106 |
| 6.4  | LMS weight update unit using SC-GDCs. . . . .   | 106 |
| 6.5  | An SR model. . . . .  | 107 |
| 6.6  | An SC-GDC array for the training of the SR model. . . . .   | 108 |
| 6.7  | System identification results. . . . .  | 110 |
| 6.8  | (a) Recognition accuracy and (b) cross entropy using the SC-GDCs. . . . .   | 111 |
| 6.9  | Fixed-point LMS weight update circuit. . . . .  | 112 |
| 6.10 | Convergence curves in misalignment of the (a) SC-GDC-based and (b) fixed-point LMS weight update units. . . . .   | 112 |
| 6.11 | (a) A multilayer NN. (b) The function of a neuron during FP. . . . .  | 115 |
| 6.12 | (a) The signal flow of local fields during BP. (b) The local field is given by the product of the derivative of the activation function and the weighted sum of local field from the next layer. . . . .  | 116 |
| 6.13 | (a) A signed SNG. (b) A signed stochastic multiplier. (c) A symbol of signed stochastic integrator. . . . .   | 118 |

|      |  |     |
|------|--|-----|
| 6.14 | Proposed signed SC-GDC. The magnitude bit is a unipolar stochastic bit<br>used to encode the absolute value of the number. . . . .         | 119 |
| 6.15 | Proposed signed SC-GDC array for the training of a multilayer NN. . . . .  | 120 |
| 6.16 | (a) Test accuracy and (b) cross entropy produced by the signed SC-GDCs,<br>the bipolar SC-GDCs and the fixed-point implementation. . . . . | 122 |
| 6.17 | (a) Test accuracy and (b) cross entropy produced by the signed SC-GDCs<br>with different widths. . . . .                                   | 123 |

# List of Abbreviations

**ADC** analog-to-digital converter

**ADDIE** ADaptive DIgital Element

**AF** adaptive filter

**APC** accumulative parallel counter

**CNN** convolutional neural network

**CPU** central processing unit

**CSC** conventional SC

**DAC** digital-to-analog converter

**DC** Design Compiler

**DNN** deep neural network

**DSC** dynamic stochastic computing

**DSM**  $\Delta - \Sigma$  modulator

**DSNG** dynamic stochastic number generator

**DSP** digital signal processing

**DSS** dynamic stochastic sequence

**DV** direction vector

**DVA** direction vector array

**EM** electro-migration

**EPO** energy per operation

**ESL** extended stochastic logic

**FA** full adder

**FIR** finite impulse response

**FSM** finite state machine

**GD** gradient descent

**GPU** graphics processing unit

**HCI** hot carrier injection

**IIR** infinite impulse response

**K-Map** Karnaugh Map

**LD** low-discrepancy

**LDPC** low-density parity check

**LFSR** linear-feedback shift register

**LMS** least-mean-square

**LSB** least significant bit

**LSZ** least significant zero

**LUT** lookup table

**MA** moving average

**MAC** multiply-accumulate

**MC** Monte Carlo



**MF** median filter

**MSB** most significant bit

**MSE** mean squared error

**NN** neural network

**ODE** ordinary differential equation

**PDE** partial differential equation

**PE** probability estimator

**PSNR** peak signal-to-noise ratio

**RMSE** root-mean-squared error

**RNG** random number generator

**SC** stochastic computing

**SC-GDC** stochastic computing-based gradient descent circuit

**SNG** stochastic number generator

**SNR** signal-to-noise ratio

**SR** softmax regression

**TDP** thermal design power

**TPA** throughput per area

**VHDL** VHSIC Hardware Description Language

# Chapter 1

## Introduction

### 1.1 Motivation

Moore's Law has successfully predicted the down scaling of transistor sizes for the past five decades, and now billions of transistors are integrated on a single chip within an area of a few hundreds of square millimeters. The scaling of transistors results in a lower power consumption with a higher performance or switching speed [5]. However, as the pursuit of Moore's Law continues, a variety of challenges emerge, such as reliability and variability issues and dark silicon. For example, the reduced critical charge, depending on process changes that can help or worsen the problem, makes a chip more vulnerable to cosmic radiation and subject to bit flips or soft errors [6]. For nanoscale technologies, the threshold and supply voltages do not scale down further, so the power density, which is proportional to the square of the supply voltage and inversely proportional to the chip area, increases as more transistors are integrated into one unit area. As a result, not all the transistors can be simultaneously powered at full performance for a given level of the thermal design power, or else more heat will be generated than that can be dissipated safely.

The reliability, variability and "dark silicon" issues can interact with each other, leading to an increased uncertainty for the operation of transistors and circuits [7, 8]. These present a great challenge to design a reliable and energy-efficient circuit while maintaining a high performance. A variety of techniques have been proposed to deal with these challenges, including the development of new device structures [9, 10], near-threshold computing [11] and various power management and protection strategies [7, 12–14]. However, none of the above methodologies is a panacea to the scaling issues. For example, near-threshold

computing provides high performance with reduced energy consumption and works well for applications with a high thread-level parallelism. However, it is highly sensitive to process variations and power supply fluctuations [12]. Power management and protection strategies have proved to be effective for certain technology nodes, which is not necessarily an indication of continued effectiveness with further scaling.

An alternative approach is to adopt different computing paradigms [15], such as approximate computing with approximate arithmetic circuits [16–18] and stochastic computing [19, 20]. In these computing paradigms, accuracy can be traded for speed, energy efficiency and error tolerance. So they are mostly used in applications where an exact answer is not required, such as image processing and data mining [15]. At a lower hardware cost, there is an increasing interest in building stochastic circuits [21–28] for neuromorphic computing [29] because stochastic circuits show great advantages over conventional computing paradigms for brain-inspired computation-intensive tasks. Stochastic circuits have also been proposed for applications in deep neural networks and convolutional neural networks for lower hardware and power consumption [24–26, 30].

In this thesis, stochastic computing (SC) is studied for its advantages in (1) relatively low hardware cost compared to conventional arithmetic circuits, (2) inherent resilience to soft errors and computation noise, and (3) biological plausibility.

- In SC, some complex computations can be implemented by simple logic circuits. For example, a multiplexing circuit is sufficient for computing a Bernstein polynomial [1], while expensive multipliers and adders are required for conventional binary circuits. A Bernstein polynomial circuit can approximate an arbitrary continuous function with values in the interval  $[0, 1]$ . The hardware cost of the SC circuits and the fixed-point 8-bit binary circuits for computing Bernstein polynomials are compared in Fig. 1.1 by transistor counts. As the sequence generation overhead can be mitigated by different sharing schemes [28, 31, 32], only the computational units are considered for SC circuits to compare the computing density without considering the overhead of stochastic number generators (SNGs) and probability estimators (PEs), which are components for converting a number between stochastic and binary representations. As shown in Fig. 1.1, a stochastic circuit can achieve one hundredth of the hardware cost of its fixed-point binary

counterpart. When the conversion components are considered, SC circuits would still have a smaller area even if those components take 80% of the area of an SC system [19]. Therefore, lower power consumption and higher clock frequencies can be expected for SC.

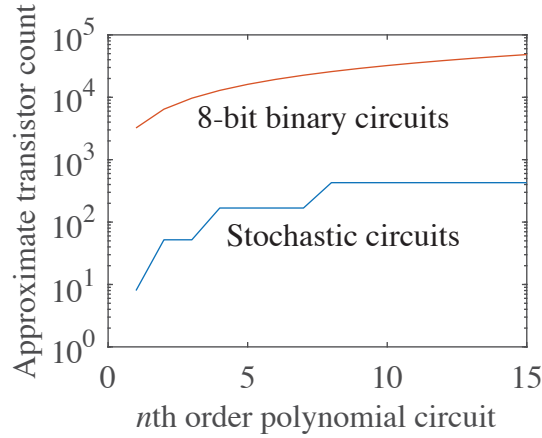


Figure 1.1. Transistor counts of stochastic circuits without considering SNGs vs. 8-bit binary circuits computing Bernstein polynomials with different orders. The binary circuits are simplified to use the least number of multipliers.

- The robustness of SC is reported in [1] and [33]. As shown in Fig. 1.2, when a bit-flip occurs in a circuit, a stochastic circuit can still produce close-enough results whereas the effects of an error in the conventional binary circuit are related to the position of the bit being affected. In other words, SC can tolerate more soft errors in a circuit than conventional arithmetic circuits. However, if a permanent error occurs (e.g., stuck-at errors), it cannot be recovered using either SC or conventional methods.
- Additionally, SC encodes a number in a similar manner to the rate coding in the brain [34]. The similarities between SC and neuron models are summarized in Table 1.1. In the brain, information is carried by neuronal spike trains. In rate coding theory, information is encoded by the number of spikes per unit time, as shown in (b) in the table; whereas in SC, the frequency of ‘1’s is used to encode a number, as shown in (a). The average-over-pool encoding system in the neuron takes multiple spike trains and information is believed to be encoded by the average spikes per unit time [34], as shown in (d) in the table. In this sense, it can be considered to be a parallel SC system, such as a parallel stochastic multiplier shown

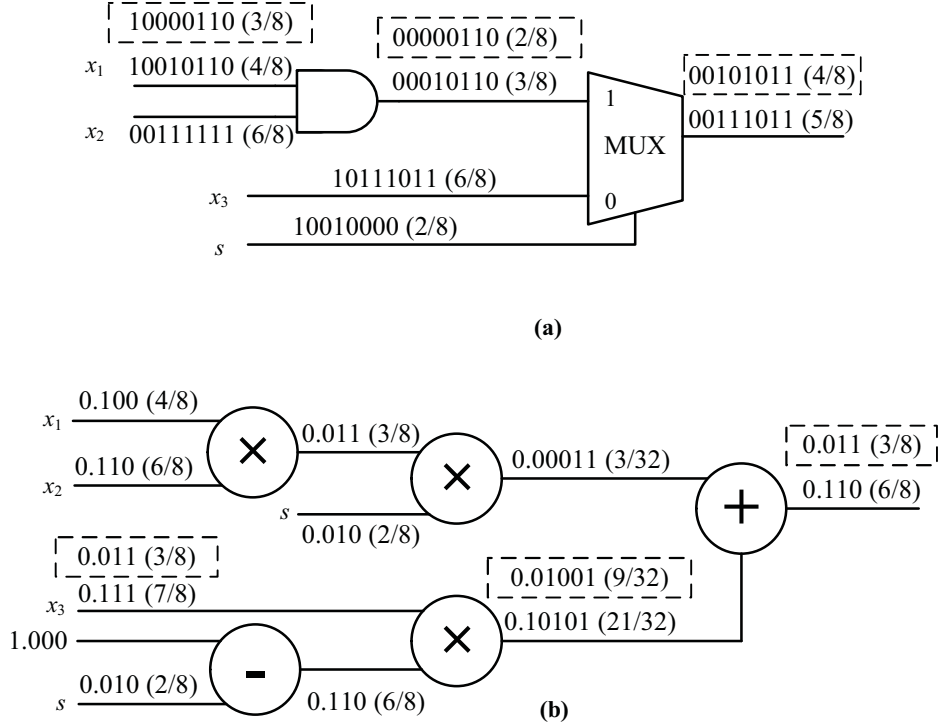
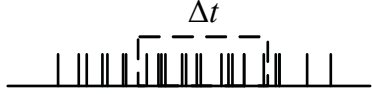
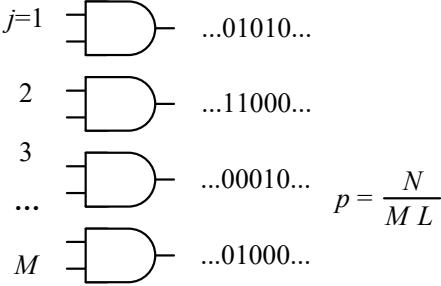
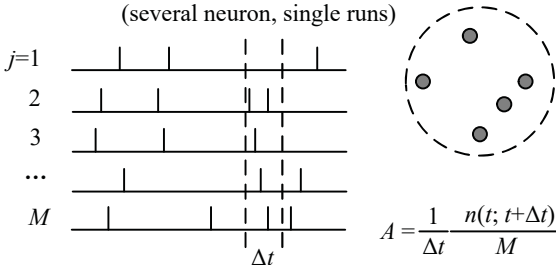
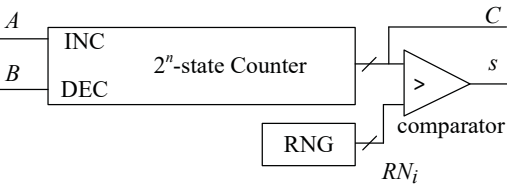
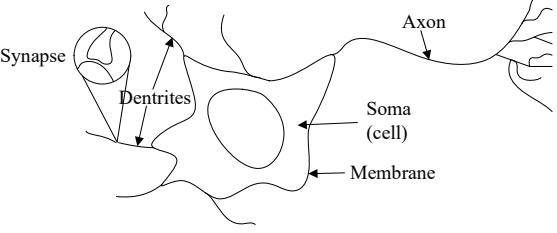


Figure 1.2. Fault tolerance against bit-flips: (a) stochastic circuits vs. (b) conventional binary circuits. Both circuits compute  $x_1x_2s + x_3(1 - s)$ . The numbers in the dotted rectangles are the faulty results (adapted from [1]). The error can be larger in the result produced by the binary circuit than the one produced by the SC circuit, especially when there is a bit-flip error on its higher bit. Note that inaccuracy can be resulted from limited sequence length in SC circuits, and it is not considered here.

in (c) in the table. Moreover, one of the basic stochastic elements, the stochastic integrator ((e) in the table), loosely resembles the integrate-and-fire neuron model in (f) [34]. The stochastic integrator “fires” randomly according to the value stored in the counter; whereas the neuron fires when the accumulated membrane potential reaches a threshold. The stochastic integrator will be discussed in detail in Chapters 4, 5 and 6. The human brain is capable of handling complex tasks with spike trains. Similarly, a neuromorphic computer using SC can potentially be established with low energy consumption and high performance. Finally, insights on how the brain works could possibly be obtained through SC.

However, conventional SC may not be a good candidate to alleviate the dark silicon problem. Firstly, to achieve a high accuracy, a long sequence is often required in conventional SC to encode a number. To fully process the long sequence, it takes a

Table 1.1. Similarities between the stochastic computing and neuron models.

| Stochastic computing   | Neuron models [34]   |
|--|--|
| $\dots 0 \overbrace{[01001010101]}^L 10100 \dots$ $p = \frac{N}{L} \quad p: \text{encoded value}$ <p><math>N</math>: number of '1's in the sequence</p> <p>(a) Stochastic encoding.</p>  |  $A = \frac{n(t; t+\Delta t)}{\Delta t} \quad A: \text{encoded value}$ <p><math>n(\cdot)</math>: number of spikes in the time window</p> <p>(b) Rate coding.</p>   |
|  <p>(c) Parallel stochastic multipliers.</p>  | <p>Rate = average over pool of equivalent neurons<br/>(several neuron, single runs)</p>  $A = \frac{1}{\Delta t} \frac{n(t; t+\Delta t)}{M}$ <p>(d) Average-over-pool.</p>   |
|  $C(t+1) = C(t) + A - B$ $\begin{cases} \text{if } C > RN_i, s = 1 \\ \text{else}, s = 0 \end{cases}$ <p>(e) A stochastic integrator [4].</p> |  $V_M(t+1) = V_M(t) + \sum \epsilon_{ij} \cdot \sum \iota_{ij}$ $\begin{cases} \text{if } V_M > V_{th}, \text{ reset and fire} \\ \text{else}, \text{ do not fire} \end{cases}$ <p><math>\epsilon_{ij}</math>: Excitatory inputs of the neuron<br/> <math>\iota_{ij}</math>: Inhibitory inputs of the neuron<br/> <math>V_M</math>: Membrane potential</p> <p>(f) A spiking neuron model.</p> |

relatively large amount of time and hence energy compared to conventional fixed-point arithmetic circuits [1]. Another reason is that the frequent switchings in a stochastic sequence may make the energy efficiency even lower due to a relatively high activity factor and hence a high power consumption. Also, the costly SNG and PE, which are used to convert a value between a fixed-point number and the corresponding stochastic sequence, consume a major portion of the hardware resources in an SC system [19].

## 1.2 Objective

This work is aimed at addressing the main disadvantages of SC, namely, the long latency and high energy consumption issues. Specifically, the following topics are covered.

- **Design of parallel Sobol sequence generators and parallel stochastic circuits for energy-efficient SC.** To reduce the sequence length, the Sobol sequence is proposed to be used in SC with a high accuracy. To further improve the performance and energy efficiency, parallel Sobol sequence generators and stochastic circuits are developed by exploiting the inherent parallelism in the Sobol sequence generation algorithm.
- **Proposal to use dynamic stochastic computing (DSC) for high-performance and energy-efficient SC-based digital signal processing (DSP).** While it could be inefficient to use a long sequence to encode one number in conventional SC, we note that a stochastic sequence can carry the information of a continuous signal with each bit encoding a sample from the signal. This new type of sequence is referred to as a dynamic stochastic sequence (DSS). In this way, the sequence length encoding each number is 1, thus it significantly improves the performance and energy efficiency of a stochastic circuit. To recover the information from a DSS, two signal reconstruction units are developed.
- **Hardware ordinary differential equation (ODE) solvers using the proposed DSC.** To implement a high-performance and energy-efficient SC-based ODE solver, the stochastic integrators are used to implement the accumulation step in the Euler method. With the derivative function implemented by using the DSS, each estimate is made by processing one bit in the sequence. Laplace's partial differential equation (PDE) is then solved by an array of stochastic Laplacian circuits to show its scalability.
- **Design of a stochastic computing-based gradient descent circuit (SC-GDC) for the efficient training of learning machines.** As machine learning models are becoming more complex, the cost to train the models is growing dramatically. To implement a gradient descent (GD) training algorithm, an SC-GDC is developed to

accumulate the gradient by using the DSS's, which encodes the gradient information. An SC-GDC array is then composed to implement the least-mean-square algorithm of an adaptive filter, a softmax regression and to train a fully connected neural network.

### **1.3 Dissertation outline**

The rest of this thesis is organized as follows. Chapter 2 reviews the basics and current developments of SC. Our work is presented in the remaining chapters. In Chapter 3, the Sobol sequence is introduced and used to generate stochastic sequences; new parallel stochastic circuit designs for using parallel Sobol sequence generators are also proposed. In Chapter 4, the concepts of DSS and DSC are introduced and a DSS is used to encode a digital signal and several DSP applications are implemented using DSC circuits. In Chapter 5, a numerical solution is obtained by using a stochastic integrator to solve an ODE/PDE. DSS is used as the inputs of the stochastic integrators for the ODE/PDE solvers. This design achieves higher energy efficiency and performance over its binary counterpart with a limited loss of accuracy. When the DSS's are used to encode the gradient information, the stochastic integrators are adapted to implement the GD algorithm in Chapter 6 and they are used to train learning machines with a high efficiency. Chapter 7 concludes this thesis and discusses promising directions for future work.



## Chapter 2

# Stochastic Computing Systems

Originally proposed in the 1960s [4, 35], stochastic computing (SC) is intended to be a low-cost alternative to conventional computing. In SC, numbers are encoded by random binary bit streams, which are referred to as stochastic sequences. The probability of each bit being ‘1’ is referred to as the probability of a stochastic sequence. Computation is often performed by using simple logic gates. As shown in Fig. 2.1, a typical SC system consists of three major parts: the stochastic number generators (SNGs) that convert binary numbers into stochastic sequences; the stochastic circuits that carry out the stochastic computation; and the probability estimators (PEs) that convert stochastic sequences back to binary numbers.

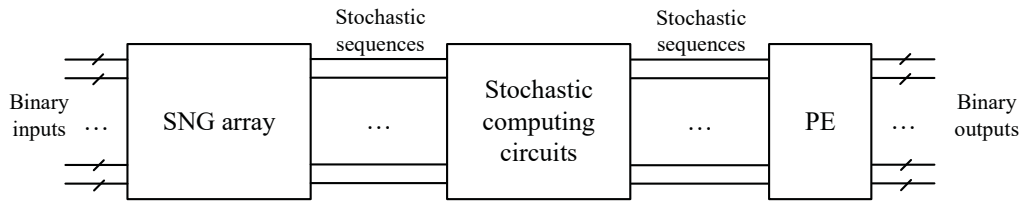


Figure 2.1. An SC system.

### 2.1 Stochastic number representations

In order to represent a real number using the probability of a stochastic sequence, different mapping schemes are used to encode numbers within certain ranges [4]. For the different mapping schemes, the computational elements are different.

### 2.1.1 Linear mapping

Assume that  $x$  is the number to be encoded and that  $p$  is the probability of a stochastic sequence. The simplest strategy is to let  $x = p$ , i.e., the probability of the stochastic sequence is used to represent a number. This is referred to as the unipolar representation. As the value of the probability lies within  $[0, 1]$ , only real numbers within  $[0, 1]$  can be represented by using the unipolar representation. In order to expand the range to include negative values, the bipolar representation takes a linear transformation of the unipolar representation by letting  $x = (p - 0.5) \times 2$ , so that the representation range is  $[-1, 1]$ .

### 2.1.2 Nonlinear mapping

A nonlinear mapping can expand the range of the stochastic representation to infinity. For example, a single-line extended unipolar representation with an infinite range can represent a number in  $[0, +\infty)$ . This can be accomplished by a nonlinear mapping of  $x = p/(1 - p)$ . So a stochastic sequence with a probability of ‘0’ represents ‘0’, whereas a stochastic sequence with a probability of ‘1’ represents  $+\infty$ . Unbounded quantities taking both positive and negative values can be represented by a single-line extended bipolar representation using the mapping strategy of  $x = (2p - 1)/[2p(1 - p)]$  [4].

However, the nonlinear mapping is rarely used due to its complex basic computational elements and the large variations when encoding large values [4]. A double-line representation or extended stochastic logic (ESL) [36] has been proposed to mitigate these problems. The ESL uses two stochastic sequences, in which the ratio of the two bipolar numbers encoded by these stochastic sequences is used to represent an unbounded quantity in  $(-\infty, +\infty)$ . Since the variation of a stochastic sequence is the largest when its probability is 0.5 [37], a bipolar number near zero ( $x_{\max} = 2p_{\max} - 1 = 0$ ) suffers the most from random fluctuations. As a result, the accuracy loss is significant when the “denominator sequence” of the ESL is near zero.

Recently, a sign-magnitude representation was proposed to expand the representation range of the unipolar range by adding an extra sign bit to a stochastic sequence [38]. The same range is achieved as the bipolar representation and the computed results are more accurate when using the sign-magnitude representation. However, it is still considered a

linear mapping since it is a modified unipolar representation. A few examples of the above-mentioned representations are shown in Table 2.1.

Table 2.1. Examples of different SC representations

|                      | Seq. 1 | Seq. 2 | Sign | Value encoded  |
|----------------------|--------|--------|------|--|
| Unipolar             | 10101  | –      | –    | 0.6  |
| Bipolar              | 10001  | –      | –    | $2 \times 0.4 - 1 = -0.2$                                      |
| Sing.-line ext. uni. | 10101  | –      | –    | $0.6 / (1 - 0.6) = 1.5$  |
| Sing.-line ext. bi.  | 01000  | –      | –    | $(2 \times 0.2 - 1) / [2 \times (1 - 0.2) \times 0.2] = -15/8$ |
| ESL                  | 10101  | 01001  | –    | $(2 \times 0.6 - 1) / (2 \times 0.4 - 1) = -1$                 |
| Sign-magnitude       | 10101  | –      | 1    | $-1 \times 0.6 = -0.6$   |

For the rest of this thesis, the unipolar representation by a linear mapping is adopted for its simple circuit implementation and high accuracy, unless stated otherwise.

## 2.2 Stochastic sequence generation

An SNG shown in Fig. 2.2 is used to generate a stochastic sequence. A random number generator (RNG) is conventionally implemented by a linear-feedback shift register (LFSR). A maximum-length  $n$ -bit LFSR traverses all the integer numbers from 1 to  $2^n - 1$  within a period of  $2^n - 1$ , i.e.,  $2^n - 1$  clock cycles. The numbers generated by an LFSR are called pseudorandom numbers because they are deterministic rather than truly random once the seed and the structure of the LFSR are determined. However, due to its statistical characteristics, a pseudorandom number can be approximately considered as a uniformly distributed random number. If the  $n$ -bit fractional number to be encoded,  $x$ , is larger than the  $n$ -bit “uniformly” distributed pseudorandom number, a ‘1’ is generated, otherwise, ‘0’ is the output. Then, the probability of generating a ‘1’ is  $x$ . The detailed description and mathematical model for the LFSR-based SNG is available in [39].

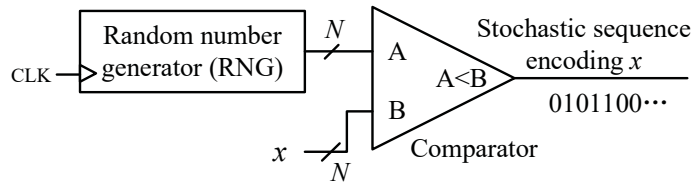


Figure 2.2. An SNG.

## 2.3 Stochastic circuits

Stochastic circuits are the core part of an SC system. There are two major categories of stochastic circuits, combinational and sequential.

### 2.3.1 Combinational circuits

The combinational circuits in an SC system are typically smaller than their fixed-point binary counterparts. For example, in Fig. 2.3(a) and (b), a single AND or an XNOR gate performs unipolar or bipolar multiplication, respectively. In Fig. 2.3(c), a stochastic scaled adder is implemented by a multiplexer.

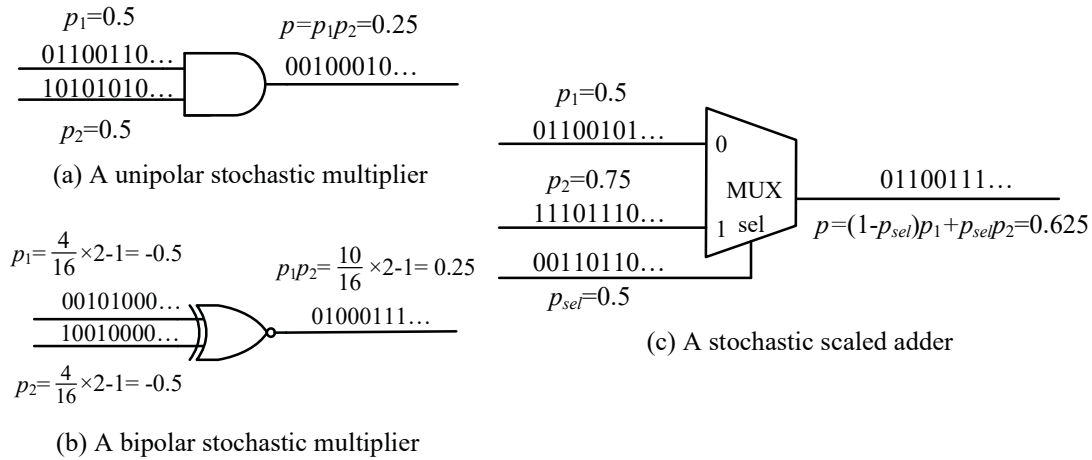


Figure 2.3. Three stochastic combinational arithmetic circuits.

For the stochastic multiplier in Fig. 2.3(a), if and only if both the random bits in the stochastic sequences encoding  $p_1$  and  $p_2$  are '1's, the AND gate produces a '1'. So the probability that a '1' is observed from the output is  $p_1 p_2$  if the stochastic sequences encoding  $p_1$  and  $p_2$  are independently generated. The function of the bipolar multiplier can be obtained similarly. By controlling the probability of the selection signal of the multiplexer,  $p_{sel}$ , in Fig. 2.3(c), the stochastic sequence encoding  $p_2$  has a chance of  $p_{sel}$  being selected, while the top one with a chance  $1 - p_{sel}$ , so that the probability that a '1' is observed from the output is obtained by

$$\begin{aligned}
 p(\text{output} = 1) &= p(\text{output} = 1 | \text{sel} = 0) + p(\text{output} = 1 | \text{sel} = 1) \\
 &= p_1(1 - p_{sel}) + p_2 p_{sel}.
 \end{aligned} \tag{2.1}$$

While it is inconvenient to design single combinational stochastic circuit case by case, a generic method is proposed for synthesizing stochastic circuits that compute Bernstein polynomials by using the multiplexing circuit shown in Fig. 2.4 [1].

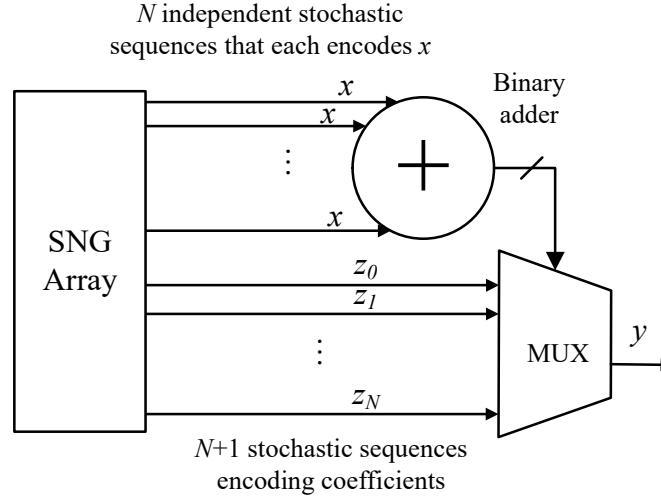


Figure 2.4. A multiplexing circuit that implements Bernstein polynomials [1].

The binary adder is used to count how many ‘1’s are in the input stochastic sequences encoding  $x$ . The sequences encoding  $x$  are independently generated, so the output of the adder follows a binomial distribution, i.e.,  $p(\text{output} = k) = \binom{N}{k} x^k (1-x)^{N-k}$ , ( $k = 0, 1, 2, \dots, N$ ). Therefore, the probability of ‘1’ in the output sequence  $y$  is given by

$$\begin{aligned}
 p(y = 1) &= p(y = 1 | \text{sel} = 0) + p(y = 1 | \text{sel} = 1) + \dots + p(y = 1 | \text{sel} = N) \\
 &= z_0 \binom{N}{0} x^0 (1-x)^N + \dots + z_k \binom{N}{k} x^k (1-x)^{N-k} + \dots + z_N \binom{N}{N} x^N (1-x)^0,
 \end{aligned} \tag{2.2}$$

which is a Bernstein polynomial.

Another synthesizing method for arbitrary multilinear polynomials can be achieved by using the Walsh-Hadamard transform [40]. A multilinear polynomial is linear on all of its variables. It can be a sum of products of multiple variables that all appear with a power of 1. It is shown that the Boolean function of a stochastic circuit can be obtained by performing the Walsh-Hadamard transformation on the coefficients of the multilinear function to be computed. It is also shown that the stochastic combinational circuits can be considered to be Monte Carlo (MC) problems [41].

### 2.3.2 Sequential circuits

There are generally two types of stochastic sequential circuits, finite state machine (FSM)-based and stochastic integrator-based.

For the FSM-based circuits, a state transition graph is shown in Fig. 2.5. The input for the circuit is a stochastic sequence with probability  $x$ . The FSM can be considered as a Markov chain if the input sequence is not autocorrelated, i.e., each bit is generated independently. Let the probability of transitioning from state  $S_i$  to state  $S_j$  be  $p_{ij}$ . Then, a square transition matrix (TM) can be used to describe the Markov chain with  $p_{ij}$  as the element in the  $i$ th column and the  $j$ th row.

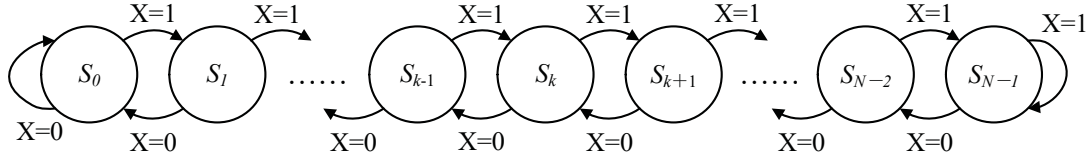


Figure 2.5. State transition graph for an FSM-based stochastic circuit.

As per Fig. 2.5, the TM of an FSM-based circuit is given by

$$\mathbf{P}_{\text{TM}} = \begin{bmatrix} 1-x & x & 0 & 0 & \dots & 0 \\ 1-x & 0 & x & 0 & \dots & 0 \\ 0 & 1-x & 0 & x & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 1-x & 0 & x \\ 0 & 0 & \dots & 0 & 1-x & x \end{bmatrix}. \quad (2.3)$$

According to (2.3), the  $i$ th step probability distribution of the states,  $\mathbf{P}_i$ , is given by

$$\mathbf{P}_i = \mathbf{P}_{\text{TM}} \mathbf{P}_{i-1}, \quad (2.4)$$

where  $\mathbf{P}_i$  is a column vector where each element  $P_{i,k}$  is the probability that the FSM is in state  $k$  and  $\sum_{k=0}^{N-1} P_{i,k} = 1$ . The stationary state of the FSM-based stochastic circuit can be obtained by [42]

$$\lim_{i \rightarrow \infty} \mathbf{P}_i = \lim_{i \rightarrow \infty} (\mathbf{P}_{\text{TM}} \mathbf{P}_{i-1}). \quad (2.5)$$

The solution of (2.5) is

$$P_k(x) = \frac{\left(\frac{x}{1-x}\right)^k}{\sum_{j=0}^{N-1} \left(\frac{x}{1-x}\right)^j}, \quad (2.6)$$

where  $P_k(x)$  stands for the probability of being in state  $S_k$ , given an input stochastic sequence with probability of  $x$ . According to (2.6), by assigning different outputs (0 or 1) to each state, linear gain, exponential, hyperbolic and absolute value functions can be approximated by the FSM-based circuits. A general synthesis method is discussed in [43]. Although it can produce stochastic circuits with lower hardware cost compared to the method in [1], it still requires long stochastic sequences to achieve a high accuracy.

In [4], stochastic integrator-based circuits are proposed to perform the integration of two stochastic sequences by using an up/down counter, as shown in Fig. 2.6(a). The RNG and comparator are used to generate the output sequence that encodes the number stored in the counter. The stochastic integrators with feedback are called ADaptive DIgital Elements (ADDIEs). In an ADDIE, when the counter in the stochastic integrator has an equal probability for increment and decrement, the number stored in the counter reaches an equilibrium state. When an equilibrium state is established, the value encoded in the counter in Fig. 2.6(c) is  $A/B$ , and so this circuit implements a stochastic divider. Similarly, the circuit in Fig. 2.6(d) computes  $B/(A+B)$ . The circuit in Fig. 2.6(d) has a similar behavior to the adaptation of human eyes when exposed to either dark or light environments [4]. Unfortunately, both circuits based on ADDIE require a relatively long “warm-up” phase before the counters reach their equilibrium states [44].

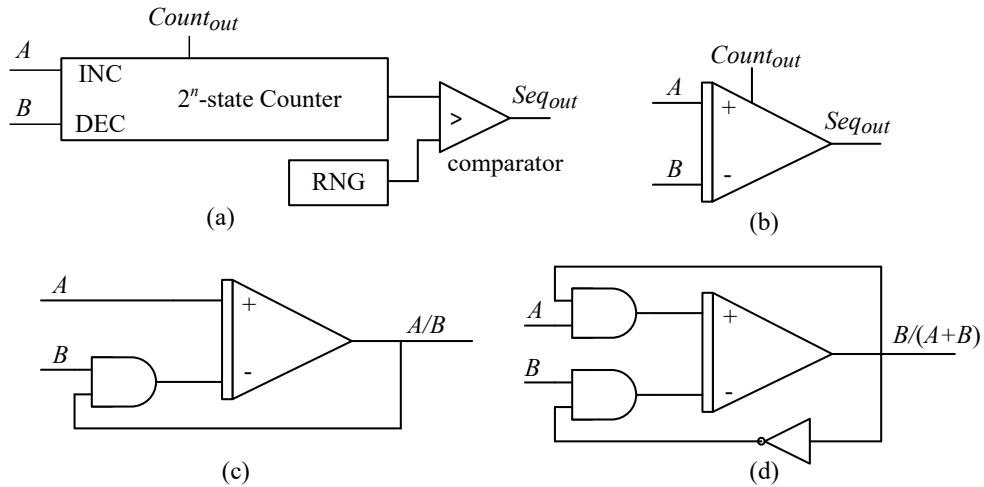


Figure 2.6. Stochastic integrator and its applications: (a) a stochastic integrator; (b) the symbol of a stochastic integrator; (c) an ADDIE-based stochastic divider; (d) an ADDIE-based adaptive stochastic converter.

## 2.4 Applications of stochastic computing

Most stochastic circuits have a disadvantage in accuracy due to their randomness, so they are mainly used in inherently fault-tolerant applications such as low-density parity check (LDPC) decoders [45–47], LU decomposition [48], control systems [49–51], signal processing [52–58], image processing [59–62], machine learning [22, 24–26, 28, 36, 50, 63–74] and invertible logic [75]. They are also used in reliability evaluation and biological models [37, 76–79]. Recently, emerging devices such as spin-based devices [80–85] and memristors [86, 87] have been used for SC, leveraging the inherent randomness in the devices.

In the stochastic LDPC decoder design, the sum-product algorithm (SPA) can be implemented by simple logic gates: the information is updated by using J-K flip-flops (JKFFs) and the parity block is implemented by XOR gates, as shown in Fig. 2.7. The function of a JKFF in the stochastic domain is obtained by using the TM discussed in the previous section and it computes the update function  $C = AB / (AB + (1 - A)(1 - B))$ . The XOR gate computes  $C = (1 - A)B + (1 - B)A$ .

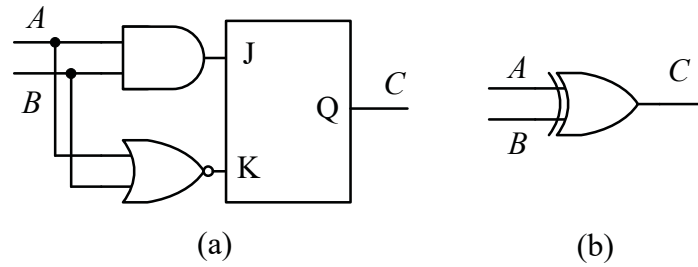


Figure 2.7. Components of a stochastic LDPC decoder: (a) a JKFF for information-update; (b) an XOR gate for parity-check.

When SC is used to implement a finite impulse response (FIR) or an infinite impulse response (IIR) filter [55, 58, 88], stochastic circuits, including the stochastic multiplier and scaled adder, are used to implement the multiply-accumulate (MAC) computation for the filter. Different strategies are adopted to improve the accuracy or the energy efficiency of the MAC operation. In [89], the transfer function of an ADDIE indicates that it is an IIR without using stochastic multipliers or adders.



In [60, 61], several specific image processing algorithms are realized by different stochastic circuits. In [61], edge detection, a median filter, contrast stretching and image segmentation are implemented by a multiplexer circuit, an FSM-based hyperbolic tangent function, an FSM-based stochastic linear-gain function and a multiplexer tree, respectively. The experimental results show that the stochastic implementations are extremely tolerant to soft errors compared to conventional implementations. In [60], the stochastic implementations are considered for real-time image processing applications and an analog comparator is used to generate the stochastic sequence from a sampled signal. An edge-detection circuit is considered as an example and progressive precision is applied to produce acceptable results with a reduced sequence length.

Inspired by the neuronal coding scheme, SC has been used to implement neural networks (NNs). All the computation required for an NN can be achieved in the stochastic domain so that no conversion is required between stochastic sequences and binary numbers, which could lead to hardware-efficient designs. The required stochastic components for implementing an NN are summarized in [22]. In [25], approximate parallel counter-based and OR gate-based inner product blocks are explored for implementing deep convolutional neural networks. In [26], integral SC is used to deal with numbers beyond  $[0, 1]$  and the stochastic circuits using integral stochastic sequences are proposed. In [24], several techniques are employed to dynamically obtain a trade-off between accuracy and energy, including near-zero weight removal, using accumulation-based activation functions and early decision termination.

## 2.5 Limitations of stochastic computing

In the conventional stochastic circuits, long stochastic sequences are required to achieve high accuracy, which leads to a high latency. It is reported in multiple sources that at least  $2^8$ -bit length sequences (or  $2^8$  clock cycles) are required to produce an acceptable quality for use in machine learning [22, 25, 26, 66, 90] and sequence lengths of at least  $2^7$  are required for image processing tasks [1, 60]. The high latency also leads to an inferior energy efficiency when compared to their conventional binary counterparts. To improve the accuracy, effort has been made to understand and reduce the impact of random

fluctuations in SC. In [91–93], correlations, autocorrelations and their impacts on SC are discussed by using probability transfer matrices. It has been found that, sometimes, the correlations can be exploited to achieve better accuracy. The correlations have also been used to build hardware-efficient dividers [94–96]. The error and variance propagation through multiple stages of stochastic circuits are studied in [97] by using variance transfer functions. In [98], a framework is developed for SC circuits to calculate the numerical deviation of the computed value with a confidence level. However, most of them provide only theoretical estimation of the error based on true random sequences, whereas in reality, the stochastic sequences are typically generated by using pseudorandom sequences. In [99], seed selection and sequence scrambling are discussed for LFSR-based SNGs to improve the accuracy; however, a selected seed and a specific permutation of a pseudorandom sequence only work for a single case. Besides, although the randomness in a Bernoulli sequence minimizes signal correlation, it leads to a rather wide distribution of the value encoded in the output stochastic sequence. A non-Bernoulli sequence is used to reduce the randomness in SC by using a fixed number of ‘1’s for reliability evaluation in [37].

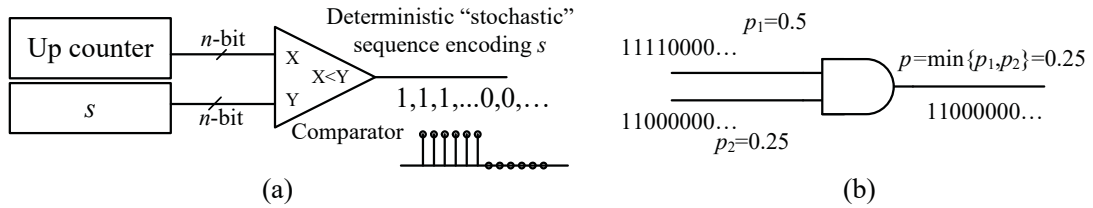


Figure 2.8. One type of deterministic sequence: (a) an SNG; (b) an AND gate produces the minimum number instead of the product when using two deterministic sequences.

Instead of using a true stochastic sequence, deterministic methods have been adopted to reduce the random fluctuations. One approach is to use organized permutations of ‘0’s and ‘1’s [69, 70, 100, 101]. The basic idea is to gather the ‘0’s (or ‘1’s) to the head (or tail) of a stochastic sequence. These deterministic “stochastic” sequences can be generated by replacing the RNG in the SNG with an up (or down, determined by the position of ‘0’s and ‘1’s) counter as shown in Fig. 2.8. The sequences generated by this method are highly correlated. It is evident in Fig. 2.8(b) that for two deterministic sequences passing the stochastic multiplier, the output is the minimum of the two values encoded by the sequences

instead of the product. Another approach is to use the Halton sequence as suggested in [41]. However, a base-conversion circuit is inevitable for the Halton sequence generator, which introduces a significant hardware overhead. Recent work also suggests using FSM-based SNGs [102], time coding [62],  $\Delta - \Sigma$  modulation [103] and Sobol-based deterministic methods [104] after the proposal of using Sobol sequences [105] to generate the sequences.

## Chapter 3

# Toward Energy-Efficient Stochastic Circuits Using Parallel Sobol Sequences

Stochastic computing (SC) often requires long stochastic sequences and, thus, a long latency to achieve accurate computation. The long latency leads to an inferior performance and low energy efficiency compared to most conventional binary designs. In this chapter, a type of low-discrepancy (LD) sequence, the Sobol sequence<sup>1</sup>, is considered for use in SC. Compared to the use of pseudorandom sequences generated by linear-feedback shift registers (LFSRs), the use of Sobol sequences improves the accuracy of stochastic computation with a reduced sequence length. The inherent feature in Sobol sequence generators enables the parallel implementation of random number generators with an improved performance and hardware efficiency. In particular, the underlying theory is formulated and circuit design is proposed for an arbitrary power of 2 degree of parallelization. In addition, different strategies are implemented for parallelizing combinational and sequential stochastic circuits. At a root-mean-squared error (RMSE) of around  $5 \times 10^{-3}$ , the  $8\times$  parallel stochastic multiplier using Sobol sequences consume approximately 1.5% of the energy per operation (EPO) of that using conventional non-parallelized LFSR-generated pseudorandom numbers. For the stochastic divider, the EPO is reduced by 99.3% at an RMSE of around  $2 \times 10^{-2}$ . Meanwhile, an average of 79 (up to 110) times improvements in throughput per area (TPA) and less than 1% runtime are achieved at the aforementioned accuracy. A sorting network is implemented for a

---

<sup>1</sup>The Sobol sequence is used to refer to the sequence of quasirandom numbers that are generated by using the method proposed by the Russian mathematician I. M. Sobol throughout this chapter. The stochastic sequences, containing only '0's and '1's, generated using the Sobol sequences are referred to as the Sobol-based stochastic sequences.

median filter (MF) as an application. For a similar image processing quality, a higher energy efficiency is obtained for an  $8\times$  parallelized stochastic MF compared to its binary counterpart at a low resolution (e.g., less than or equal to 7-bit resolution).

### 3.1 Introduction

Although an SC circuit can be simple, its performance is undermined by the required sequence length [19]. Since only one bit is generated in each clock cycle, it takes  $L$  clock cycles to fully generate and process a stochastic sequence with  $L$  bits. As a result, the energy consumption increases proportionally with  $L$  and the throughput decreases in an inversely proportional manner with  $L$ ; therefore, a large  $L$  leads to a low energy efficiency. The accuracy of SC can be improved by increasing the sequence length, but the stochastic error only decreases with  $1/\sqrt{L}$  [37].

In [41], the Halton sequence is introduced for use in SC. This LD sequence requires a shorter length for achieving the same accuracy compared to LFSR-generated pseudorandom sequences. When several independent sequences are required, however, the generation of Halton sequences relies on the use of counters with different radices, and thus base conversion becomes necessary for a binary circuit. The base conversion imposes additional hardware overhead on the stochastic circuit. In [105], the Sobol sequence is introduced to replace LFSR-generated sequences to improve the efficiency of an SC circuit. It is shown that in most cases, the use of Sobol sequences leads to a better energy efficiency with a similar accuracy compared to the use of LFSR-generated sequences. However, the improvement is not as significant when compared to the use of Halton sequences.

In this chapter, the inherent feature of Sobol sequence generation is exploited for an efficient parallel implementation of the generator. In the proposed  $2^m \times (m = 0, 1, 2, \dots)$  parallel generator, only a few extra XOR gates are required to implement the parallelization. Both parallel combinational and sequential circuits are then designed for SC. With  $8\times$  parallelization, a circuit using Sobol sequences consumes approximately 1% of the energy consumption of an LFSR-based circuit with more than 49 times of the TPA to achieve a similar accuracy. A stochastic MF is implemented for removing noise in images. At a

similar quality, the parallel stochastic Sobol design achieves a higher energy efficiency than its binary counterpart at a relatively low resolution.

## 3.2 Review

### 3.2.1 Low-discrepancy sequences

LD sequences were first proposed to accelerate the convergence process of Monte Carlo (MC) integration [106, 107], which is referred to as quasi-MC. MC integration requires  $S$ -dimensional (or  $S$ -independent) random sequences to estimate an  $S$ -dimensional numerical integration. Using random sequences with a sufficient length, MC integration can provide an estimate of the result for a numerical integration. It has been shown that a lower discrepancy in the random samples leads to a smaller error in MC integration [106]. An SC circuit can be considered as an MC problem. It is shown in [41] that a stochastic circuit using Halton sequences as LD sequences produces a smaller error than a circuit using pseudorandom sequences.

Discrepancy is a measure indicating how evenly a random sequence is distributed in the sample space. For a random sequence  $P$ , it can be quantitatively measured by the star discrepancy  $D^*(P)$ . For a random sequence with  $L$  random points, it is given by [41, 106]

$$D^*(P) = \max_B \left| \frac{A(B;P)}{L} - \lambda(B) \right|, \quad (3.1)$$

where  $B$  is any  $s$ -dimensional region in the form  $\prod_{i=1}^s [0, u_i)$  within an  $s$ -dimensional unit cube  $\prod_{i=1}^s [0, 1]$ ;  $A(B;P)$  is a function that counts the number of points satisfying  $P \in B$ , and  $\lambda(B)$  is the Lebesgue measure of  $B$ : it is the length of  $B$  if  $s = 1$  or is the area of  $B$  if  $s = 2$ .

The  $D^*(P)$  is bounded by  $O(\log(L)^{s-1}/L)$  for an LD sequence. Thus, using a longer sequence (with a larger  $L$ ) and/or fewer independent sequences (with a smaller  $s$ ) implies a smaller error in an SC circuit. For a small  $s$  and a large  $L$ , the error in quasi-MC integration asymptotically converges to  $O(1/L)$ , whereas it is approximately proportional to  $1/\sqrt{L}$  for using pseudorandom sequences. Thus, the stochastic circuits using LD sequences can produce more accurate results with shorter sequences.

Several methodologies have been developed to generate different types of LD sequences, including Halton, Sobol and Faure sequences. Software-based generation

methods have been developed, but few have been implemented in hardware. Examples of the aforementioned sequences are shown in Fig. 3.1.

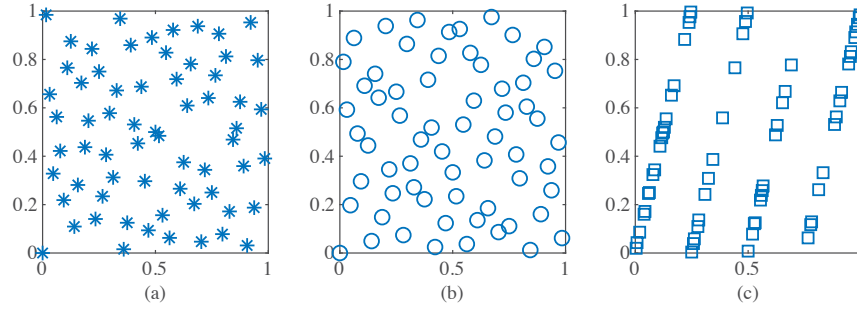


Figure 3.1. Examples of 2-dimensional (a) Sobol sequence; (b) Halton sequence; and (c) LFSR-based pseudorandom sequences. The LD sequences are more evenly distributed than the pseudorandom sequences.

### 3.2.2 Sobol sequence generation

A direction vector array (DVA)  $\{V_k\}$  ( $k = 0, 1, \dots, N - 1$ ) is a group of intermediate variables; these variables can be generated by using primitive polynomials [108]. The direction vectors (DVs) are pre-calculated and stored in the DVA in the hardware implementation. Uncorrelated Sobol sequences are generated by using multiple DVAs derived by different primitive polynomials. At least  $\lceil \log_2 L \rceil$  DVs are required for generating a Sobol sequence of length  $L$ . An algorithm for generating Sobol sequence with a specific DVA is elaborated in [108] and is briefly summarized in Fig. 3.2. In Fig. 3.2,  $\{R_i\}$ , with  $i = 0, 1, 2, \dots, L - 1$ , is a Sobol sequence of length  $L$ , and “LSZ” stands for “least significant zero”.

At each iteration of the loop, the  $i$ th quasirandom number,  $R_i$ , is XOR-ed with one of the DVs,  $V_k$ , to produce  $R_{i+1}$ . The DV index  $k$  is determined by the position of the least

- |   |   |
|---|---|
| 1: $R_0 = 0$ ;<br>2: <b>for</b> $i = 0$ <b>to</b> $L - 2$ <b>do</b><br>3: $k = \text{LSZ position of } i$ ;<br>4: $R_{i+1} = R_i \oplus V_k$ ;<br>5: <b>end for</b> ;<br>6: <b>return</b> $\{R_n\}, n = 0, 1, \dots, L - 1$ | <div style="text-align: right;">▷ Initialization</div> <div style="text-align: right;">▷ Detection of LSZ</div> |
|---|---|

Figure 3.2. Sobol sequence generation algorithm.

Table 3.1. Truth table of a 4-to-2 priority encoder for LSZ detection

| Inputs |       |       |       | Outputs |       |
|--------|-------|-------|-------|---------|-------|
| $D_3$  | $D_2$ | $D_1$ | $D_0$ | $Q_1$   | $Q_0$ |
| X      | X     | X     | 0     | 0       | 0     |
| X      | X     | 0     | 1     | 0       | 1     |
| X      | 0     | 1     | 1     | 1       | 0     |
| 0      | 1     | 1     | 1     | 1       | 1     |

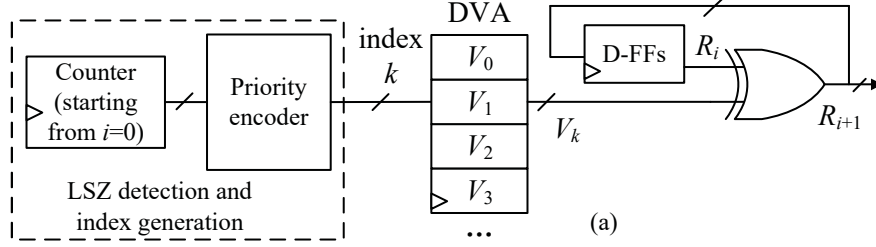
significant zero (LSZ) in the binary form of  $i$ . For example, if  $i = 11$ ,  $i$  is first converted to the binary representation  $(1011)_2$ . Then, the LSZ of  $i$  is at bit 2, which implies that  $k = 2$ . Accordingly,  $V_2$  in the DVA is XOR-ed with  $R_{11}$  to produce  $R_{12}$ . The LSZ detection is of complexity  $O(\log N)$ , where  $N$  is the bit width of a number, using shift-and-count in a software implementation, whereas a priority encoder can detect the LSZ much faster in hardware. The truth table of a 4-to-2 priority encoder for the LSZ detection is shown in Table 3.1. Note that “X” stands for “don’t care”.

For the algorithm in Fig. 3.2, a hardware Sobol sequence generator is proposed in [2], as shown in Fig. 3.3. A counter counts  $i$  in the for-loop. The priority encoder is used to detect the LSZ. The obtained index  $k$  is then passed to the component that stores the values in DVA for retrieving  $V_k$  at each clock cycle. The XOR gates and D flip-flops (FFs) are used to perform  $R_{i+1} = R_i \oplus V_k$  for each iteration of the for-loop. An  $N$ -bit generator can produce non-repeated Sobol sequences with a length of  $L = 2^N$ . Parallelization can be implemented on the Sobol sequence generator in Fig. 3.3, however only a maximum degree of  $4\times$  is achieved in [2].

### 3.3 Parallel Sobol sequence generator

A method to mitigate the long-latency problem is to parallelize the computation by duplicating the stochastic number generators (SNGs) and stochastic circuits. However, the energy efficiency may not be improved because the power consumption would be increased such that the energy consumption would remain nearly the same. Additionally, the hardware cost would also be increased. However, Sobol sequence generation is inherently parallelizable so that better energy efficiency can indeed be obtained by implementing a high degree of parallelism.





Working example of a Sobol sequence generator

| CLK   | $i$ | $k$ | $V_k$    | $R_i$    | $R_{i+1}$ |
|-------|-----|-----|----------|----------|-----------|
| reset | 0   | 0   | 00000000 | 00000000 | 00000000  |
| 1     | 1   | 1   | 10000000 | 00000000 | 10000000  |
| 2     | 2   | 0   | 11000000 | 10000000 | 01000000  |
| 3     | 3   | 2   | 10000000 | 01000000 | 11000000  |
| 4     | 4   | 0   | 11100000 | 11000000 | 00100000  |
| 5     | 5   | 1   | 10000000 | 00100000 | 10100000  |
| ...   | ... | ... | ...      | ...      | ...       |

The D-flip flops are reset for an extra clock cycle to compensate the clock cycle required to fetch the DV.

(b)

Example of a DVA

| $k$ | $V_k$    |
|-----|----------|
| 0   | 10000000 |
| 1   | 11000000 |
| 2   | 11100000 |
| 3   | 11110000 |
| 4   | 11111000 |
| 5   | 11111100 |
| ... | ...      |

(c)

Figure 3.3. (a) A Sobol sequence generator, adapted from [2]. (b) An example is given by using (c) the designated DVA. The example reflects actual hardware operation instead of a mathematical model.

### 3.3.1 Formulation

In what follows, the unique feature of Sobol sequence generation is exploited for parallelization. Specifically, the LSZs of continuous non-negative integers follow a regular pattern. For the ease of interpretation, let  $L(i)$  indicate the LSZ position of  $i$  in its binary format ( $i = 0, 1, \dots$ ). The LSZs of continuous integers are listed in Table 3.2. Following the algorithm in Fig. 3.2, the pattern of the LSZs is explored to generate multiple or multi-dimensional Sobol sequences.

As shown in Table 3.2,  $L(i)$  is “nearly periodic” with a period of 8, except for the  $i$ ’s with a remainder of 7 when divided by 8, i.e.,  $i \equiv 7 \pmod{8}$ . Next, we show that  $L(i)$  is “nearly periodic” with a period of  $2^m$  ( $m = 0, 1, 2, \dots$ ), except for the  $i$ ’s that  $i \equiv (2^m - 1) \pmod{2^m}$ . It is proven that the LSZ for the residue class modulo  $2^m$  is

$$L(i) = \begin{cases} L(j) + m & \text{when } i \equiv (2^m - 1) \pmod{2^m}, \\ & \text{for } j = \lfloor i/2^m \rfloor. \\ L(l) & \text{when } i \equiv l \pmod{2^m}, \\ & l = 0, 1, \dots, 2^m - 2. \end{cases} \quad (3.2)$$

Table 3.2. LSZ positions of continuous non-negative integers

|        |          |          |        |          |        |        |        |          |
|--------|----------|----------|--------|----------|--------|--------|--------|----------|
| $i$    | 0        | 1        | 2      | 3        | 4      | 5      | 6      | 7        |
| $L(i)$ | <b>0</b> | <b>1</b> | 0      | <b>2</b> | 0      | 1      | 0      | <b>3</b> |
| $i$    | 8        | 9        | 10     | 11       | 12     | 13     | 14     | 15       |
| $L(i)$ | 0        | 1        | 0      | 2        | 0      | 1      | 0      | <b>4</b> |
| $i$    | ...      |          |        |          |        |        |        |          |
| $L(i)$ | ...      |          |        |          |        |        |        |          |
| $i$    | $8j$     | $8j+1$   | $8j+2$ | $8j+3$   | $8j+4$ | $8j+5$ | $8j+6$ | $8j+7$   |
| $L(i)$ | 0        | 1        | 0      | 2        | 0      | 1      | 0      | $L(j)+3$ |

An example of (3.2) is shown in Fig. 3.4. The detailed mathematical proof is provided as follows.

$$\begin{array}{ll}
 i = (\underbrace{XX \dots XX}_{\text{Higher bits, } j} \underbrace{111 \dots 1}_{m \text{ '1's}})_2 & i = (\underbrace{XX \dots XX X \dots X}_{\text{Lower } m \text{ bits, } l})_2 \\
 L(i) = L(j) + m & L(i) = L(l) \\
 j = \lfloor i/2^m \rfloor & i = l \bmod 2^m \\
 \text{(a)} & \text{(b)}
 \end{array}$$

Figure 3.4. LSZ position for the residue class modulo  $2^m$ : (a) case 1: the lower  $m$  bits are all '1's; (b) case 2: the lower  $m$  bits are not all '1's.

**Lemma 3.3.1.**  $L(i) = k$  is equivalent to [2]:

$$i \equiv (2^k - 1) \pmod{2^{k+1}}, (k = 0, 1, \dots). \quad (3.3)$$

Fig. 3.5 shows an example for Lemma 3.3.1.

$$\begin{array}{c}
 L(i)=k \\
 \uparrow \\
 \text{the } k\text{th bit} \\
 \uparrow \\
 i = (\underbrace{XX \dots X}_{\text{Don't-care bits}} 0 \underbrace{111 \dots 1}_{k \text{ '1's}})_2 \Leftrightarrow i \equiv 2^k - 1 \pmod{2^{k+1}} \\
 \downarrow \\
 2^k - 1
 \end{array}$$

Figure 3.5. An example of Lemma 3.3.1.

**Corollary 3.3.1.1.** For a number  $i = 2^m \cdot j + 2^m - 1, (i, j, m \in \mathbb{Z}_{\geq 0})$  or  $i \equiv 2^m - 1 \pmod{2^m}$ ,  $L(i) = L(j) + m$ .

*Proof.* Let  $L(j) = k$ . Per Lemma 3.3.1, we have  $j \equiv 2^k - 1 \pmod{2^{k+1}}$ , that is,  $j = h \cdot 2^{k+1} + 2^k - 1$ ,  $h \in \mathbb{Z}_{\geq 0}$ , so that  $i = 2^m \cdot j + 2^m - 1 = 2^m \cdot (h \cdot 2^{k+1} + 2^k - 1) + 2^m - 1 = h \cdot 2^{m+k+1} + 2^{m+k} - 1$ . Therefore,

$$\begin{aligned} i &\equiv (h \cdot 2^{m+k+1} + 2^{m+k} - 1) \pmod{2^{k+m+1}} \\ &\equiv (2^{m+k} - 1) \pmod{2^{k+m+1}}. \end{aligned} \quad (3.4)$$

By applying Lemma 3.3.1, we obtain  $L(i) = m + k = L(j) + m$ .  $\square$

**Corollary 3.3.1.2.** For a number  $i \equiv l \pmod{2^m}$ , ( $i, m, l \in \mathbb{Z}_{\geq 0}, l \neq 2^m - 1$ ),  $L(i) = L(l)$ .

*Proof.* Since  $i \equiv l \pmod{2^m}$  and  $l \neq 2^m - 1$ ,  $0 \leq l \leq 2^m - 2$ , and  $i$  can be represented by  $i = 2^m \cdot j + l$ ,  $j = 0, 1, \dots$ . Let  $L(l) = k$ . First, it is clear that  $k < m$ , which can be proved by contradiction. Per Lemma 3.3.1, we also have  $l \equiv 2^k - 1 \pmod{2^{k+1}}$ . The LSZ of  $i$  can be obtained by

$$i \equiv (2^m \cdot j + l) \pmod{2^{k+1}}. \quad (3.5)$$

Since  $m > k$ , and  $m, k \in \mathbb{Z}_{\geq 0}$ , so  $m \geq k + 1$ . Then, the first term on the right hand side of (3.5) can be removed due to  $(2^m \cdot j) \equiv 0 \pmod{2^{k+1}}$ . Equation (3.5) becomes

$$\begin{aligned} i &\equiv l \pmod{2^{k+1}} \\ &\equiv 2^k - 1 \pmod{2^{k+1}}. \end{aligned} \quad (3.6)$$

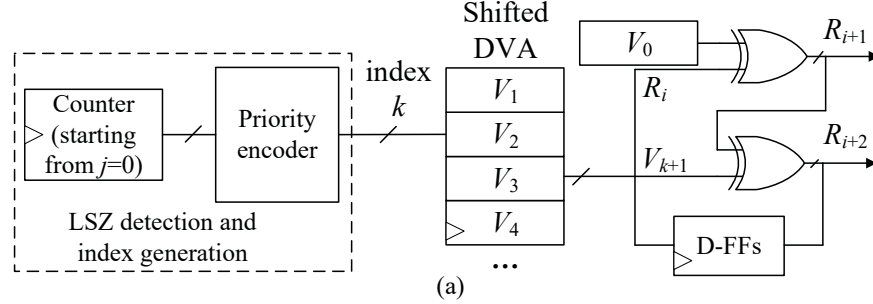
Again, we obtain  $L(i) = k = L(l)$  with the application of Lemma 3.3.1.  $\square$

By Corollaries 3.3.1.1 and 3.3.1.2, (3.2) is explained.

### 3.3.2 Parallelized Sobol SNG and probability estimator

As per Table 3.2, it is clear that  $L(i) = L(0) = 0$  for every even number  $i$  because an even number ends with ‘0’ in the binary format, which is the second case of (3.2) when  $m = 1$ . Therefore, the first DV,  $V_0$ , can be pre-loaded instead of being computed for every other clock cycle as shown in Fig. 3.3, i.e., to perform  $R_{i+1} = R_i \oplus V_0$ . As per (3.2), when  $i$  is an odd number, the same LSZ detection and index generation unit can be used, with the counter counting  $j = \lfloor i/2 \rfloor$ , ( $i = 1, 3, 5, \dots$  so  $j = 0, 1, 2, \dots$ ) instead of counting  $i$ . The ‘+ $m$ ’ term (or ‘+1’ in this case) in (3.2) can be offset by shifting the index of DVA instead of changing the LSZ detection and index generation unit as in [2]. Thus, the index of the

DV loaded from the shifted DVA is given by  $k + 1 = L(j) + 1$ , which is in accordance with the first case in (3.2). The XOR gate at the bottom is used to perform  $R_{i+2} = R_{i+1} \oplus V_k$ . Accordingly, a  $2 \times$  parallel Sobol sequence generator is designed as shown in Fig. 3.6.



Working example of a  $2 \times$  parallel Sobol sequence generator

| CLK   | $j$ | $k$ | $R_i$    | $V_0$    | $R_{i+1}$ | $V_{k+1}$ | $R_{i+2}$ |
|-------|-----|-----|----------|----------|-----------|-----------|-----------|
| reset | 0   | 0   | 00000000 | 10000000 | 10000000  | 00000000  | 10000000  |
| 1     | 1   | 1   | 00000000 | 10000000 | 10000000  | 11000000  | 01000000  |
| 2     | 2   | 0   | 01000000 | 10000000 | 11000000  | 11100000  | 00100000  |
| 3     | 3   | 2   | 00100000 | 10000000 | 10100000  | 11000000  | 01100000  |
| 4     | 4   | 0   | 01100000 | 10000000 | 11100000  | 11110000  | 00010000  |
| 5     | 5   | 1   | 00010000 | 10000000 | 10010000  | 11000000  | 01010000  |
| ...   | ... | ... | ...      | ...      | ...       | ...       | ...       |

The D-flip flops are reset for an extra clock cycle to compensate the clock cycle required to fetch the DV. The DVA used is the same as the one in Fig. 3(c).

(b)

Figure 3.6. (a) Proposed  $2 \times$  parallel Sobol sequence generator. The D-FFs at the final stage are used for recursively generating the Sobol sequence. (b) A working example shows how the generator works.

A  $4 \times$  parallel Sobol sequence generator can similarly be constructed for  $m = 2$ . When  $i \equiv l \pmod{4}$  ( $l = 0, 1, 2$ ), as per (3.2), the LSZ position for  $i$ ,  $L(i)$ , yields  $L(i) = L(l)$ . Equivalently,

$$L(i) = \begin{cases} 0 & \text{when } i \equiv 0 \pmod{4}, \\ 1 & \text{when } i \equiv 1 \pmod{4}, \\ 0 & \text{when } i \equiv 2 \pmod{4}. \end{cases} \quad (3.7)$$

Accordingly,  $V_0$  and  $V_1$  are preloaded to perform the XOR operations. When  $i \equiv 3 \pmod{4}$ , i.e.,  $i = 3, 7, 11, \dots$ , the shifted DVA produces  $V_{k+2}$  with  $k = L(j)$ , where  $j = \lfloor i/4 \rfloor$ . The counter in Fig. 3.7 is used for counting  $j$  ( $j = 0, 1, \dots$ ). A  $4 \times$  parallel Sobol sequence generator is designed and the diagram is shown in Fig. 3.7.

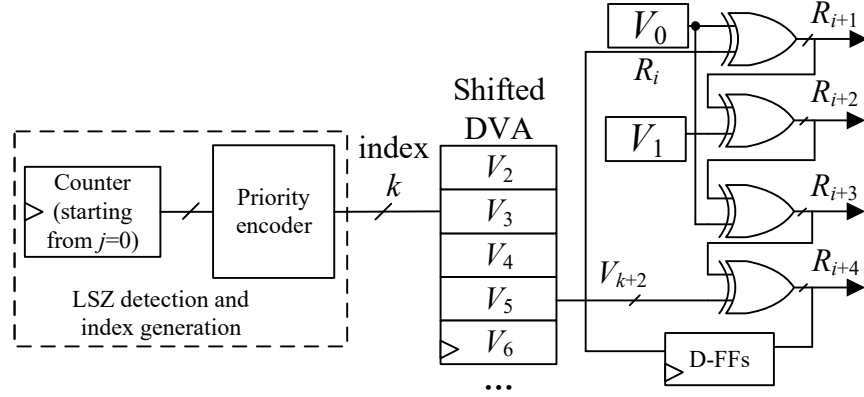


Figure 3.7. Proposed  $4 \times$  parallel Sobol sequence generator.

Similarly, an arbitrary degree of  $2^m \times (m = 3, 4, \dots)$  parallelization can be implemented by exploring the regular pattern of the LSZ positions. Only several additional XOR gates are required to implement the parallelization. When multiple Sobol sequences are required, a second DVA different from the existing one is inserted, and so are the XOR gate array and the D-FFs. The LSZ detection and index generation components can be shared since the LSZs are the same for the different Sobol sequence generations [2]. A generator for two uncorrelated Sobol sequences is shown in Fig. 3.8.

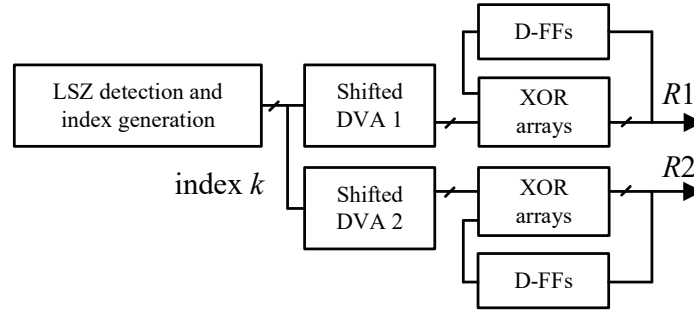


Figure 3.8. Two uncorrelated Sobol sequences are generated by the same LSZ detection and index generation component, but with different DVAs.

An SNG is composed of a random number generator (RNG) and a comparator. Similar to Fig. 2.2, a Sobol SNG can be implemented by an  $N$ -bit Sobol sequence generator and a comparator. For a parallel SNG,  $2^m$  comparators are required to implement  $2^m \times$  parallelization, and  $2^m$  stochastic sequences encoding the same value are generated. Because the circuit for generating additional Sobol sequences is small (using a few XOR

gates), the hardware cost of the comparators will dominate, especially when the degree of parallelization is high. A  $2^m \times$  parallel Sobol SNG is shown in Fig. 3.9.

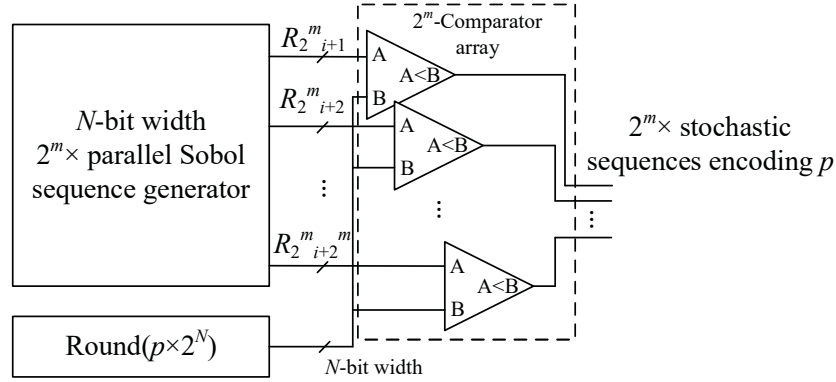


Figure 3.9. A  $2^m \times$  parallel Sobol SNG,  $i = 0, 1, \dots$

The probability estimator (PE) can be implemented by an accumulative parallel counter (APC) as proposed in [3]. The APC can take multiple stochastic sequences in one clock cycle and obtain the total number of ‘1’s in parallel stochastic sequences. The diagram is shown in Fig. 3.10.

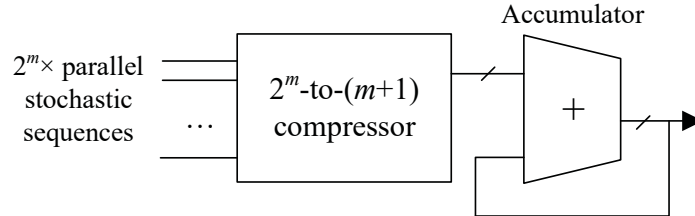


Figure 3.10. An APC adapted from [3].

## 3.4 Parallel stochastic circuits

### 3.4.1 Basic computing elements

To compare the hardware efficiency of using different types of random sequences in SC, several basic stochastic elements are considered: (a) an AND gate implementing a multiplier, as shown in Fig. 2.3(a); (b) a multiplexing circuit computing the Bernstein polynomial [1] as a high-dimensional case, as shown in Fig. 2.4; (c) a divider based on stochastic integrator (with an up/down counter) as a stochastic sequential element, as shown in Fig. 2.6(c).

In Fig. 2.3(a), given uncorrelated stochastic sequences encoding  $x_1$  and  $x_2$ , the probability of the output of the AND gate is  $y = x_1x_2$  in the unipolar representation. The multiplexing circuit in Fig. 2.4 is used to calculate an  $N$ th-order Bernstein polynomial  $f(x) = \sum_{i=0}^N z_i B_{i,N}(x)$ , where  $B_{i,N}(x) = \binom{N}{i} x^i (1-x)^{N-i}$ . The selection signal is produced by a binary adder summing up the independent stochastic bit streams encoding  $x$ . A stochastic divider employs the converged value of the up/down counter to estimate the quotient of two numbers [22]. As shown in Fig. 2.6(c), when an equilibrium state is reached, the probabilities of counting-up and counting-down are equal.

### 3.4.2 Parallel computing elements

As shown in Chapter 1, Table 1.1, a parallel stochastic circuit works similarly as the average-over-pool coding scheme in a neuron. The simultaneously arriving “spikes” produced by the parallel SNG using Sobol sequences are processed by the following proposed parallel stochastic circuits.

#### Parallel combinational elements

The implementation of parallel stochastic combinational elements is straightforward. In general,  $2^m$  duplicates of the original stochastic circuit can implement  $2^m \times$  parallelization. Fig. 3.11 shows a  $4 \times$  unipolar stochastic multiplier, which is implemented by four duplicates of the AND gate.

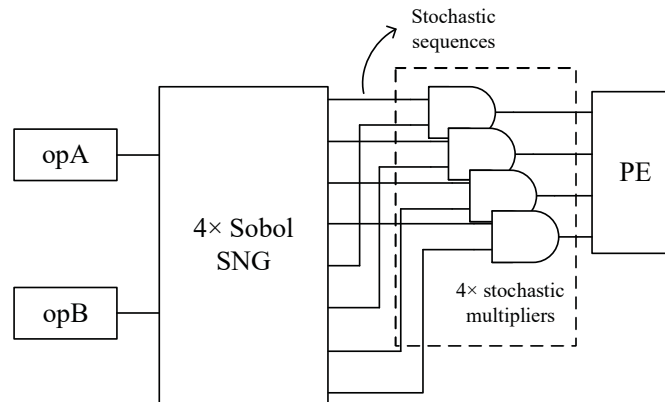


Figure 3.11. A  $4 \times$  unipolar stochastic multiplier. The SNG generates parallel stochastic sequences for the multiplier and multiplicand, opA and opB, respectively. PE stands for a probability estimator.

Similarly, a parallel stochastic Bernstein polynomial circuit can also be implemented by duplication.

### Parallel sequential elements

Stochastic sequential elements mainly consist of two categories: finite state machine (FSM)-based and stochastic integrator-based. For the FSM-based circuits, the functionalities are based on the theory of Markov chains, that is, the current state of the FSM is only directly related with its last state. This type of circuit requires that each bit in the input stochastic sequence is independently generated. However, the bits in a Sobol sequence are generated from the previous bits, and this violates the requirement and will not produce accurate results. For example, a stochastic tanh (Stanh) circuit using a Sobol sequence creates a hard-threshold function instead of an S-shaped curve as shown in Fig. 3.12. However, the use of Sobol sequences improves the accuracy of stochastic integrator-based circuits. The stochastic divider is considered as an illustrative example.

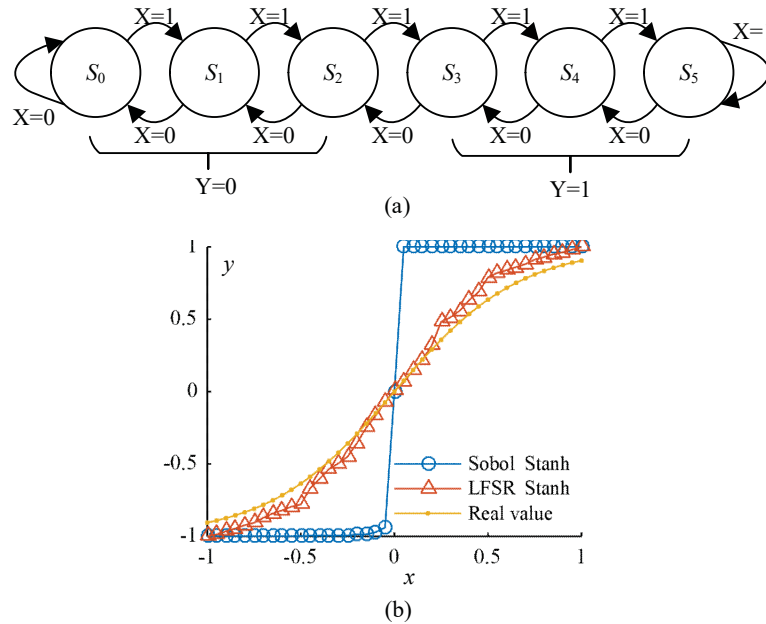


Figure 3.12. Using Sobol sequences in the FSM-based Stanh circuit produces inaccurate results: (a) the state transition graph of Stanh; (b) the results of  $\tanh(3x)$  computed by using Sobol and LFSR-generated sequences. Note that  $x$  is encoded by the stochastic sequence  $X$ , while  $y$  is encoded by the sequence  $Y$ .

Because it does not increase the convergence speed of a stochastic divider by simply duplicating the circuit, the parallelization can be implemented by doubling the input



sequences of the up/down counter to accelerate the computation, as shown in Fig. 3.13. To help understand the underlying theory, the mathematical model of a stochastic divider is analyzed as follows.

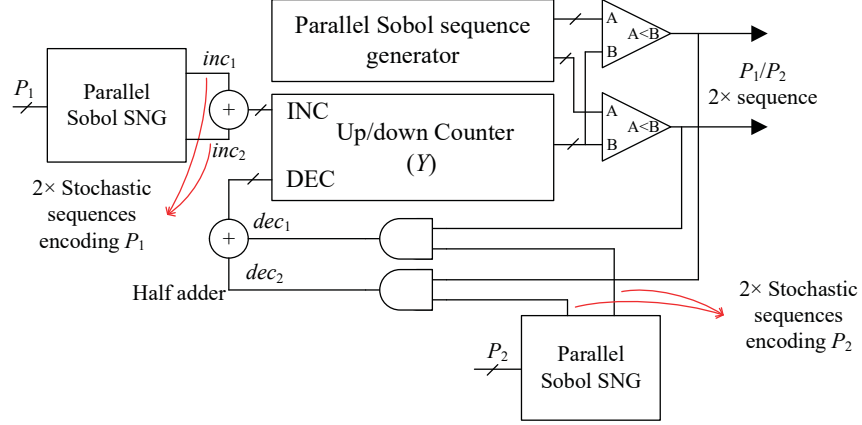


Figure 3.13. A  $2 \times$  parallel stochastic divider.

In the stochastic divider in Fig. 2.6(c) [22], the computation of the quotient relies on the convergence of the stochastic integrator until its equilibrium state is reached. Let  $p_{1,i}$ ,  $p_{2,i}$  and  $q_i$  be the  $i$ th bit in the stochastic sequences encoding  $P_1$ ,  $P_2$  and  $P_1/P_2$ . Let the integer stored in the  $N$ -bit counter be  $Y_i$ . The probability value carried by the counter is then  $y_i = Y_i/2^N$ , where  $N$  is the bit width of the up/down counter. The AND gate serves as a stochastic multiplier, and the output of the AND gate encodes  $p_{2,i} \cdot q_i$ . The up/down counter is updated by the rule [89]:

$$Y_{i+1} = Y_i + p_{1,i} - p_{2,i} \cdot q_i. \quad (3.8)$$

If the initial value stored in the counter is  $Y_0$ , then the value of  $Y_k$  at an arbitrary  $k$ th clock cycle is obtained by accumulating (3.8) for  $i = 0, 1, \dots, k-1$  as

$$Y_k = Y_0 + \sum_{i=0}^{k-1} (p_{1,i} - p_{2,i} \cdot q_i). \quad (3.9)$$

Additionally, the sequence  $\{q_i\}$  is generated by comparing the number stored in the counter and the uniformly distributed random number generated by the RNG, in a similar manner to an SNG, so the expectation of  $q_i$  is  $\mathbb{E}[q_i] = y_i = Y_i/2^N$ . Then, the expectation of  $Y_k$  is given as

$$\mathbb{E}[Y_k] = Y_0 + \sum_{i=0}^{k-1} (P_1 - P_2 y_i). \quad (3.10)$$

Substituting  $Y_k$  and  $Y_0$  by  $y_k$  and  $y_0$ , (3.10) becomes

$$\mathbb{E}[y_k] = y_0 + \frac{1}{2^N} \sum_{i=0}^{k-1} (P_1 - P_2 y_i). \quad (3.11)$$

The Euler method is a first-order iterative algorithm that solves an ordinary differential equation (ODE). For an ODE  $\frac{dy(t)}{dt} = f(t)$ , a one-step solution is calculated from the previous estimate as follows,

$$\hat{y}_{i+1} = \hat{y}_i + hf(t_i), \quad (3.12)$$

where  $h$  is the step size and  $t_i = hi$ .  $\hat{y}_i$  is the Euler numerical solution at the  $i$ th step. Given an initial condition of an ODE  $y_0$ , the  $k$ th step numerical solution is calculated by accumulating (3.12) through  $i = 0, 1, \dots, k-1$ , such that

$$\hat{y}_k = y_0 + h \sum_{i=0}^{k-1} f(t_i). \quad (3.13)$$

By comparing (3.11) and (3.13), it can be seen that the stochastic integrator of the divider provides an unbiased estimate to the Euler solution of the ODE

$$\frac{dy(t)}{dt} = P_1 - P_2 y(t) \quad (3.14)$$

with a step size of  $h = 1/2^N$ . By solving (3.14) analytically, the convergence process of the counter is approximated by

$$y(t) = \frac{P_1}{P_2} - \frac{1}{P_2} (P_1 - P_2 y_0) e^{-P_2 t}, \quad (3.15)$$

where  $t$  is discretized to the number of clock cycles, i.e.,  $t_i = h \cdot i = i/2^N$ ,  $i = 0, 1, \dots$  and  $y_0$  is set to the initial value of the counter. As  $t$  approaches infinity, the exponential term approaches 0 so that  $y(t)$  converges to the quotient of  $P_1$  and  $P_2$ . The convergence process is governed by an exponential function, and the speed of convergence is determined by the exponent.

The convergence process of the proposed parallel stochastic divider in Fig. 3.13 can similarly be evaluated. The up/down counter updates its value by  $Y_{i+1} = Y_i + inc_1 + inc_2 - dec_1 - dec_2$ , where signals  $\{inc_1, inc_2, dec_1, dec_2\}$  can only be 0 or 1. The expectation of the “INC” input parallel stochastic sequences, i.e.,  $inc_1 + inc_2$ , can be obtained from the distribution of  $\sum P_1$  in Table 3.3, as  $\mathbb{E}[\text{INC}] = 0 \times (1 - P_1)^2 + 2P_1(1 - P_1) + 2 \times P_1^2 = 2P_1$ .

Similarly,  $\mathbb{E}[\text{DEC}] = 2P_2y(t)$ . Then the expectation of the value stored in the counter at the  $k$ th clock cycle is given as

$$\mathbb{E}[Y_k] = Y_0 + \sum_{i=0}^{k-1} (2P_1 - 2P_2y_i). \quad (3.16)$$

Table 3.3. Probability distribution of  $\sum P_1$  in a parallel stochastic divider.

| $\sum P_1$  | 0             | 1               | 2       |
|-------------|---------------|-----------------|---------|
| probability | $(1 - P_1)^2$ | $2P_1(1 - P_1)$ | $P_1^2$ |

Due to (3.16), the parallel stochastic divider actually solves

$$\frac{dy(t)}{dt} = 2P_1 - 2P_2y(t). \quad (3.17)$$

The solution for (3.17) is

$$y(t) = \frac{P_1}{P_2} - \frac{1}{P_2}(P_1 - P_2y_0)e^{-2P_2t}. \quad (3.18)$$

Compared to (3.15), the exponent is doubled, and so the convergence process of the  $2 \times$  parallel divider design is twice as fast as the one that does not use any parallelization.

### Convergence time of a stochastic divider

The time when the value of  $y(t)$  converges, referred to as the convergence time, can be estimated by (3.15) or (3.18). Since the value stored in the counter is an  $N$ -bit number, the resolution is  $1/2^N$  for encoding a probability in  $[0, 1]$ . When the absolute value of the exponential term in (3.15) (or (3.18)) is smaller than the resolution of the counter, the state of the counter can be considered converged. The convergence time can then be estimated. Additionally, both the divisor and dividend are shifted to the left by the same number of bits such that the most significant bit (MSB) of the divisor is '1'. By doing so, the divisor,  $P_2$ , is amplified, so that the term containing the exponential expression in (3.15) or (3.18) approaches to 0 faster, while the quotient is not changed. Assume that the convergence time is  $t_{conv}$ , an inequality is composed to find  $t_{conv}$  for the original stochastic divider as per (3.15),

$$\left| (P_1 - P_2y_0) \frac{1}{P_2} e^{-P_2t_{conv}} \right| < \frac{1}{2^N}. \quad (3.19)$$

By solving the inequality,  $t_{conv}$  is estimated to be

$$t_{conv} \geq \frac{1}{P_2} (N \log 2 + \log \left| \frac{P_1 - P_2 y_0}{P_2} \right|) \quad (3.20)$$

This approach is used for estimating the runtime of a stochastic divider. Also, it can be applied to any stochastic integrator-based circuits, whose convergence follows an exponential function.

## 3.5 Experiments and results

### 3.5.1 Metrics

The performance of the SC elements are examined by EPO, TPA [88], and runtime. The RMSE is used to measure the accuracy.

Given the sequence length  $L$  and degree of parallelization  $P$  for a stochastic arithmetic operation, the EPO for an SC element can be calculated by

$$\text{EPO} = \text{Total Power} \times T_{clk} \times L/P, \quad (3.21)$$

where  $T_{clk}$  is the clock period and power is measured at the corresponding  $T_{clk}$ .  $P = 1$  when no parallelization is applied. Similarly, the EPO of an SNG is measured by the energy consumption for generating one bit. Throughput is used to measure how much information a system can process during a unit time. The TPA of an SNG is measured by the number of stochastic bits generated in a unit time per unit area. The TPA of a stochastic circuit is measured by the number of computation results produced in a unit time per unit area. The runtime is considered as  $t_c \times L/P$  for producing one result for a combinational circuit, where  $t_c$  is the critical path delay. The sequence length of a stochastic divider is determined by the convergence time. Subsequently, the TPA is given as

$$\text{TPA} = 1 \text{ bit} \times P/t_c / \text{area} \quad (3.22)$$

for an SNG and

$$\text{TPA} = 1/(t_c \times L/P) / \text{area} \quad (3.23)$$

for a stochastic arithmetic circuit.

The runtime,  $T$ , is evaluated by

$$T = t_c \times L/P. \quad (3.24)$$

The critical path delay, area and power consumption are first obtained by the Synopsys Design Compiler (DC) with a 28-nm industrial process. The same temperature and process corners are applied to all the circuits. Detailed parameters are listed in Appendix A. The EPO, TPA and runtime are then computed. The RMSE is obtained by using 10,000 random trials for each circuit.

### 3.5.2 Evaluation of the proposed Sobol SNG

#### Accuracy

The accuracy of an 8-bit Sobol SNG is compared with an 8-bit LFSR-based SNG to show how accurately a real number is encoded by using those two different types of SNGs with different sequence lengths. The numbers to be encoded are chosen randomly. For a different sequence length, the ratio of ‘1’s in the stochastic sequences generated by the two SNGs are calculated. The difference between the ratio and the number to be encoded is measured by RMSE and the results are shown in Fig. 3.14.

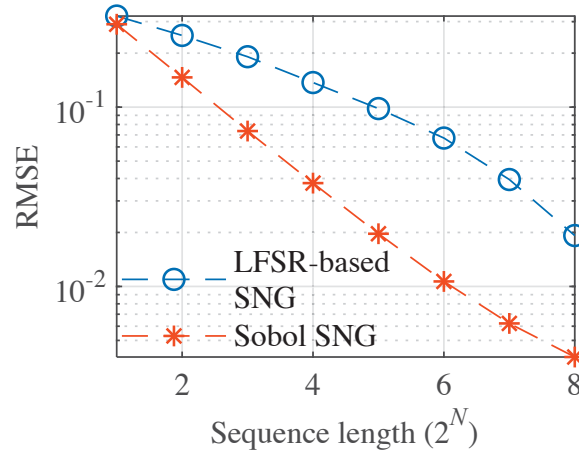


Figure 3.14. Accuracy of the Sobol SNG and LFSR-based SNG.

The accuracy of a stochastic sequence generated by a Sobol SNG is consistently higher than that generated by an LFSR-based SNG. Note that the parallelization does not affect the accuracy since the total number of 1’s remains the same, but with a faster generation rate.

### Hardware efficiency

A Sobol SNG consists of a sequence generator and comparators. A  $2^m \times$  parallelization requires approximately  $(2^m - 1) \times N$  more XOR gates than a single Sobol SNG, where  $N$  is the bit width of the generator. The number of comparators required is the same as the degree of parallelization, i.e.,  $2^m$ . Table 3.4 shows the components required in a  $2^m \times$  parallel Sobol SNG.

Table 3.4. Hardware cost for a  $2^m \times$  parallel Sobol SNG

| Parallelism  | LSZ detection & DVA | XOR gates | Comparators |
|--------------|---------------------|-----------|-------------|
| $1 \times$   | 1                   | $N$       | 1           |
| $2 \times$   | 1                   | $2N$      | 2           |
| $4 \times$   | 1                   | $4N$      | 4           |
| $\dots$      | $\dots$             | $\dots$   | $\dots$     |
| $2^m \times$ | 1                   | $2^m N$   | $2^m$       |

The hardware efficiency of a Sobol SNG is measured in EPO, TPA and generation time. For the Sobol SNG, different degrees of parallelization are implemented. The results are shown in Fig. 3.15 for an 8-bit SNG. The performance of an 8-bit LFSR-based SNG is used as a reference.

When no parallelization is applied, as can be seen in Fig. 3.15, the Sobol SNG has a lower hardware efficiency compared with the LFSR-based SNG. However, the EPO and TPA of a Sobol SNG increase with the degree of parallelization because of its small hardware cost.

### 3.5.3 Stochastic combinational circuits

#### Accuracy

For the AND-based stochastic multiplier, two independent stochastic sequences are required. For a third-order Bernstein polynomial circuit, at least 4-dimensional or 4 independent stochastic sequences are required. The accuracy comparisons for the stochastic multiplier and Bernstein polynomial circuit are shown in Fig. 3.16, in which another type of LD sequences, Halton sequence, is also considered. The Halton sequence generators are implemented by inversely-mapped counters using different bases [41].

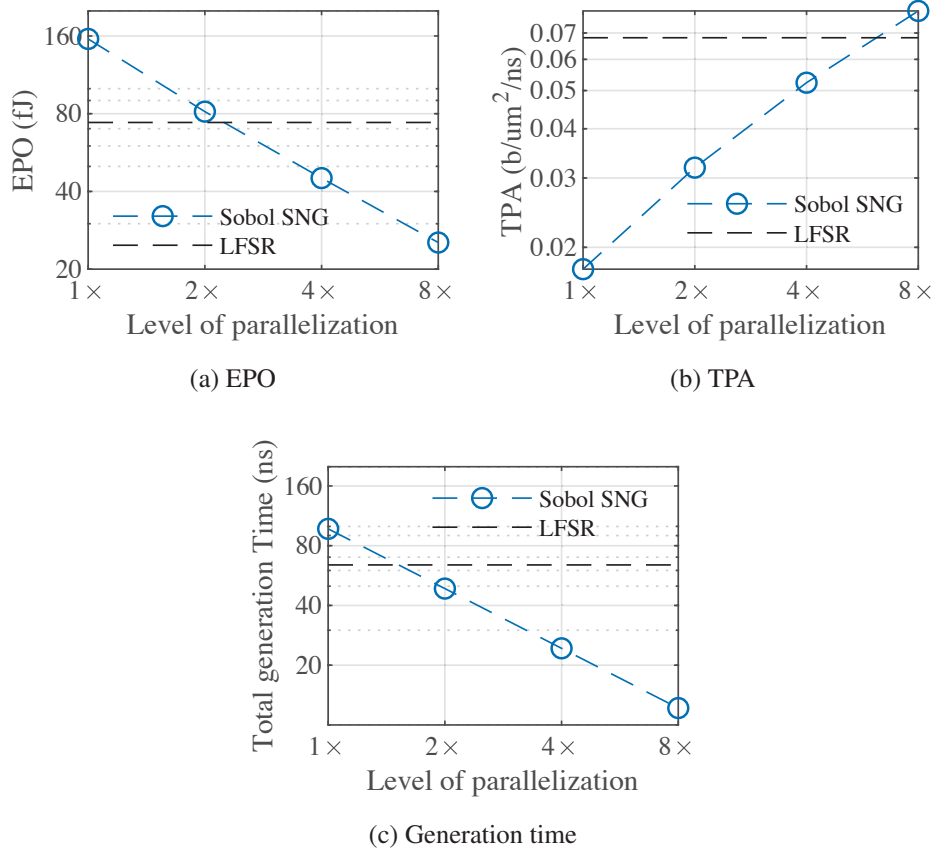


Figure 3.15. Hardware measurements of parallel Sobol SNGs.

As shown in Fig. 3.16, for the same sequence length, the accuracy of the results produced by using Sobol sequences are mostly higher than those obtained using LFSR-generated and Halton sequences. For the stochastic multiplier, it is also observed that as the sequence length increases, the RMSE of the LD sequences-based circuits decreases faster than that of the LFSR-based design. However, due to the increased dimension of the sequences, the RMSE of the Bernstein polynomial circuit using LD sequences does not converge as fast as the multiplier.

### Hardware efficiency

The hardware efficiency of stochastic multipliers are shown in Figs. 3.17, 3.18 and 3.19. For the same accuracy, a lower EPO, a larger TPA and a shorter runtime indicate a more efficient hardware design. From Figs. 3.17, 3.18 and 3.19, most designs using Sobol sequences show a higher efficiency than Halton and LFSR-based designs. This is due to

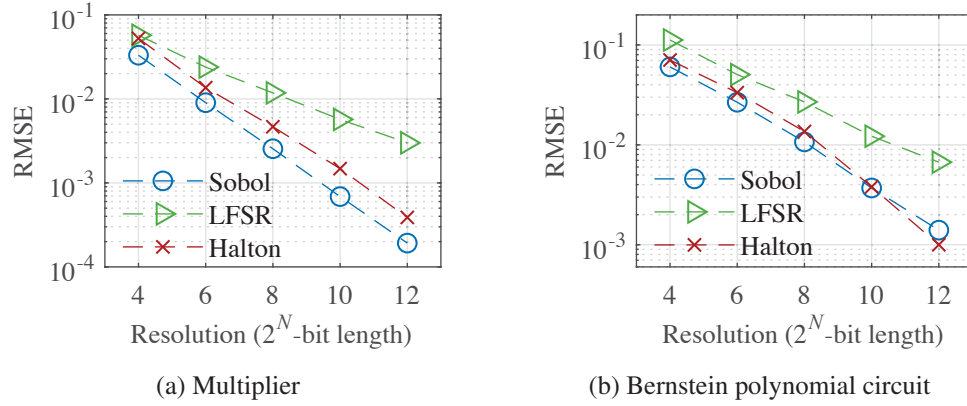


Figure 3.16. Accuracy of combinational stochastic circuits using Sobol, Halton and LFSR-generated sequences.

the shorter sequence length required in a design using Sobol sequences to achieve a similar RMSE. Also, the parallelization in Sobol sequence generation makes a design more efficient because of the small hardware cost to implement the parallelization. The stochastic multipliers using Halton and Sobol sequences consume a similar energy when no parallelization is applied, whereas a parallel design using Sobol sequences is more energy-efficient than the design using Halton sequences. If parallelization is implemented for the Halton- or LFSR-based designs, the EPO and TPA would not change much due to the extra power consumption and hardware cost, albeit with a reduction in runtime.

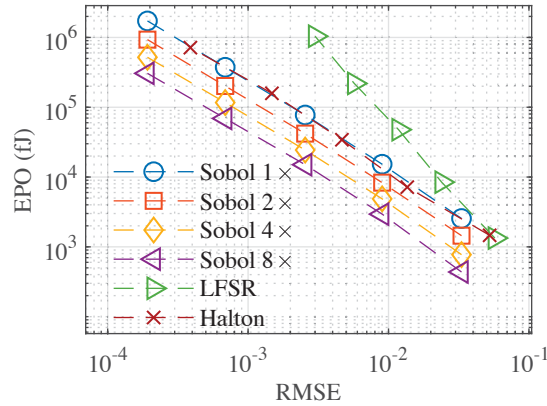


Figure 3.17. EPO of stochastic multipliers using  $2^{12}$ -bit,  $2^{10}$ -bit,  $2^8$ -bit,  $2^6$ -bit and  $2^4$ -bit length from left to the right.

The stochastic multiplier using  $2^8$ -bit Sobol sequences has a similar RMSE to a design using  $2^{12}$ -bit LFSR-generated sequences. The EPO of the  $8\times$  parallel Sobol multiplier



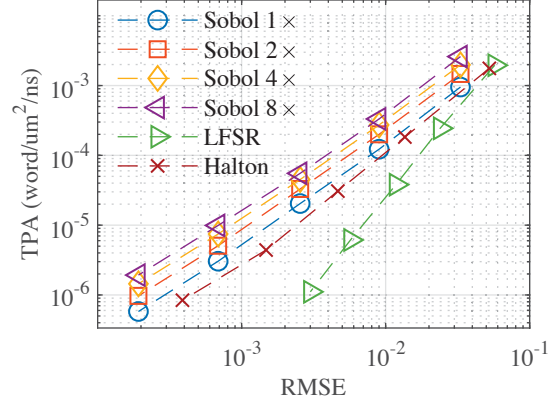


Figure 3.18. TPA of stochastic multipliers using  $2^{12}$ -bit,  $2^{10}$ -bit,  $2^8$ -bit,  $2^6$ -bit and  $2^4$ -bit length from left to the right.

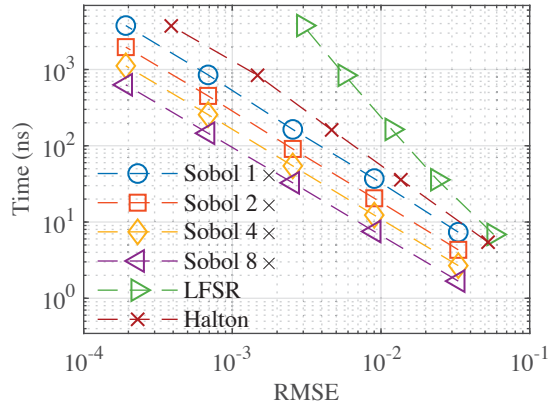


Figure 3.19. Runtime of stochastic multipliers using  $2^{12}$ -bit,  $2^{10}$ -bit,  $2^8$ -bit,  $2^6$ -bit and  $2^4$ -bit length from left to the right.

costs only 1.44% of the energy of the LFSR-based design with 49.80 times of the TPA. The Bernstein polynomial circuits are also evaluated, and the results show a similar trend to the multipliers.

### Curse of dimensionality

The benefits using LD sequences can diminish as the dimension of the sequences increases [106], which is referred to as the “curse of dimensionality”. To investigate how the number of dimensions affects the accuracy and the hardware efficiency of a design using Sobol sequences,  $n$ -dimensional random sequences are employed to implement  $(n - 1)$ th-order Bernstein polynomial with the same sequence length. A total of 10,000 MC simulation runs were carried out, with the coefficient of the Bernstein polynomial randomly chosen. As

shown in Fig. 3.20(a), the RMSEs of LFSR-based designs oscillate around 0.04 throughout all dimensions. Although the RMSE of a design using Sobol sequences increases, it does not exceed that of an LFSR-based design for up to 20 dimensions. Also, the RMSE of the designs using Halton sequences are slightly larger than that of the designs using Sobol sequences.

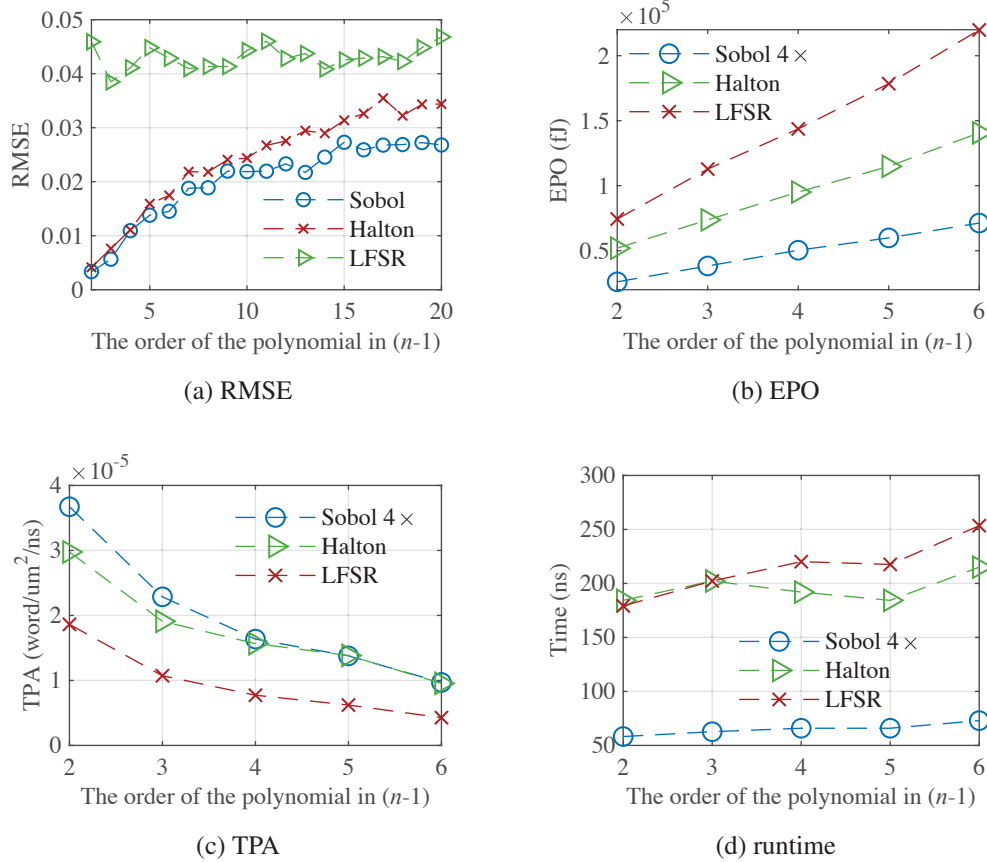


Figure 3.20. Accuracy, EPO, TPA and runtime of an  $(n-1)$ th-order Bernstein polynomial circuit using different random sequences with the same sequence length. An  $n$ -dimensional random sequence is required to implement an  $(n-1)$ th-order Bernstein polynomial circuit.

The EPO, TPA and runtime are reported in Figs. 3.20(b), (c) and (d). As can be seen, a  $4\times$  parallel design using Sobol sequences always results in the lowest EPO, the largest TPA and the shortest runtime. The EPO of the design using Halton sequences increases quickly with the order of the polynomial because a larger counter is required in the larger base for a higher-dimensional Halton sequence. The EPO of the design using Sobol sequences

increases rather slowly because the LSZ detection and index generation unit is shared to generate multidimensional Sobol sequences, as shown in Fig. 3.8.

### 3.5.4 Stochastic sequential circuits

#### Verification of the proposed parallel stochastic divider

The parallel stochastic divider design proposed previously is first verified using hardware simulation. An 8-bit counter is used and different degrees of parallelization are applied. The result produced using LFSRs is also considered for comparison as shown in Fig. 3.21. The predicted convergence time is calculated by using (3.19) with  $P_1 = 0.8$ ,  $P_2 = 0.9$  and  $y_0 = 0.5$ .

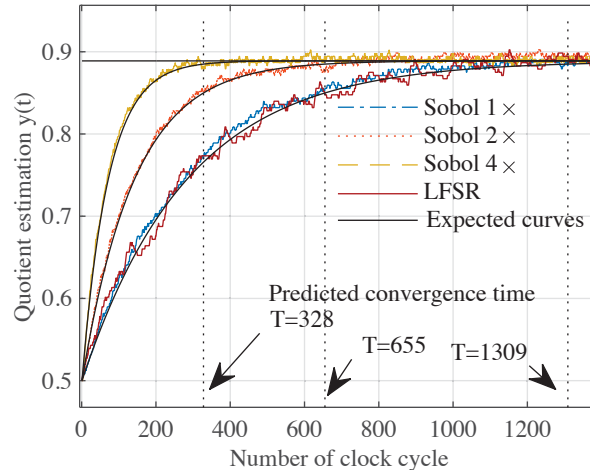


Figure 3.21. Convergence processes of stochastic dividers using Sobol sequences with  $1\times$ ,  $2\times$  and  $4\times$  parallelization, computing  $0.8 \div 0.9$ .  $y(t)$  is initialized with  $y_0 = 0.5$ . The LFSR-based design is also considered for comparison.

From Fig. 3.21, all the parallel divider designs converge to the accurate result with different speeds. The Sobol design using  $4\times$  parallelization is the fastest, followed by the designs using  $2\times$  parallelization and no parallelization. The result produced by using LFSRs fluctuates significantly and leads to an ambiguous result, as shown in Fig. 3.21. However, the results produced by Sobol sequences are more stable. Additionally, the predicted convergence time fits well with the simulation results with less glitches for Sobol sequences.

## Accuracy

The accuracy is measured with different degrees of parallelization and different bit widths for the up/down counter. The results are shown in Fig. 3.22.

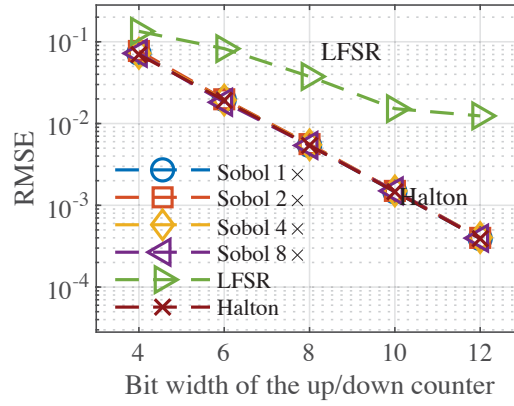


Figure 3.22. Accuracy of stochastic dividers using Sobol sequences with  $1\times$ ,  $2\times$ ,  $4\times$  and  $8\times$  parallelization with different sizes of the up/down counter. The RMSEs for using Halton and LFSR-generated sequences without parallelization are also compared.

As shown in Fig. 3.22, the parallelization has little effect on the accuracy of dividers using Sobol sequences, and the RMSEs of those designs almost overlap with one another. The RMSE of a design using Sobol sequences decreases linearly with the bit width of the up/down counter and it is consistently smaller than that of an LFSR-based design. A divider using Halton sequences shows a similar accuracy to its counterpart using Sobol sequences.

## Hardware efficiency

For different operands, the convergence time (in the number of clock cycles) of the stochastic divider is different. An average convergence time is estimated for randomly chosen operands and the average EPO, TPA and runtime are obtained by using the average convergence time. Different degrees of parallelization are applied to evaluate the improvement in performance and hardware efficiency. The results are shown in Figs. 3.23, 3.24 and 3.25.

As can be seen, as the degree of parallelization increases, the energy and hardware efficiency improve. However, the improvement of a higher degree parallelization is not as significant as at a lower degree parallelization. It is evident in Fig. 3.24 that the TPAs of  $8\times$  parallel stochastic dividers are close to  $4\times$  ones, whereas the TPAs of  $2\times$  parallel

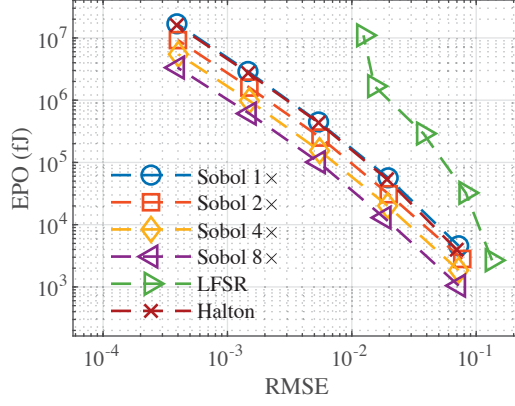


Figure 3.23. EPO of stochastic dividers using Sobol sequences with  $1\times$ ,  $2\times$ ,  $4\times$  and  $8\times$  parallelization with 12-bit, 10-bit, 8-bit, 6-bit and 4-bit up/down counter from left to the right. The LFSR-based design without parallelization is also shown.

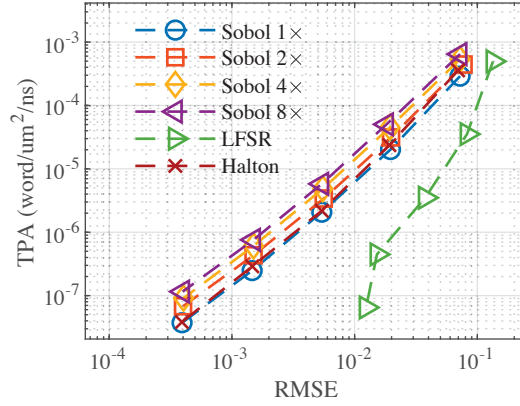


Figure 3.24. TPA of stochastic dividers using Sobol sequences with  $1\times$ ,  $2\times$ ,  $4\times$  and  $8\times$  parallelization with 12-bit, 10-bit, 8-bit, 6-bit and 4-bit up/down counter from left to the right. The LFSR-based design without parallelization is also shown.

stochastic dividers are well separated from the ones without parallelization. This occurs because the number of comparators increases linearly with the degree of parallelization, as shown in Table 3.4. Thus, they dominate the energy and hardware cost of the design.

For the designs using Halton sequences, they have a similar EPO, TPA and runtime to their Sobol counterparts without parallelization. However, the design using Sobol sequences benefits from the efficient implementation of parallelization, so it outperforms its Halton counterpart in all considered metrics.

For the LFSR-based designs, it takes at least two more bit widths to achieve a similar accuracy to the designs using Sobol sequences. Therefore, it results in a longer runtime

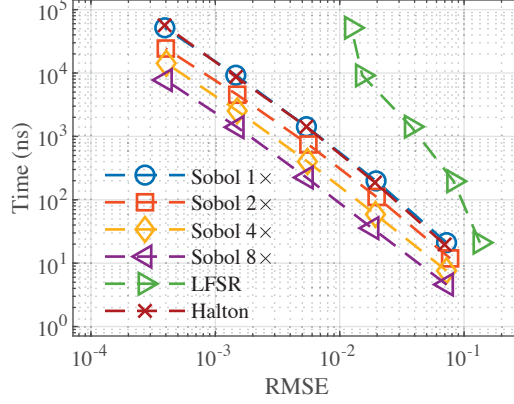


Figure 3.25. Runtime of stochastic dividers using Sobol sequences with  $1\times$ ,  $2\times$ ,  $4\times$  and  $8\times$  parallelization with 12-bit, 10-bit, 8-bit, 6-bit and 4-bit up/down counter from left to the right. The LFSR-based design without parallelization is also shown.

and a lower hardware efficiency. When a 12-bit LFSR-based design is compared with an 8-bit Sobol design, the  $8\times$  parallel design using Sobol sequences costs 0.92% of the energy consumption and produces 89 times of the TPA of an LFSR-based design. Additionally, the accuracy of an 8-bit divider using Sobol sequences is higher than that of the 12-bit LFSR-based divider.

The basic stochastic elements are not compared with their binary counterparts because this would either favor the SC design without considering the SNGs and PEs or, otherwise, it would impose a large overhead on the stochastic arithmetic elements. Instead, the hardware efficiency is compared at the application level in the next section.

## 3.6 Application

### 3.6.1 A sorting network and median filter design

A sorting network rearranges a list of values in an ascending or descending order. It is widely used in modern computer systems for file matching, data searching and filtering, among other applications [109]. A sorting network can be implemented in hardware by comparators and multiplexers for comparing and swapping the values. However, the hardware cost is very high for large volumes of input data. The depth of a sorting network is defined as the largest number of comparators that an input goes through in the network. It is in the order of  $O(n\log(n)^2)$  [110], where  $n$  is the size of the input data. If the

compare-and-swap is executed in parallel, the runtime is proportional to the network depth. Here, a stochastic circuit is used to implement the sorting network with an energy-efficient compare-and-swap unit.

The design of a basic stochastic unit is shown in Fig. 3.26. The underlying principle can be explained by considering the AND gate as an example. The probability of its output sequence is given as

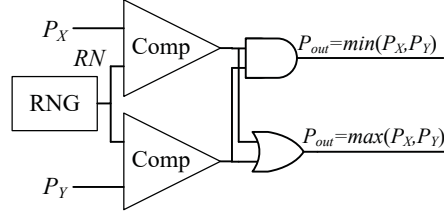


Figure 3.26. Stochastic sorter circuit. The stochastic sequence encoding a larger value is moved downward.

$$\begin{aligned} P[P_{out} = 1] &= P[(P_X > RN) \wedge (P_Y > RN)] \\ &= P[\min\{P_X, P_Y\} > RN]. \end{aligned} \quad (3.25)$$

Thus, the probability of the output sequence is the lesser value of  $P_X$  and  $P_Y$ . Similarly, the OR gate can be shown to obtain the larger value of  $P_X$  and  $P_Y$ :

$$\begin{aligned} P[P_{out} = 1] &= P[(P_X > RN) \vee (P_Y > RN)] \\ &= P[\max\{P_X, P_Y\} > RN]. \end{aligned} \quad (3.26)$$

The output stochastic sequence can be used for further comparing and swapping with the values in another stochastic sequence generated from the same RNG such that (3.25) is satisfied. Hence, a sorting network can be constructed by using just one RNG generating the stochastic sequences for all the input data and the AND and OR gates for comparing and swapping the stochastic sequences.

Salt-and-pepper impulse noise can occur due to the malfunction of image sensors, errors in memory or a noisy channel [111]. Unwanted white and black pixels appear in an image upon the occurrence of this noise. An MF is often used to reduce this type of noise in images without losing the edges. It replaces each pixel in an image with the median value of the surrounding pixels. A  $3 \times 3$  MF can be implemented by a sorting network [112]. A design is shown in Fig. 3.27 with the stochastic sorter as its basic unit.

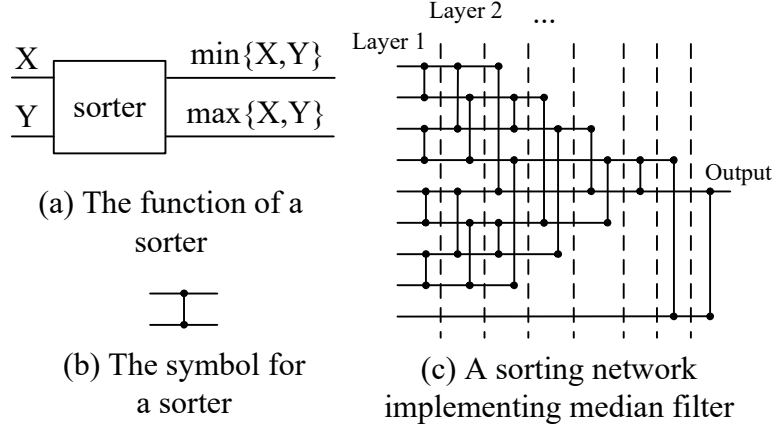


Figure 3.27. The basic unit for implementing an MF.

Since the input sequences are required to be generated by the same RNG as per (3.25), only one RNG is used for the first layer of the sorters as shown in Fig. 3.28. Passing through the first layer of sorters, the output sequences are still the original stochastic sequences with the same permutations of '0's and '1's for a certain value, but with a different order. This means that the output sequences can be directly used as the input sequences in the next compare-swap layers instead of being regenerated by an SNG. As a result, only one RNG is required for the stochastic MF. Additionally, the LD characteristic is maintained after stages of computations. Both Sobol sequences and LFSR-generated sequences can be used for this scheme.

### 3.6.2 Experimental results of the median filter

This stochastic MF design is tested on the 8-bit grey image “cameraman” corrupted by salt-and-pepper noise with a density of 0.1. A conventional binary design is also considered for comparison. Different levels of approximation are applied to the binary design by truncating the bit width. Since there are in total 8 layers of comparing and sorting, the binary MF takes 8 clock cycles to produce the final result. The original and polluted images are depicted in Fig. 3.29 and the filtering results are shown in Fig. 3.30.

It can be seen that both binary and stochastic circuits can filter the noise with a high quality. However, when the sequence length is reduced to 16 bits for a stochastic design or the bit width is reduced to 4 for a binary design, some severe distortions become evident. For the stochastic MF using LFSR-generated sequences, the image can be either darker or



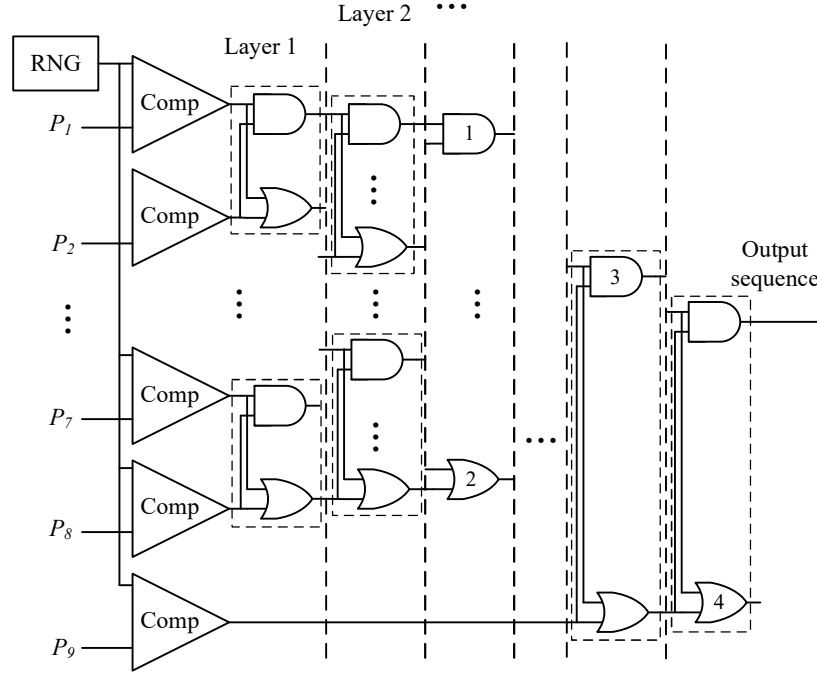


Figure 3.28. Stochastic MF based on the sorting network in Fig. 3.27(c). Since the output stochastic sequences from gates 1, 2, 3 and 4 are not used, they can be removed to save hardware. Each sorter unit is marked by a dotted rectangle.



(a) Original image



(b) Corrupted image

Figure 3.29. Original and corrupted “cameraman” images.

brighter than the original image due to random fluctuations. There is a loss of detail in Figs. 3.30(f) and (g) for 16-bit and 32-bit stochastic designs using LFSRs, respectively.

The quality of the output image can be measured quantitatively by the peak signal-to-noise ratio (PSNR). The results are illustrated in Fig. 3.31. Because of the random noise and random fluctuations in LFSR-based stochastic circuits, 100 MC simulation runs were carried out to allow the mean and standard deviation of the PSNRs to be measured. For both the Sobol and binary designs, the deviation is purely introduced by the randomly

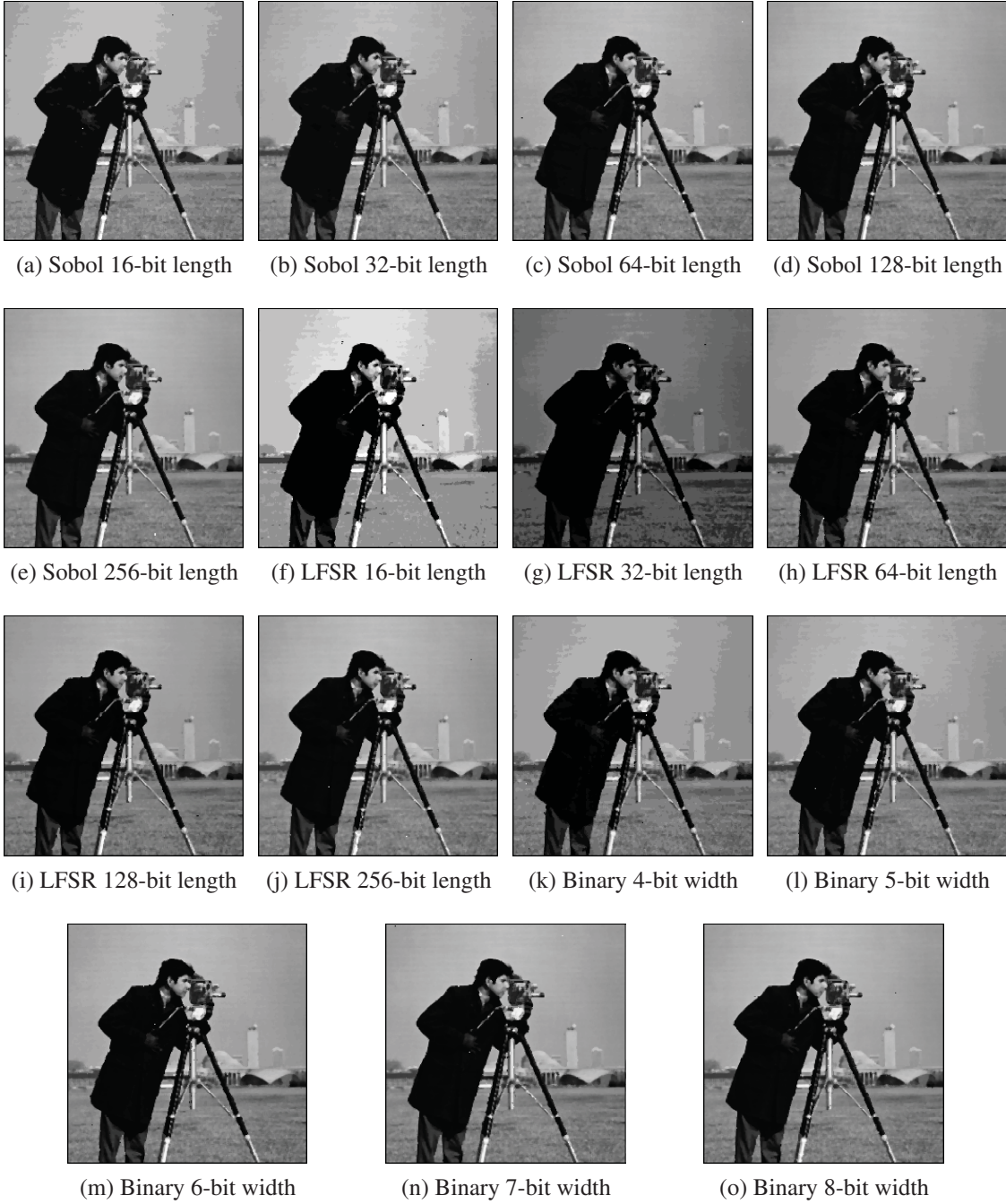


Figure 3.30. Filtering results using the stochastic and binary implementations of an MF.

added noise rather than the design itself due to their deterministic nature, thus the deviation is very small. As can be seen, the LFSR-based stochastic design results in the lowest PSNR because of the randomness in the generation of the random sequences. The design using Sobol sequences has overall the best PSNR even over the binary design because a

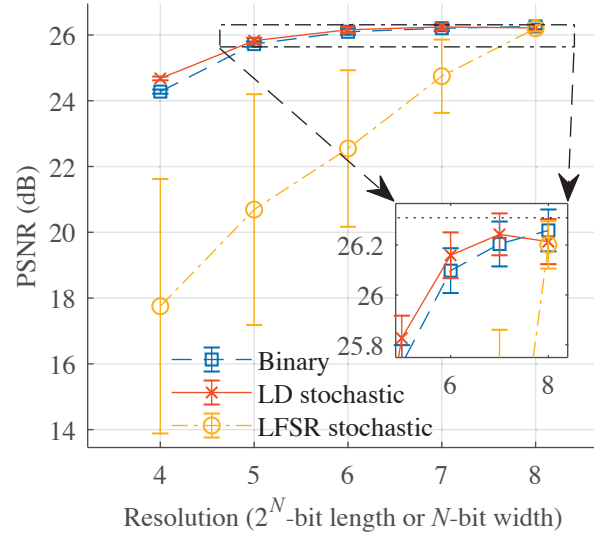


Figure 3.31. PSNR comparison of different median filter implementations.

simple truncation in the binary design introduces a bias and the resulting value is equal to or smaller than its actual value. On the other hand, the results produced by the LD-based designs can either be larger or smaller than the actual value. The distribution of the error of a stochastic sorter is shown in Fig. 3.32.

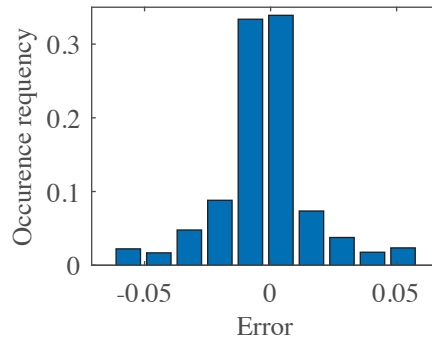


Figure 3.32. Distribution of the errors produced by the stochastic sorter.

Since a base-2 Halton sequence is the same as a one-dimensional Sobol sequence, they produce results with a similar accuracy. The efficiency of the hardware implementation is again measured by EPO, TPA and runtime. The results are shown in Figs. 3.33, 3.34 and 3.35.

As can be seen from Fig. 3.33, the energy efficiency is improved using parallel designs using Sobol sequences. The EPO of the  $8\times$  parallel stochastic MFs are smaller than their

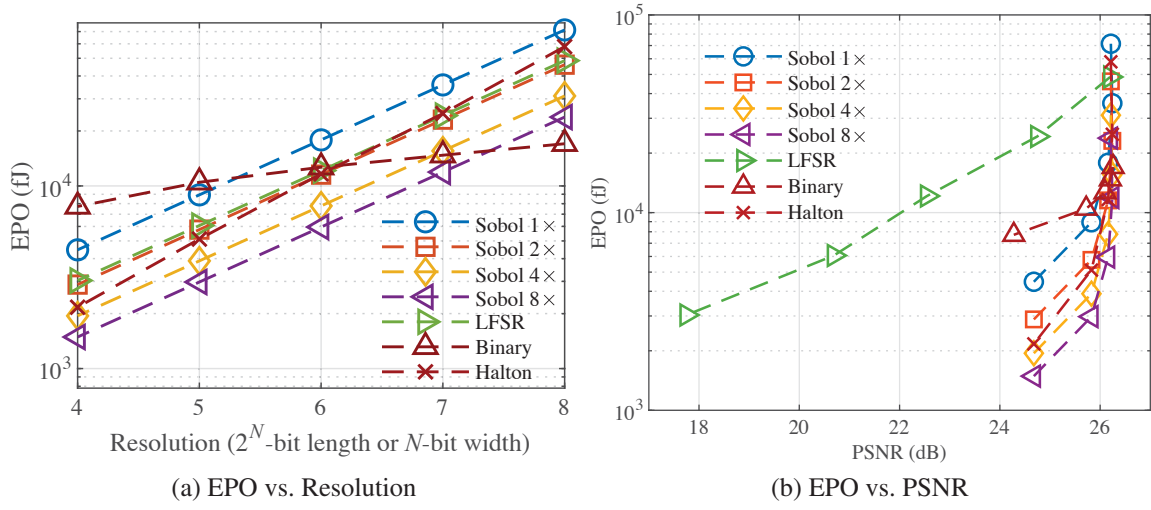


Figure 3.33. EPO comparison of different median filter implementations.

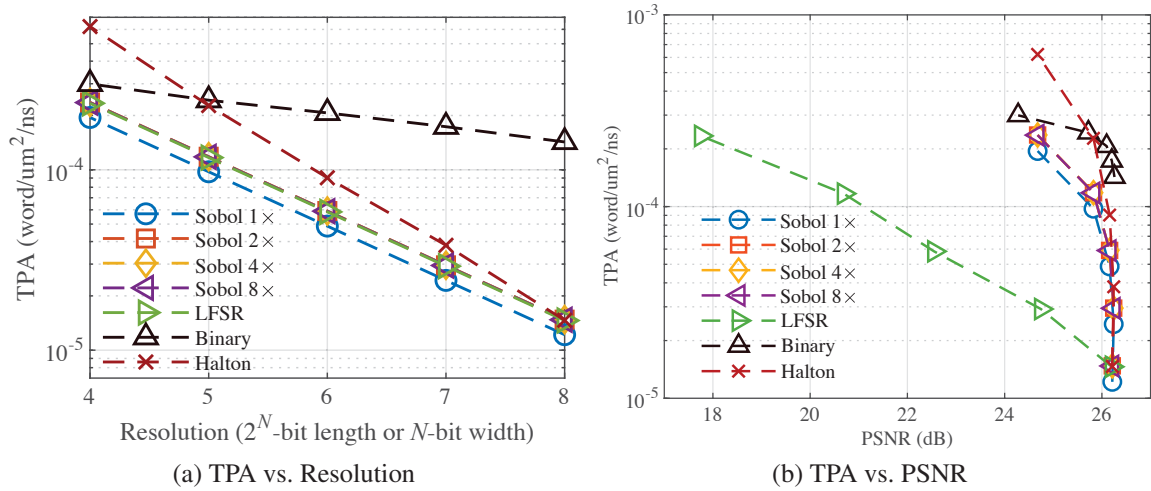


Figure 3.34. TPA comparison of different median filter implementations.

binary counterparts at a similar processing quality (PSNR as shown in Fig. 3.33(b)) except for the 8-bit binary design. For the same resolution, the hardware efficiency in terms of EPO, TPA and runtime for the binary circuits is proportional to the bit width  $N$ , whereas it is proportional to  $2^N$  for the stochastic circuits. Therefore, the slope of the EPO curve in Fig. 3.33 for the binary circuits is less steep than those for the stochastic circuits.

However, the TPA and runtime still have the shortcomings of a stochastic MF. The stochastic MF is only faster with 16-bit sequences and  $8\times$  parallelization than 4-bit binary design. Due to the extra hardware cost of the comparators and the increased critical path

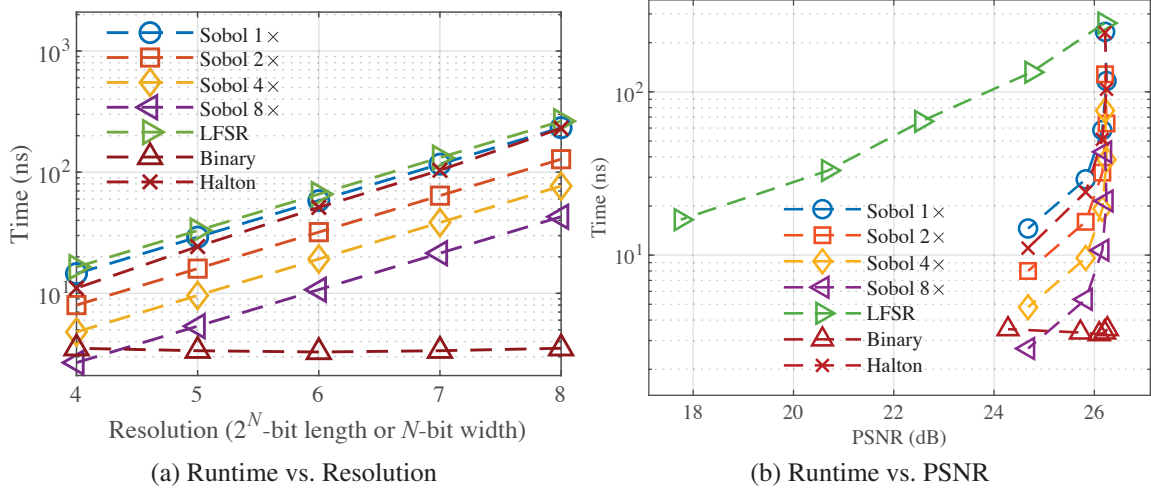


Figure 3.35. Runtime comparison of different median filter implementations.

delay, the TPA of a Sobol MF design is not significantly improved by the parallelization. However, all parallel Sobol designs consistently show advantages over the LFSR-based designs for all considered metrics with a better image processing quality. The designs using Halton sequences require a similar EPO to  $2\times$  parallel Sobol designs, while they are not as efficient as designs using Sobol sequences with  $4\times$  or larger parallelization.

### 3.7 Summary

The use of parallel Sobol sequences in SC achieves higher energy efficiency, higher throughput and shorter runtime than the use of conventional LFSR-generated sequences and the Halton sequence, another type of LD sequence. The parallelization exploits the regular patterns of the LSZ positions of continuous nonnegative integers. As a result, it imposes only a relatively small hardware overhead of a few XOR gates.

The proposed stochastic circuits using parallel Sobol sequences are applied and evaluated in a sorting network and an MF. The stochastic MF using Sobol sequences shows a higher energy efficiency than its binary counterparts with resolutions of 7 bits and below and it consistently outperforms an LFSR-based MF with a higher filtering quality.

## Chapter 4

# Dynamic Stochastic Computing for Digital Signal Processing Applications

Stochastic computing (SC) utilizes a random binary bit stream to encode a number by counting the frequency of 1's in the stream (or sequence). Typically, a small circuit is used to perform a bit-wise logic operation on the stochastic sequences. Energy efficiency, however, is a challenge for SC due to the long sequences required for accurately encoding numbers. To overcome this challenge, we consider to use a stochastic sequence to encode a continuously varying signal instead of a fixed number to achieve higher accuracy, higher energy efficiency and greater flexibility. Specifically, one single bit is used to encode a sample from a signal for efficient processing. This type of sequences encodes constantly varying values, so it is referred to as dynamic stochastic sequence (DSS). The DSS enables the use of SC circuits to efficiently perform tasks such as frequency mixing, function estimation, filtering and integration of the sampled signal.

### 4.1 Introduction

In SC, a random binary bit stream or a stochastic sequence is used to encode a number. The arithmetic circuits that are used to process the sequences are area- and power-efficient because complex functions can be implemented by simple logic circuits to process one bit at each clock cycle. However, most of the designs rely on relatively long stochastic bit streams to obtain a high accuracy, thus resulting in an inferior performance and low energy efficiency compared to conventional arithmetic circuits. This performance bottleneck stems from the fundamental principle of the digital encoding of analog or

continuous signals in SC, which, by itself, is likely overly ambitious to attain without efficient encoding techniques. In [89, 103], a  $\Delta - \Sigma$  modulated bit stream is used as a stochastic sequence for filtering, multiplication and image processing. However, the  $\Delta - \Sigma$  modulated bit streams suffer from high signal correlations and degrade the accuracy of the results [103].

In this chapter, we propose a new type of SC that uses a DSS to encode a continuously variable signal rather than a static number. In such a sequence, the bit values show a dynamical pattern that constantly changes with the signal amplitude. To further improve the accuracy, quasirandom numbers are used to generate the dynamic sequences. In the resulting dynamic stochastic computing (DSC), the length of the sequence is dramatically reduced compared to conventional stochastic sequences. This reduction significantly saves the energy consumption compared to conventional SC. This new encoding technique enables the use of single-bit SC circuits for low-power computing of many digital signal processing (DSP) functions.

## 4.2 Digital signal processing systems

A DSP system usually requires analog-to-digital converters (ADCs) and digital-to-analog converters (DACs). An ADC is used to convert an analog signal to a digital signal while a DAC converts a digital signal to an analog signal. Due to the conversion from an analog signal to a digital signal, the difference between an analog signal,  $x$ , and the quantized digital signal,  $X[n]$ , is referred to as a quantization error ( $e$  in Fig. 4.1), caused by the limited resolution of the digital representation. It can be viewed as an additive noise, as shown in Fig. 4.1(b). The digital signal from the ADC is stored or directly processed by a DSP circuit. After the processing, it is stored or converted back by a DAC to an analog signal, as shown in Fig. 4.1(a). However, the DSP circuits often consist of relatively expensive multipliers and adders.

One type of the sampling techniques uses oversampling to reduce the quantization error; it is referred to as the  $\Delta - \Sigma$  modulation. For the  $\Delta - \Sigma$  modulated bit stream, it is generated by a  $\Delta - \Sigma$  modulator (DSM), and it can be used directly for signal processing [113]. The block diagram of a DSM is shown in Fig. 4.2(a). The error between the original digital



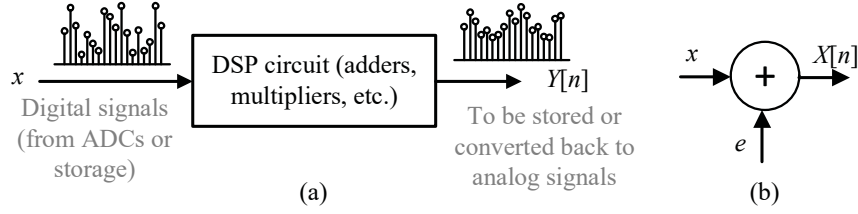


Figure 4.1. (a) A typical DSP system; (b) a model of quantization error.

signal  $x(n)$  and the feedback signal,  $X[n]$ , is integrated over time, producing the signal  $u(n)$ .  $u(n)$  is then quantized to generate next bit in the sequence. The quantization noise ( $e(n)$ ) produced by the quantizer can be viewed as an additive white noise as shown in Fig. 4.1(b). Then, the DSM in Z-domain can be modeled as Fig. 4.2(b). When the quantization noise is zero, the signal transfer function (STF) in z-domain is given by [114]

$$\text{STF} = \frac{X(z)}{x(z)} = z^{-1}; \quad (4.1)$$

while, similarly, the quantization noise transfer function (NTF) is given by [114]

$$\text{NTF} = \frac{X(z)}{e(z)} = 1 - z^{-1}. \quad (4.2)$$

Since the STF has a value of  $z^{-1}$ , it indicates that the DSM preserves the original information of the signal with a single delay; however, for the quantization noise, the DSM serves as a high-pass filter as per (4.2). It is called a “noise-shaping” effect: it “pushes” the low-frequency power of the quantization noise to a high frequency. Later, a low-pass filter can be used to reduce the high-frequency power of the quantization noise to achieve a high accuracy [115]. This type of bit stream has recently been used in SC for filtering, multiplication and image processing [103].

### 4.3 Proposed dynamic stochastic computing systems

In a DSC system, a digital signal is first converted to a DSS by a dynamic stochastic number generator (DSNG). Then the bit sequence is processed by the stochastic circuits and converted back to a digital signal by a signal reconstruction unit, as shown in Fig. 4.3. It is generally assumed that a digital signal is available from an ADC or storage. If an analog signal is directly used as an input, the DSNG can be replaced with an analog



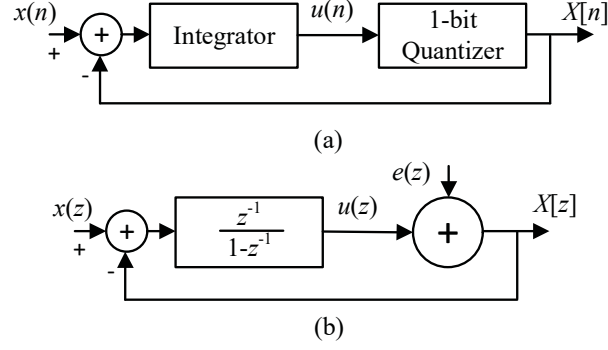


Figure 4.2. (a) Block diagram of a DSM; (b) Z-domain model of the DSM.

design such as the stochastic number generator (SNG) in [60]. In this section, the generation of the DSS, the reconstruction of the output signal and general aspects of DSC circuits are discussed.

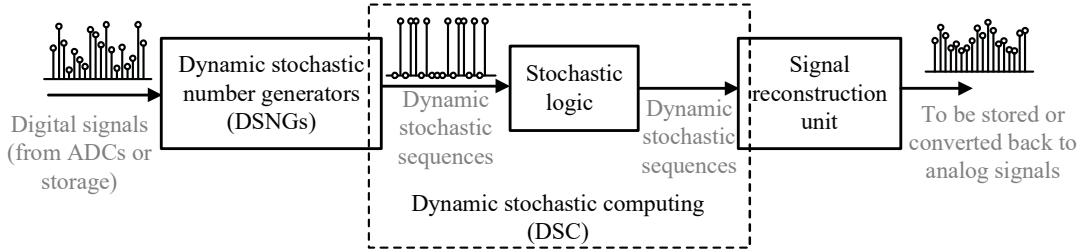


Figure 4.3. A DSC system.

### 4.3.1 Generation of the dynamic stochastic sequence

**Definition 1.** Let  $0 \leq f(t) \leq 1$  be a continuous time-domain signal. After a sampling with a clock period of  $T$ , it can be converted to a sequence of  $\{f(kT)\}$ ,  $k = 0, 1, \dots$ . A DSS  $\{A_k\}$  encoding  $f(t)$  satisfies that the  $k$ th bit in the random binary sequence has the expectation,

$$\mathbb{E}[A_k] = f(kT), \quad (4.3)$$

where  $T^1$  is the sampling period and  $1/T$  is the sampling rate.

A DSS can be generated by comparing each element in the sequence ( $\{f(kT)\}$ ) with a uniformly distributed random number within  $[0, 1]$ . Therefore, the expectation of the  $k$ th

<sup>1</sup> $T$  will be used as the notation for sampling period throughout this chapter.

bit in a DSS is  $f(kT)$ . Similar to a conventional SNG, the diagram of a DSNG is shown in Fig. 4.4. Also, a DSS can also be generated by an SC-based ordinary differential equation

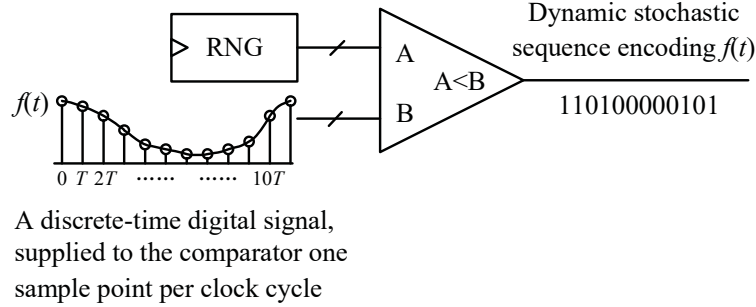


Figure 4.4. A DSNG.

(ODE) solver, as presented in Chapter 5. The generation of a  $\Delta - \Sigma$  modulated signal can be considered as a Bernoulli process [113] so that  $\Delta - \Sigma$  modulated signal can be considered as a type of DSS's as well. In these cases, a  $\Delta - \Sigma$  modulated signal or the stochastic sequence produced by a stochastic integrator can be considered to form approximately a Bernoulli sequence satisfying (4.3) [116]. Fig. 5.1 presents several examples of using the ODE solvers to generate DSS's following the formulation in (5.13) and the definition of a DSS. For example, in Fig. 5.1(b), the solution is given by  $y_1(t) = 0.5 - 0.5\cos(t)$  and  $y_2(t) = 0.5 + 0.5\sin(t)$ . According to the function of a stochastic integrator, the expectation of the  $k$ th bit in  $Y_1$ ,  $\mathbb{E}[Y_{1,k}] \approx 0.5 - 0.5\cos(k/2^n)$ , where  $n$  is the width of the counter in the stochastic integrator. So  $Y_1$  approximately encodes  $0.5 - 0.5\cos(t)$  with a sampling period of  $1/2^N$  and, similarly,  $Y_2$  approximately encodes  $0.5 + 0.5\sin(t)$ . In Fig. 5.1(c), the solution of the ODE is given by  $y(t) = te^{-t}$ , so that  $Y$  approximately encodes  $te^{-t}$ .

If the signal value lies within  $[-1, 1]$ , a linear mapping can be applied for the bipolar representation.

### 4.3.2 Reconstruction of the DSS

The outputs of combinational stochastic circuits are still stochastic sequences. To convert a DSS back into a digital signal, a reconstruction unit is required. The DSS can be reconstructed to a digital signal by using either a moving average (MA) circuit, an exponential smoothing circuit [117] or a stochastic integrator.

## Moving average

The DSS's can be reconstructed by an MA operation as

$$\hat{x}(t_k) = \frac{1}{2^N} \sum_{i=k-2^{N-1}}^{k+2^{N-1}-1} X_i, \quad (4.4)$$

where  $\hat{x}(t_k)$  is the reconstructed signal at time  $t_k = kT$ .  $2^N$  is the length of the averaging window, which allows for an easy division by shifting.

In this design, the MA is implemented by an up/down counter and a  $2^N$ -bit shift register, as shown in Fig. 4.5. The input is the DSS to be decoded. The stochastic bits  $X_{k+2^{N-1}-1}$  and  $X_{k-2^{N-1}-1}$  are the input and the output of the shift register respectively, with all the other bits between these two bits stored in the shift register by  $N$ .

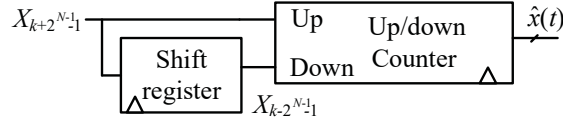


Figure 4.5. An MA circuit.

From (4.4), it can be obtained that

$$\begin{aligned} \hat{x}(t_k) - \hat{x}(t_{k-1}) &= \frac{1}{2^N} \left[ \sum_{i=k-2^{N-1}}^{k+2^{N-1}-1} X_i - \sum_{i=k-2^{N-1}-1}^{k+2^{N-1}-2} X_i \right] \\ &= \frac{1}{2^N} (X_{k+2^{N-1}-1} - X_{k-2^{N-1}-1}), \end{aligned} \quad (4.5)$$

which is computed by the up/down counter. If  $X_{k+2^{N-1}-1}$  is 1 and  $X_{k-2^{N-1}-1}$  is 0, the counter counts up; if  $X_{k+2^{N-1}-1}$  is 0 and  $X_{k-2^{N-1}-1}$  is 1, the counter counts down; otherwise, the counter keeps its value. The value stored in the counter is then normalized by  $2^N$  to obtain  $\hat{x}(t_k)$ . A major drawback of this method is that the hardware cost of the shift register grows exponentially with  $N$  to store a long sequence [117]. However, this reconstruction scheme works similar to the rate coding as shown in Table 1.1.

## Exponential smoothing

The exponential smoothing operation is implemented by an ADaptive DIgital Element (ADDIE) for a signal reconstruction [117], as shown in Fig. 4.6.

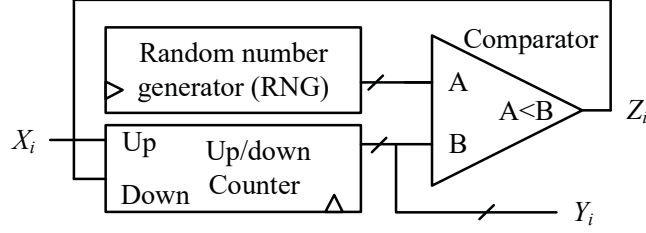


Figure 4.6. An ADDIE.

Following [117], the exponential smoothing function is formulated as follow. Let the output of the comparator be  $Z_i$  (0 or 1) at clock cycle  $i$ , the input bit be  $X_i$ , and the multi-bit integer value stored in the counter be  $Y_i$ . The random number generator (RNG) and the comparator work as an SNG, so the probability of generating a ‘1’ is  $Y_i/2^N$ , where  $N$  is the bit-width of the counter, i.e.,  $y_i = \mathbb{E}[Z_i] = Y_i/2^N$ . As per the function of the up/down counter,  $Y_{i+1} = Y_i + X_i - Z_i$  [89]. Therefore, given  $Y_i$ , the expectation of  $Y_{i+1}$  is

$$\mathbb{E}[Y_{i+1}] = (2^N - 1)Y_i/2^N + \mathbb{E}[X_i] \quad (4.6)$$

by taking expectations of the  $X_i$  and  $Z_i$ . Assume  $Y_0$  is 0, and let  $Y_{i+1} \approx \mathbb{E}[Y_{i+1}]$  for  $i = 0, 1, \dots$ , (4.6) can be rewritten as

$$y_{i+1} = \frac{1}{2^N}Y_{i+1} \approx \frac{1}{2^N} \sum_{k=0}^i \left[ \left( \frac{2^N - 1}{2^N} \right)^k \mathbb{E}[X_{i-k}] \right], \quad (4.7)$$

where the coefficients of  $\{\mathbb{E}[X_{i-k}]\}$  form a geometric sequences. It indicates an exponential smoothing. Then,  $\{y_i\}$  is an estimate of the encoded signal. However, to reconstruct the signal instead of filtering it, the width of the counter  $N$  needs to be carefully selected because the same circuit can be used as a low-pass infinite impulse response (IIR) filter [89], which may attenuate the encoded signal. A sinusoidal signal reconstructed by an ADDIE is shown in Fig. 4.7. It requires a warm-up phase to allow the integrator to follow the signal if it is initialized with a random value other than the actual initial value. It is shown in Fig. 4.7 at the start of the signal. On the other hand, a long shift register is not required for the ADDIE, thus reducing hardware cost in most cases (with a large  $N$  value) compared to the MA circuit.

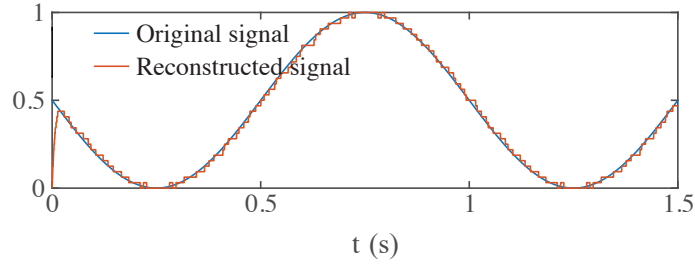


Figure 4.7. Original and the reconstructed signals by using an ADDIE.

### Stochastic integrator

When a stochastic integrator produces the output of a DSC system, a reconstruction unit is not required as the integration results encoded by the counter, i.e.,  $Count_{out}$  in Fig. 2.6(a), can directly be used as the output. It is applicable to certain tasks, such as a numerical integration or an IIR filter as discussed later in 4.4.3 and 4.4.4.

### 4.3.3 Dynamic stochastic computing circuits

DSC circuits are efficient and flexible to implement a series of function compositions. For example, a combinational SC circuit implementing the function  $f(x)$  can be used to implement the function composition  $f[\chi(t)]$  with the input DSS encoding the signal  $\chi(t)$ . This principle also applies to multi-input combinational circuits.

For a finite state machine (FSM)-based sequential circuit, as shown in Fig. 3.12, however, the output does not accurately encode  $f[\chi(t)]$  because the FSM-based circuit requires that each bit in the stochastic sequence is generated independently. The adjacent bits in a DSS are correlated and violate this requirement, as discussed in Chapter 3. On the other hand, a stochastic integrator does not require this due to the accumulation of bit values, so it works well with the DSS's.

## 4.4 DSC-based DSP

Based on the proposed DSS and DSC system, several applications in DSP using DSC are discussed in this section.

### 4.4.1 Frequency mixer

Conventionally, a frequency mixer is implemented by an analog multiplier consisting of nonlinear components. New signals at the summation and difference of the original frequencies are then produced. Using the DSS's, a stochastic multiplier is proposed to implement a frequency mixer, as shown in Fig. 4.8(a). The x-axis denotes time, which is measured by seconds<sup>1</sup>. As per (4.3), if the input sequences  $X$  and  $Y$  are statistically independent and  $X_k$ ,  $Y_k$  and  $Z_k$  are the  $k$ th bits in the sequences  $X$ ,  $Y$  and  $Z$ , respectively, we obtain  $\mathbb{E}[Z_k] = \mathbb{E}[X_k]\mathbb{E}[Y_k] = x(kT)y(kT)$  at clock cycle  $k$  for the output sequence. Therefore, the output sequence  $Z$  encodes  $z(t) = x(t)y(t)$ , which is the product of the two signals,  $x(t)$  and  $y(t)$ .

Fig. 4.8(b) shows the output results when multiplying two sinusoidal signals with frequencies of 1 Hz and 6 Hz, both sampled at a rate of  $2^{12}$  Hz. The DSS is reconstructed to a digital signal by an MA operation with an averaging window of length 128.

As shown in Fig. 4.8(b), the results produced by the stochastic circuit are very close to the results produced by using double precision numbers with a signal-to-noise ratio (SNR) of 25 dB. It also shows that there are more 1's (or 0's) in the DSS's when the original signal is closer to 1 (or 0). The DSS's are decimated for clarity.

### 4.4.2 Approximation of functions

Bernstein or multilinear polynomials have been implemented in SC either by a multiplexing circuit or a Boolean function with auxiliary inputs [1, 40]. By using the DSS's, more complex functions can be implemented with the same circuit. In this section, a multiplexing circuit consisting of an accumulator and a multiplexer is discussed as an example. Fig. 4.9(a) shows a multiplexing circuit that computes a Bernstein polynomial,  $f(x) = 1/11(2x^3 + 3x^2 + 6x)$ , where all the input sequences to the accumulator encode a fixed value  $x$ . However, the same circuit can be used to compute the function composition,  $f[x(t)]$ , when DSS's encoding  $x(t)$  serve as the inputs to the accumulator. It is shown as follows. The expectation of the  $k$ th bit in the output sequence  $Y$  can be obtained as  $\mathbb{E}[Y_k] = \sum_{i=0}^n b_i \binom{n}{i} x^i(t) (1 - x(t))^{n-i}$ , where  $\{b_i\}$  are the Bernstein coefficients

---

<sup>1</sup>This also applies to later figures that show the time-domain results.

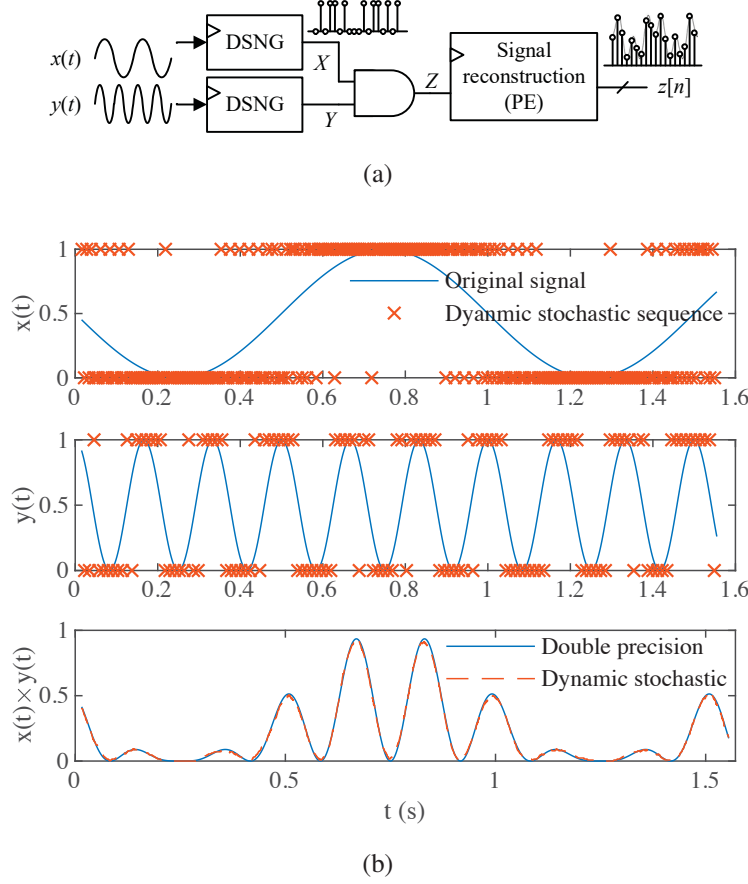


Figure 4.8. A frequency mixer by using a stochastic multiplier with DSS's as inputs, (a) circuit and (b) mixed output.

( $\{1, 2/11, 5/11, 0\}$  in Fig. 4.9(a)) encoded by the inputs of the multiplexer and  $n$  is the order of the Bernstein polynomial [1]. The value of the function  $f[x(t)]$  at  $t = kT$  is then equal to  $\mathbb{E}[Y_k]$ , so the output sequence encodes  $f[x(t)]$ .

When  $x(t) = e^{-2t}$ , the output sequence of the multiplexer encodes  $f[x(t)] = 1/11(2e^{-6t} + 3e^{-4t} + 6e^{-2t})$ . Fig. 4.9(b) shows the reconstructed result that is produced when using an MA with an averaging window of length 256. The reconstructed signal has an SNR of 29 dB. The sequences for the coefficients of the Bernstein polynomial, i.e., the inputs to the multiplexer, are generated by an SNG as in a conventional SC circuit.

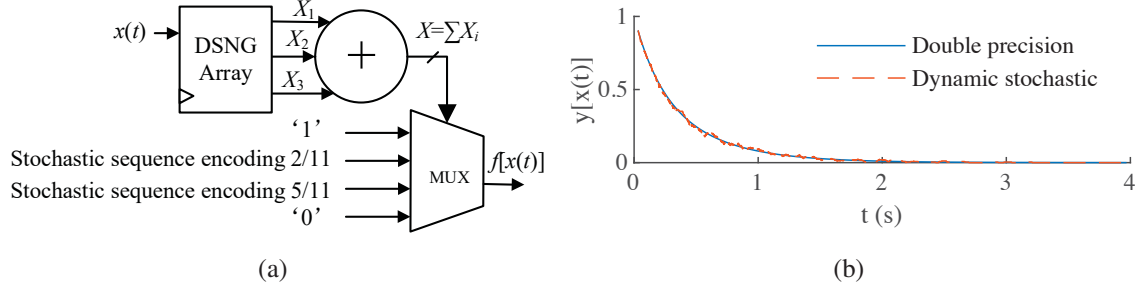


Figure 4.9. A summation of several exponential functions computed by (a) a multiplexing circuit and (b) the output results.

### 4.4.3 Numerical integration

Integration is an important operation in DSP. To process the single-line DSS encoding the signal to be integrated, the original two-input stochastic integrator is modified in this design. For the unipolar representation, the “DEC” port of the stochastic integrator is connected to ‘0’ in Fig. 2.6(a), while the other input is the DSS encoding the signal to be integrated; for the bipolar representation, the stochastic integrator is modified as shown in Fig. 4.10. When the input is a ‘1’, the counter counts up; otherwise, it counts down.

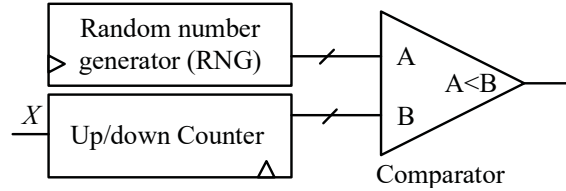


Figure 4.10. A single-input bipolar stochastic integrator for numerical integration.

The modified stochastic integrator provides an unbiased estimate to the Riemann sum [118], which is an approximation of the integral. As the modified bipolar stochastic integrator is similar to the unipolar design, only the function of the unipolar design is shown as follows.

By the Riemann sum, the integral over the interval from 0 to  $nT$  is approximated by

$$\int_0^{nT} f(\tau) d\tau = T \sum_{k=0}^{n-1} f(kT) \quad (4.8)$$

where  $\{f(kT)\}$  ( $k = 0, 1, \dots, n-1$ ) is the sampled signal. Taking (4.3) into (4.8) leads to

$$\int_0^{nT} f(\tau) d\tau = T \sum_{k=0}^{n-1} \mathbb{E}[A_k], \quad (4.9)$$



where  $A_k$  is the  $k$ th bit in the DSS, which encodes the signal to be integrated. Thus, the integral can be approximated by

$$\int_0^{nT} f(\tau) d\tau \approx T \sum_{k=0}^{n-1} A_k. \quad (4.10)$$

The approximation is partly reduced due to the bit accumulation on the right hand side of (4.10) and the accumulation is performed by the counter in Fig. 4.10.  $T$  can be set to a 2's negative power for ease of computation.

Fig. 4.11 shows the results of an integral of a sinusoidal signal. The signal to be integrated is sampled at a rate of 256 Hz and converted to a DSS. The integration is achieved by using the bipolar stochastic integrator in Fig. 4.10. The SNR of the result reaches 34 dB. Since the integrated results are directly provided by the counter, a reconstruction unit is not required in this case.

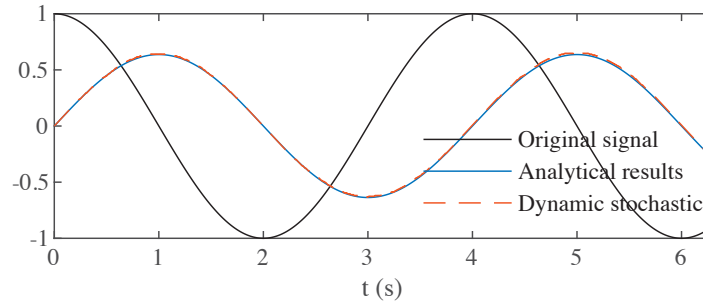


Figure 4.11. The input sequence (not shown) encodes the original signal  $\cos(\pi t/2)$ . The results produced by the stochastic integrator are close to the analytical results.

#### 4.4.4 Low-pass IIR filter

In [89], it is shown that an ADDIE [4] with different parameters can be used as a low-pass IIR filter with different frequency responses. The ADDIE is essentially a stochastic integrator with a feedback signal as its input, as shown in Fig. 4.6.

If the counter is  $N$ -bit wide, the transfer function of the IIR filter in the  $Z$ -domain is

$$H(Z) = \frac{1}{2^N Z + 1 - 2^N}, \quad (4.11)$$

which indicates a stable first-order low-pass IIR filter [89]. To filter a signal, it is first converted to a DSS; the obtained sequence is then the input to the port  $X$  of ADDIE. The

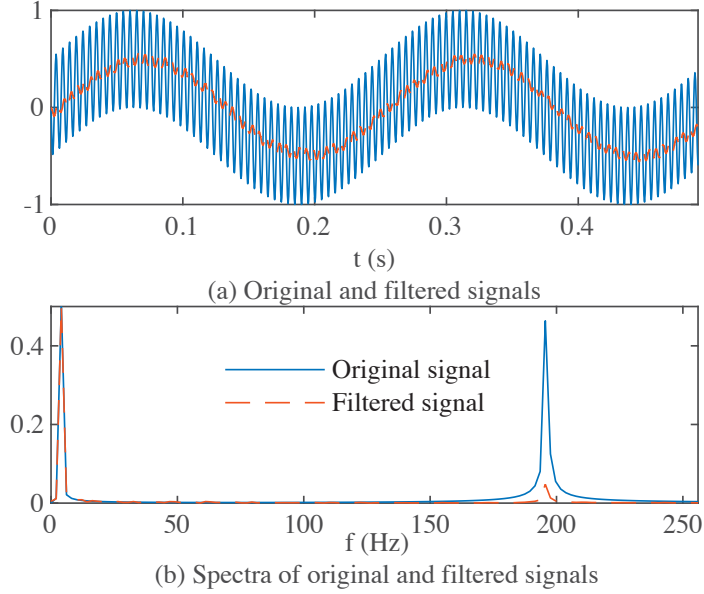


Figure 4.12. The filtering results in (a) time domain and (b) frequency domain.

output of the counter in the ADDIE provides the filtering result by a normalization factor of  $2^N$ .

In this section, the IIR filter is tested by filtering mixed sinusoidal signals of  $f_1 = 4$  Hz and  $f_2 = 196$  Hz. The input DSS encodes the mixture by a sampling rate of 65.5k Hz and it is filtered by an 8-bit ADDIE. The filtering results are shown in Fig. 4.12 in both the time and frequency domains. As shown in the results, the signal with a higher frequency is almost filtered out. Again, since the output signal is provided by the counter in the ADDIE, a reconstruction unit is not required.

## 4.5 Optimization of sequence generation and signal reconstruction

### 4.5.1 Optimal sampling rates and reconstruction parameters

The DSS can be reconstructed to a digital signal by either the MA or the exponential smoothing. In an MA, a long averaging window, i.e., a large  $N$  in (4.4), leads to a strong smoothing effect. However, the interval  $[(k - 2^{N-1})T, (k + 2^{N-1})T]$  is larger when  $N$  is larger; therefore, it may not provide an accurate estimate to  $x(t)$  at  $t = kT$  by considering too much information before and after time  $t$ . For the exponential smoothing using ADDIE,

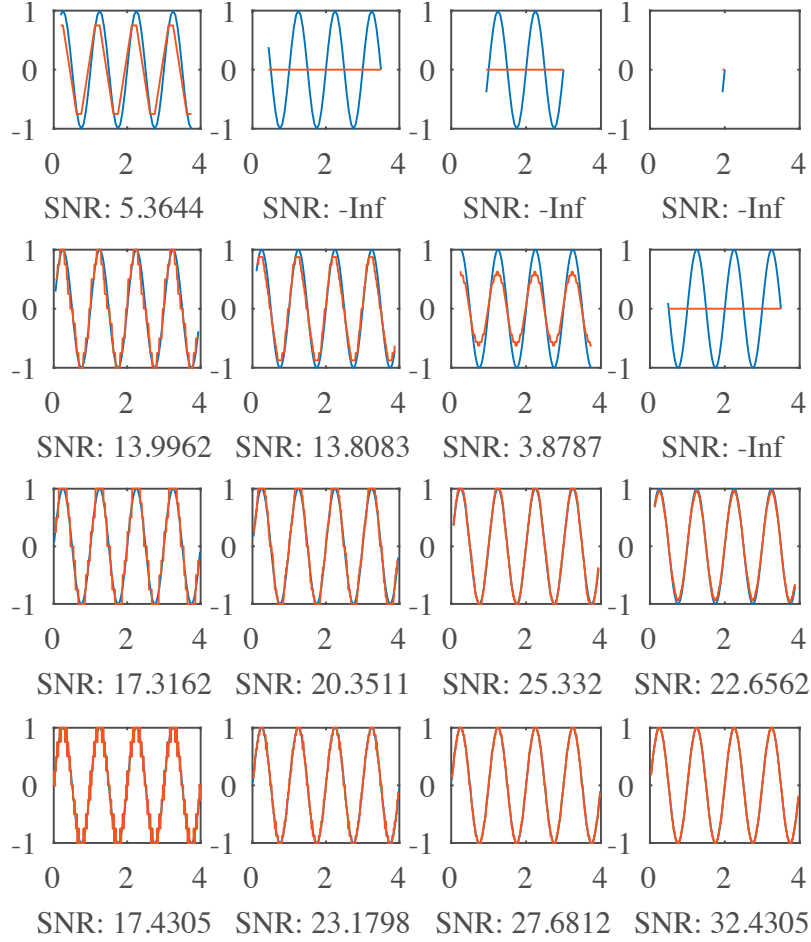


Figure 4.13. Original signal (blue line) and the reconstructed signal (red line) by an MA with different parameters under different sampling rates. Sampling rates from top to bottom rows:  $2^4, 2^6, 2^8, 2^{10}$  Hz. MA window lengths from left to right columns:  $2^3, 2^4, 2^5, 2^6$ . The SNRs are measured in dB.

the reconstructed signal tends to be polluted by the quantization error when the width,  $N$ , of the ADDIE is small; on the other hand, when the width is large, the ADDIE functions more like a filter than a reconstruction unit as per (4.11). So, the optimal reconstruction parameter  $N$  must be numerically analyzed for both cases with respect to the sampling rate of the original signal. For a better accuracy and more stable measurement, the Sobol sequences are used for the analysis in this section. Different sampling rates and reconstruction parameters are explored for different methods, as shown in Figs. 4.13 and 4.14. The bipolar representation is used and the accuracy is measured by SNR.

Fig. 4.13 shows that the smoothing effect is too strong, such that the signal is attenuated or filtered, when the averaging window is too long. For example, when the sampling rate

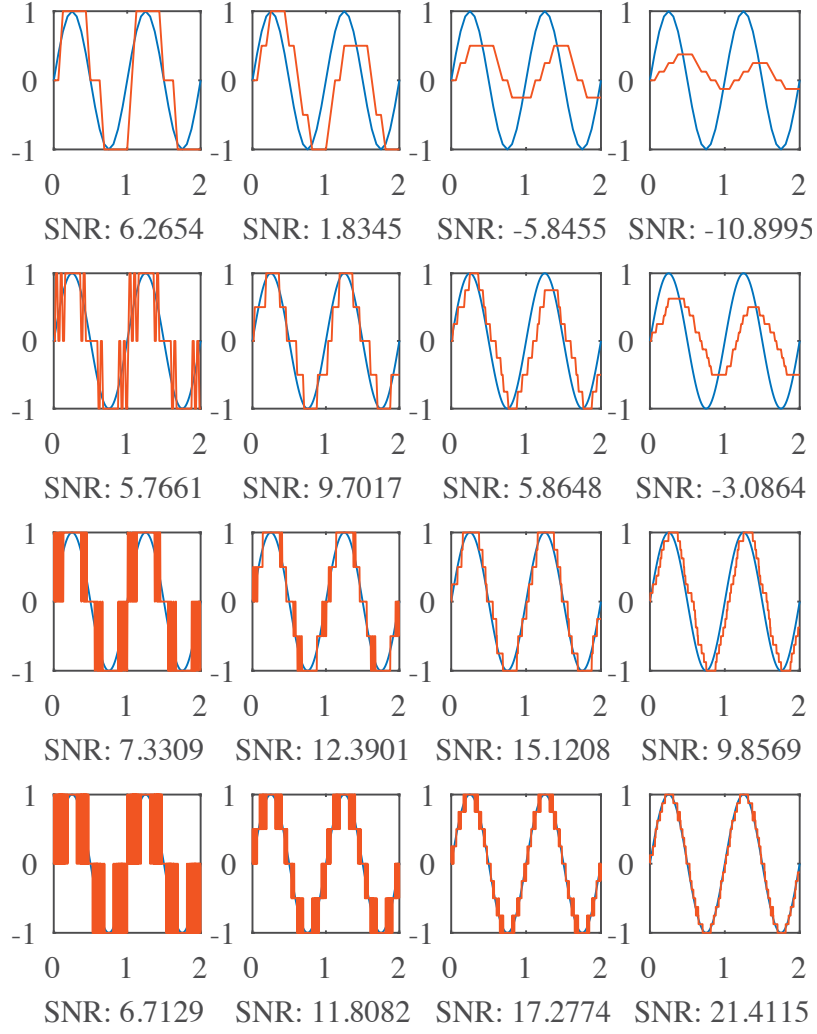


Figure 4.14. Original signal (blue) and the reconstructed signal (red) by ADDIEs with different parameters under different sampling rates. Sampling rates from top to bottom:  $2^4, 2^6, 2^8, 2^{10}$  Hz. Bit widths of the ADDIE from left to right: 2, 3, 4, 5, including one sign bit. The SNRs are measured in dB.

is  $2^6$  Hz, a window length of  $2^5$  or  $2^6$  is too long to fully reconstruct the signal. Moreover, since the original signal is periodic, the results become a straight line when the length of the window is a multiple of the period of the sampled signal. For example, when the sampling rate is  $2^4$  Hz and the averaging window length is  $2^4$  or  $2^5$ , the MA produces a straight line instead of a sinusoidal signal. Also, when the sampling rate is  $2^4$  with a window length of  $2^6$ , the signal between the short time interval,  $[0, 4]$ , can hardly be reconstructed due to the short time interval and a lack of sample points. Given a fixed window length, a larger sampling rate tends to result in a larger SNR. Generally, the optimal averaging window

length is about  $2^{n+1}$  when the sampling rate is  $2^{2n}$  Hz as empirically determined from the experimental results, and a sampling rate of at least 32 Hz is required to reconstruct a 1 Hz signal with an SNR higher than 13 dB. However, this result may vary for different types of encoded signals.

The results using ADDIEs (exponential smoothing) for signal reconstruction are shown in Fig. 4.14. When the width of the ADDIE is not well chosen, the amplitude of the reconstructed signal is reduced by an unwanted filtering effect. Also, the optimal width for the ADDIE has a strong linear relationship with the logarithm of the sampling rate. The optimal width of the ADDIE for reconstructing the signal is about  $n$  when the sampling rate is  $2^{2n}$  Hz, as empirically determined from the experimental results.

#### **4.5.2 Pseudorandom sequences vs. quasirandom sequences vs. $\Delta - \Sigma$ modulated sequences**

In [41, 119], it is shown that the use of quasirandom numbers improves the accuracy of SC with shorter stochastic sequences. Also, a  $\Delta - \Sigma$  modulated sequence can be considered as a Bernoulli sequence [116]. It resembles a DSS and, therefore, can be used in SC [89]. In Fig. 4.15, these types of stochastic sequences are tested for sequence generation and signal reconstruction. The original signal used is a 1-Hz sinusoidal signal. It is sampled with a sampling rate of  $2^{10}$  Hz to reduce the inaccuracy caused by under-sampling. Different methods for reconstruction, i.e., by an MA (or an ADDIE) with an optimal averaging window length (or an optimal width), are also considered. The SNRs are reported to indicate the accuracy.

As shown in Fig. 4.15, the signals generated or reconstructed by the linear-feedback shift registers (LFSRs) are most heavily corrupted by random noise except that they are both generated and reconstructed by the same LFSR. In Fig. 4.15(i), the signal is generated and reconstructed using the same sequence so only one LFSR is required and the accuracy is improved due to the reduced variance [120]; so is the case with Fig. 4.15(e). The signal reconstructed by an MA with an averaging window length of 64 generally has a higher accuracy than the ones reconstructed by a 5-bit ADDIE because the use of a wider counter provides a higher resolution. For the ADDIE, a higher sampling rate is required to obtain an optimal reconstruction if a larger counter is desired for a

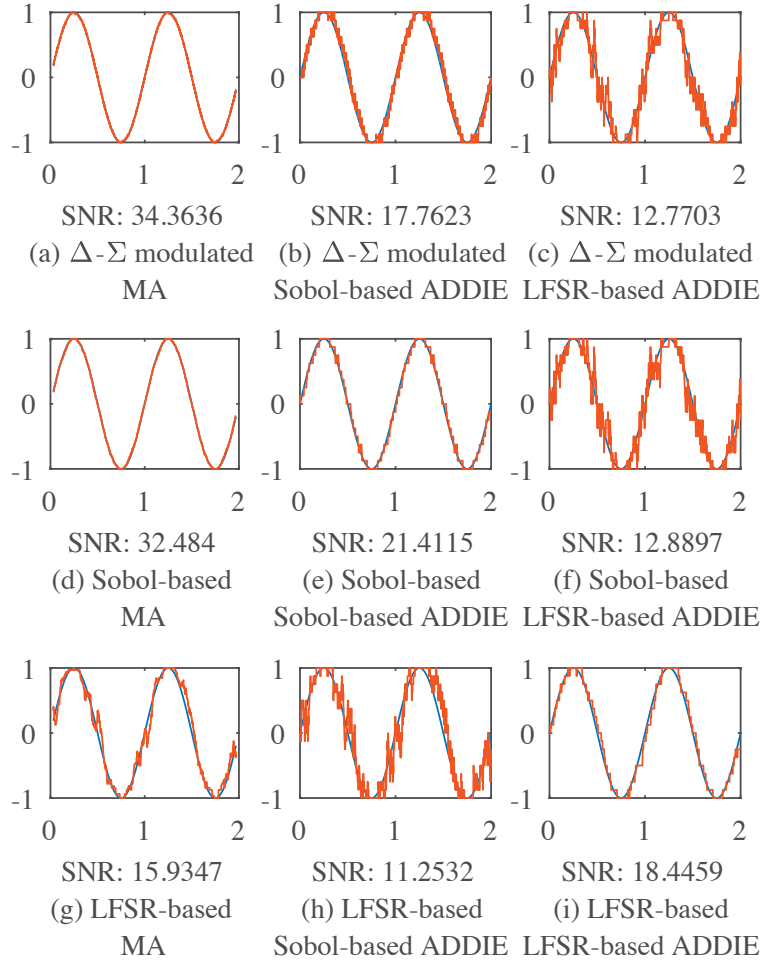


Figure 4.15. DSS's generated and reconstructed using different methods. The first line of the sub-caption indicates how the sequences are generated and the second line indicates how they are reconstructed. The blue line shows the original signal while the red line shows the reconstructed signals. The results in the same row use the same generation method and the results in the same column share the same reconstruction method. The MAs all have a window length of  $2^6$  and the ADDIEs all have a width of 5. The SNRs are measured in dB.

better resolution. The signal generated by  $\Delta - \Sigma$  modulation and reconstructed by an MA has the highest accuracy overall because it is generated and reconstructed in a deterministic manner, thus being the least influenced by the random noise.

## 4.6 Hardware efficiency assessment

Conventionally, signal multiplication, function approximation and signal filtering are implemented using either analog or digital circuits. However, an analog circuit is hard to

integrate with digital systems and is vulnerable to noise and mismatch; a digital circuit may require expensive multipliers in a conventional binary implementation. The proposed DSC provides a potential solution for efficient signal processing. In this section, the hardware efficiency is evaluated for the frequency mixer and the function estimator. The resolution of the results (in bit width) are matched. The area, power and critical path delay are measured using Synopsys Design Compiler (DC) with a 28-nm industrial process. The detailed parameters are summarized in Appendix A.

#### 4.6.1 Frequency mixer

Two sinusoidal signals with frequencies of 1 Hz and 6 Hz are multiplied to measure the SNR of the results produced by different SC systems. Although an MA circuit produces a higher accuracy than the ADDIE-based signal re-constructor with the same bit width, its hardware consumption increases exponentially with the bit width, as discussed in [117], so an ADDIE is used to reconstruct the signal for a smaller hardware cost. With a sampling rate of  $2^{16}$  Hz (resulting in a dynamic sequence length of  $k \cdot 2^{16}$  bits for a  $k$ -second signal) and a 5-bit ADDIE, the DSC circuit produces an SNR of 24.20 dB, as shown in Figure 4.16(a).

For the conventional SC (CSC) circuit, the same sampling rate is used, and then the same low-discrepancy sequences are applied. With a sequence length of  $2^5$  encoding each sampling point, the CSC circuit produces an SNR of 23.30 dB, as shown in Fig. 4.16(b). A binary implementation using 5-bit fixed-width multiplication is also considered and the result is shown in Fig. 4.16(c).

As shown in Fig. 4.16, the results produced by the DSC circuit show stronger variations because the signal reconstructor consistently tracks the input signal, which makes it very sensitive to the change of the input sequence. The DSC frequency mixer produces a similar or a slightly higher SNR than the CSC circuit. However, due to the effect of limited precision, the fixed-width multiplication by a binary circuit produces the lowest quality.

The hardware evaluation results are shown in Table 4.1. For the stochastic designs, the RNG (low-discrepancy (LD) sequence generator) can be shared to generate multiple stochastic sequences, so the cost is negligible when considering a large SC system and it is

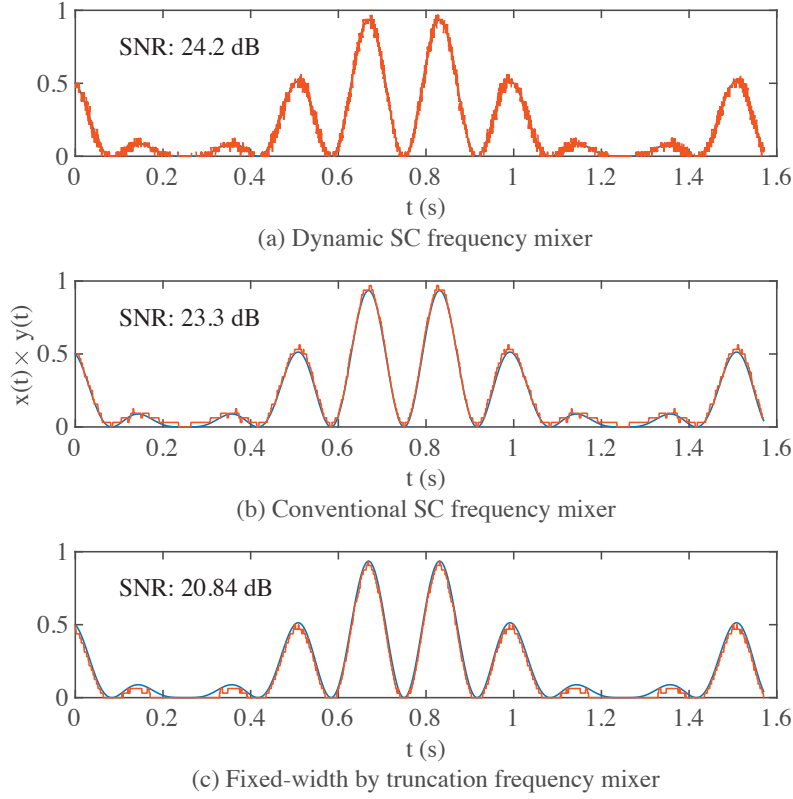


Figure 4.16. Frequency mixer: DSC vs. CSC vs. fixed-width binary implementation.

Table 4.1. Hardware efficiency evaluation of dynamic and conventional stochastic frequency mixer producing one result

|                    | Area<br>( $\mu m^2$ ) | Power<br>( $\mu W$ ) | No. of clk<br>cycles | Minimum<br>time (ns) | Energy<br>(fJ) | SNR<br>(dB) |
|--------------------|-----------------------|----------------------|----------------------|----------------------|----------------|-------------|
| Dynamic SC         | 81.76                 | 12.70                | 1                    | 0.70                 | 50.79          | 24.20       |
| Conventional SC    | 66.59                 | 10.27                | 32                   | 9.28                 | 1314.85        | 23.30       |
| Fixed-width binary | 79.64                 | 13.79                | 1                    | 0.72                 | 55.18          | 20.84       |
| Ratio (Dyn.:Conv.) | 1.23                  | 1.24                 | 0.031                | 0.075                | 0.039          | –           |

not included in the hardware evaluation [119]. However, the hardware cost of comparators are counted since every signal requires a comparator to generate a sequence and they cannot be shared. The signal reconstruction units/probability estimators (PEs) are also counted for the dynamic/conventional stochastic designs.

As shown in Table 4.1, the DSC circuits have a slightly higher area and power consumption than the conventional stochastic circuit due to the use of ADDIE-based signal reconstructor, while the SNG (comparator) and the multiplier are the same for those



two circuits. However, this disadvantage is negligible when considering the dominating factor, the sequence length or the number of clock cycles for producing one result: it is  $2^5$  for CSC; whereas it is only 1 for DSC. Due to the significant reduction in sequence length, the DSC frequency mixer consumes only 3.9% of the energy and 7.5% of the time required by the conventional design with a similar accuracy.

Compared to the fixed-width binary design, the DSC circuit has a slightly larger hardware cost, while it requires a slightly lower energy and shorter time to produce the result. Also, the SNR of the results produced by the proposed method exceeds that of the binary design by 3.38 dB. However, when the input data are sampled at the Nyquist frequency, i.e., 14 Hz considering that the mixed frequencies are 5 Hz and 7 Hz, the fixed-width binary design processes the signal of  $\pi/2$  seconds (s) with 22 multiplications, as shown in Fig. 4.17(b). On the other hand, the DSC cannot produce adequately accurate results with signals sampled at the Nyquist rate. It takes the DSC design 25,736 bit-wise AND operations to process the same signal at a sampling rate of  $2^{14}$  Hz to produce results with a similar accuracy to the 5-bit fixed-width binary design. This result is shown in Fig. 4.17(a). In this case, the latency of the DSC circuit is much longer, and the energy efficiency is much lower than the fixed-width design when processing a signal with the same length. The hardware efficiency is compared in Table 4.2.

Table 4.2. Hardware efficiency evaluation of DSC and fixed-width frequency mixer processing the same period of signal with different sampling rates

|                    | Sampling rate (Hz) | No. of clk cycles | Minimum time (ns)  | Energy (fJ)        | SNR (dB) |
|--------------------|--------------------|-------------------|--------------------|--------------------|----------|
| Dynamic SC         | $2^{14}$           | 25,736            | $1.49 \times 10^4$ | $1.12 \times 10^6$ | 19.16    |
| Fixed-width binary | 14                 | 22                | 15.84              | $1.21 \times 10^3$ | 20.69    |

#### 4.6.2 Function estimator using multiplexing circuits implementing Bernstein polynomials

The function  $y(t) = 1/11(2e^{-6t} + 3e^{-4t} + 6e^{-2t})$  is used for the accuracy evaluation of the dynamic and conventional SC systems. For the DSC system, a multiplexing circuit in Fig. 4.9(b) with DSS's is used to perform the function estimation with DSS's encoding

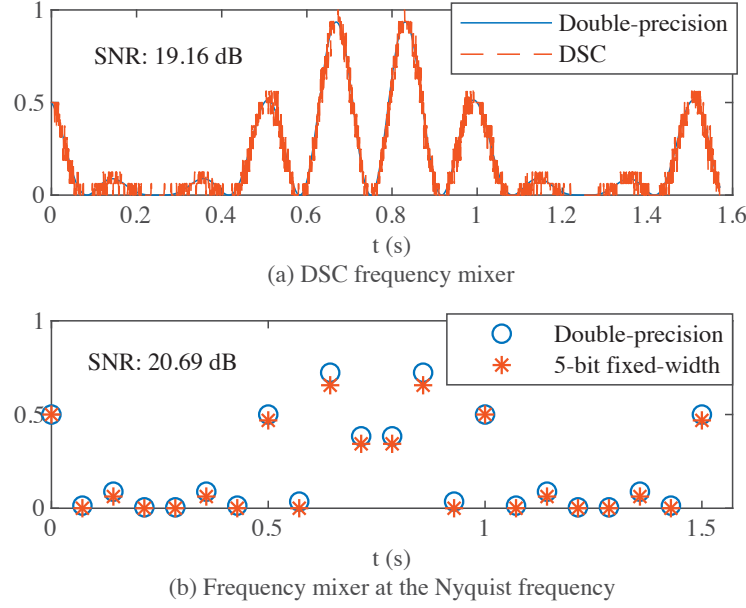


Figure 4.17. A DSC-based frequency mixer using oversampled signals vs. a fixed-width design using signals sampled at the Nyquist frequency.

$x(t) = e^{-2t}$  as the inputs. With a sampling rate of  $2^{16}$  Hz and 6-bit ADDIE as the signal reconstructor, the SNR produced by the DSC system is 26.60 dB as shown in Fig. 4.18(a).

For the CSC system, a multiplexing circuit proposed in [1] is used to approximate the function with a minimum-order Bernstein polynomial to reduce the hardware cost of the SNGs (comparators), which constitutes a major part of the SC system [19]. To estimate the function  $y(t)$  using conventional SC circuit, it is approximated by a Bernstein polynomial,  $y_1(t) = 17/256t^3 + 159/256t^2(1-t) + 66/256t(1-t)^2 + 249/256(1-t)^3$ . Using a sequence length of 64, the SNR of the estimate is 25.90 dB, as shown in Fig. 4.18(b).

A fixed-width binary circuit is also considered to implement the polynomial. The binary circuit is optimized to use the least number of multipliers to reduce the hardware cost. The resulting 6-bit design produces an SNR of 27.81 dB, as shown in Fig. 4.18(c), which is slightly higher than that of the result produced by the DSC circuit.

The hardware evaluation of the two SC and the binary circuits is reported in Table 4.3. As shown in Table 4.3, the DSC circuit has a slightly higher hardware cost but with a similar power consumption compared to the CSC circuit. However, for the CSC circuit, the long sequence undermines the performance and energy efficiency. The DSC takes only about

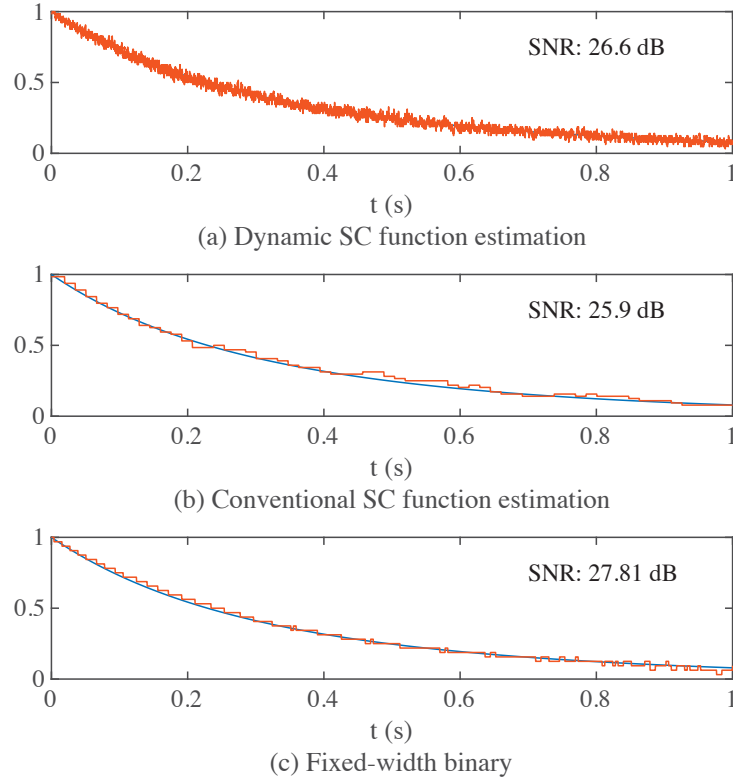


Figure 4.18. Function estimator: DSC vs. CSC vs. fixed-width binary implementation.

3.2% of the time and 1.6% of the energy of the CSC to achieve a higher accuracy and attains about 60% savings in energy and time compared to the binary circuit. Since the processed signal is not band-limited, the fixed-width binary design that uses signals sampled at the Nyquist rate is not included for comparison. However, when a lower sampling rate is used, the fixed-point design can achieve a lower total energy cost while maintaining a high accuracy; whereas this is not feasible for the DSC circuits.

Table 4.3. Hardware efficiency evaluation of dynamic and conventional stochastic function estimator producing one result

|                    | Area<br>( $\mu m^2$ ) | Power<br>( $\mu W$ ) | No. of clk<br>cycles | Minimum<br>time (ns) | Energy<br>(fJ) | SNR<br>(dB) |
|--------------------|-----------------------|----------------------|----------------------|----------------------|----------------|-------------|
| Dynamic SC         | 175.60                | 20.54                | 1                    | 0.7                  | 82.17          | 26.60       |
| Conventional SC    | 158.63                | 20.02                | 64                   | 21.76                | 5125.12        | 25.90       |
| Fixed-width binary | 301.76                | 50.77                | 1                    | 1.75                 | 203.08         | 27.81       |
| Ratio (Dyn.:Conv.) | 1.11                  | 1.03                 | 0.016                | 0.032                | 0.016          | –           |

## 4.7 Discussion

Since DSS's and  $\Delta - \Sigma$  modulated bit streams can both be considered to be Bernoulli sequences and used for SC [113], their similarities and differences are discussed. To gain insights into the DSS, an analysis is performed in the frequency domain first.

Compared to the DSM, the DSNG has a similar “noise-shaping” effect. Fig. 4.19 shows the results of an FFT analysis of the DSS and the  $\Delta - \Sigma$  modulated signal encoding the same sinusoidal signal. Only the quantization noise is considered in the experiment.

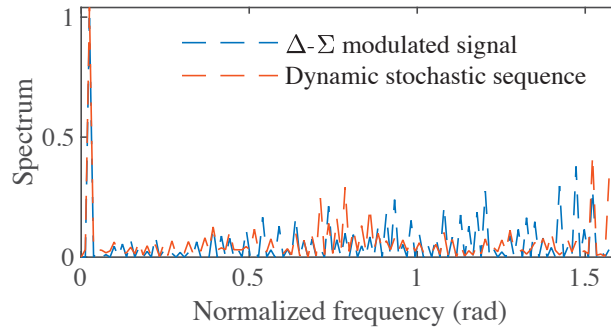


Figure 4.19. Spectrum:  $\Delta - \Sigma$  modulated vs. dynamic stochastic sequence.

In Fig. 4.19, the signal encoded is a sinusoid, and its spectrum is shown as a spectral peak near the frequency at 0 rad/s, which has an amplitude of 1. After either a  $\Delta - \Sigma$  modulation or a DSNG, the original spectrum of the signal is well preserved; while the spectrum of quantization noise is shifted and concentrated in the high-frequency domain. This is the reason why it has a similar behaviour to the  $\Delta - \Sigma$  modulated bit stream. However, instead of relying on the feedback loop as a high-pass filter in a DSM, it is achieved by introducing the random sequences, specifically, the Sobol sequences, when generating the DSS's. Compared to the  $\Delta - \Sigma$  modulation, it does not cause any stability issues without including a feedback loop.

Also, the DSC can avoid correlation issues by using independent Sobol sequences, whereas correlation exists in the  $\Delta - \Sigma$  modulated signals and it can degrade the accuracy seriously [103]. For example, if multiplying two signals with different frequencies, the DSC produces a much higher accuracy than using the  $\Delta - \Sigma$  modulated signals even multiplying signals with different frequency, as shown in Fig. 4.20. However, multiplying

two  $\Delta - \Sigma$  modulated signals that encode identical values leads to the original signal instead of their product [103].

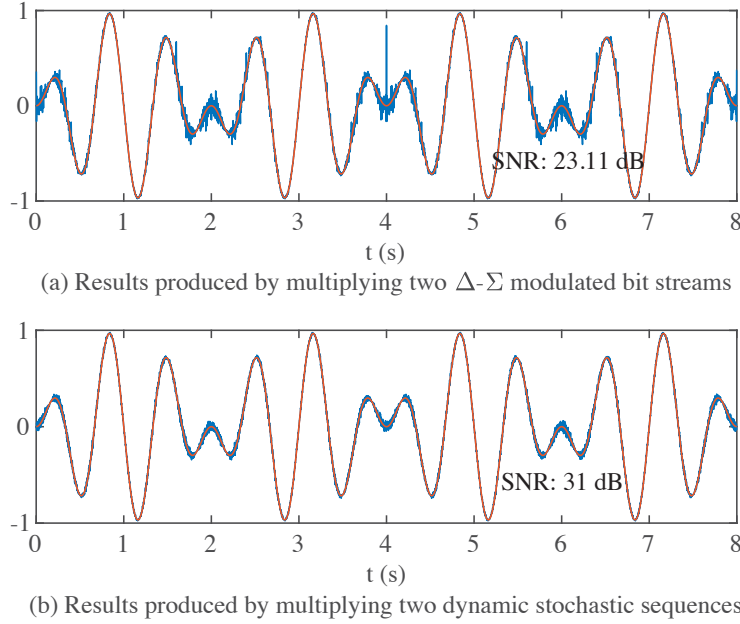


Figure 4.20.  $\Delta - \Sigma$  modulated vs. dynamic stochastic sequence for frequency mixer.

## 4.8 Summary

In this chapter, DSC is proposed that leverages an efficient encoding technique using DSS's and simple digital SC circuits to implement complex DSP functions. Frequency mixing, function approximation, filtering and numerical integration are implemented by using a stochastic multiplier, a multiplexing circuit, an ADDIE and a stochastic integrator, respectively. The optimal parameters for reconstructing the signals are numerically analyzed. Moreover, the generation of the DSS's is discussed. It is shown that using either a quasirandom sequence or a deterministic  $\Delta - \Sigma$  modulated sequence improves the accuracy of the results compared to the use of an LFSR-generated sequence. Compared to CSC, the proposed DSC achieves a speedup and an energy efficiency improvement over conventional SC of more than  $13\times$  and  $25\times$ , respectively, for signal multiplication at a better accuracy. For function estimation, the improvement is even larger, which are  $31\times$  and  $62\times$ , respectively. With a similar accuracy, DSC using a 6-bit resolution ADDIE also

achieves a saving in energy and time by 60% compared to a 6-bit conventional binary circuits, when dealing with complex tasks such as function estimation using the oversampled data. However, due to the requirement of over-sampling, the proposed method has a lower energy efficiency and higher latency compared with fixed-width circuits using signals sampled at the Nyquist rate.

# Chapter 5

## ODE Solvers Using Stochastic Circuits

A novel ordinary differential equation (ODE) solver is proposed that uses a stochastic integrator to implement the accumulative function of the Euler method. We show that a stochastic integrator is an unbiased estimator for a Euler numerical solution. Unlike in conventional stochastic computing (SC) circuits, in which relatively long stochastic bit streams are required to produce a result with a high accuracy, the proposed stochastic ODE solver provides an estimate of the solution for every bit in a stochastic sequence or a dynamic stochastic sequence (DSS), thus significantly reducing the latency and energy consumption of the circuit. Complex ODE solvers are constructed for solving nonhomogeneous ODEs, systems of ODEs and higher-order ODEs. Experimental results show that the stochastic ODE solvers provide very accurate solutions compared to their binary counterparts, with on average an energy saving of 46% (up to 74%),  $8\times$  throughput per area (up to nearly  $12\times$ ) and a runtime reduction of 72% (up to 82%). Furthermore, through solving Laplace's equation, it is shown that this method is scalable and can be used for a large scale partial differential equation (PDE) solving.

### 5.1 Introduction

ODEs are widely used in the modeling of natural processes in physics, chemistry and biology, as well as in solving problems in many engineering and social studies such as scientific computing and economics. Various algorithms have been developed for solving an ODE. However, most of the algorithms are computationally intensive, especially when solving problems for a large-scale system. While a general-purpose processor is often

used, acceleration has been achieved by using the massively parallel structure of graphics processing units (GPUs) [121, 122]. However, a GPU is not well suited for mobile and embedded applications because it is large and power hungry compared to a tailored application-specific integrated circuit (ASIC) design.

Conventional digital ODE solvers require the use of adders, multipliers, registers and complex control circuitry, such as in a digital differential analyzer (DDA) [123]. Although more efficient implementations have been proposed by using concurrent processing paths [124], a conventional binary design still requires a complex datapath and control circuitry.

In this chapter, a novel ODE solver is proposed by using SC circuits. Detailed formulation shows that a single stochastic ODE solver provides an unbiased estimate of the Euler numerical solution for an ODE. Three error reduction schemes are further proposed and verified by both theory and simulations. The approach to building more complex ODE solvers is demonstrated by constructing the circuits for solving three typical ODEs. With a limited loss of accuracy, the proposed stochastic ODE solvers show significant advantages in energy consumption and throughput compared to their binary counterparts. A modified stochastic Laplacian circuit is also proposed to solve a large scale PDE.

## 5.2 Proposed stochastic ODE solvers

### 5.2.1 Formulation

The circuit diagram and a symbol of the stochastic integrator are shown in Fig. 2.6(a) and (b) [4]. A key component in a stochastic integrator is a  $2^n$ -state up/down counter. A random number generator (RNG) and a comparator work as a stochastic number generator (SNG) for generating a stochastic bit stream,  $Seq_{out}$ , to encode the value stored in the counter. The bit stream can be used as a feedback for itself or as an input bit stream for subsequent stochastic integrators. If not used, the RNG and the comparator can be removed to reduce hardware cost. The input signals  $A$  and  $B$  carry stochastic bit streams, which determine whether to increase, decrease or keep the value of the counter.

The counter is typically a  $2^n$ -state counter with  $n$ -bit width, counting from 0 to  $2^n - 1$ . The initial value is determined by the input of the application. Let  $I$  denote the integer value



stored in the counter. The probability to be encoded by the output stochastic bit stream is  $I/2^n$  in the unipolar representation or  $2 \times I/2^n - 1$  in the bipolar representation. Without loss of generality, only the unipolar representation is considered in this chapter; designs for the bipolar representation can similarly be derived by a linear mapping.

Let the two bits in the input streams  $A$  and  $B$  be  $a_i$  and  $b_i$  respectively at the  $i$ th clock cycle. The function of the up/down counter is then

$$I_{i+1} = \begin{cases} I_i + 1 & \text{if } a_i = 1 \text{ and } b_i = 0 \\ I_i & \text{if } a_i = b_i \\ I_i - 1 & \text{if } a_i = 0 \text{ and } b_i = 1 \end{cases}, \quad (5.1)$$

where  $I_i$  and  $I_{i+1}$  are the integers stored in the counter at the  $i$ th and  $(i+1)$ th clock cycles. Equivalently, we have

$$I_{i+1} = I_i + a_i - b_i. \quad (5.2)$$

The expectation of  $I_{i+1}$ ,  $\mathbb{E}[I_{i+1}]$ , is given by

$$\mathbb{E}[I_{i+1}] = \mathbb{E}[I_i + a_i - b_i] = \mathbb{E}[I_i] + \mathbb{E}[a_i] - \mathbb{E}[b_i]. \quad (5.3)$$

Let the probability that  $a_i$  is “1” at the  $i$ th clock cycle be  $p_{a,i}$  and the probability that  $b_i$  is “1” be  $p_{b,i}$ , (5.3) becomes [89]:

$$\mathbb{E}[I_{i+1}] = \mathbb{E}[I_i] + p_{a,i} - p_{b,i}. \quad (5.4)$$

To convert an integer into a stochastic number in the unipolar representation, both sides of (5.4) are normalized by  $\frac{1}{2^n}$ . Hence, (5.4) is transformed to

$$\mathbb{E}[s_{i+1}] = \mathbb{E}[s_i] + \frac{1}{2^n}(p_{a,i} - p_{b,i}), \quad (5.5)$$

where  $s_i = I_i/2^n$  and  $s_{i+1} = I_{i+1}/2^n$ . If the initial value in the counter is  $s_0$ , then by an iterative accumulation of (5.5), the expected value of  $s_k$  at the (arbitrary)  $k$ th clock cycle ( $k = 1, 2, \dots$ ) is obtained as

$$\mathbb{E}[s_k] = s_0 + \frac{1}{2^n} \sum_{i=0}^{k-1} (p_{a,i} - p_{b,i}). \quad (5.6)$$

For an ODE  $\frac{dy(t)}{dt} = f(t, y(t))$ , the numerical solution for a given  $t$  can be estimated by considering the derivative of  $y(t)$  at a discrete  $t_i$  as

$$\left. \frac{dy(t)}{dt} \right|_{t=t_i} = \lim_{\Delta t \rightarrow 0} \frac{y(t_i + \Delta t) - y(t_i)}{\Delta t} \approx \frac{y(t_i + h) - y(t_i)}{h}, \quad (5.7)$$

where  $h$  is a small value for the time interval  $\Delta t$ . Let  $t_{i+1} = t_i + h$  ( $t_i = h \cdot i$  when  $t_0 = 0$  and  $h$  is a constant,  $i = 0, 1, 2, \dots$ ), (5.7) leads to the solution by the Euler method [125]:

$$\hat{y}_{i+1} = y_i + hf(t_i, y_i), \quad (5.8)$$

where  $\hat{y}_{i+1}$  is the numerical estimation of the function value of  $y(t)$  at  $t_{i+1}$ , i.e.,  $y(t_{i+1})$ , and  $h$  is the step size for the estimate. Let  $t$  start from  $t = 0$ , with  $h = \frac{1}{2^n}$  and  $f(t, y(t)) = p_a(t) - p_b(t)$ , (5.8) is simplified to

$$\hat{y}_{i+1} = y_i + \frac{1}{2^n} [p_a(\frac{i}{2^n}) - p_b(\frac{i}{2^n})]. \quad (5.9)$$

If the initial condition of the ODE is  $y_0$ , the estimate of the solution at the  $k$ th step is given by an iterative accumulation of (5.9) over  $i$ , which leads to

$$\hat{y}_k = y_0 + \frac{1}{2^n} \sum_{i=0}^{k-1} [p_a(\frac{i}{2^n}) - p_b(\frac{i}{2^n})]. \quad (5.10)$$

Hence, for the ODE

$$\frac{dy(t)}{dt} = p_a(t) - p_b(t), \quad (5.11)$$

the Euler numerical solution  $\hat{y}_k$  provides an estimated value of the function  $y(t)$  at  $t = k/2^n$ ,  $k = 0, 1, 2, \dots$ , i.e.,

$$y(\frac{k}{2^n}) \approx \hat{y}_k. \quad (5.12)$$

Let the input sequences of the stochastic integrator encode the probabilities  $p_a(t)$  and  $p_b(t)$  at  $t = i/2^n$ , i.e.,  $p_{a,i} = p_a(\frac{i}{2^n})$  and  $p_{b,i} = p_b(\frac{i}{2^n})$ ; as per (5.6) and (5.10), the normalized expected value of the up/down counter at the  $k$ th clock cycle,  $\mathbb{E}[s_k]$ , provides an unbiased estimate of the Euler solution at the  $k$ th time step,  $\hat{y}_k$ , for the same initial condition, i.e.,  $y_0 = s_0$ . By (5.12), we obtain

$$y(\frac{k}{2^n}) \approx \mathbb{E}[s_k], \quad (5.13)$$

that is,  $\mathbb{E}[s_k]$  ( $k = 1, 2, \dots$ ) provides an approximate solution of the ODE (5.11) with a step size of  $1/2^n$ .

The input bit streams only serve as the control signals for the counter, while the output of the counter provides the Euler estimate, one estimate at each time step or per clock cycle, thus achieving great efficiency.

## 5.2.2 Stochastic ODE solver designs

In this section, several designs are proposed as typical ODE solvers. Note that all the ODEs and parameters are chosen so that the solution lies in the range  $[0, 1]$  of the stochastic unipolar representation; otherwise, it is considered that an overflow occurs. To evaluate accuracy, the analytical solutions are also obtained for comparison.

### Nonhomogeneous ODEs

Nonhomogeneous ODEs refer to the type of ODEs that involves time, i.e., using  $t$ -related terms.

As per (5.11) and (5.13), if  $p_a(t) = 1$  and  $p_b(t) = 0$ , a stochastic integrator solves the ODE,

$$\frac{dy(t)}{dt} = 1 - 0, \quad (5.14)$$

with  $y(0) = 0$ , i.e., the counter is initialized to “0”. The analytical solution for (5.14) is  $y(t) = t$ . It is produced by the output of the counter, as shown in the stochastic integrator at the first stage in Fig. 5.1(a). The output sequence generated from the first stage is a DSS that encodes  $y(t) = t$  as per (4.3) and the formulation of the stochastic integrator at the first stage. If the output sequence from the stochastic integrator is connected to a subsequent stochastic integrator (with 0 as another input), the cascaded structure, as shown at the first two stages in Fig. 5.1(a), solves the ODE,

$$\frac{dy(t)}{dt} = t, \quad (5.15)$$

with  $y(0) = 0$ . It is because that the second stage of the circuit can be viewed as a numerical integrator as shown in Chapter 4, and it can also be explained by the formulation of the stochastic integrator as shown in the previous section. The analytical solution for (5.15) is  $y(t) = (1/2)t^2$ , estimated in the circuit by the output of the second counter. Similarly, the three stages of the cascaded structure solve

$$\frac{dy(t)}{dt} = \frac{1}{2}t^2, \quad (5.16)$$

with  $y(0) = 0$ . The analytical solution for (5.16) is  $y(t) = (1/6)t^3$ , estimated by the output of the third counter.

Hence, the cascaded stochastic integrators in Fig. 5.1(a) are solvers for a set of nonhomogeneous ODEs. In the cascaded structure, the integrators in the earlier stages are used to generate the stochastic bit stream for the  $t$ -related terms in the ODE to be solved.

The cascading of the stochastic integrators can be continued for solving an ODE with a higher-order polynomial as its solution. Note that the output sequence from the integrator at the last stage is not connected to any other components, so the RNG and comparator in this stochastic integrator can be removed.

### Systems of ODEs

A system of ODEs can be solved by using multiple stochastic integrators. For a system of ODEs such as

$$\begin{cases} \frac{dy_1(t)}{dt} = y_2(t) - 0.5, \\ \frac{dy_2(t)}{dt} = 0.5 - y_1(t), \end{cases} \quad (5.17)$$

with  $y_1(0) = 0$  and  $y_2(0) = 0.5$  as the initial values of the two counters. For this system, two cross-coupled stochastic integrators provide a solution by utilizing the output bit stream from one integrator as an input of the other integrator, as shown in Fig. 5.1(b). These sequences are DSS's encoding  $y_1(t)$  and  $y_2(t)$  respectively. In this design, the other inputs are set to 0.5, as determined by (5.17). The analytical solution for (5.17) is  $y_1(t) = 0.5 - 0.5 \cos(t)$  and  $y_2(t) = 0.5 + 0.5 \sin(t)$ . Other values other than 0.5 can be used in (5.17) if they do not result in an overflow.

### Higher-order ODEs

To solve an  $m$ th order ODE, at least  $m$  stochastic integrators are required since one stochastic integrator performs a single integration. For a second order ODE such as

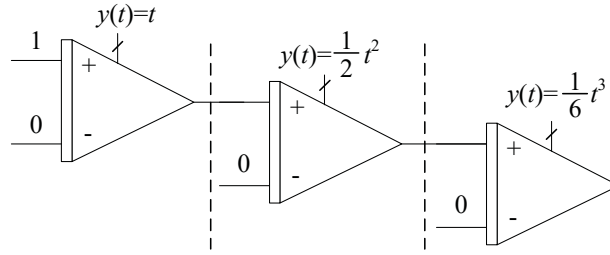
$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) = 0, \quad (5.18)$$

with  $y(0) = 0$ ,  $\frac{dy(t)}{dt}|_{t=0} = 1$ , an additional function,  $z(t) = \frac{dy(t)}{dt} + 2y(t)$ , is introduced for the first two terms in (5.18) such that  $\frac{dz(t)}{dt} = \frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt}$ . By doing so, the order of the ODE is lowered and (5.18) is converted into two first-order ODEs

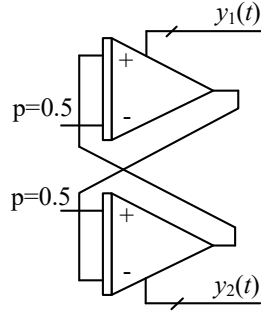
$$\begin{cases} \frac{dz(t)}{dt} = -y(t), \\ \frac{dy(t)}{dt} = z(t) - 2y(t). \end{cases} \quad (5.19)$$

In Fig. 5.1(c), the first stochastic integrator solves the first ODE in (5.19) by initializing the counter to  $z(0) = \frac{dy(t)}{dt}|_{t=0} + 2y(0) = 1$ . The inverter serves as a stochastic subtractor that computes the function  $1 - y(t)$ . The second stochastic integrator solves the second ODE by initializing the counter to  $y(0) = 0$ .

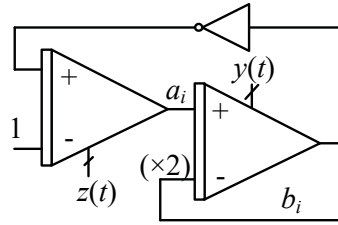
Due to the  $\times 2$  factor for  $y(t)$ , the updating rule of the second stochastic integrator becomes:  $I_{i+1} = I_i + a_i - 2b_i$ , where  $a_i$  and  $b_i$  are two input bits at the  $i$ th clock cycle.  $2b_i$  is implemented by a 1-bit left-shift of  $b_i$ . The second stochastic integrator is then modified to accommodate the modified updating rule. The analytical solution for (5.18) is  $y(t) = te^{-t}$ ;  $z(t) = (t+1)e^{-t}$  is computed by the first stochastic integrator as an intermediate result. The output DSS of the first stochastic integrator encoding  $z(t)$  is then connected to the second stochastic integrator to compute  $y(t)$ .



(a) For (5.14), (5.15) and (5.16).



(b) For (5.17).



(c) For (5.18).

Figure 5.1. Designs of stochastic ODE solvers.

### 5.3 Error reduction schemes

One major disadvantage in SC is the loss of accuracy [126]. Usually, it is believed that multiple independent RNGs need to be utilized for a higher accuracy, which inevitably

increases the hardware overhead. However, we show here that sharing RNGs can reduce the variance in the solution of a stochastic ODE solver, thus reducing the error.

The error in the solution of a stochastic ODE solver is mainly caused by: (1) the Euler numerical method and (2) the randomness in SC.

In the Euler method, the error is measured by the local truncation error (LTE) and global truncation error (GTE). The LTE refers to the error introduced in a single step of estimation and the GTE refers to the error caused in multiple steps. The LTE and GTE are proportional to  $h^2$  and  $h$  respectively, where  $h$  is the step size [125]. Thus a simple solution is to increase the bit width of the up/down counter to reduce the step size ( $h = 1/2^n$ , where  $n$  is the bit width of the counter), thereby reducing error due to the Euler method. Note that increasing the bit width of the counter leads to a better granularity in the final solution, but it does not significantly increase the latency of the stochastic circuit, unlike in conventional SC.

The error introduced by SC is related to the variance of the derivative term,  $a_i - b_i$ . When independent RNGs are used to generate  $a_i$  and  $b_i$ , the variance at a single step is given by

$$\text{Var}[a_i - b_i] = p_{a,i}(1 - p_{a,i}) + p_{b,i}(1 - p_{b,i}). \quad (5.20)$$

The total variance is the sum of variances at each step if each random number is independently generated, as is approximately the case for using an linear-feedback shift register (LFSR).

However, if the same RNG is used to generate  $A$  and  $B$ , the probability distribution of  $a_i - b_i$  is shown in Table 5.1 (assume  $p_{a,i} \geq p_{b,i}$ ). In this case, the variance of  $a_i - b_i$  can be derived as:

$$\text{Var}_s[a_i - b_i] = \mathbb{E}[(a_i - b_i - \mathbb{E}(a_i - b_i))^2] = (p_{a,i} - p_{b,i})(1 - p_{a,i} + p_{b,i}). \quad (5.21)$$

Table 5.1. Probability distribution of  $a_i - b_i$  when the same RNG is used to generate  $A$  and  $B$

| $a_i - b_i$ | Probability             |
|-------------|-------------------------|
| -1          | 0                       |
| 0           | $1 - p_{a,i} + p_{b,i}$ |
| 1           | $p_{a,i} - p_{b,i}$     |

Because  $\text{Var}_s[a_i - b_i] - \text{Var}[a_i - b_i] = -2p_{b,i}(1 - p_{a,i}) \leq 0$ , we obtain  $\text{Var}_s[a_i - b_i] \leq \text{Var}[a_i - b_i]$  for any  $i = 0, 1, 2, \dots$ . Therefore, sharing the use of RNGs to generate input stochastic bit streams improves the accuracy. The same conclusion can similarly be obtained for  $p_{a,i} < p_{b,i}$ .

Further improvement of the accuracy can be achieved by using low-discrepancy (LD) sequences for a faster convergence and thus better progressive precision [41, 127].

## 5.4 Experiments and results

### 5.4.1 Validation of the proposed designs

In this section, the proposed stochastic ODE solvers are validated by hardware simulations using designs specified in VHSIC Hardware Description Language (VHDL). The designs are synthesized by Synopsys Design Compiler (DC) and analyzed by Mentor Graphic ModelSim with the STM 28-nm technology library. The detailed parameters and setting are available in Appendix A. The numerical solution is produced by using 8-bit counters with a step size of  $1/2^8$ .

Fig. 5.2(a) shows the results produced by the circuit in Fig. 5.1(a) for solving (5.14), (5.15) and (5.16), in comparison with the analytical results. Note that for  $t > 1$ , the result is not shown as it exceeds the range of the unipolar representation in SC. The simulation results are depicted in Fig. 5.2(b) for two full periods along with the analytical solution for (5.17). A full period of the sine/cosine function can be generated within 1609 clock cycles ( $\lceil 2 \times \pi \times 2^8 \rceil$ ), while a conventional SC function generation method requires 1024-bit sequences to produce a single result [128]. Thus, a stochastic ODE solver can be used as an efficient function generator. The simulation results produced by the circuit in Fig. 5.1(c) are depicted in Fig. 5.2(c), in comparison with the analytical results.

As seen from the results, the 8-bit stochastic ODE solvers produce very accurate solutions when compared to the analytical results. A quantitative evaluation of the results using the root-mean-squared error (RMSE) is reported next.

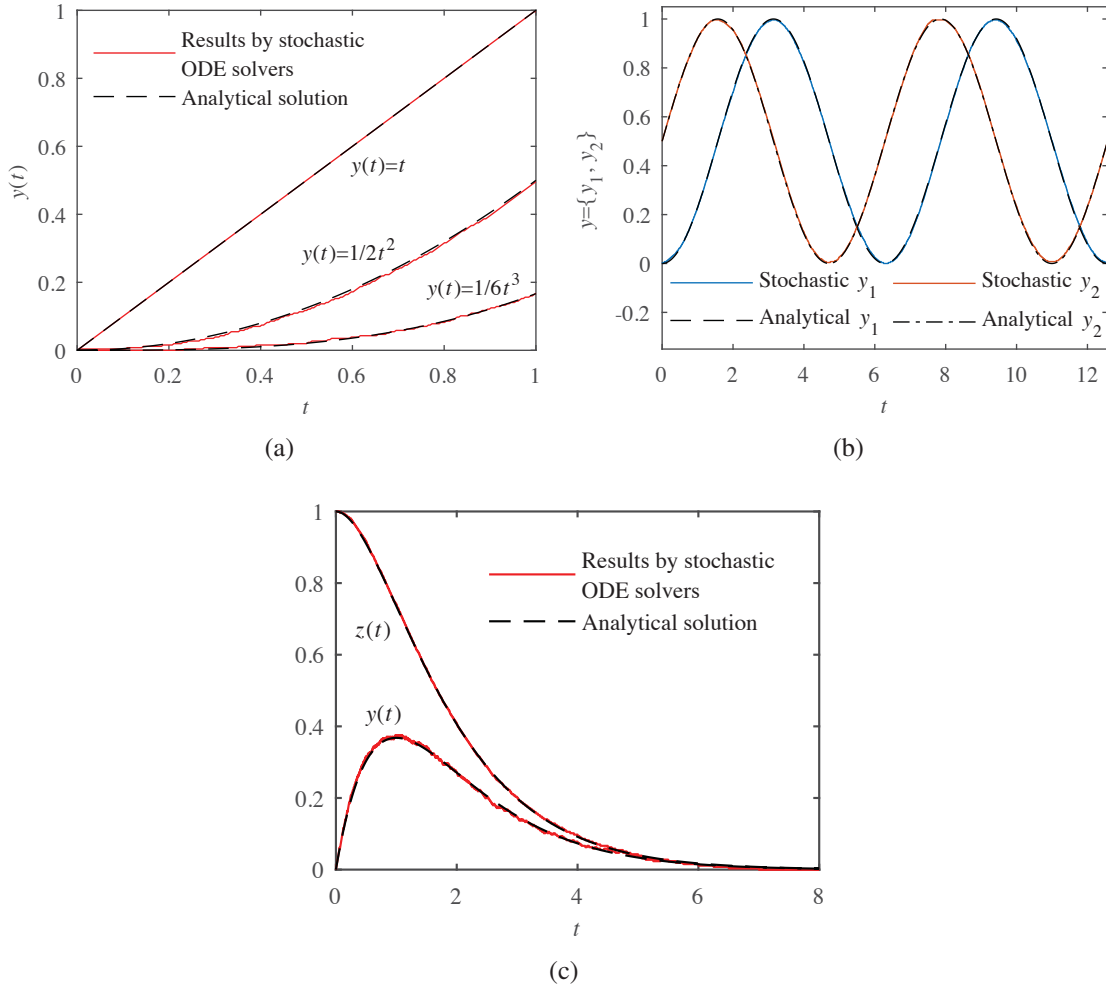


Figure 5.2. Simulation results of stochastic ODE solvers. Analytical solution vs. hardware results produced by the stochastic ODE solvers for (a) (5.14), (5.15) and (5.16); (b) (5.17) and (c) (5.18).

#### 5.4.2 Validation of the error reduction schemes

The accuracy of the stochastic ODE solvers with different configurations is measured to verify the three error reduction schemes by: (1) increasing the bit width; (2) sharing the use of RNGs; (3) using LD Sobol sequences [105]. The proposed design in Fig. 5.1(b) is considered for an RMSE analysis in the first full period of the functions  $y_1(t)$  and  $y_2(t)$ . The results are shown in Fig. 5.3.

As can be seen, the circuit with a larger width tends to have a lower RMSE. In general, the circuits using LD sequences produce more accurate results than those using pseudorandom (PR) sequences generated by LFSRs. For the same bit width, the circuits



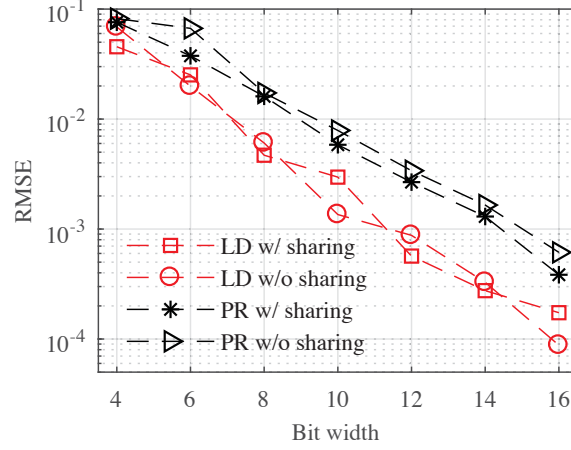


Figure 5.3. RMSE of  $y_1(t)$  and  $y_2(t)$  for the stochastic ODE solver in Fig. 5.1(b) under different configurations.

using PR sequences with shared RNGs provide more accurate numerical solutions than those using independent RNGs. When LD sequences are used, the RMSE is not significantly affected by using shared RNGs. Nevertheless, LD sequences are adopted with the RNG-sharing scheme, resulting in a reduced hardware cost and energy consumption [105].

### 5.4.3 Performance evaluation

The accuracy and hardware cost of the stochastic ODE solvers are evaluated and compared with their binary counterparts. The binary circuits are implemented by using a second-order Runge Kutta (RK2) (midpoint) numerical method<sup>1</sup> [125]. The RK2 method is also known as a modified Euler method with GTE in  $O(h^2)$  and LTE in  $O(h^3)$ . The numerical solution of an ODE in the form of  $\frac{dy(t)}{dt} = f(y(t), t)$  using the RK2 method is given by

$$\hat{y}_{i+1} = y_i + hf(t_i + \frac{h}{2}, y_i + \frac{h}{2}f(t_i, y_i)), \quad (5.22)$$

where  $h$  is the step size of the estimate. The bit width of the stochastic ODE solver is set to 8, so that  $h$  is  $1/2^8$  for the stochastic ODE solver. The binary ODE solvers also employ 8-bit designs with a step size of  $1/2^8$  for comparison. The RK2 method performs iterative additions and multiplications. For a step size of  $1/2^8$ , however, the multiplication can be

<sup>1</sup>RK2 is used instead of the original Euler method because using conventional binary circuits with truncation-based accumulations for the Euler method leads to a large loss of accuracy.

simplified by shifting. Thus, a binary RK2 ODE solver can be implemented by iterative shifting and additions [129].

For the stochastic solvers, the RNG in a stochastic integrator is implemented by the simplest Sobol sequence generator, a reversely mapped counter [41]. The RNG is shared to reduce the hardware cost. As a result, only one RNG is required to generate stochastic bit streams for each design.

The accuracy of the hardware ODE solvers is measured by RMSE. The hardware efficiency is measured by energy per operation (EPO), glstpa and minimum runtime. The EPO is the total energy consumed for completing one estimate of the solution, obtained as the power multiplied by the clock period. The throughput per area (TPA) is the maximum throughput per area and per time unit, given by  $1/(\text{area} \times \text{critical path delay})$ . The minimum runtime is obtained by multiplying the critical path delay and the number of clock cycles needed to obtain the solutions in Figs. 5.2(a), 5.2(b) and 5.2(c) respectively. The power, area and critical path delay are first measured to compute the EPO, TPA and minimum runtime.

Table 5.2. Hardware performance comparison of the ODE solvers

| ODE    | Metric                    | SC      | RK2 (binary) | Improvement |
|--------|---------------------------|---------|--------------|-------------|
| (5.15) | EPO (fJ)                  | 144.49  | 201.05       | 28%         |
|        | TPA ( $w/\mu s/\mu m^2$ ) | 13.84   | 3.86         | 258%        |
|        | Runtime (ns)              | 104.96  | 263.68       | 60%         |
| (5.16) | EPO (fJ)                  | 186.10  | 253.05       | 26%         |
|        | TPA ( $w/\mu s/\mu m^2$ ) | 9.76    | 0.94         | 934%        |
|        | Runtime (ns)              | 104.96  | 586.24       | 82%         |
| (5.17) | EPO (fJ)                  | 201.21  | 466.00       | 56%         |
|        | TPA ( $w/\mu s/\mu m^2$ ) | 4.75    | 0.58         | 716%        |
|        | Runtime (ns)              | 2573.59 | 8557.20      | 70%         |
| (5.18) | EPO (fJ)                  | 156.04  | 591.62       | 74%         |
|        | TPA ( $w/\mu s/\mu m^2$ ) | 5.68    | 0.44         | 1184%       |
|        | Runtime (ns)              | 1597.44 | 6819.84      | 76%         |

As shown in the simulation results in Table 5.2, all of the stochastic ODE solvers have a smaller EPO, minimum runtime and a larger TPA than their binary counterparts. The improvement is obtained as the relative difference between the measures of the SC and binary circuits with respect to the measure of the binary circuit. The stochastic ODE solver

achieves up to 74% in energy saving, up to nearly  $12\times$  throughput enhancement and up to 82% drop in runtime. On average, the stochastic designs achieve an energy saving of 46%,  $7\times$  TPA improvement and a 72% reduction of runtime due to the smaller delay and area. Due to the random fluctuations in an SC circuit, the RMSE is larger for a stochastic ODE solver than for a binary ODE solver, as shown in Table 5.3. However, the RMSE is in the order of  $10^{-3}$ , which indicates that the accuracy obtained by a stochastic ODE solver is quite high.

Table 5.3. Accuracy comparison of the ODE solvers (RMSE in  $10^{-3}$ )

| ODE    | SC   | RK2  |
|--------|------|------|
| (5.15) | 5.66 | 3.70 |
| (5.16) | 3.88 | 2.37 |
| (5.17) | 4.69 | 3.59 |
| (5.18) | 5.86 | 2.11 |

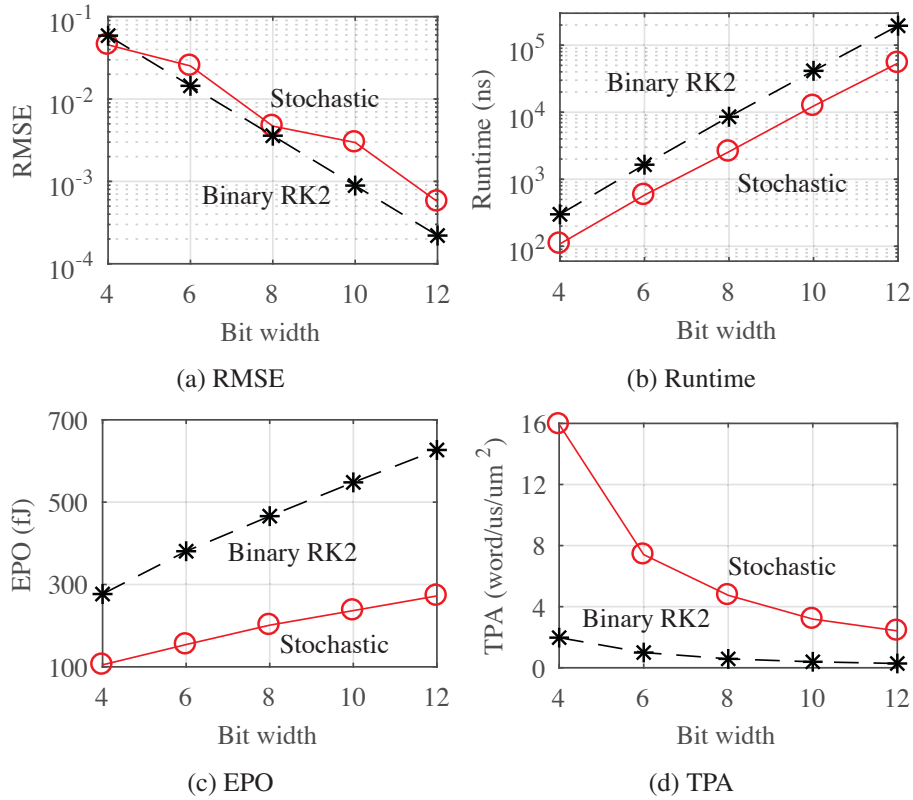


Figure 5.4. Comparison of stochastic and binary ODE solvers with different bit widths.

To further test the hardware efficiency at a similar accuracy for the stochastic and binary designs, the design shown in Fig. 5.1(b) is used. Fig. 5.4 shows the accuracy and hardware efficiency of the stochastic designs compared to their binary counterparts with various bit widths and step sizes. In Fig. 5.4(a), the stochastic design shows a smaller RMSE when the bit width is 4 for both circuits. However, the accuracy of a binary design becomes higher for a larger bit width. When a similar RMSE is achieved (e.g., for 8-bit binary and 10-bit stochastic designs), the stochastic design shows considerable advantages in EPO and TPA over the binary design, despite having a slightly longer runtime.

## 5.5 Accuracy vs. sequence length

Although the stochastic ODE solver can be used to achieve a higher hardware efficiency than its binary counterpart, extending the sequence length for each value encoded by the DSS can still improve the accuracy of the computed result. For example, one bit is used to encode a point in a signal in a DSS. However, multiple bits can be added to encode the same value as in conventional SC for a better accuracy. In Fig. 5.5, the sequence lengths encoding each number are extended and different ODE solvers with different bit widths are tested.

As shown in Fig. 5.5, extending the sequence length tends to improve the accuracy of the results. However, irregular patterns appear when solving (5.16), i.e., a longer sequence can lead to a lower accuracy. This occurs because the circuit lacks a feedback, thus making the solution unstable. When solving (5.17) and (5.18), longer sequences generally produce higher accuracy for the solvers that use Sobol sequences. Due to the randomness in the LFSR-generated pseudorandom numbers, the accuracy of the solvers that use LFSRs is more irregular except for the one that solves (5.18).

## 5.6 Stochastic Laplacian circuit for Laplace's equation

A PDE is a type of differential equations that contains multiple variables and their derivatives. For example, Laplace's equation is a second order PDE that can be used to describe many natural phenomena, such as thermodynamics, fluid-dynamics and gravitation, thus it is very important in mathematical physics [130]. A 2-dimensional

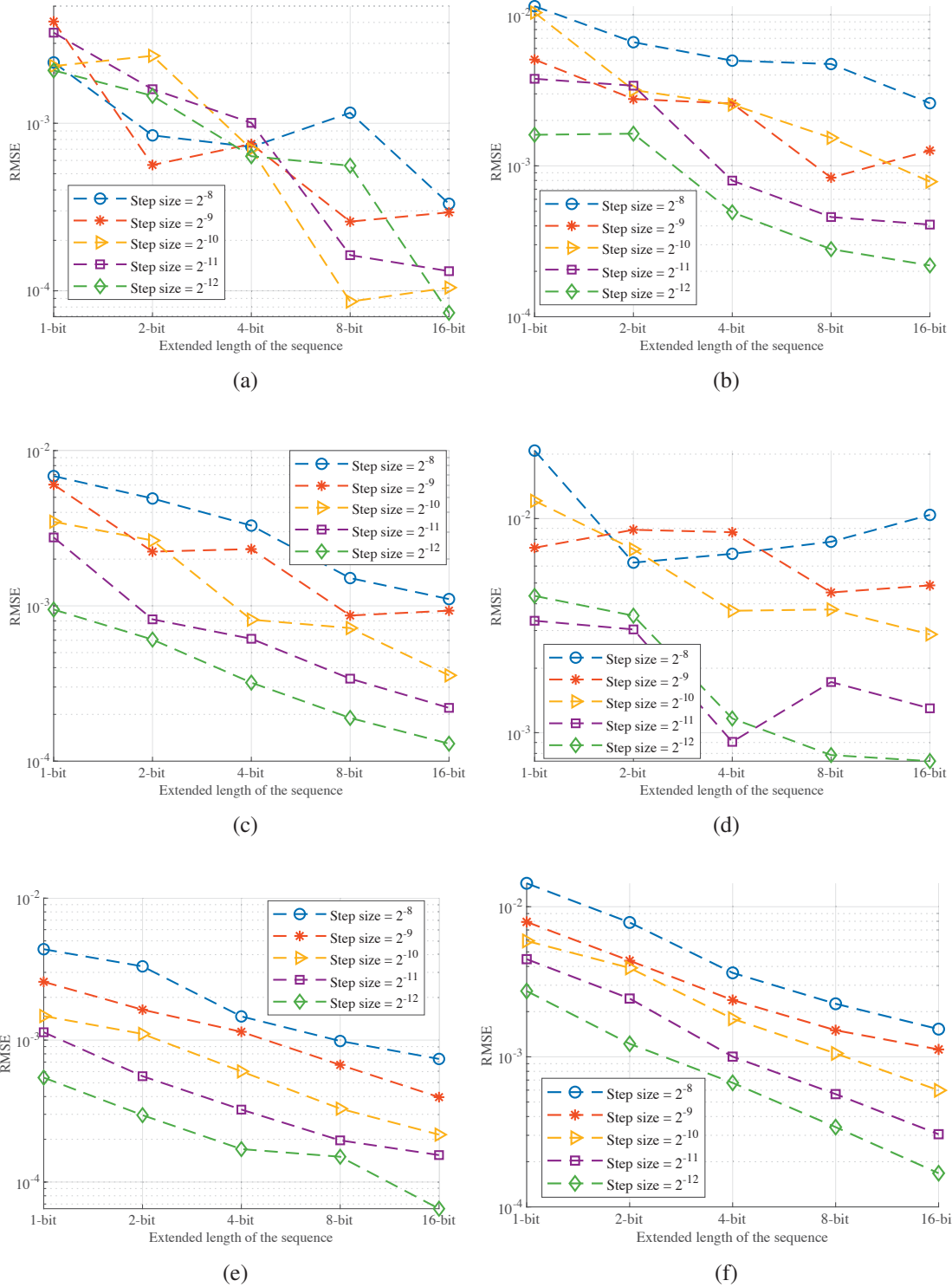


Figure 5.5. Stochastic ODE solvers with different sequence lengths that solve (a) (5.16) using Sobol sequences; (b) (5.16) using pseudorandom numbers; (c) (5.17) using Sobol sequences; (d) (5.17) using pseudorandom numbers; (e) (5.18) using Sobol sequences and (f) (5.18) using pseudorandom numbers.

Laplace's equation is given by

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad (5.23)$$

where  $\nabla^2$  is referred to as the Laplacian operator. A numerical solution of (5.23) can be obtained by dividing the 2-dimensional space into square grids and computing the value at each grid numerically. For example, a 2-dimensional space can be divided as shown in Fig. 5.6 with a horizontal interval of  $\Delta x$  and vertical interval of  $\Delta y$  and  $\Delta x = \Delta y$ . The value at location  $(i, j)$  is denoted by  $u_{i,j}$ . Then, the second-order partial derivative terms in (5.23) can be approximate by a second-order finite difference scheme,

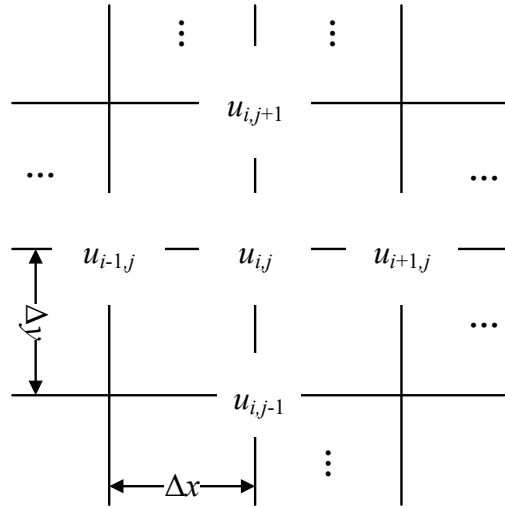


Figure 5.6. The 2-dimensional space is divided into grids and the value at each point is solved numerically. Each point is in the center of the grid.

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2}, \quad (5.24)$$

which can be derived by a Taylor expansion. Similarly,

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta y^2}. \quad (5.25)$$

Since  $\Delta x = \Delta y$ , combining (5.23), (5.24) and (5.25) leads to [130]

$$u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j} = 0. \quad (5.26)$$

By solving the linear equation set consisting of (5.26) for each grid point, a steady-state numerical solution of Laplace's equation is obtained. Boundary conditions are required to obtain a unique numerical solution, which are similar to the initial value in the Euler method.

### 5.6.1 Proposed stochastic Laplacian circuit design

In [4], a stochastic Laplacian circuit has been proposed, as shown in Fig. 5.7. In Fig. 5.7,  $u_{i,j}$  is used to denote the result at point  $(i,j)$  and the upper case  $U_{i,j}$  represents the corresponding stochastic sequence encoding the value. The input stochastic sequences are randomly chosen by the multiplexer with equal probability to compute  $1/4(U_{i,j-1} + U_{i-1,j} + U_{i,j+1} + U_{i+1,j})$ .

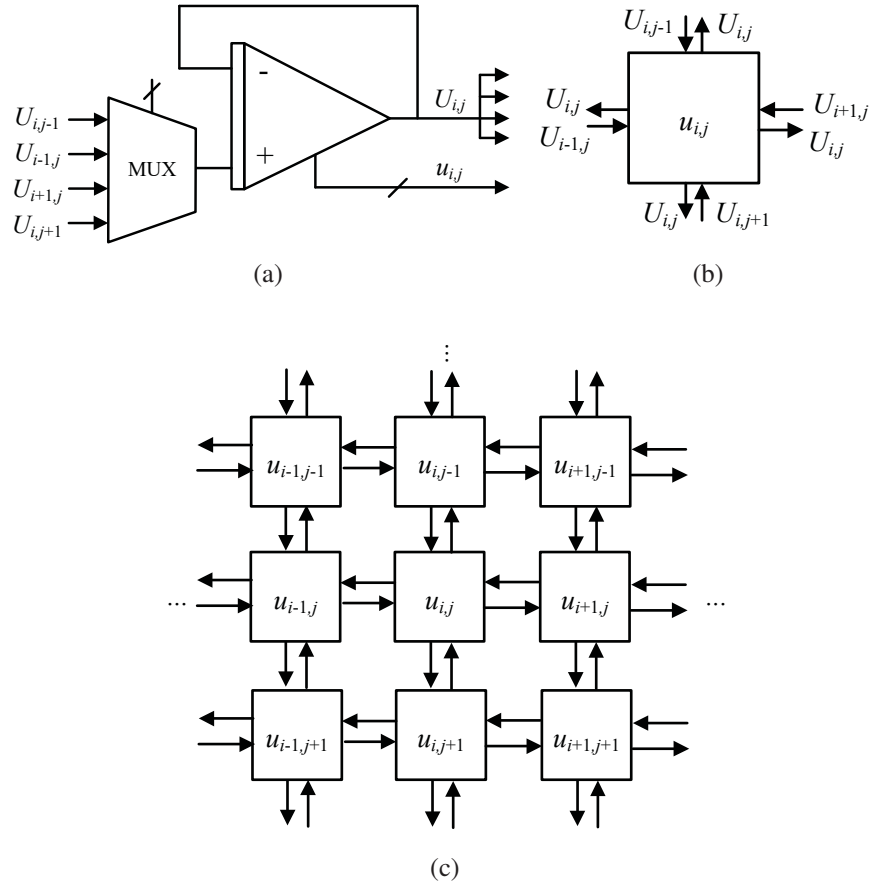


Figure 5.7. Stochastic Laplacian circuits: (a) the circuit diagram of a single element; (b) a symbol and (c) an array of Laplacian circuits solving Laplace's equation (adapted from [4]).

It is equivalent to obtain the steady-state solution by a finite-difference method using an iterative accumulation,

$$u_{n+1}(i, j) = u_n(i, j) + \frac{1}{4}\lambda [u_n(i-1, j) + u_n(i+1, j) + u_n(i, j-1) + u_n(i, j+1) - 4u_n(i, j)], \quad (5.27)$$

where  $u_n(i, j)$  is the estimate of the solution at iteration  $n$  at point  $(i, j)$  and  $\lambda$  is the step size. It can also be interpreted as solving a steady-state heat equation given by

$$\frac{du}{dt} = \nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}. \quad (5.28)$$

After a long enough period of time, the heat stops to flow, i.e.,  $du/dt = 0$ , and the temperature at each point becomes a fixed value given a fixed boundary condition. Then the function of the circuit in Fig. 5.7(a) can be formulated as in a stochastic ODE solver that solves (5.28). Since the value of each point is supplied to the adjacent four points to accomplish the update, four copies of the DSS's ( $U_{i,j}$ 's in Fig. 5.7(b)) that encode the value are required and they are connected as shown in Fig. 5.7(c).

However, due to the loss of accuracy and inefficiency using a multiplexer, an accumulative parallel counter (APC) and a shifter are used to improve the accuracy and efficiency of stochastic additions [51]. The modified stochastic Laplacian circuit is shown in Fig. 5.8. The expectation of the 3-bit binary number from the APC is then  $\mathbb{E}[U_{sum}] = [u_n(i-1, j) + u_n(i+1, j) + u_n(i, j-1) + u_n(i, j+1)]$ , and the  $1/4$  factor is implemented by a 2-bit right-shift. The stochastic integrator is modified to update  $u_{i,j}$  with  $u_{i,j} - \lambda [U_{i,j} + 1/4 U_{sum}]$  for each clock cycle.

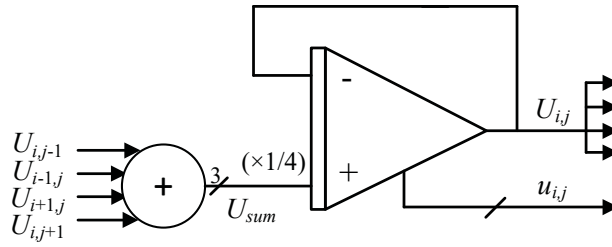


Figure 5.8. Modified stochastic Laplacian circuit using APC.

## 5.6.2 Experiments and results

A steady-state heat equation is given by (5.28). To solve it numerically, an area of interest,  $[0, 1]^2$  or a unit square, is selected, and let the boundary condition be

$$u(x, y) = \begin{cases} 1 & \text{when } y = 0 \ (0 < x < 1) \text{ or } x = 1 \ (0 < y < 1), \\ 0 & \text{when } y = 1 \ (0 < x < 1) \text{ or } x = 0 \ (0 < y < 1). \end{cases} \quad (5.29)$$



The area is divided into  $20 \times 20$  squares so  $\Delta x = \Delta y = 0.05$ . The values at the corner of the unit square area are not considered to avoid contradiction. Since the values on the edges of the square area are defined by the boundary condition, so a  $18 \times 18$  stochastic Laplacian circuit array can solve the problem. Fig. 5.9(a) shows the results produced by the stochastic circuits and Fig. 5.9(b) is the solution provided by a MATLAB software program with double precision. By using a 10-bit counter in the stochastic integrator, the RMSE of the results is as low as  $1.2 \times 10^{-3}$ .

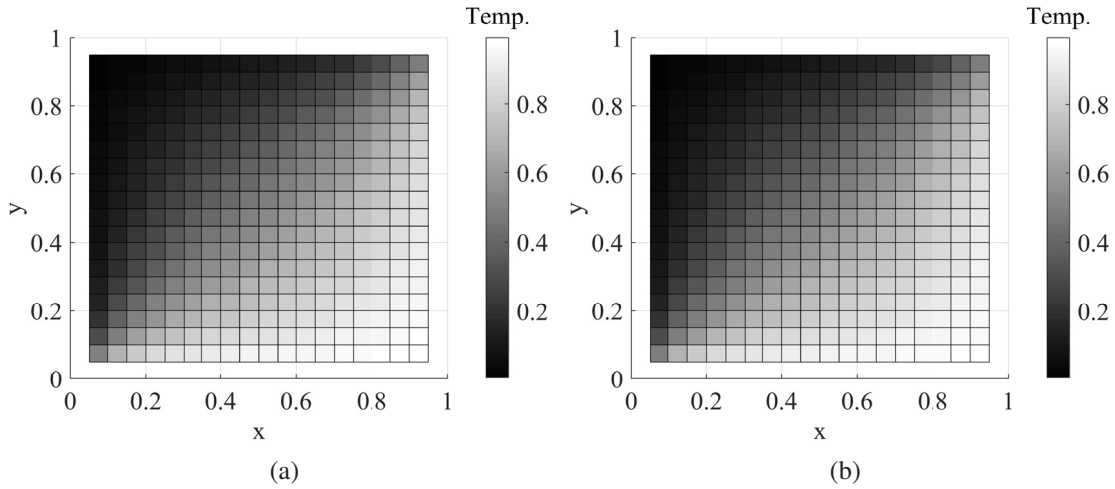


Figure 5.9. Simulation results for a steady-state heat equation produced by (a) the stochastic Laplacian circuits and (b) MATLAB program using double precision.

## 5.7 Summary

A novel ODE solver is proposed that uses a stochastic integrator to implement the accumulative function of the Euler method. We show that a stochastic integrator is an unbiased estimator for an Euler numerical solution. Unlike in conventional stochastic circuits, in which long stochastic sequences are required to produce a result with a high accuracy, the proposed stochastic ODE solver provides an estimate of the solution for every bit in the stochastic sequence or DSS, thus significantly reducing the latency and energy consumption of the circuit. Complex ODE solvers are constructed for solving nonhomogeneous ODEs, systems of ODEs and higher-order ODEs. Experimental results show that with limited loss of accuracy, the stochastic ODE solvers using 8-bit counters

provide an average energy saving of 46% (up to 74%),  $8\times$  throughput per area (up to nearly  $12\times$ ) and a runtime reduction of 72% (up to 82%) compared to their 8-bit binary counterparts. Additionally, a modified stochastic Laplacian circuit is proposed that uses the stochastic integrator and the APC to solve a large-scale steady-state heat equation with a high accuracy.

## Chapter 6

# Gradient Descent Using Stochastic Circuits for Efficient Training of Learning Machines

Gradient descent (GD) is a widely used optimization algorithm in machine learning. In this chapter, a novel stochastic computing-based gradient descent circuit (SC-GDC) is proposed with a dynamic stochastic sequence (DSS) encoding the gradient information. Inspired by the spiking neuron model in Table 1.1, a stochastic integrator is used to optimize the weights in a learning machine by its “inhibitory” and “excitatory” inputs. Specifically, two AND (or XNOR) gates for the unipolar representation (or the bipolar representation) and one stochastic integrator are, respectively, used to implement the multiplications and accumulations in a GD algorithm. Thus, the SC-GDC is very area- and power-efficient. As per the formulation of the proposed SC-GDC, it provides unbiased estimates of the optimized weights in a learning algorithm. The proposed SC-GDC is then used to implement an least-mean-square (LMS) algorithm in an adaptive filter (AF) and an softmax regression (SR). With a similar accuracy, the proposed design achieves more than  $30\times$  improvement in throughput per area (TPA) and consumes less than 13% of the energy per training sample, compared with a fixed-point implementation. Moreover, a signed SC-GDC is proposed for training complex neural networks (NNs). It is shown that for a 784-128-128-10 fully-connected NN, the signed SC-GDC produces a similar training result to its 16-bit fixed-point counterpart, while achieving more than 89.1% energy saving and 66% reduction in training time with about  $38\times$  improvement in TPA.

## 6.1 Introduction

Deep learning utilizes a computational model to automatically discover intricate structures from large raw data by following a general-purpose training procedure. By using a multiple-layer computational model, it has produced many promising results for various tasks including object recognition, natural language processing and autonomous driving [131]. However, a larger computational load is imposed on training a learning machine as a model becomes more complex; e.g., tens of millions of parameters or weights need to be optimized for image recognition in AlexNet [132]. To improve performance, graphics processing units (GPUs) with massively parallel computing resources have widely been used for machine learning.

To further improve performance, machine learning specific chips have been developed, such as the TrueNorth neuro-chip [133], the tensor processing unit (TPU) [134], and the Minerva [135]. However, most of these chips are designed for inference rather than training or optimizing the weights in a learning process. A recent study shows that only three days are required to train a network playing the Go game using four TPUs [136]. Nevertheless, this is still not energy-efficient for mobile or embedded applications.

To reduce the energy consumption in a learning system, quantization and binarization have been shown to be effective [137, 138]. However, both methodologies are mostly implemented in software and no dedicated hardware is available for training. In [137], binarized weights and activations are used to drastically reduce memory usage during the inference phase, whereas in the backpropagation, the binarization is not applicable, and real values are used to compute the optimal weights by using GD.

In this chapter, a novel SC-GDC is proposed for the efficient training of learning machines using stochastic circuits. In the proposed design, the gradient information of a training sample is carried by DSS's. It is different from the conventional belief that the gradient value used during training cannot tolerate much inaccuracy [137].

By using the proposed SC-GDCs, the loss functions of an LMS AF and an SR are minimized to obtain optimized weights. The simulation results show that SC-GDC-based LMS weight update unit achieves a higher accuracy with less than 0.1% of the computation time than a previous stochastic design. For handwritten-digit recognition, the proposed SR

unit using an SC-GDC array produces a similar test accuracy to a software implementation using the same SR model. It takes only 42.6% of the computation time and less than 16% of the energy of a fixed-point design. A more complex 784-128-128-10 NN is trained by a signed SC-GDC array. The signed SC-GDC achieves more than 88% energy saving and 82% reduction in time compared to its 16-bit fixed-point implementation while preserving a similar test accuracy.

The variance bound for the proposed SC-GDC is given by an error analysis. Moreover, it is shown that sharing the random number generators (RNGs) for generating some of the input stochastic sequences reduces the variance of the computed result.

## 6.2 Background

### 6.2.1 Gradient descent

As a basic optimization algorithm, GD has widely been used in machine learning to optimize the weights of a model by minimizing the loss function. Let  $L(\mathbf{w})$  be a multivariate differentiable loss function and the vector  $\mathbf{w}$  be the weights in a learning machine, GD computes the local minimum of the loss function by the following iterative optimization [139]:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta \nabla L(\mathbf{w}_i), (i = 0, 1, 2, \dots), \quad (6.1)$$

where  $\mathbf{w}_i$  is the optimized weight vector at the  $i$ th step;  $\eta$  is a constant or variable step size, or learning rate, which determines how fast the model learns;  $\nabla L(\mathbf{w}_i)$  is the gradient of the loss function at  $\mathbf{w} = \mathbf{w}_i$ .  $\nabla L(\mathbf{w})$  is given by

$$\nabla L(\mathbf{w}) = \left[ \frac{\partial L(\mathbf{w})}{\partial w_1}, \frac{\partial L(\mathbf{w})}{\partial w_2}, \dots, \frac{\partial L(\mathbf{w})}{\partial w_j}, \dots \right], \quad (6.2)$$

where  $w_j$  is the  $j$ th element in vector  $\mathbf{w}$ . If the step size is constant, the optimization result at the  $k$ th step can be obtained by accumulating (6.1) from  $i = 0$  to  $k - 1$ ,

$$\mathbf{w}_k = \mathbf{w}_0 - \eta \sum_{i=0}^{k-1} \nabla L(\mathbf{w}_i), \quad (6.3)$$

where the vector,  $\mathbf{w}_0$ , is usually initialized randomly [139].

### 6.2.2 Stochastic integrator

Stochastic integrators are sequential stochastic computing (SC) elements that accumulate the difference between two stochastic sequences [4]. As shown in Fig. 4.6, a stochastic integrator consists of an  $n$ -bit counter, an RNG and a comparator. The counter updates its value by [89] where  $a_i$  and  $b_i$  are the  $i$ th bits from the stochastic sequences  $A$  and  $B$  respectively, i.e., they are the values of  $A$  and  $B$  at the  $i$ th clock cycle since each bit is generated per clock cycle.  $C_i$  and  $C_{i+1}$  are the  $n$ -bit binary numbers stored in the counter at the  $i$ th and  $(i + 1)$ th clock cycles. Equivalently, we have

$$C_{i+1} = C_i + a_i - b_i. \quad (6.4)$$

As in an SNG, the output stochastic sequence is generated by comparing the  $n$ -bit binary number with an  $n$ -bit random number generated by the RNG. So, at the  $i$ th clock cycle, the probability generating a 1 equals to  $2^{-n}C_i$ , i.e., the value encoded by the  $n$ -bit binary number is  $P_i = 2^{-n}C_i$  in the unipolar representation. Normalizing (6.4) by  $2^{-n}$  leads to

$$P_{i+1} = P_i + 2^{-n}(a_i - b_i). \quad (6.5)$$

Assume the initial value is  $P_0$ , then by an iterative accumulation of (6.5) from  $i = 0$  to  $k - 1$ , the value encoded by the counter at the  $k$ th clock cycle is obtained as

$$P_k = P_0 + 2^{-n} \sum_{i=0}^{k-1} (a_i - b_i). \quad (6.6)$$

Taking the expectation of (6.6) gives us

$$\mathbb{E}[P_k] = P_0 + 2^{-n} \sum_{i=0}^{k-1} (\mathbb{E}[a_i] - \mathbb{E}[b_i]). \quad (6.7)$$

Comparing (6.7) and (6.3), the stochastic integrator provides an unbiased estimate of a weight optimized by the GD algorithm, i.e.,  $\mathbb{E}[P_k] = \mathbf{w}_k$ , under the conditions that: 1) the stochastic integrator is initialized with  $\mathbf{w}_0$ ; 2) the expectation of the difference of the stochastic sequences equals to the negative of gradient, i.e.,  $\mathbb{E}[a_i] - \mathbb{E}[b_i] = -\nabla L(\mathbf{w}_i)$ ; and 3) the step size equals to  $2^{-n}$  [140].

## 6.3 Proposed SC-GDC design

### 6.3.1 SC-GDC circuit design

Fig. 6.1 shows the proposed unipolar SC-GDC for optimizing a weight,  $w_{i,j}$ , i.e., the  $j$ th element in the vector  $\mathbf{w}$  at time step  $i$ . If there are  $N$  elements in  $\mathbf{w}$ ,  $N$  SC-GDCs are required to optimize the  $N$  weights. In Fig. 6.1,  $F(\mathbf{w}_i, \mathbf{x}_i)$  is the inferred value given by the model, and  $t_i$  is the target or desired output value for input  $\mathbf{x}_i$ , which is used to supervise the training of a model.  $\partial F(\mathbf{w}_i, \mathbf{x}_i)/\partial w_{i,j}$  is the input signal for a linear model. For an NN model, these signals can be obtained by a back-propagation. The stochastic number generators (SNGs) are used to stochastically binarize the inputs of SC-GDC, and the stochastic multiplier and integrator are used to efficiently compute (6.1). The SC-GDC works in an online manner, which means that it uses  $\{F(\mathbf{w}_i, \mathbf{x}_i), \partial F(\mathbf{w}_i, \mathbf{x}_i)/\partial w_{i,j}, t_i\}$  of training sample  $\mathbf{x}_i$  sequentially to update the weight. When the bipolar representation is used, the AND gates are replaced by XNOR gates for multiplication.

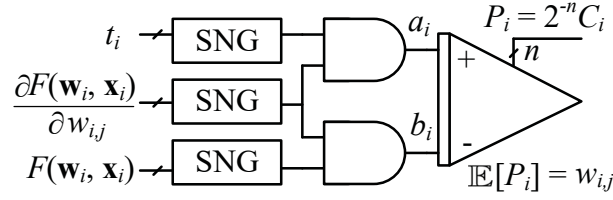


Figure 6.1. Proposed unipolar SC-GDC.

The circuit design of a stochastic integrator is shown in Fig. 6.2. One  $n$ -bit adder is used to compute (6.4) by taking advantage of SC. Since  $a_i$  can only be 0 or 1, it is used as the carry input of the adder.  $n$  copies of  $b_i$  are used to perform “ $-b_i$ ” in (6.4). Specifically,  $(b_i b_i \dots b_i)_2$  represents -1 in 2’s complement when  $b_i$  is 1, otherwise it represents 0. The random numbers are generated by a linear-feedback shift register (LFSR) that works as an RNG. The LFSR and the comparator can be removed if the output sequence is not used for further stochastic computation.

### 6.3.2 Formulation of SC-GDC

To train a learning machine by using GD, the loss function is defined first. The quadratic error between the inferred value and the target value is a commonly used loss function,

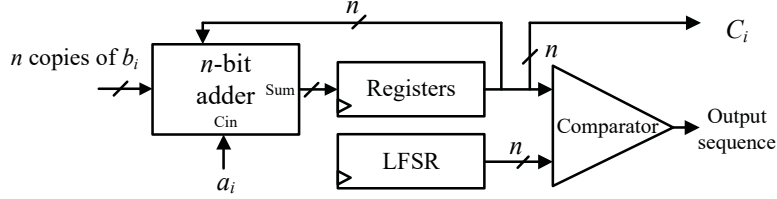


Figure 6.2. Circuit design of a stochastic integrator.

which is also known as the LMS error, given by

$$L(\mathbf{w}_i) = 0.5[t_i - F(\mathbf{w}_i, \mathbf{x}_i)]^2. \quad (6.8)$$

The gradient is the partial derivative of the loss function as per (6.2). For  $w_{i,j}$ , it is the partial derivative of the loss function with respect to  $w_{i,j}$ , i.e.,

$$\frac{\partial L(\mathbf{w}_i)}{\partial w_{i,j}} = -\frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} t_i + \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} F(\mathbf{w}_i, \mathbf{x}_i). \quad (6.9)$$

Equation (6.3) is then transformed to

$$w_{k,j} = w_{0,j} - \eta \sum_{i=0}^{k-1} \left[ -\frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} t_i + \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} F(\mathbf{w}_i, \mathbf{x}_i) \right]. \quad (6.10)$$

As shown in Fig. 6.1, two AND gates are used to implement the multiplications in (6.9), and we have

$$\begin{aligned} \mathbb{E}[a_i] &= \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} t_i = f^-(i), \\ \mathbb{E}[b_i] &= \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} F(\mathbf{w}_i, \mathbf{x}_i) = f^+(i), \end{aligned} \quad (6.11)$$

where  $a_i$  and  $b_i$  are the two output bits of the AND gates and functions  $f^+(i)$  and  $f^-(i)$  contain the gradient information of training sample  $i$ . Then, sequences  $a$  and  $b$  can be considered as DSS's encoding  $f^-(i)$  and  $f^+(i)$  respectively. On the other hand, since each training sample is randomly picked for each iteration or for different  $i$ ,  $f^+(i)$  and  $f^-(i)$  are random signals. Sequences  $a$  and  $b$  encoding the gradient information are, subsequently, used as the inputs for the stochastic integrator to implement the accumulation in (6.10). As per (6.7) and (6.11), the expectation of the value encoded by the counter at the  $k$ th clock cycle is given by

$$\mathbb{E}[P_k] = P_0 + \frac{1}{2^n} \sum_{i=0}^{k-1} \left[ \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} t_i - \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} F(\mathbf{w}_i, \mathbf{x}_i) \right]. \quad (6.12)$$



As per (6.10) and (6.12), if the counter is initialized with the value of  $w_{0,j}$ , then the proposed SC-GDC provides an unbiased estimate to the weight to be optimized with a step size of  $2^{-n}$ , i.e.,

$$\mathbb{E}[P_k] = w_{k,j}, \text{ for } \eta = 2^{-n} \text{ and } P_0 = w_{0,j}. \quad (6.13)$$

Therefore, the SC-GDC is used to perform GD-based online learning, and the weights are stochastically optimized by the SC-GDC. Each bit from the DSS accounts for  $2^{-n}$  in the unbiased estimate of the optimized weight and the randomness of each bit can be canceled out during the accumulation. As a result, this method leads to a high accuracy using a DSS encoding the gradient of the training samples.

## 6.4 Error analysis

### 6.4.1 Single-step variance for SC-GDC

The variance of the estimated weights produced by the SC-GDC is obtained by analyzing the probability mass function (PMF) of the result by one-step optimization, i.e., the value of  $w_{i+1}$  updated from  $w_i$ . For the proposed unipolar design in Fig. 6.1, let  $Y = F(\mathbf{w}_i, \mathbf{x}_i)$ ,  $X = \partial F(\mathbf{w}_i, \mathbf{x}_i) / \partial w_{i,j}$ , and  $T = t_i$  for simplicity; further let  $y_s$ ,  $x_s$  and  $t_s$  be the stochastic bits encoding these values, generated by the three SNGs. Thus,  $\mathbb{E}[y_s] = Y$ ,  $\mathbb{E}[x_s] = X$  and  $\mathbb{E}[t_s] = T$ .  $y_s$  and  $x_s$  are independently generated to ensure the correctness of the multiplications, so are  $t_s$  and  $x_s$ . However,  $y_s$  and  $t_s$  are not necessarily independent. If  $y_s$  and  $t_s$  are independently generated, the PMF of  $w_{i+1}$  is listed in Table 6.1. As per (6.5), only when  $t_s = 1$ ,  $x_s = 1$  and  $y_s = 0$ , is  $w_i$  increased by  $2^{-n}$ . When  $t_s = 0$ ,  $x_s = 1$  and  $y_s = 1$ ,  $w_i$  is decreased by  $2^{-n}$ ; otherwise,  $w_i$  does not change.

The variance of a random variable  $x$  is given by

$$\text{Var}[x] = \mathbb{E}[(x - \mathbb{E}[x])^2]. \quad (6.14)$$

Table 6.1. Probability distribution of  $w_{i+1}$  when  $y_s$ ,  $x_s$  and  $t_s$  are independently generated.

| $w_{i+1}$      | Probability          |
|----------------|----------------------|
| $w_i - 2^{-n}$ | $(1 - T)XY$          |
| $w_i$          | $1 - TX - XY + 2TXY$ |
| $w_i + 2^{-n}$ | $(1 - Y)TX$          |

As per Table 6.1 and (6.14), the variance of  $w_{i+1}$  is computed as

$$\text{Var}_{\text{ind}}[w_{i+1}] = 2^{-2n}[XY(1 - XY) + TX(1 - TX) + 2TYX(1 - X)]. \quad (6.15)$$

However, when the same RNG is used to generate  $y_s$  and  $t_s$ , they are not statistically independent. The variance of  $w_{i+1}$  can be computed similarly by using its PMF. The variance of  $w_{i+1}$  is directly given here, by

$$\text{Var}_{\text{share}}[w_{i+1}] = 2^{-2n}(|T - Y|X)(1 - |T - Y|X). \quad (6.16)$$

The variance is reduced by  $2^{-2n+1} \min\{T, Y\}(1 - \max\{T, Y\})$  compared to the case when  $y_s$  and  $t_s$  are independently generated. Therefore, it reduces the variance, thus improving the accuracy when using the same RNG to generate stochastic bits  $y_s$  and  $t_s$ .

## 6.4.2 Multiple-step variance bound

If every stochastic bit is independently generated at each step, i.e., they are temporally independent, the multi-step variance is the summation of the single-step variances [141]. When the RNG is shared, the maximum single-step variance is  $2^{-2n-2}$  only when  $|T - Y|X = 0.5$ . Thus, the variance bound of the estimated optimized weights after  $k$  steps is given by

$$\text{Var}_{\text{bound}}[w_k] = 2^{-2n-2}k. \quad (6.17)$$

As per (6.17), the variance bound exponentially decreases with the bit width  $n$ . When the bipolar representation is used, the variance can be derived similarly, and the bound is given by  $2^{-2n}k$ .

## 6.5 Applications

### 6.5.1 System identification using least-mean-square adaptive filters

To assess the efficiency of the proposed SC-GDC, it is used in an LMS AF for system identification. The block diagram of an AF is shown in Fig. 6.3. An AF system consists of a linear filter and an optimization module that adjusts the weights of the linear filter. It can be considered as a simple learning machine with one neuron and a linear activation

function. An AF has been implemented in approximate arithmetic circuits as a cerebellar model to control eye movement [142, 143].

In an AF system, the output of the linear filter at the  $i$ th step,  $y_i$ , is given by

$$y_i = F(\mathbf{w}_i, \mathbf{x}_i) = \mathbf{w}_i \mathbf{x}_i = \sum_{j=0}^{M-1} w_{i,j} x_{i-M+j+1}, \quad (6.18)$$

where  $M$  is the length of the filter;  $\mathbf{w}_i$  is a vector of  $M$  weights,  $\mathbf{w}_i = [w_{i,0}, w_{i,1}, \dots, w_{i,j}, \dots, w_{i,M-1}]$ ; and  $\mathbf{x}_i$  is the input vector at the  $i$ th step,  $\mathbf{x}_i = [x_{i-M+1}, \dots, x_{i-M+j}, \dots, x_i]^T$ . The desired signal,  $t_i$ , guides the estimation of the weights in the target system [144].

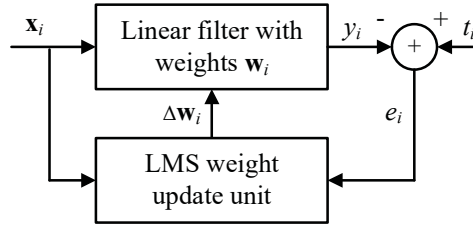


Figure 6.3. An AF.

In the LMS algorithm, the loss function is the quadratic error between  $t_i$  and  $y_i$ , which is given by (6.8) with  $F(\mathbf{w}_i, \mathbf{x}_i) = y_i = \mathbf{w}_i \mathbf{x}_i$ . As per (6.18),  $\partial F(\mathbf{w}_i, \mathbf{x}_i) / \partial w_{i,j} = x_{i-M+j+1}$ . Thus, the weight update unit is constructed from the SC-GDCs as shown in Fig. 6.4.  $M$  SC-GDCs are used to minimize the loss function and hence, to estimate the weights of the target system.

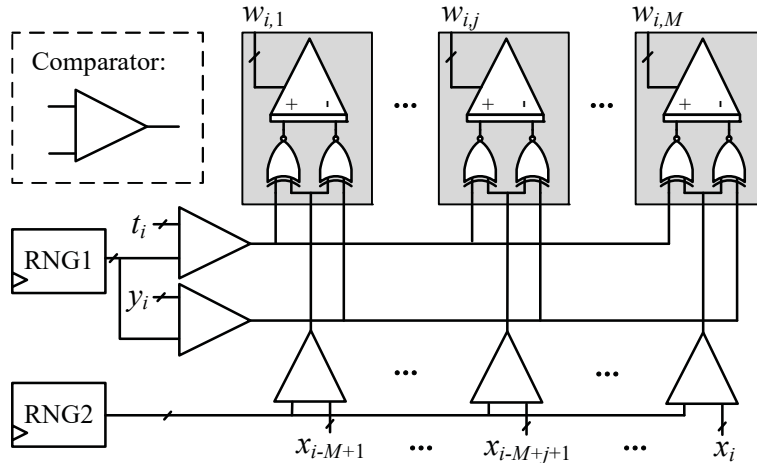


Figure 6.4. LMS weight update unit using SC-GDCs.

Since  $\mathbf{x}_i$ ,  $\mathbf{w}_i$  and  $y_i$  take values within  $[-1, 1]$ , the bipolar representation is used. Thus, the stochastic multipliers are implemented by XNOR gates. The DSS's encoding  $y_i$ ,  $t_i$  and

$\mathbf{x}_i$  are generated by the RNGs and comparators. As discussed in Section 6.4, to reduce the variance as well as the hardware cost, the RNG is shared to generate the sequences for  $y_i$  and  $t_i$ . As the  $M$  SC-GDCs are independent of each other, the RNG generating sequences for the vector  $\mathbf{x}_i$  is also shared. For the same reason, the stochastic sequences encoding  $y_i$  and  $t_i$  are, respectively, shared among different SC-GDCs.

## 6.5.2 Handwritten-digit recognition using softmax regression

A softmax layer is usually the output layer in NNs for multi-class classification, which can be trained by using the GD algorithm. A softmax layer by itself can be considered as a learning machine, which can perform classification of a relatively simple dataset. Fig. 6.5 shows an SR model, where  $w_{m,j}$  denotes the weight of the connection between the  $j$ th input and the  $m$ th neuron. The output of the neurons  $\mathbf{a} = [a_1, \dots, a_M]^T$  is given by

$$\mathbf{a} = \mathbf{w}\mathbf{x} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,J} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M,1} & w_{M,2} & \cdots & w_{M,J} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_J \end{bmatrix}. \quad (6.19)$$

The probability of input  $\mathbf{x}$  belonging to class  $m$ ,  $P(\mathbf{x} \in m|\mathbf{w})$ , is then estimated by the softmax function [145],

$$y_m = P(\mathbf{x} \in m|\mathbf{w}) = \frac{e^{a_m}}{\sum_{k=1}^M e^{a_k}}. \quad (6.20)$$

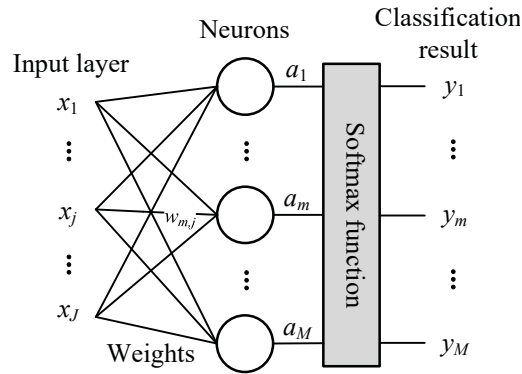


Figure 6.5. An SR model.

The loss function of an SR model is evaluated by the cross entropy, given by [145]

$$L_{ce}(\mathbf{w}) = \sum_{m=1}^M -t_m \log y_m, \quad (6.21)$$

where  $m$  is the class label, and  $t_m$  is the actual classification result in a one-hot code.  $t_m$  is 1 when the input data belongs to class  $m$ ; otherwise, it is 0. The partial derivative of the cross entropy with respect to  $w_{m,j}$  is obtained by

$$\frac{\partial L_{ce}(\mathbf{w})}{\partial w_{m,j}} = \frac{\partial L_{ce}(\mathbf{w})}{\partial y_m} \frac{\partial y_m}{\partial a_m} \frac{\partial a_m}{\partial w_{m,j}} = -(t_m - y_m)x_j, \quad (6.22)$$

which is similar to the gradient of the quadratic error loss function. Thus, a GD-based SR unit can be implemented by using an SC-GDC array with  $y_m$ ,  $x_j$  and  $t_m$  as its inputs. Since  $x_j, t_m, y_m \in [0, 1]$ , the unipolar SC-GDCs are used. The SR unit is shown in Fig. 6.6 for training the MNIST handwritten-digit dataset.

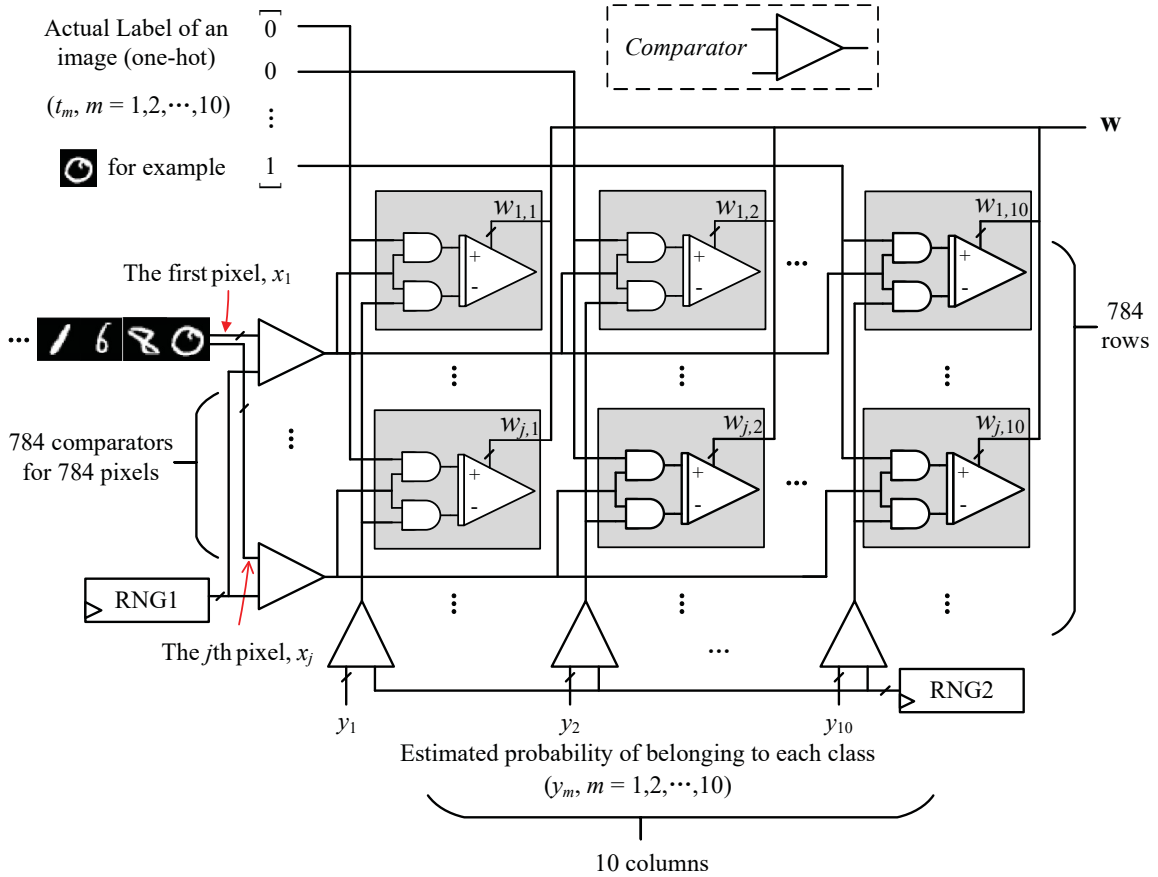


Figure 6.6. An SC-GDC array for the training of the SR model.

In Fig. 6.6, the input samples are  $28 \times 28$  gray-scale images, which are flattened into vectors of 784 values. There are 10 classes of digits, so  $784 \times 10$  weights are to be optimized by at least  $784 \times 10$  SC-GDCs if the weights are updated in parallel. The RNGs and comparators are shared to the maximum extent to reduce the hardware cost and

improve the accuracy as in the LMS weight update unit. Thus, two independent RNGs are used.

At clock  $i$ , one of the training samples, image  $i$ , is loaded to the SR model. Corresponding triples  $\{t_{m,i}, y_{m,i}, x_{j,i}\}$  are computed and connected to the SC-GDC array. Meanwhile, in the SC-GDC array,  $\{t_{m,i}, y_{m,i}, x_{j,i}\}$  are stochastically binarized and the DSS's encoding the gradient information are used to update the weights stored in the stochastic integrators. Since only one clock cycle is used to train one image and to update the weights, the performance of the SC-GDC array is much higher than conventional stochastic circuits where a long sequence is usually used to ensure the accuracy.

However, if a counter needs to count beyond the maximum/minimum value it can reach, an overflow occurs. In this design, an overflow is avoided by adding extra bits and using the 2's complement representation. For example, for an 8-bit counter, it can encode a negative number or a number larger than 1 by adding 2 bits to its most significant bit, e.g., "01 0000 0001" encodes  $257/2^8$  and "11 1111 1111" encodes  $-1/2^8$  in the extended counter. If the weights are still out of the representation range of the extended counter in some other applications, the counter can always be further extended.

## 6.6 Experiments and results

### 6.6.1 Accuracy evaluation

#### System identification using an LMS AF

The proposed LMS weight update unit is used to perform system identification for a high pass finite impulse response (FIR) filter (target system) with 103 weights, so 103 SC-GDCs are required. Pseudorandom numbers are used to generate the DSS's for the simulations of the LMS AF and also the designs below. The frequency response of the AF after training is shown in Fig. 6.7. It indicates that the results produced by the SC-GDCs are very close to the target system. After  $2^{20}$  steps of training using a 15-bit counter<sup>1</sup> (for a step size of  $2^{-15}$ ), the root-mean-squared error (RMSE) between the optimized and actual weights is around  $6.45 \times 10^{-4}$ , and the maximum absolute error is  $2.90 \times 10^{-3}$  for 100 runs. According to

---

<sup>1</sup>Since the weights used in the AF are within  $[-1, 1]$ , a wrapping counter can be used without overflow.

the  $3\text{-}\sigma$  rule and the theoretical bound of variance derived in Section 6.4, the maximum error, in this case, is under the  $3\text{-}\sigma$  bound, i.e.,  $2.90 \times 10^{-3} \ll 3 \times \sqrt{2^{20}/2^{2 \times 15}}$ .

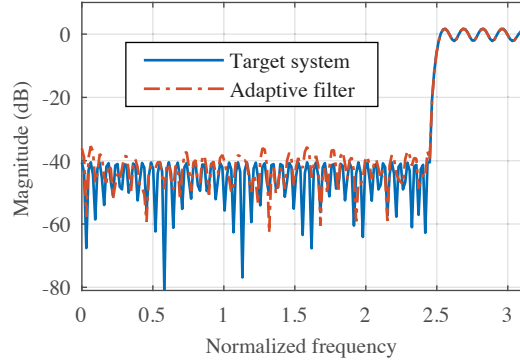


Figure 6.7. System identification results.

### Handwritten-digit recognition using SR

The proposed SR unit in Fig. 6.6 is used to recognize the handwritten digits in the MNIST dataset. 60,000 images are used for training by using the SC-GDCs, and 10,000 images are used to evaluate the optimized weights without cross-validation. In the SC-GDCs, 9-bit saturating counters are used for the experiments with the highest bit representing the integer and the remaining bits representing the fractional part.<sup>2</sup> The weights are initialized with random values.

An epoch of training is completed when the model is exposed to every training sample exactly once. The recognition accuracy and the cross entropy are shown against the number of training epochs in Fig. 6.8. The accuracy and cross entropy are reported every 10,000 steps or training samples.

Fig. 6.8(a) shows that the optimized weights produce a recognition accuracy around 92% for both the training and test data, which is similar to a software implementation using the same SR model [145]. The cross entropy converges rapidly at the first 10,000 samples and it becomes stable after about 4 epochs of training.

<sup>2</sup>A saturating counter is used so that the value represented by the counter is limited between  $[0, 2)$ .

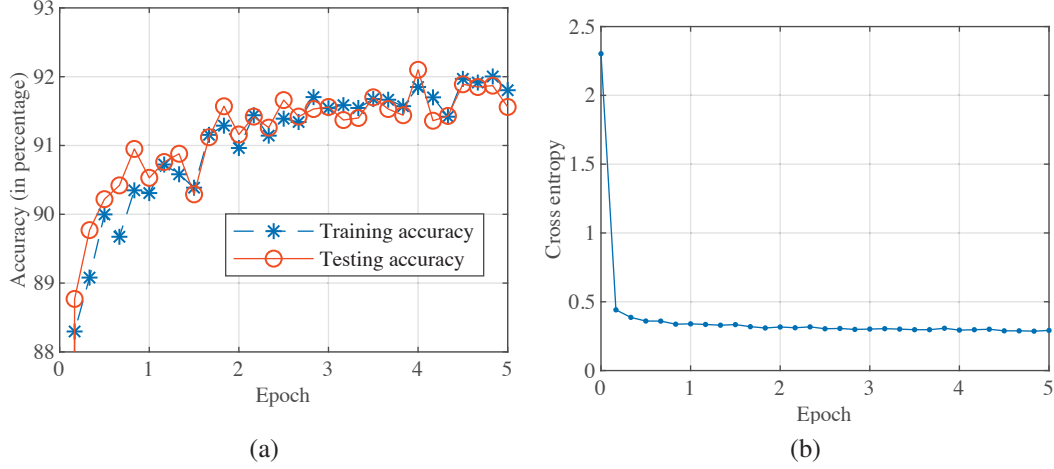


Figure 6.8. (a) Recognition accuracy and (b) cross entropy using the SC-GDCs.

## 6.6.2 Hardware evaluation

The hardware efficiency of the proposed SC-GDC is evaluated in terms of speed, throughput and energy consumption. The proposed designs are implemented in VHSIC Hardware Description Language (VHDL) and synthesized in Synopsys Design Compiler (DC) with a 28-nm STM process. The other parameters and settings are listed in Appendix A. As per the formulation of the SC-GDC, one training sample is loaded to the circuit per clock cycle, and it does not require a long sequence to compute one result as in a conventional SC circuit. Therefore, the proposed design is more efficient than the conventional SC design.

The GD algorithm can be considered as walking from point A (initial weights) to point B (optimal weights) in a high-dimensional space. A larger step size leads to a smaller number of steps to reach point B, thus to a lower latency for the gradient descent circuit. Therefore, to optimize the energy efficiency and speed, a larger step size is preferable. However, if the step size is too large, the optimal point could be missed. It then may incur instability. Therefore, for a fair comparison, the maximum step size in a power of 2 that does not incur instability is used for each application by an exhaustive search, so that their energy efficiency and speed are optimized.



## LMS weight update unit

The LMS weight update unit is compared with an existing SC design [51] and a fixed-point implementation for the same task. The fixed-point GD circuit is shown in Fig. 6.9, consisting of a subtractor, a multiplier, a shifter, an adder and registers. The shifter is used as a multiplier for multiplying the step size,  $2^{-k}$ , where  $k$  is a positive integer.

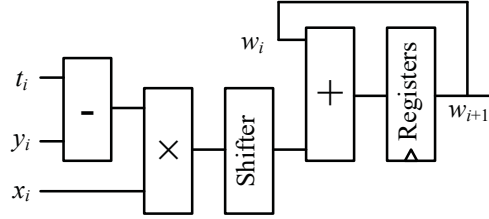


Figure 6.9. Fixed-point LMS weight update circuit.

The accuracy of the fixed-point circuit and the proposed design is matched by observing the convergence of the misalignment. The misalignment is defined as the normalized mean squared error between the optimized weight  $\hat{\mathbf{w}}$  and the actual value  $\mathbf{w}$  of the target system,

$$\text{Misalignment} = \frac{\mathbb{E}[(\mathbf{w} - \hat{\mathbf{w}})^2]}{\mathbb{E}[\mathbf{w}^2]}. \quad (6.23)$$

The convergence curves in misalignment are shown in Fig. 6.10.

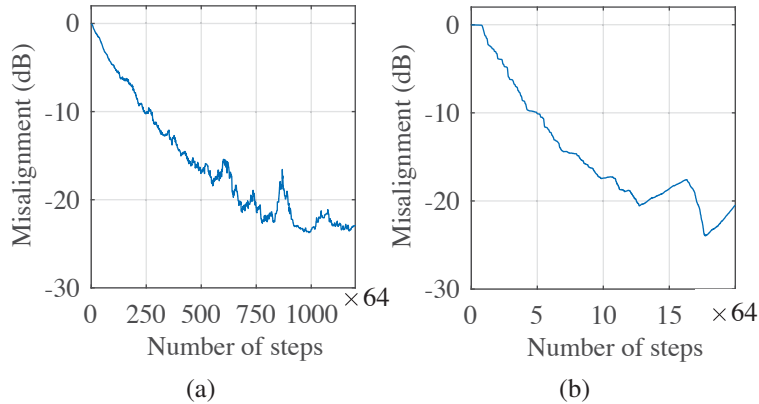


Figure 6.10. Convergence curves in misalignment of the (a) SC-GDC-based and (b) fixed-point LMS weight update units.

For the stochastic design, the step size is  $2^{-11}$  by using 11-bit counters in the SC-GDCs. Fig. 6.10(a) shows that the misalignment for the stochastic design converges to

-23 dB after about 64,000 steps of training. For the fixed-point circuit, a 16-bit design<sup>3</sup> is used with a step size of  $2^{-6}$ , and the misalignment converges to around -23 dB after about 1,100 steps. One hundred simulations are performed to measure the average minimum steps required for the misalignment to decrease to -23 dB. The average minimum steps and the critical path delay are then used to estimate the minimum computation time (“Min. time” in Table 6.2). Also, the energy per operation (EPO) is used to evaluate the energy consumption for training one sample, and the total energy is used to measure the energy cost of the circuits for the whole training process. The TPA is used to evaluate the hardware efficiency by computing maximum number of samples that can be trained by the circuit per unit time and per unit area.

The results in Table 6.2 show that the proposed stochastic design consumes only about 0.047% of the total energy and 0.1% of the computation time of the stochastic design in [51] with a higher accuracy. Compared to the fixed-point implementation, the proposed design achieves 87.4% energy saving for each training sample and  $35.3\times$  TPA. However, the large number of steps leads to a large total energy cost for the proposed design.

Table 6.2. Hardware evaluation of the LMS weight update units.

| Metrics                                 | SC-GDCs              | [51]                 | 16-bit Fixed-point   | Ratio      |
|---|----------------------|----------------------|----------------------|------------|
| Step size                               | $2^{-11}$            | $2^{-10}$            | $2^{-6}$             | -          |
| Steps                                   | 58504                | 16384                | 912                  | -          |
| Min. time (ns)                          | $6.0 \times 10^4$    | $6.5 \times 10^7$    | $3.6 \times 10^3$    | 17:18056:1 |
| EPO (fJ)                                | $1.2 \times 10^4$    | $9.1 \times 10^7$    | $9.5 \times 10^4$    | 1:7583:8   |
| Total energy (fJ)                       | $7.0 \times 10^8$    | $1.5 \times 10^{12}$ | $8.8 \times 10^7$    | 8:17045:1  |
| TPA (Sa./ $\mu\text{s}/\mu\text{m}^2$ ) | $6.7 \times 10^{-5}$ | $1.0 \times 10^{-8}$ | $1.9 \times 10^{-6}$ | 6700:1:190 |
| Misalign. (dB)                          | -23                  | -6                   | -23                  | -          |

## SR unit

For the SR unit, the design in [146] realizes only the inference phase of an SR. To the best of our knowledge, no previous stochastic design is available for training an SR model, thus the proposed design is only compared with a fixed-point implementation using the GD circuit in Fig. 6.9. The weights are considered as converged when the cross entropy is below 0.3.

<sup>3</sup>According to our experiments, 16-bit width for the fixed-point truncation-based implementation is the minimum width that does not incur divergence of the naïve GD algorithm. It also applies to the below applications.

The proposed design takes about 4 epochs to converge when the step size is  $2^{-8}$ , whereas 2 epochs are required for the fixed-point design with a step size of  $2^{-7}$ . One epoch takes 60,000 clock cycles for both the stochastic and fixed-point circuits. The training samples are randomly shuffled at each epoch. Since the RNGs and the comparators are shared among the SC-GDCs, they are omitted in the hardware simulation. As shown in Table 6.3, the hardware evaluation results indicate that the proposed design costs 42.55% computation time of its fixed-point counterpart without any accuracy loss (“Aver. test Accu.” in the table). Meanwhile, 84.4% total energy saving and more than  $70\times$  TPA are achieved by using the SC-GDCs. The small difference in accuracy could be caused by the random initialization and different order of the image samples.

Table 6.3. Hardware evaluation of the SR units.

| Metrics                    | SC-GDCs              | 16-bit Fixed-point   | Ratio  |
|----------------------------|----------------------|----------------------|--------|
| Step size                  | $2^{-8}$             | $2^{-7}$             | -      |
| Epochs                     | 4                    | 2                    | -      |
| Area ( $\mu m^2$ )         | $7.2 \times 10^5$    | $1.1 \times 10^7$    | 0.07:1 |
| Min. time (ns)             | $2.0 \times 10^5$    | $4.7 \times 10^5$    | 1:2.35 |
| EPO (fJ)                   | $5.9 \times 10^5$    | $7.5 \times 10^6$    | 1:13   |
| Total energy (fJ)          | $1.4 \times 10^{11}$ | $9.0 \times 10^{11}$ | 1:6.4  |
| TPA (images/s/ $\mu m^2$ ) | $1.6 \times 10^3$    | $2.3 \times 10^1$    | 70:1   |
| Aver. test Accu.           | 91.76%               | 91.73%               | -      |

### 6.6.3 Discussion

The hardware evaluation results indicate that for weight estimation tasks such as system identification, the proposed stochastic design using SC-GDCs obtains an adequate accuracy, although it takes a larger number of steps than its 16-bit fixed-point counterpart. However, for applications such as image recognition that can tolerate more errors, the number of epochs required for the proposed stochastic design and conventional fixed-point design are on the same level, which indicates a high-performance and energy-efficient stochastic design. However, the accuracy of the image recognition of the MNIST dataset is quite low (around 92%) compared to the state-of-the-art result (around 99%). This is due to the inherent simplicity of the SR model rather than the training method or the SC-GDC. Next, a complex NN model that produces higher recognition

accuracy is used to test the performance of the SC-GDCs, where hundreds of thousands of weights are to be optimized.

## 6.7 Signed SC-GDC units training complex NNs

### 6.7.1 Background for back-propagation

An NN consists of a set of neurons and the connections between them; the neurons are typically organized layer-by-layer. Fig. 6.11 shows an NN with one input layer, two hidden layers and one output layer. The output signals of an NN are generated based on the input signals and the weights of the connections. For an image recognition task, the input signals are the pixels of an image, and the output signals are the classification results. To produce a correct classification for an image, a GD algorithm can be used to train an NN by adjusting the weights of the connections. However, for the hidden layers in an NN, the target value and the loss function cannot be computed directly. Typically, it requires both forward- and backward-propagation (FP and BP) algorithms to obtain the gradients.

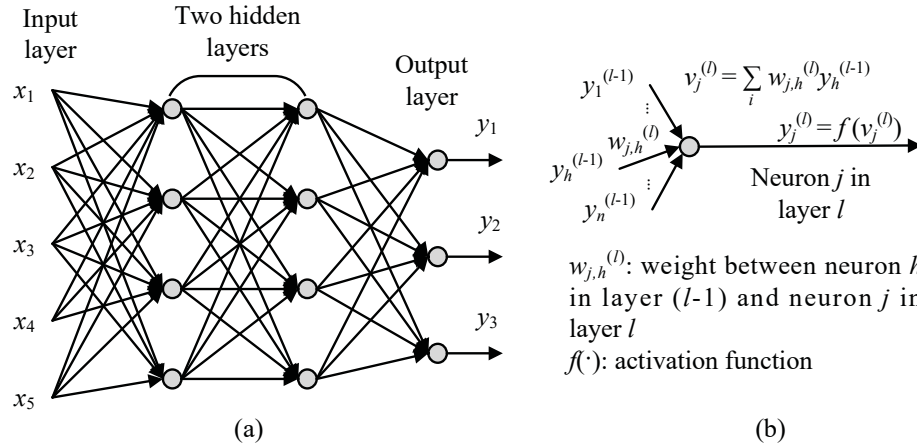


Figure 6.11. (a) A multilayer NN. (b) The function of a neuron during FP.

In FP, each neuron computes the weighted sum of the outputs from the previous layer (or the training data from the input layer). An activation function is then used to decide whether the neuron is activated based on the weighted sum result as shown in Fig. 6.11(b). The output layer is usually a softmax layer for a multi-class classification task. During FP, the weights remain unaltered.

In BP, the error signals,  $\{e_j\}$ , are first obtained as the differences between the outputs and the class-labels of the training data,

$$e_j = t_j - y_j, \quad (6.24)$$

where  $y_j$  is the  $j$ th output of the NN and  $t_j$  is the target output, i.e., the actual class-label in one-hot code.  $t_j$  is 1 when the input data belongs to class  $j$ ; otherwise, it is 0. Then, the local field,  $\delta_j^{(l)}$  for neuron  $j$  in layer  $l$  is computed using the error signals and the weights by

$$\delta_j^{(l)} = \begin{cases} e_j & \text{for neuron } j \text{ in output layer} \\ f'(v_j^{(l)}) \sum_m \delta_m^{(l+1)} w_{m,j}^{(l+1)} & \text{for neuron } j \text{ in hidden layer } l, \end{cases} \quad (6.25)$$

where  $v_j^{(l)}$  is the weighted sum of neuron  $j$  in layer  $l$ ,

$$v_j^{(l)} = \sum_h w_{j,h}^{(l)} y_h^{(l-1)}. \quad (6.26)$$

$w_{j,h}^{(l)}$  denotes the weight of the connection between neuron  $h$  in layer  $l-1$  and neuron  $j$  in layer  $l$ . When neuron  $j$  is in the softmax output layer, its local field equals the error signal, as shown in the first equation in (6.25). The signal flow of the local field during BP in an NN is shown in Fig. 6.12.

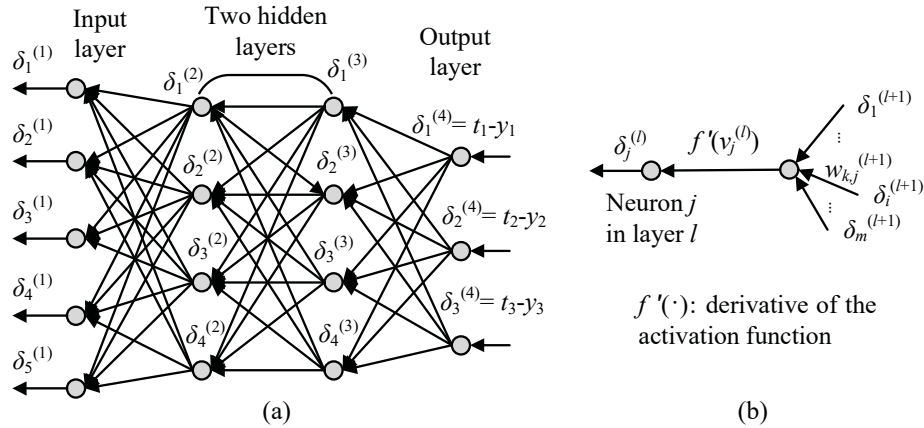


Figure 6.12. (a) The signal flow of local fields during BP. (b) The local field is given by the product of the derivative of the activation function and the weighted sum of local field from the next layer.

Then, the gradient with respect to each weight is given by

$$\nabla w_{j,h}^{(l)} = -\delta_j^{(l)} y_h^{(l-1)}. \quad (6.27)$$

Note that when  $l = 1$ ,  $y_h^{(l-1)}$  is the training data from the input layer.

Finally, the GD algorithm can be used to optimize the weights with the gradient function of  $\nabla w_{j,h}^{(l)}$ .

### 6.7.2 Design of signed SC-GDCs

In an NN, data are usually normalized to have a mean of 0 and the weights are initialized with small random numbers near 0 to improve the training efficiency [139]. This results in a lot of near-0 intermediate results during the computation. Meanwhile, the bipolar SC suffers the most from accuracy loss when representing near-0 numbers due to its large variance [119]. The reason is as follow. Let the probability of 1's in a stochastic sequence be  $p$ . The variance of the sequence is given by  $p(1-p)/L$ , where  $L$  is the sequence length. When  $p = 0.5$ , the variance reaches its maximum value, which indicates a possibly large error. For the bipolar representation,  $x = 2p - 1$  ( $x \in [-1, 1]$ ). So the stochastic sequence encodes  $x$  in the least accurate manner when  $x = 0$  (or  $p = 0.5$ ). It means that using the bipolar representation will dramatically increase the variance of the results, thus leading to increased error. However, it is necessary to be able to encode negative numbers by using stochastic sequences in this application. Thus, instead of using the bipolar representation, a sign bit is added to the unipolar representation to encode a negative number, which leads to the sign-magnitude representation [69]. The variance  $p(1-p)/L$  approaches 0 when  $p$  (or  $x$ ) approaches 0 when using the unipolar representation. In this way, the variance is very small for the encoded values near 0, so the computed results are more accurate than the ones using the bipolar representation.

The stochastic circuits are adjusted to work with the sign-magnitude representation. The signed SNG and multiplier are shown in Figs. 6.13(a) and (b) respectively. In the signed SNG, the  $n$ -bit input  $x$  is in 2's complement. Thus, the sign bit for  $x$  is its most significant bit,  $x[n-1]$ , denoted as  $X_{\text{sign}}$ . The absolute value of a negative number,  $|x|$ , is approximated by flipping all bits using inverters. The RNG and the comparator are then used to generate the magnitude bit,  $X$ . In the signed multiplier, the XOR gate is used to compute the sign bit and the AND gate serves as a unipolar stochastic multiplier. The symbol of a signed stochastic integrator using the sign-magnitude representation is shown in Fig. 6.13(c). The counter in the signed stochastic integrator updates its value according

to Table 6.4. In this way, the signed stochastic integrator implements the same function as an ordinary stochastic integrator for (6.4), where  $a_i$  and  $b_i$  can take either 0, -1 or +1.

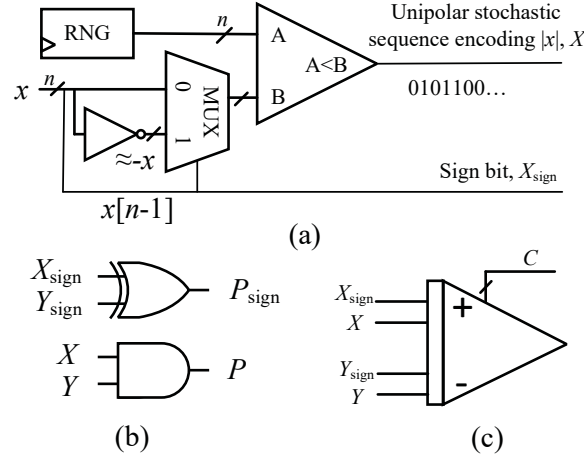


Figure 6.13. (a) A signed SNG. (b) A signed stochastic multiplier. (c) A symbol of signed stochastic integrator.

Table 6.4. The logic of signed stochastic integrator.

| $C_{i+1}$ | $A_{\text{sign}}$ | $A$ | $B_{\text{sign}}$ | $B$ | $C_{i+1}$ | $A_{\text{sign}}$ | $A$ | $B_{\text{sign}}$ | $B$ |
|-----------|-------------------|-----|-------------------|-----|-----------|-------------------|-----|-------------------|-----|
| $C_i$     | 0                 | 0   | 0                 | 0   | $C_i$     | 1                 | 0   | 0                 | 0   |
| $C_{i-1}$ | 0                 | 0   | 0                 | 1   | $C_{i-1}$ | 1                 | 0   | 0                 | 1   |
| $C_i$     | 0                 | 0   | 1                 | 0   | $C_i$     | 1                 | 0   | 1                 | 0   |
| $C_{i+1}$ | 0                 | 0   | 1                 | 1   | $C_{i+1}$ | 1                 | 0   | 1                 | 1   |
| $C_{i+1}$ | 0                 | 1   | 0                 | 0   | $C_{i-1}$ | 1                 | 1   | 0                 | 0   |
| $C_i$     | 0                 | 1   | 0                 | 1   | $C_{i-2}$ | 1                 | 1   | 0                 | 1   |
| $C_{i+1}$ | 0                 | 1   | 1                 | 0   | $C_{i-1}$ | 1                 | 1   | 1                 | 0   |
| $C_{i+2}$ | 0                 | 1   | 1                 | 1   | $C_i$     | 1                 | 1   | 1                 | 1   |

The signed stochastic integrator takes “differential” signals to update its value, i.e., one signal to increase the value and another to decrease the value. Thus, the local field signal in (6.25) has to be rewritten as a differential pair to work with the stochastic integrator. By applying the distributive law of multiplication, (6.24) and (6.25) are combined and rewritten as

$$\begin{aligned}
 \delta_{j,+}^{(l)} &= \begin{cases} t_j & \text{for neuron } j \text{ in output layer} \\ f'(v_j^{(l)}) \sum_k \delta_{k,+}^{(l+1)} w_{k,j}^{(l+1)} & \text{for neuron } j \text{ in hidden layer } l, \end{cases} \\
 \delta_{j,-}^{(l)} &= \begin{cases} y_j & \text{for neuron } j \text{ in output layer} \\ f'(v_j^{(l)}) \sum_k \delta_{k,-}^{(l+1)} w_{k,j}^{(l+1)} & \text{for neuron } j \text{ in hidden layer } l, \end{cases}
 \end{aligned} \tag{6.28}$$

and  $\delta_j^{(l)} = \delta_{j,+}^{(l)} - \delta_{j,-}^{(l)}$ . Thus (6.27) can be rewritten as

$$\nabla w_{j,h}^{(l)} = -(\delta_{j,+}^{(l)} - \delta_{j,-}^{(l)})y_h^{(l-1)}. \quad (6.29)$$

As per the formulation of the SC-GDC, the signed SC-GDC is proposed to calculate the gradients and to update the weights as shown in Fig. 6.14. In a signed SC-GDC, two signed stochastic multipliers and a signed stochastic integrator are used. Also, the magnitude stochastic bits of  $\delta_{j,+}^{(l)}$  and  $\delta_{j,-}^{(l)}$ , i.e., the unipolar stochastic sequences encoding  $|\delta_{j,+}^{(l)}|$  and  $|\delta_{j,-}^{(l)}|$  are generated by the same RNG to reduce hardware cost and variance of the results. It is assumed that  $\delta_{j,+}^{(l)}$  and  $\delta_{j,-}^{(l)}$ , i.e., the local fields are available to the signed SC-GDC by a BP. It means that (6.28) is computed by other methods (such as a systolic array) other than SC, because it results in a significant accuracy loss if the entire BP algorithm is computed in SC. However, the updating of the weights is purely implemented by using the signed SC-GDCs. The overflow of the counters is handled in a similar manner to the SR units.

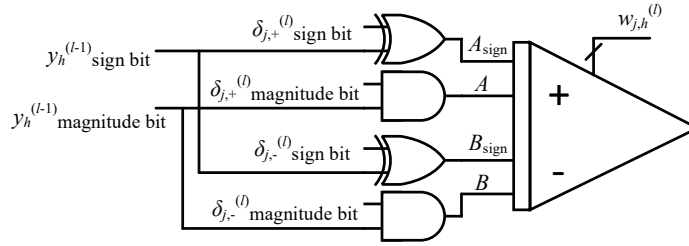


Figure 6.14. Proposed signed SC-GDC. The magnitude bit is a unipolar stochastic bit used to encode the absolute value of the number.

### 6.7.3 Handwritten-digit recognition

The MNIST handwritten-digit dataset is used to test the effectiveness of the proposed signed SC-GDCs. The input data are pre-processed to have a mean value of 0. A 784-128-128-10 fully connected NN is used and hyperbolic tangent (tanh) function is selected as the activation function for the hidden layers, which is given by

$$y = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad (6.30)$$

and its derivative function is

$$\tanh'(x) = 1 - y^2. \quad (6.31)$$



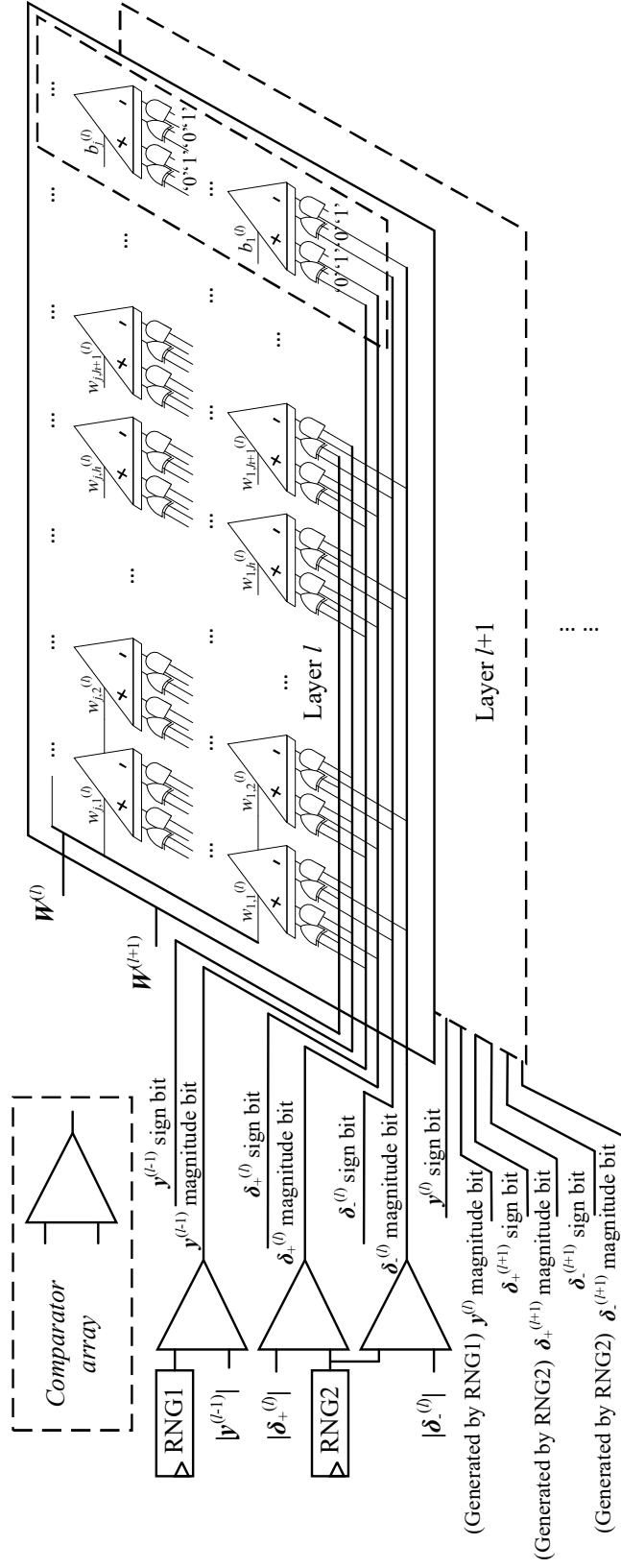


Figure 6.15. Proposed signed SC-GDC array for the training of a multilayer NN.

For this activation function,  $f'(v_j^{(l)})$  in (6.28) is given by (6.31). “Fully connected” means that each neuron in layer  $l$  has connections with every neuron in layer  $l - 1$ . Typically, a fully connected NN has more complex connections to be trained, while a convolutional neural network (CNN) has sparse connections between two adjacent convolutional layers. So, training a fully connected NN can be a more challenging task for the proposed circuitry than training a CNN. Therefore, a fully connected NN is selected instead of a CNN to evaluate the proposed design, though CNN has a better test accuracy in most cases. However, the basic processes of training a fully connected NN and a CNN are similar, i.e., computing the gradients and applying an optimization algorithm, such as the GD, to compute the weights. Therefore, once the gradients are obtained, the proposed method can potentially be used to train a CNN as well. However, some modifications are required to implement a modified GD algorithm for a CNN, such as GD with momentum and adaptive step size.

Similar to the circuit for the training of the SR model in Fig. 6.6, an array of signed SC-GDCs are used to train the weights in all layers. In total,  $784 \times 128 + 128 \times 128 + 128 \times 10 = 118016$  weights and  $128 + 128 + 10 = 266$  biases are trained by 118282 signed SC-GDCs. Only the “naïve” GD algorithm in (6.1) is considered to train the NN model. Other optimization techniques and variants of the GD algorithm, such as cross-validation, weight regularization, momentum terms and adaptive optimization, are not considered. The array of signed SC-GDCs works in the same manner as in the SR training unit. One clock cycle is required to train one sample. Also, the RNGs and comparators can be shared among the inputs and two RNGs are sufficient to generate the stochastic sequences: one for  $\{y_h^{(l-1)}\}$ ; one for  $\{\delta_{j,+}^{(l)}\}$  and  $\{\delta_{j,-}^{(l)}\}$ . So, the cost of SNGs is omitted. The proposed circuit for the training of a multilayer NN is shown in Fig. 6.15. The signed SC-GDCs are organized layer-wise to show their connections and signals instead of the actual mapping of the circuit. The rightmost column of the signed SC-GDCs in each layer is used to train the biases, which can be considered as “weights” with an input of constant 1. Hence, the input signals,  $\mathbf{y}^{(l-1)}$ , for these SC-GDCs encode a 1.

## 6.7.4 Experiments and results

The average test accuracy and cross entropy against epoch are shown in Fig. 6.16<sup>4</sup>. It illustrates that the stochastic circuit using the bipolar representation has a relatively slow convergence for the cross-entropy, thus leading to a low accuracy. Unless the model is trained with a smaller step size, the test accuracy produced by the bipolar stochastic circuits remains at below 95% after 20 epochs. Compared to the fixed-point implementation, the signed SC-GDCs produce a slightly lower accuracy with the same step size of  $2^{-10}$ ; however, it has a similar convergence speed. A double precision floating-point implementation is also compared as a reference; it shows a similar accuracy to the fixed-point design.

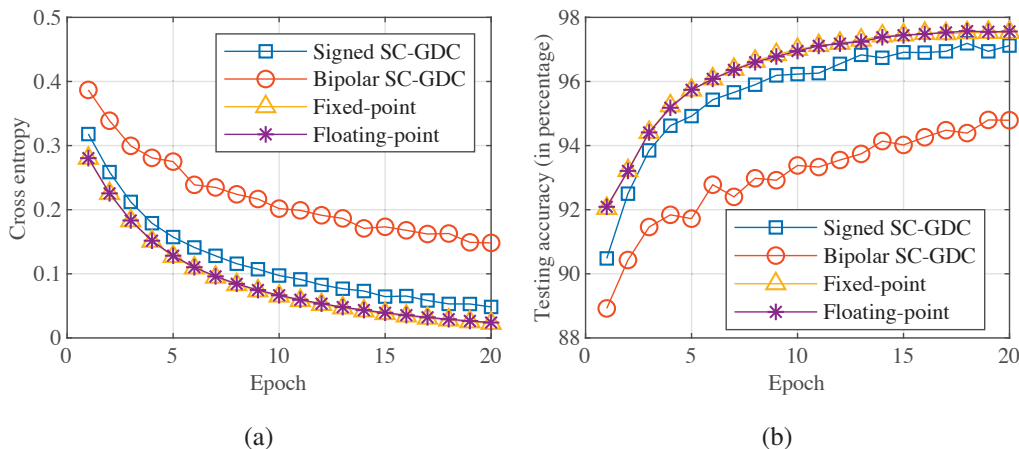


Figure 6.16. (a) Test accuracy and (b) cross entropy produced by the signed SC-GDCs, the bipolar SC-GDCs and the fixed-point implementation.

Fig. 6.17 shows the classification accuracy and cross entropy for different widths of SC-GDCs, for which 8-, 10- and 12-bit SC-GDCs with the same initial values for weights are considered. The 8-bit design converges the fastest during the first few epochs. However, due to the large step size ( $1/2^8$ ), it loses its advantage in accuracy after a few epochs to the 10-bit design, and converges to a slightly lower value. On the other hand, the 12-bit design can produce a finer estimate of the optimal weights. However, it takes a longer time to converge, which would incur a higher energy consumption.

<sup>4</sup>Saturating counters are used here in a fixed-point manner (1 sign bit, 1 most significant bit for the integer part and the other bits for the fractional part), such that they return values within  $[-2, 2)$  for both software and hardware simulations.

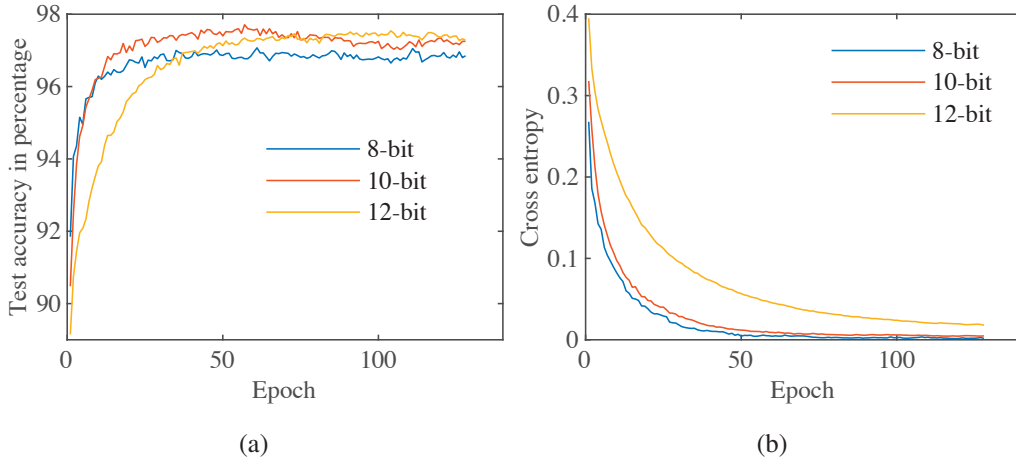


Figure 6.17. (a) Test accuracy and (b) cross entropy produced by the signed SC-GDCs with different widths.

The hardware cost of the circuits are measured and estimated as shown in Table 6.5 by using the same method as for the AF and SR applications. The fixed-point implementation used for comparison is shown in Fig. 6.9, where the additions and accumulations are implemented by fixed-point circuits. Table 6.5 shows that the signed SC-GDC-based design consumes 10.9% of the energy and 25.5% the computation time of the fixed-point implementation, while providing about 38 times TPA, with a similar test accuracy<sup>5</sup> The total sequence length of the DSS's for training 60,000 images for 20 epochs is  $60,000 \times 20 = 1,200,000$ , and it takes the same amount of accumulations for the fixed-point implementation to accomplish this.

## 6.7.5 Related work and discussion

In [28], extended stochastic logic is used to implement both FP and BP of a multilayer perceptron. By using a binary search, a reconfigurable stochastic computational activation unit and an LFSR sharing scheme, the design achieves lower area and energy consumption compared to the binarized neural network, and the floating- and fixed-point

<sup>5</sup>The results are slightly different from [120] since the overflow is not considered in a single SC-GDC in [120] while the overflow is avoided by a saturating counter here. It is assumed that an overflow causes an interrupt and can be processed by a higher level controller in [120], which is more practical due to the rare occurrence of the overflow. Additionally, we found that the comparators can be shared among the SC-GDCs, so they are not considered here, which is different from [120]. Detailed comparisons are shown in Appendix B.

Table 6.5. Hardware evaluation of the signed SC-GDC array training a 784-128-128-10 neural network.

| Metrics                   | Signed SC-GDCs    | 16-bit Fixed-point | Ratio  |
|---------------------------|-------------------|--------------------|--------|
| Step size                 | $2^{-10}$         | $2^{-10}$          | -      |
| Epochs                    | 20                | 20                 | -      |
| Area ( $\mu m^2$ )        | $1.3 \times 10^7$ | $1.7 \times 10^8$  | 0.08:1 |
| Min. time (ns)            | $1.6 \times 10^6$ | $4.7 \times 10^6$  | 1:3    |
| EPO (fJ)                  | $1.2 \times 10^7$ | $1.1 \times 10^8$  | 1:9.2  |
| TPA (image/s/ $\mu m^2$ ) | $5.7 \times 10^1$ | 1.5                | 38:1   |
| Aver. test Accu.          | 97.04%            | 97.49%             | -      |

implementations. With a similar network size and structure, a similar accuracy is obtained in this chapter compared to [28]. However, a relatively long sequence is required for [28] to achieve a high accuracy, which incurs a long latency. Specifically, 256 clock cycles with  $16\times$  parallelization are required to handle one image with a maximum operation frequency of 112.4 MHz, while it takes only 1 clock cycle in this work to accomplish the gradient accumulation of one training sample with a maximum operation frequency of 1.03 GHz. Also, the use of extended stochastic logic in [28] requires an extra stochastic divider and more computation time to convert a stochastic sequence back to a binary number. However, a converter is not required in this design since the value stored in the SC-GDCs is already in 2's complement format.

In [137], the weights and activations are binarized to +1 or -1 to reduce the power consumption and hardware resources. However, the binarization is only applicable to the FP of an NN. In fact, by using real-valued variables and gradient during the training process, a larger workload is imposed on training binarized weights and activations. Compared to [137], this chapter focuses on the efficient training of an NN by stochastic binarization of the gradient instead of the weights and activations. Also, dedicated hardware using stochastic circuits are proposed to perform the GD algorithm. It makes the SC-GDC applicable to most optimization tasks that can be solved by a GD algorithm, such as system identification that clearly cannot be solved by binarizing the weights.

Recently, a TernGrad method is proposed to reduce the communication cost for synchronizing gradients and parameters in distributed training [138]. In TernGrad, the gradient is compressed to only three levels,  $\{-1, 0, +1\}$ , stochastically. Compared to

[138], this work focuses on enhancing the computation efficiency instead of reducing the communication cost. Therefore, the activations and local fields used for training the NN are stochastically binarized, and the stochastic bits are used in the computation of gradients by stochastic circuits instead of fixed- or floating-point multipliers and adders. Due to the simplicity of the proposed stochastic circuits, significant energy saving and hardware efficiency are achieved compared to conventional arithmetic circuits. Generated by the stochastic circuits, the stochastic bits encoding the gradients in an SC-GDC are similar to the ternary gradients, and can be directly used to reduce the communication cost.

The proposed design can also be adapted to deal with more complex learning tasks. For example, to implement a variant of GD algorithm using a dynamically adjustable step size, an additional stochastic sequence can be used to encode the step size. Then, one more stochastic multiplier can be used for the SC-GDC to multiply the additional stochastic sequence. To implement batch learning, multiple stochastic sequences encoding the gradient information can be used as the inputs of the SC-GDCs. It resembles a spiking neuron, in which the probability of an action potential occurring in a postsynaptic neuron is determined by multiple excitatory and inhibitory synapses in the presynaptic neurons.

## 6.8 Summary

In this chapter, a novel SC-GDC for online learning is proposed by using stochastic circuits to implement the GD algorithm. By encoding the gradient information using DSS's, the SC-GDC provides an unbiased estimate for the optimized weights in a learning algorithm. The proposed SC-GDC units are then utilized in system identification and handwritten-digit recognition using an SR model. Compared to a conventional SC system identification design, the proposed design provides  $6.7 \times 10^3$  times TPA improvement,  $1000\times$  speed-up and 99.9% energy reduction. For the SR model, the proposed SC-GDC consumes 42.6% of the computation time and less than 16% of the energy with more than  $70\times$  of the TPA of a 16-bit fixed-point design, while providing a similar accuracy.

Moreover, a signed SC-GDC is proposed to improve the accuracy of the bipolar SC-GDC; it is then used to implement the training of a complex NN. For a 784-128-128-10 fully connected NN, the use of the sign-magnitude representation significantly improves

the accuracy of SC, thus leading to a faster convergence compared to the use of the bipolar representation. Compared to its fixed-point counterpart, a similar accuracy is obtained while 89.1% energy saving per training sample, 66% reduction in training time and about  $38\times$  improvement in TPA are achieved.

The proposed circuitry can be useful for online learning systems where real-time interaction with the environment is required [147] with an energy constraint. It can also be used to train a machine learning model using private or security-critical data on mobile devices if data are sensitive and cannot be uploaded to a cloud computer.

# Chapter 7

## Conclusion and future work

### 7.1 Summary

In this dissertation, the basics and recent developments of stochastic computing (SC) are first reviewed. Two novel schemes are proposed to improve the efficiency of an SC system as well as the accuracy of an SC circuits, i.e., generating low-discrepancy (LD) stochastic sequences using the Sobol sequences and by using the dynamic stochastic sequence (DSS).

To reduce the sequence length while maintaining a high accuracy, Sobol sequences are used to generate LD stochastic sequences. Inspired by the average-over-pool coding scheme in the brain and by exploiting the inherent parallelism of Sobol sequence generation, a parallel Sobol-based stochastic number generator (SNG) is implemented to further improve the energy efficiency and reduce the computation time. It is shown that the stochastic circuits using  $8\times$  parallelism consume approximately 1% of the energy per operation (EPO), achieves up to 89 times improvements in throughput per area (TPA), and requires 1% of the runtime of the conventional linear-feedback shift register (LFSR)-based non-parallelized circuits.

To achieve a high-performance and energy-efficient SC system, a DSS is proposed and used in SC. In the DSS, each bit can have a different probability to be ‘1’ and a varying digital signal (or equivalently, with different probabilities) can be encoded by the DSS. The DSS is used for energy-efficient digital signal processing (DSP) applications, such as frequency mixing, function generation and infinite impulse response (IIR) filtering. An energy and time saving of up to 60% is obtained compared to a 6-bit conventional binary circuit for a function estimation application when processing the same oversampled signal.



However, when processing the same period of signals, the binary circuits consume much lower total energy and require a shorter time using signals sampled at the Nyquist rate than the dynamic stochastic computing (DSC) circuits using oversampled signals.

In addition, the DSS is efficiently used in the design of an ordinary differential equation (ODE) solver and a stochastic computing-based gradient descent circuit (SC-GDC). In these designs, an accumulation of the stochastic bits in DSS's is accomplished by a stochastic integrator that implements the Euler method and a gradient descent algorithm. The randomness can be greatly reduced during the accumulation. A higher energy efficiency, shorter runtime and a higher TPA than their fixed-point implementation are observed in solving a non-homogeneous ODE, a set of ODEs, a second-order ODE and training a softmax regression model with a similar accuracy. Arrays of the stochastic ODE solvers and the SC-GDC are used to solve Laplace's equation and to train a fully connected neural network (NN), respectively. Furthermore, the use of Sobol sequences in a stochastic ODE solver and DSP applications can improve the accuracy of the computed results.

## **7.2 Future work**

Although the proposed methods greatly improve the energy efficiency and performance of the SC circuits, the improvements over their binary counterparts are not as significant. There is still design space unexplored for SC to achieve an even higher energy efficiency and performance.

The connection between SC and a spiking neuron is a topic of interest. In this thesis, it is shown that the proposed DSS is similar to a rate-coded spiking train in a neuron and that the moving average circuit used as a signal reconstruction unit in Chapter 4 can be utilized for decoding the information of a spike train based on rate coding. Similarly, different types of neuronal coding schemes, such as temporal coding, can be studied and used for high-performance and energy-efficient SC. On the other hand, by tracking the behavior of an SC circuit using different neuronal coding schemes, insight could be gained to understand how information is processed in the brain.

The parallel Sobol-based stochastic sequence works similarly to the average-over-pool coding in a neuron, and the stochastic integrator can be viewed as a spiking neuron with one inhibitory and one excitatory synapse. As discussed in Chapters 4 and 6, this “naïve neuron” can be used for IIR filtering and training an NN. Furthermore, a parallel stochastic integrator with multiple inputs is used in a stochastic divider to achieve high performance and energy efficiency in Chapter 3; however, it is a single arithmetic component and far from being competitive with a real neuron in terms of the scale of computing capacity. In reality, numerous synapses co-exist for ultra-high density information processing in one neuron. So a parallel stochastic integrator-based circuit with a higher degree of parallelization, i.e., with a larger number of inputs and more complex connections, could be designed for more complex tasks such as deep learning. Finally, a neuromorphic SC-based computer or accelerator could be built using high-degree parallel stochastic integrators with reconfigurable logic and connections to realize various functions.

# Bibliography

- [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*, 2011, vol. 60, no. 1, pp. 93–105.
- [2] I. L. Dalal, D. Stefan, and J. Harwayne-Gidansky, “Low discrepancy sequences for Monte Carlo simulations on reconfigurable platforms,” in *International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 2008, pp. 108–113.
- [3] P.-S. Ting and J. P. Hayes, “Stochastic logic realization of matrix operations,” in *17th Euromicro Conference on Digital System Design (DSD)*, 2014, pp. 356–364.
- [4] B. R. Gaines, *Stochastic Computing Systems*. Boston, MA: Springer US, 1969, pp. 37–172.
- [5] G. E. Moore *et al.*, “Cramming more components onto integrated circuits,” *Proceedings of the IEEE*, 1998, vol. 86, no. 1, pp. 82–85.
- [6] N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, B. Gill, and J. Maiz, “Radiation-induced soft error rates of advanced CMOS bulk devices,” in *2006 IEEE International Reliability Physics Symposium Proceedings*, March 2006, pp. 217–225.
- [7] J. Keane, X. Wang, D. Persaud, and C. H. Kim, “An all-in-one silicon odometer for separately monitoring HCI, BTI, and TDDB,” *IEEE Journal of Solid-State Circuits*, 2010, vol. 45, no. 4, pp. 817–829.
- [8] J. Henkel, S. Pagani, H. Khdr, F. Kriebel, S. Rehman, and M. Shafique, “Towards performance and reliability-efficient computing in the dark silicon era,”

in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe (DATE)*, 2016, pp. 1–6.

- [9] D. Hisamoto, W.-C. Lee, J. Kedzierski, H. Takeuchi, K. Asano, C. Kuo, E. Anderson, T.-J. King, J. Bokor, and C. Hu, “FinFET-a self-aligned double-gate MOSFET scalable to 20 nm,” *IEEE Transactions on Electron Devices*, 2000, vol. 47, no. 12, pp. 2320–2325.
- [10] J. Cartwright, “Intel enters the third dimension,” *Nature News*, 2011.
- [11] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, “Near-threshold computing: Reclaiming Moore’s law through energy efficient integrated circuits,” *Proceedings of the IEEE*, 2010, vol. 98, no. 2, pp. 253–266.
- [12] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, “The EDA challenges in the dark silicon era: Temperature, reliability, and variability perspectives,” in *Proceedings of the 51st Annual Design Automation Conference (DAC)*, 2014, pp. 1–6.
- [13] A. Deval, A. Ananthakrishnan, and C. Forbell, “Power management on 14 nm intel® Core- M processor,” in *IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, 2015, pp. 1–3.
- [14] V. M. van Santen, J. Martin-Martinez, H. Amrouch, M. M. Nafria, and J. Henkel, “Reliability in super- and near-threshold computing: A unified model of RTN, BTI, and PV,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, Jan 2018, vol. 65, no. 1, pp. 293–306.
- [15] A. M. Kadin and the IEEE Rebooting Computing Committee, “Rethinking structures of computing,” IEEE Rebooting Computing, Tech. Rep., 2014.
- [16] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.

- [17] S. Mittal, “A survey of techniques for approximate computing,” *ACM Comput. Surv.*, Mar. 2016, vol. 48, no. 4, pp. 62:1–62:33.
- [18] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, “A review, classification, and comparative evaluation of approximate arithmetic circuits,” *ACM Journal on Emerging Technologies in Computing Systems*, Aug. 2017, vol. 13, no. 4, pp. 60:1–60:34.
- [19] A. Alaghi and J. P. Hayes, “Survey of stochastic computing,” *ACM Trans. Embed. Comput. Syst.*, May 2013, vol. 12, no. 2s, pp. 92:1–92:19.
- [20] 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*, 2017, vol. 37, no. 8, pp. 1515–1531.
- [21] P. S. Burge, M. R. van Daalen, B. J. P. Rising, and J. S. Shawe-Taylor, “Pulsed neural networks,” in *Pulsed Neural Networks*. Cambridge, MA, USA: MIT Press, 1999, ch. Stochastic Bit-stream Neural Networks, pp. 337–352.
- [22] B. D. Brown and H. C. Card, “Stochastic neural computation. I. computational elements,” *IEEE Transactions on Computers*, 2001, vol. 50, no. 9, pp. 891–905.
- [23] S. Gupta, V. Sindhvani, and K. Gopalakrishnan, “Learning machines implemented on non-deterministic hardware,” *CoRR*, 2014, vol. abs/1409.2620. [Online]. Available: <http://arxiv.org/abs/1409.2620>
- [24] 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 *Proceedings of the 53rd Annual Design Automation Conference (DAC)*, 2016, pp. 124:1–124:6.
- [25] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, “SC-DCNN: Highly-scalable deep convolutional neural network using stochastic computing,” in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017, pp. 405–418.

- [26] 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*, 2017, vol. 25, no. 10, pp. 2688–2699.
- [27] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Analysis and characterization of inherent application resilience for approximate computing,” in *Proceedings of the 50th Annual Design Automation Conference (DAC)*, 2013, pp. 1–9.
- [28] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, “A stochastic computational multi-layer perceptron with backward propagation,” *IEEE Transactions on Computers*, 2018, vol. 67, no. 9, pp. 1273–1286.
- [29] 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 *et al.*, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, 2014, vol. 345, no. 6197, pp. 668–673.
- [30] Y. Liu, Y. Wang, F. Lombardi, and J. Han, “An energy-efficient stochastic computational deep belief network,” in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 1175–1178.
- [31] F. Neugebauer, I. Polian, and J. P. Hayes, “Building a better random number generator for stochastic computing,” in *2017 Euromicro Conference on Digital System Design (DSD)*, Aug 2017, pp. 1–8.
- [32] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue, “Compact and accurate stochastic circuits with shared random number sources,” in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, Oct 2014, pp. 361–366.
- [33] P. Li and D. J. Lilja, “Using stochastic computing to implement digital image processing algorithms,” in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, 2011, pp. 154–161.
- [34] W. Maass and C. M. Bishop, *Pulsed neural networks*. MIT press, 2001.

- [35] W. Poppelbaum, C. Afuso, and J. Esch, “Stochastic computing elements and systems,” in *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*, 1967, pp. 635–644.
- [36] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rosselló, “A new stochastic computing methodology for efficient neural network implementation,” *IEEE Transactions on Neural Networks and Learning Systems*, 2016, vol. 27, no. 3, pp. 551–564.
- [37] J. Han, H. Chen, J. Liang, P. Zhu, Z. Yang, and F. Lombardi, “A stochastic computational approach for accurate and efficient reliability evaluation,” *IEEE Transactions on Computers*, 2014, vol. 63, no. 6, pp. 1336–1350.
- [38] A. Zhakatayev, S. Lee, H. Sim, and J. Lee, “Sign-magnitude SC: getting 10x accuracy for free in stochastic computing for deep neural networks,” in *Proceedings of the 55th Annual Design Automation Conference (DAC)*, 2018, pp. 1–6.
- [39] P. K. Gupta and R. Kumaresan, “Binary multiplication with PN sequences,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1988, vol. 36, no. 4, pp. 603–606.
- [40] A. Alaghi and J. P. Hayes, “STRAUSS: Spectral transform use in stochastic circuit synthesis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015, vol. 34, no. 11, pp. 1770–1783.
- [41] ———, “Fast and accurate computation using stochastic circuits,” in *Proceedings of the conference on Design, Automation & Test in Europe (DATE)*, 2014, pp. 1–4.
- [42] 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*, 2014, vol. 63, no. 6, pp. 1474–1486.
- [43] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, “The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic,” in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 480–487.

- [44] P. Ting and J. P. Hayes, “On the role of sequential circuits in stochastic computing,” in *Proceedings of the on Great Lakes Symposium on VLSI (GLSVLSI)*, 2017, pp. 475–478.
- [45] V. C. Gaudet and A. C. Rapley, “Iterative decoding using stochastic computation,” *Electronics Letters*, 2003, vol. 39, no. 3, pp. 299–301.
- [46] 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*, 2010, vol. 58, no. 9, pp. 4883–4896.
- [47] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, “Delayed stochastic decoding of LDPC codes,” *IEEE Transactions on Signal Processing*, 2011, vol. 59, no. 11, pp. 5617–5626.
- [48] 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*, 2015, vol. 24, no. 4, pp. 1391–1401.
- [49] S. T. Marin, J. Q. Reboul, and L. G. Franquelo, “Digital stochastic realization of complex analog controllers,” *IEEE Transactions on Industrial Electronics*, 2002, vol. 49, no. 5, pp. 1101–1109.
- [50] D. Zhang and H. Li, “A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms,” *IEEE Transactions on Industrial Electronics*, 2008, vol. 55, no. 2, pp. 551–561.
- [51] H. Jiang, C. Shen, P. Jonker, F. Lombardi, and J. Han, “Adaptive filter design using stochastic circuits,” in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 122–127.
- [52] J. F. Keane and L. E. Atlas, “Impulses and stochastic arithmetic for signal processing,” in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2, 2001, pp. 1257–1260.



- [53] R. Kuehnel, “Binomial logic: extending stochastic computing to high-bandwidth signals,” in *Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers*, vol. 2, 2002, pp. 1089–1093.
- [54] N. Yamamoto, H. Fujisaka, K. Haeiwa, and T. Kamio, “Nanoelectronic circuits for stochastic computing,” in *6th IEEE Conference on Nanotechnology (IEEE-NANO)*, vol. 1, 2006, pp. 306–309.
- [55] K. K. Parhi and Y. Liu, “Architectures for IIR digital filters using stochastic computing,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 373–376.
- [56] B. Yuan, Y. Wang, and Z. Wang, “Area-efficient scaling-free DFT/FFT design using stochastic computing,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2016, vol. 63, no. 12, pp. 1131–1135.
- [57] R. Wang, J. Han, B. F. Cockburn, and D. G. Elliott, “Stochastic circuit design and performance evaluation of vector quantization for different error measures,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2016, vol. 24, no. 10, pp. 3169–3183.
- [58] H. Ichihara, T. Sugino, S. Ishii, T. Iwagaki, and T. Inoue, “Compact and accurate digital filters based on stochastic computing,” *IEEE Transactions on Emerging Topics in Computing*, 2016, vol. 7, no. 1, pp. 31–43.
- [59] T. Hammadou, M. Nilson, A. Bermak, and P. Ogunbona, “A 96/spl times/64 intelligent digital pixel array with extended binary stochastic arithmetic,” in *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, 2003.
- [60] A. Alaghi, C. Li, and J. P. Hayes, “Stochastic circuits for real-time image-processing applications,” in *Proceedings of the 50th Annual Design Automation Conference (DAC)*, 2013, pp. 136:1–136:6.
- [61] 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*, 2014, vol. 22, no. 3, pp. 449–462.

- [62] 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*, 2017, vol. 25, no. 5, pp. 1644–1657.
- [63] Y.-C. Kim and M. A. Shanblatt, "Architecture and statistical model of a pulse-mode digital multilayer neural network," *IEEE Transactions on Neural Networks*, 1995, vol. 6, no. 5, pp. 1109–1118.
- [64] J. Zhao and J. Shawe-Taylor, "Stochastic connection neural networks," in *4th International Conference on Artificial Neural Networks*. IET, 1995, pp. 35–39.
- [65] J. Zhao, J. Shawe-Taylor, and M. van Daalen, "Learning in stochastic bit stream neural networks," *Neural Networks*, 1996, vol. 9, no. 6, pp. 991–998.
- [66] Y. Ji, F. Ran, C. Ma, and D. J. Lilja, "A hardware implementation of a radial basis function neural network using stochastic logic," in *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*, 2015, pp. 880–883.
- [67] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan, "DSCNN: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *IEEE 34th International Conference on Computer Design (ICCD)*, 2016, pp. 678–681.
- [68] A. Ren, Z. Li, Y. Wang, Q. Qiu, and B. Yuan, "Designing reconfigurable large-scale deep learning systems using stochastic computing," in *IEEE International Conference on Rebooting Computing (ICRC)*, 2016, pp. 1–7.
- [69] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017, pp. 1–6.
- [70] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze, "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 13–18.

- [71] R. Hojabr, K. Givaki, S. Tayaranian, P. Esfahanian, A. Khonsari, D. Rahmati, and M. H. Najafi, “SkippyNN: An embedded stochastic-computing accelerator for convolutional neural networks,” in *Proceedings of the 56th Annual Design Automation Conference (DAC)*, 2019, pp. 1–6.
- [72] Y. Liu, Y. Wang, F. Lombardi, and J. Han, “An energy-efficient online-learning stochastic computational deep belief network,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Sep. 2018, vol. 8, no. 3, pp. 454–465.
- [73] Y. Liu, L. Liu, F. Lombardi, and J. Han, “An energy-efficient and noise-tolerant recurrent neural network using stochastic computing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Sep. 2019, vol. 27, no. 9, pp. 2213–2221.
- [74] R. Cai, A. Ren, O. Chen, N. Liu, C. Ding, X. Qian, J. Han, W. Luo, N. Yoshikawa, and Y. Wang, “A stochastic-computing based deep learning framework using adiabatic quantum-flux-parametron superconducting technology,” in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA ’19. New York, NY, USA: ACM, 2019, pp. 567–578.
- [75] S. C. Smithson, N. Onizawa, B. H. Meyer, W. J. Gross, and T. Hanyu, “Efficient cmos invertible logic using stochastic computing,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, June 2019, vol. 66, no. 6, pp. 2263–2274.
- [76] J. Liang and J. Han, “Stochastic boolean networks: an efficient approach to modeling gene regulatory networks,” *BMC systems biology*, 2012, vol. 6, no. 1, p. 113.
- [77] P. Zhu, J. Han, L. Liu, and M. J. Zuo, “A stochastic approach for the analysis of fault trees with priority AND gates,” *IEEE Transactions on Reliability*, 2014, vol. 63, no. 2, pp. 480–494.
- [78] P. Zhu, J. Han, L. Liu, and F. Lombardi, “A stochastic approach for the analysis of dynamic fault trees with spare gates under probabilistic common cause failures,” *IEEE Transactions on Reliability*, 2015, vol. 64, no. 3, pp. 878–892.

- [79] —, “Reliability evaluation of phased-mission systems using stochastic computation,” *IEEE Transactions on Reliability*, 2016, vol. 65, no. 3, pp. 1612–1623.
- [80] N. Onizawa, D. Katagiri, W. J. Gross, and T. Hanyu, “Analog-to-stochastic converter using magnetic-tunnel junction devices,” in *Proceedings of the 2014 IEEE/ACM International Symposium on Nanoscale Architectures*, 2014, pp. 59–64.
- [81] R. Perricone, Y. Liu, A. Dingler, X. S. Hu, and M. Niemier, “Design of stochastic computing circuits using nanomagnetic logic,” *IEEE Transactions on Nanotechnology*, 2016, vol. 15, no. 2, pp. 179–187.
- [82] 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 *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 1438–1443.
- [83] Y. Qu, J. Han, B. F. Cockburn, W. Pedrycz, Y. Zhang, and W. Zhao, “A true random number generator based on parallel STT-MTJs,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 606–609.
- [84] H. Lee, A. Lee, F. Ebrahimi, P. K. Amiri, and K. Wang, “Analog to stochastic bit stream converter utilizing voltage-assisted spin hall effect,” *IEEE Electron Device Letters*, 2017, vol. 38, no. 9, pp. 1343–1346.
- [85] Y. Qu, B. F. Cockburn, Z. Huang, H. Cai, Y. Zhang, W. Zhao, and J. Han, “Variation-resilient true random number generators based on multiple STT-MTJs,” *IEEE Transactions on Nanotechnology*, Nov 2018, vol. 17, no. 6, pp. 1270–1281.
- [86] M. Suri, D. Querlioz, O. Bichler, G. Palma, E. Vianello, D. Vuillaume, C. Gamrat, and B. DeSalvo, “Bio-inspired stochastic computing using binary CBRAM synapses,” *IEEE Transactions on Electron Devices*, 2013, vol. 60, no. 7, pp. 2402–2409.

- [87] P. Knag, W. Lu, and Z. Zhang, “A native stochastic computing architecture enabled by memristors,” *IEEE Transactions on Nanotechnology*, 2014, vol. 13, no. 2, pp. 283–293.
- [88] R. Wang, J. Han, B. F. Cockburn, and D. G. Elliott, “Design, evaluation and fault-tolerance analysis of stochastic FIR filters,” *Microelectronics Reliability*, 2016, vol. 57, pp. 111–127.
- [89] N. Saraf, K. Bazargan, D. J. Lilja, and M. D. Riedel, “IIR filters using stochastic arithmetic,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014, pp. 1–6.
- [90] B. Li, M. H. Najafi, and D. J. Lilja, “An FPGA implementation of a restricted boltzmann machine classifier using stochastic bit streams,” in *IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2015, pp. 68–69.
- [91] A. Alaghi and J. P. Hayes, “Exploiting correlation in stochastic circuit design,” in *IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 39–46.
- [92] T.-H. Chen, A. Alaghi, and J. P. Hayes, “Behavior of stochastic circuits under severe error conditions,” *it-Information Technology*, 2014, vol. 56, no. 4, pp. 182–191.
- [93] T.-H. Chen and J. P. Hayes, “Analyzing and controlling accuracy in stochastic circuits,” in *IEEE 32nd International Conference on Computer Design (ICCD)*, 2014, pp. 367–373.
- [94] S. Chu, “New divider design for stochastic computing,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2019, pp. 1–5, early access.
- [95] T. Chen and J. P. Hayes, “Design of division circuits for stochastic computing,” in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 116–121.

- [96] D. Wu and J. S. Miguel, “In-stream stochastic division and square root via correlation,” in *Proceedings of the 56th Annual Design Automation Conference (DAC)*, ser. DAC ’19, 2019, pp. 162:1–162:6.
- [97] C. Ma, S. Zhong, and H. Dang, “Understanding variance propagation in stochastic computing systems,” in *IEEE 30th International Conference on Computer Design (ICCD)*, 2012, pp. 213–218.
- [98] F. Neugebauer, I. Polian, and J. P. Hayes, “Framework for quantifying and managing accuracy in stochastic circuit design,” in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 1–6.
- [99] 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 (DATE)*, 2016, pp. 1550–1555.
- [100] D. Braendler, T. Hendtlass, and P. O’Donoghue, “Deterministic bit-stream digital neurons,” *IEEE Transactions on Neural Networks*, 2002, vol. 13, no. 6, pp. 1514–1525.
- [101] D. Jenson and M. Riedel, “A deterministic approach to stochastic computation,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–8.
- [102] H. Sim and J. Lee, “Cost-effective stochastic mac circuits for deep neural networks,” *Neural Networks*, 2019, vol. 117, pp. 152 – 162.
- [103] P. Gonzalez-Guerrero, X. Guo, and M. Stan, “SC-SD: Towards low power stochastic computing using sigma delta streams,” in *2018 IEEE International Conference on Rebooting Computing (ICRC)*, 2018, pp. 1–8.
- [104] S. R. Faraji, M. H. Najafi, B. Li, D. J. Lilja, and K. Bazargan, “Energy-efficient convolutional neural networks with deterministic bit-stream processing,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 1757–1762.

- [105] S. Liu and J. Han, “Energy efficient stochastic computing with Sobol sequences,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 650–653.
- [106] H. Niederreiter, *Random Number Generation and quasi-Monte Carlo Methods*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.
- [107] D. P. Kroese, T. Taimre, and Z. I. Botev, *Handbook of Monte Carlo methods*. John Wiley & Sons, 2013, vol. 706.
- [108] P. Bratley and B. L. Fox, “Algorithm 659: Implementing Sobol’s quasirandom sequence generator,” *ACM Trans. Math. Softw.*, Mar. 1988, vol. 14, no. 1, pp. 88–100.
- [109] D. E. Knuth, *The art of computer programming: sorting and searching*. Pearson Education, 1998, vol. 3.
- [110] K. E. Batcher, “Sorting networks and their applications,” in *Proceedings of the Spring Joint Computer Conference*, 1968, pp. 307–314.
- [111] R. H. Chan, Chung-Wa Ho, and M. Nikolova, “Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization,” *IEEE Transactions on Image Processing*, Oct 2005, vol. 14, no. 10, pp. 1479–1485.
- [112] S. Wolfram, *A New Kind of Science*. Wolfram Media, 2002. [Online]. Available: <http://www.wolframscience.com>
- [113] F. Maloberti, “Non conventional signal processing by the use of sigma delta technique: a tutorial introduction,” in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 6, May 1992, pp. 2645–2648.
- [114] M. Kozak and I. Kale, *Oversampled delta-sigma modulators: Analysis, applications and novel topologies*. Springer Science & Business Media, 2003.
- [115] R. Schreier, G. C. Temes *et al.*, *Understanding delta-sigma data converters*. IEEE press Piscataway, NJ, 2005, vol. 74.

- [116] V. da Fonte Dias, “Sigma-delta signal processing,” in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, May 1994, pp. 421–424.
- [117] P. Mars and W. J. Poppelbaum, *Stochastic and deterministic averaging processors*. Peter Peregrinus Press, 1981.
- [118] W. G. McCallum, D. Hughes-Hallett, A. M. Gleason, D. O. Lomen, D. Lovelock, J. Tecosky-Feldman, T. W. Tucker, D. Flath, T. Thrash, K. R. Rhea *et al.*, *Multivariable calculus*. Wiley, 1997, vol. 200, no. 1.
- [119] S. Liu and J. Han, “Toward energy-efficient stochastic circuits using parallel sobol sequences,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, July 2018, vol. 26, no. 7, pp. 1326–1339.
- [120] S. Liu, H. Jiang, L. Liu, and J. Han, “Gradient descent using stochastic circuits for efficient training of learning machines,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Nov 2018, vol. 37, no. 11, pp. 2530–2541.
- [121] M. Harris, “Fast fluid dynamics simulation on the GPU,” *GPU gems*, 2004, vol. 1, pp. 637–665.
- [122] M. Januszewski and M. Kostur, “Accelerating numerical solution of stochastic differential equations with CUDA,” *Computer Physics Communications*, 2010, vol. 181, no. 1, pp. 183 – 188.
- [123] K. Olynyk, “System and method for improved digital differential analyzer,” 2003, US Patent 6,510,442. [Online]. Available: <https://www.google.ca/patents/US6510442>
- [124] W. Bong, “Digital differential analyzer,” Apr. 23 2002, US Patent 6,377,265. [Online]. Available: <https://www.google.com/patents/US6377265>
- [125] J. C. Butcher, *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*. New York, NY, USA: Wiley-Interscience, 1987.



- [126] J. P. Hayes, “Introduction to stochastic computing and its challenges,” in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, 2015, pp. 59:1–59:3.
- [127] M. Gerber and N. Chopin, “Sequential quasi Monte Carlo,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2015, vol. 77, no. 3, pp. 509–579.
- [128] Y. Liu and K. K. Parhi, “Computing complex functions using factorization in unipolar stochastic logic,” in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2016, pp. 109–112.
- [129] B. R. Land, “DDA on FPGA,” <http://people.ece.cornell.edu/land/courses/ece5760/DDA/index.htm>, 2006, accessed: 2016-10-12.
- [130] S. J. Farlow, *Partial differential equations for scientists and engineers*. Courier Corporation, 1993.
- [131] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, 2015, vol. 521, no. 7553, pp. 436–444.
- [132] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proceedings of Conference on Neural Information Processing Systems (NIPS)*, 2012.
- [133] F. Akopyan and *et al.*, “TrueNorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015, vol. 34, no. 10, pp. 1537–1557.
- [134] N. P. Jouppi and *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.
- [135] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, “Minerva: Enabling low-power, highly-accurate

deep neural network accelerators,” in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2016, pp. 267–278.

- [136] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, “Mastering the game of go without human knowledge,” *Nature*, 2017, vol. 550, no. 7676, pp. 354–359.
- [137] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1,” *arXiv preprint arXiv:1602.02830*, 2016.
- [138] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, “TernGrad: Ternary gradients to reduce communication in distributed deep learning,” in *Proceedings of Conference on Neural Information Processing Systems (NIPS)*, 2017, pp. 1509–1519.
- [139] S. S. Haykin, *Neural networks and learning machines*. Pearson Upper Saddle River, NJ, USA:, 2009, vol. 3.
- [140] S. Liu and J. Han, “Hardware ODE solvers using stochastic circuits,” in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017, pp. 1–6.
- [141] W. Qian, C. Wang, P. Li, D. J. Lilja, K. Bazargan, and M. D. Riedel, “An efficient implementation of numerical integration using logical computation on stochastic bit streams,” in *Proceedings of International Conference On Computer Aided Design (ICCAD)*, 2012, pp. 156–162.
- [142] H. Jiang, L. Liu, and J. Han, “An efficient hardware design for cerebellar models using approximate circuits: special session paper,” in *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2017, pp. 1–2.
- [143] H. Jiang, L. Liu, P. P. Jonker, D. G. Elliott, F. Lombardi, and J. Han, “A high-performance and energy-efficient FIR adaptive filter using approximate distributed arithmetic circuits,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, Jan 2019, vol. 66, no. 1, pp. 313–326.

- [144] B. Farhang-Boroujeny, *Adaptive filters: theory and applications*. John Wiley & Sons, 2013.
- [145] “MNIST for ML beginners,” [https://www.tensorflow.org/get\\_started/mnist/beginners](https://www.tensorflow.org/get_started/mnist/beginners), 2017, accessed: 2017-10-25.
- [146] Z. Yuan, J. Li, Z. Li, C. Ding, A. Ren, B. Yuan, Q. Qiu, J. Draper, and Y. Wang, “Softmax regression design for stochastic computing based deep convolutional neural networks,” in *Proceedings of ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2017, pp. 467–470.
- [147] R. Nishihara, P. Moritz, S. Wang, A. Tumanov, W. Paul, J. Schleier-Smith, R. Liaw, M. Niknami, M. I. Jordan, and I. Stoica, “Real-time machine learning: The missing pieces,” in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, ser. HotOS ’17. ACM, 2017, pp. 106–110.

# Appendix A

In this dissertation, all the circuits were implemented by VHSIC Hardware Description Language (VHDL) code, and the functions were validated by circuit simulation and MATLAB code. The functions of the circuits were simulated by using both MATLAB R2018b and ModelSim SE-64 6.6c software.

The VHDL code was synthesized by Synopsys Design Compiler (DC) version F-2011.09-SP4. A 28-nm low power bulk technology from STMicroelectronics with regular threshold was used for the synthesis to measure the dynamic power consumption, hardware cost and the critical path delay. The IC development kit and standard cell versions are identified as “C32\_SC\_12\_CORE\_LR\_C28@1.1@20111209.0”. The temperature was set at 25 °C and the supply voltage was 1.0 V. The “typical-typical” process corner was used for simulating the transistors. The wire cost was not considered during the simulation.

When performing the simulations, the power consumption was measured at a clock period of 4 ns. Since a default activity factor was used instead of computing the actual activity factor in a stochastic computing (SC) circuit, a lower estimate of power consumption for the SC circuits might be expected due to the frequent transitions between ‘0’ and ‘1’ in a stochastic sequence as opposed to conventional logic. A high-effort compiler was used for all the designs with the default optimization option for overall performance.

## Appendix B

The synthesis results of the 12-bit stochastic computing-based gradient descent circuits (SC-GDCs) using wrapping and saturating counters are shown in Table B.1. The area, power and critical path delay of the design using a saturating counter increase due to the extra logic gates to detect the overflows compared to those of the design using a wrapping counter.

Table B.1. Synthesis results of an SC-GDC training a neural network.

| SC-GDC using             | wrapping counters | saturating counters |
|--------------------------|-------------------|---------------------|
| Area ( $\mu\text{m}^2$ ) | 106.2             | 113.3               |
| Power ( $\mu\text{W}$ )  | 24.8              | 25.4                |
| Critical path delay (ns) | 1.1               | 1.3                 |

The synthesis results of the 16-bit fixed-point implementations using wrap-around and saturated values are shown in Table B.2. Although a slight increase in power consumption is expected for a similar reason as in an SC-GDC, a slight reduction is observed for the fixed-point designs using adders with saturated values for the accumulations of gradients. This could be due to a random anomaly in the synthesis tools.

Table B.2. Synthesis results of a fixed-point implementation training a neural network.

| Fixed-point implementation using | wrap-around values | saturated values |
|----------------------------------|--------------------|------------------|
| Area ( $\mu\text{m}^2$ )         | 1402.5             | 1424.4           |
| Power ( $\mu\text{W}$ )          | 240.0              | 236.0            |
| Critical path delay (ns)         | 3.9                | 3.9              |

All the above results are obtained by using the settings listed in Appendix A.

# Appendix C

## Journal Publications

1. **S. Liu**, H. Jiang, L. Liu, and J. Han, “Gradient descent using stochastic circuits for efficient training of learning machines,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Nov. 2018, vol. 37, no. 11, pp. 2530–2541.
2. **S. Liu** and J. Han, “Toward energy-efficient stochastic circuits using parallel Sobol sequences,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, July 2018, vol. 26, no. 7, pp. 1326–1339.
3. Y. Liu, **S. Liu**, Y. Wang, F. Lombardi, and J. Han, “A stochastic computational multi-layer perceptron with backward propagation,” *IEEE Transactions on Computers*, 2018, vol. 67, no. 9, pp. 1273–1286.

## Conference Publications

1. **S. Liu** and J. Han, “Energy efficient stochastic computing with Sobol sequences,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, Lausanne, Switzerland, pp. 650–653.
2. **S. Liu** and J. Han, “Hardware ODE solvers using stochastic circuits,” in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017, Austin, TX, USA, 6 pages.