Design and Evaluation of Stochastic Circuits for FIR Filters and Vector Quantizers

by

Ran Wang

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

Master of Science

in

Microsystems and Nanodevices

Department of Electrical and Computer Engineering
University of Alberta

Abstract

The compact arithmetic units in stochastic computing can potentially lower the implementation cost with respect to silicon area and power consumption. In addition, stochastic computing provides inherent tolerance of transient errors at the cost of a less efficient signal encoding. Much effort has been put into stochastic circuit designs for application-specific and general-purpose computation. However, the accuracy issue of these stochastic implementations still remains to be understood with respect to appropriate sequence lengths. As the first part of this thesis, the performance of stochastic arithmetic elements (such as stochastic adders, multipliers and absolute subtractors) and a stochastic sum-of-products (SOP) circuit are investigated and compared. Due to the long sequences used in the stochastic implementations, the stochastic circuits are not competitive in terms of throughput per area (TPA) or energy per operation (EPO) compared with conventional binary circuits. These stochastic arithmetic elements can be used as basic components in two useful applications: finite impulse response (FIR) filters and vector quantization (VQ) for image compression.

FIR filters are key elements in digital signal processor (DSP) due to their linear phase-frequency response. Two novel stochastic FIR filter designs are proposed based on regular or simplified multiplexers. The performance of stochastic and conventional binary FIR filters are implemented using an industrial 28-nm cell library. Silicon area, power and maximum clock frequency were obtained to allow an evaluation of TPA and EPO. Required stochastic sequence lengths are determined by matching the performance of the proposed stochastic FIR filters with that of the conventional binary FIR filter. Different signal resolutions were investigated with the goal of finding the break-even point between the stochastic and binary implementations by criteria such as the TPA and EPO. For equivalent filtering performance, the stochastic FIR filters underperform the conventional binary design in terms of TPA and EPO, although the stochastic design shows a better graceful degradation in performance with a significant reduction in energy consumption. A detailed analysis is performed to evaluate the accuracy of stochastic FIR filters and to determine the required stochastic sequence length. The fault-tolerance of the stochastic design is compared with that of the binary circuit using triple modular redundancy (TMR). The stochastic designs are more reliable than the conventional binary design and its TMR implementation with unreliable voters, but they are less reliable than the binary TMR implementation when the voters are fault-free.

Unlike the FIR filters for which accuracy is more critical, VQ is a general data compression technique that can tolerate some loss of information. The VQ technique has a simply scalable implementation complexity and potentially high compression rate. A stochastic implementation of VQ is proposed and evaluated against corresponding conventional binary designs. The effect of varying the stochastic sequence length is studied with respect to TPA and EPO. The stochastic implementations are shown to have higher EPOs than the conventional binary implementations due to their long latencies. For the stochastic implementations using errors measured by the L^1 norm, squared L^2 norm or 3^{rd} law. When a shorter encoding sequence with 512 bits is used in exchange for lower quality performance, the TPA is about 1.16 to 2.60 times that of the binary implementation with the same compression quality. Thus, the stochastic implementation outperforms the conventional binary design in terms of TPA for a reduced compression quality. By exploiting the progressive precision feature of a stochastic circuit, a readily scalable processing quality can be simply attained by halting the computation after different numbers of clock cycles.

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisors Dr. Bruce Cockburn and Dr. Jie Han for the continuous support of my study and research as a Master student. Their patience, motivation and passion guided me all the time in the past two years. Without their profound knowledge and insightful suggestions, this work would be extremely difficult to be finished.

My sincere thanks also go to Dr. Duncan Elliott for his knowledgeable lectures and remarkable research ideas. I would like to thank all the professors in my thesis committee for their consistent encouragement and constructive comments. Besides, I want to thank my fellow students in the VLSI lab, Jinghang Liang, Zhixi Yang, Peican Zhu, Cong Liu and Honglan Jiang, for sharing their research experience.

Last but not the least, I would like to thank my parents who have always been backing me up both spiritually and physically. Their unconditional love has been encouraging me throughout my life.

Contents

List of Figures

List of Tables

List of Acronyms

AAE average absolute error

APE average penalized error

ASIC application-specific integrated circuit

CMOS complementary metal oxide semiconductor

CWA conventional weighted average

DCT discrete cosine transform

DSP digital signal processor

EPO energy per operation

FIR finite impulse response

FSM finite state machine

HWA hard-wired weighted average

IIR infinite impulse response

LBG Linde, Buzo and Gray

LFSR linear-feedback shift register

MSB most significant bit

MUX multiplexer

MWA multiple-stage weighted average

PDF probability density function

PMM performance matching multiplier

PR passband ripples

RMSE root mean squared error

RTL	resistor-transistor logic
SA	stopband attenuations
SNG	stochastic number generator
SNR	signal-to-noise ratio
SOP	sum-of-products
TMR	triple modular redundancy
TPA	throughput per area
VHDL	very-high-speed-integrated-circuit hardware description language
VLSI	very large scale integration
VQ	vector quantization

List of Math Symbols

1. $P(S)$: probability encoded by a stochastic sequence S
2. N_b: the bit resolution for the binary filter
3. N_s: the number of bits in stochastic sequence S
4. N_f: the number of taps in the FIR filter
5. X: a binary number, signed or unsigned
6. $X[n]$: the binary value of a time-series variable X at time n
7. $S(X)$: a stochastic binary sequence encoding the probability X/L, where L is the sequence length
8. e_q, e_{qs}: quantization error in conventional and stochastic computing, respectively
9. $e_c^{(i)}$, $e_s^{(i)}$: injected error in conventional and stochastic computing, respectively
10. e_{fs}: random fluctuation error in stochastic computing
11. $\mathit{\Delta}$, $\mathit{\Delta}_s$: the minimum distance between two representable numbers in conventional and stochastic computing, respectively
12. σ_q, σ_{qs}: the standard deviation of the quantization error in conventional and stochastic computing, respectively

Chapter 1

Introduction

1.1 Background and Motivation

The continued scaling of complementary metal oxide semiconductor (CMOS) devices in the nanoscale regime makes reliability a major issue that can not be overlooked. Design variations, such as manufacturing process, voltage, temperature, can affect very large scale integration (VLSI) circuits. Because of these variations and random noises, the functionality of a circuit shows a statistical or probabilistic behavior. Conventional digital design methodologies, especially for arithmetic circuits, are usually based on deterministic device operations in a system using a radix 2 representation (i.e. binary numbers). To consider the statistical or probabilistic behavior of nanoscale devices, stochastic computing was proposed as an unconventional probabilistic computing paradigm [1], where digital circuits (logic gates) in stochastic computing can have alternative functions compared to their roles in conventional binary circuits.

The major objective in the VLSI design are constantly related to minimizing the silicon area, low power consumption and high speed. There have been much effort in industrial and academic researches investigating design methodologies to achieve these goals. For instance, one of the approaches to overcome the power minimization problem is to use supply voltage scaling [2]. However, this may cause issues such as a loss in signal to noise ratio. Approximate computing, which is useful for applications that can tolerate errors to some degree, is able to trade off some accuracy for a potential improvement in silicon area, power consumption and/or delay [3], [4] and [5]. All these approaches have delivered improvement in one or two aspects of this multiple dimensional optimization problem. Similarly, stochastic computing is of great interest as it shows direct advantage using compact arithmetic elements. Moreover, it is inherently fault-tolerant due to a unique encoding of numerical values, which can make computation more robust against soft errors.

Research on stochastic computing has focused on two aspects: application-specific stochastic circuit designs and accuracy issues. Promising results showed that stochastic computing can be applied to many fields such as image processing [6] and the evaluation of conventional polynomials [7]. On the other hand, researchers have revealed potential problems of stochastic computing in terms of the relatiionship between accuracy and efficiency [8,9]. However, it is still unclear when and where the stochastic computing can be a better choice than the conventional binary approach. To answer these questions, a detailed study on the required sequence length in stochastic computing is performed. Preliminary studies on basic stochastic arithmetic circuits are performed to facilitate the building of stochastic-based systems for large-scale and practical applications. This thesis reports a detailed comparative study that focuses on the implementations and evaluations of FIR filters and VQ encoders.

1.2 Recent Work on Stochastic Computing

In the book chapter by Gaines [1], several useful arithmetic operations were discussed for both bipolar and unipolar stochastic representations in the implementations of adders, multipliers, dividers and integrators. In [6], Li et al. built sequential stochastic computational elements using finite state machines (FSM). Various arithmetic functions were built using state transition diagrams such as a stochastic exponentiation function, a stochastic *tanh* function, and a stochastic linear gain function [10]. These functions are more complex compared to those implemented with combinational logic gates, and FSM-based stochastic implementations were shown to be efficient for these complex functions. Using these functions, Li et al. implemented five different image processing kernels. Among them, Kernel Density Estimation-based image segmentation provided a smaller area-delay product compared with the binary implementation. Stochastic implementations have the advantage of being inherently fault tolerant and can provide progressive image quality without extra hardware cost. Alaghi et al. investigated an edge-detection algorithm for real-time image processing [11]. Correlated sequences were employed in their design while XOR gates were used for subtraction and the absolute value function. It was shown that the area-delay product of the stochastic edge detection circuit is only 8.7% of that of a conventional binary circuit. Qian and Riedel compared stochastic hardware implementations of arithmetic polynomials [7]. In [12], a synthesis method using two-input two-dimensional FSM was proposed and tested on a function for a low density parity check decoder. The area was shown to be smaller than the traditional one-dimensional FSM method at the cost of using more auxiliary circuits to generate two inputs. Chang and Parhi investigated novel designs for both FIR and infinite impulse response (IIR) filters based on stochastic logic [13]. Several low-pass and high-pass filters with different cut-off frequencies were considered. In those filters, the coefficients are encoded as stochastic selection signals of multiplexers and XOR gates are used to invert the inputs when the corresponding coefficients are negative numbers. Using the stochastic number generator (SNG) designs for both unipolar and bipolar encodings in Figures 2.7 and 2.8, novel stochastic FIR filters proposed in Chapter 3 directly works with both unsigned and signed stochastic inputs without additional operations. Combinational stochastic circuits are shown to be linked to their stochastic functions in [14], which provides the theoretical support of the idea that stochastic circuits can be built by combining basic stochastic arithmetic elements. By introducing sequential stochastic circuits, it was verified in [14] that rational functions in the form of $f(x) = \frac{P(x)}{Q(x)}$ can be implemented. To minimize the cost of auxiliary circuits such as SNGs, Ding et al. took advantage of the fact that stochastic sequences for multiplexer-based systems can tolerate correlations [15]. Two algorithms were proposed to generate stochastic numbers using low-cost circuits. The impact of correlations in stochastic computing was discussed in [14] and the stochastic computing correlation or SCC was proposed as a suitable correlation measure. The SCC played an important role in affecting the actual function that a digital logic system implements.

Random bit streams representing probabilities used in stochastic computing will inevitably introduce variance when conventional binary numbers are encoded and decoded. In addition to the design of stochastic arithmetic elements, another major issue in stochastic computing is stochastic number generation and representation. How can stochastic sequences be generated efficiently and accurately? How does stochastic computing perform in comparison with the conventional binary approach in terms of accuracy? Researchers have been investigating these questions from different points of view. Moons and Verhelst discussed how various noises in stochastic computing affect the accuracy and energy consumption [9]. The performance of a stochastic multiplier was compared to a binary multiplier using variation models and quantitative analysis. The RMSE was used as an important metric in the comparison of the stochastic and binary multipliers. A formula was derived to address the relation between the required energy and the RMSE. It was shown that stochastic computing is not as efficient as the binary approach in terms of energy consumption. Only binary resolutions below 3 or 4 bits could be favorable for the stochastic computing. In [8], a general methodology to evaluate the accuracy of stochastic computing systems with multiple stages was proposed. Since stochastic computing is based on probabilities with restrictions in sequence randomness and correlation, accuracy issues can arise when a stochastic computing system has too many stages or even feedback loops. It was demonstrated that the inherent noise in stochastic computing with multiple stages becomes much higher than the binary approach. The signal-to-noise ratio (SNR) becomes even lower for stochastic circuits as the number of stages increases. A multi-stage discrete cosine transform (DCT) block in a JPEG encoder is implemented in [16]. The DCT block was used to evaluate the accuracy of a multi-stage stochastic computing system. Although the accuracy degradation can be compensated by using longer sequences, it will inevitably affect the overall performance compared to the binary approach.

In some applications, stochastic computing has been shown to have advantages with respect to important measures of circuit performance [10]. These advantages include potentially simpler arithmetic hardware and inherent tolerance of transient signal errors. The benefits of stochastic circuits must, however, outweigh the longer latencies caused by the stochastic data encoding. Because the stochastic sequence length is closely related to accuracy, researchers focus on those scenarios where some accuracy can be safely sacrificed, such as multimedia information displayed to humans. For example, image processing algorithms using stochastic methods have been shown to match their binary counterparts in terms of cost and speed while providing a similar experience to humans [11]. Other applications such as the fault tree analysis were also investigated using the stochastic approach. A stochastic model for priority AND gates was proposed to efficiently implement the dynamic fault tree in [17]. Unlike the popular Bernoulli sequences in stochastic computing, non-Bernoulli sequences were realized by random permutations of fixed numbers of 1's and 0's. The stochastic approach using the non-Bernoulli sequences becomes more competitive compared to Monte Carlo simulation [18].

1.3 Major Contributions of this Thesis

In this thesis, a comparative study of stochastic arithmetic elements, novel stochastic FIR filter and VQ designs and implementations are presented. The accuracy and circuit performance are evaluated experimentally and theoretically. The major contributions of this thesis can be described as follows.

1. Although the functionality of the stochastic arithmetic elements has been justified, there lacks direct comparison with their conventional binary counterparts. Circuit performances of some primitive stochastic elements are reported. Both scenarios are considered where auxiliary circuits such as counters and SNG are included and excluded.

2. A theoretical analysis is performed to discuss the accuracy issues of the binary and stochastic approaches. The analytical results are then used to estimate the minimum stochastic sequence length that is required to ensure that the performance of the stochastic approach matches that of the conventional binary approach. This is important because the sequence length used in stochastic computing can affect computational efficiency significantly.

3. Novel architectures for stochastic FIR filters and stochastic VQ encoders are proposed. Traditional stochastic arithmetic elements are modified for specific applications such as FIR filters and VQ encoders. Various structures are considered for more efficient and yet reliable stochastic designs.

4. An increasing number of bit resolutions are investigated to determine the competitive range of the stochastic approach against the conventional binary approach. Different measurements related to accuracy are reported in the experiments and sequence length is therefore determined by comparing stochastic implementations with conventional

binary implementations. The experimental results can be explained by the theoretical analysis.

1.4 An Overview of this Thesis

This thesis is composed of five chapters. Followed by this introduction chapter, Chapter 2 first provides background on stochastic computing in general. Auxiliary circuits such as the SNG for both unipolar and bipolar encoding will then be introduced and the functionality of some stochastic arithmetic elements will be investigated. The performances of stochastic arithmetic elements (such as adders, multipliers and absolute subtractors) and SOP functions is evaluated as a preliminary study to facilitate application-oriented stochastic designs. In Chapters 3 and 4, stochastic and binary FIR filters and stochastic VQ encoders will be designed, built and simulated. Detailed analysis on accuracy of the stochastic implementations will be investigated and the required sequence lengths are determined. The circuit performance will be compared for various bit resolutions and stochastic sequence lengths. Features in stochastic computing such as the progressive quality improvement and fault tolerance will also be discussed. Finally, Chapter 5 concludes the dissertation with a discussion of possible directions for future work in this field.

Chapter 2

Introduction and Evaluations of Stochastic Arithmetic Elements

2.1 Stochastic Computing

2.1.1 A Stochastic Computing System

In stochastic computing systems (See Figure 2.1), binary numbers must be scaled to lie within a certain range and then converted to their corresponding stochastic representations in the first place (See the first two blocks in Figure 2.1). A usually simpler digital network then performs stochastic computing to implement the desired arithmetic operations. The final outputs of the digital network are then typically converted back to binary numbers as the final results (See the last two blocks in Figure 2.1). However, the conversion from binary numbers to stochastic sequences is not necessary when the inputs are sampled and digitized stochastically. Similarly, stochastic sequences could be used directly as the final results and a backward conversion to binary would not be needed in this case. Stochastic computing elements (Figures 2.2 and 2.3) can often be built using very small circuits with low power consumption compared to the binary circuits implementing similar functions.

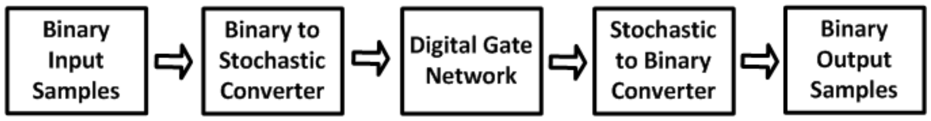


Figure 2.1: A basic stochastic computing system.

2.1.2 Unipolar and Bipolar Encodings

Stochastic computing involves processing numbers that are encoded as probabilities, which are represented using stochastic bit streams. Let N_1 denote the number of 1's in a stochastic bit stream with N_s bits. In a unipolar or unsigned representation, the bit stream represents the number N_1/N_s. The same bitstream represents $(2N_1 - N_s)/N_s$ in a bipolar (signed) representation [19] and [1]. For example, $''0001100101''$ denotes 2/5 in unipolar and -1/5 in bipolar because $N_s = 10$ and $N_1 = 4$. To convert a unipolar stochastic number P_u to a bipolar stochastic number P_b, Equation 2.1 can be used.

$$P_b = 2 \cdot P_u - 1. \tag{2.1}$$

Circuits for generating unipolar and bipolar stochastic sequence are described in Section 2.3.

2.2 Stochastic Computing Elements

In this section, the designs of common stochastic number generators are reviewed. Key stochastic computing elements are then discussed including adders, multipliers, absolute subtractors and a stochastic *tanh* function.

2.2.1 Combinational Stochastic Elements

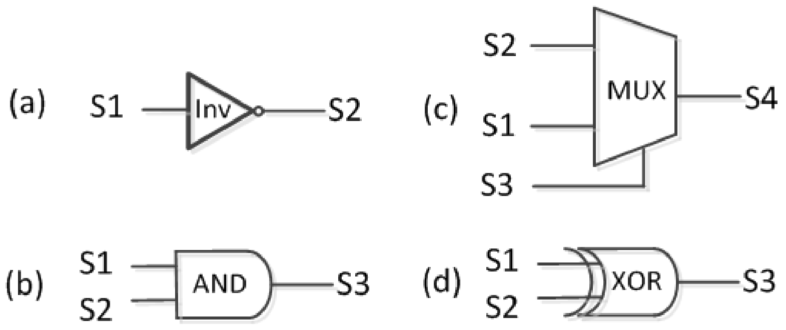


Figure 2.2: Stochastic arithmetic units in unipolar (unsigned) encoding: (a) $S2 = 1 - S1$; (b) $S3 = S1 \cdot S2$, ($S1$ and $S2$ are uncorrelated sequences); (c) $S4 = S3 \cdot S1 + (1 - S3) \cdot S2$ ($S3$ is a unipolar sequence); (d) $S3 = |S1 - S2|$ ($S1$ and $S2$ are correlated sequences).

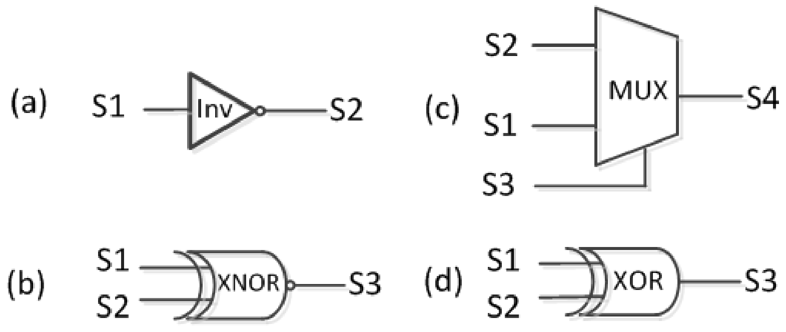


Figure 2.3: Stochastic arithmetic units in bipolar (signed) encoding: (a) $S2 = -S1$; (b) $S3 = S1 \cdot S2$, ($S1$ and $S2$ are uncorrelated sequences); (c) $S4 = S3 \cdot S1 + (1 - S3) \cdot S2$ ($S3$ is a unipolar sequence); (d) $S3 = -S1 \cdot S2$ ($S1$ and $S2$ are uncorrelated sequences).

Inverters

An inverter can have different functions in stochastic computing for the unipolar and bipolar encodings. As shown in Figures 2.2 and 2.3 [1], a sequence S1 using the unipolar encoding can be converted to represent $1-S1$ using a single inverter. For a bipolar-encoded sequence S1, however, the output of the inverter would become $-S1$. Therefore, an inverter can be used to obtain the additive inverse of a stochastic number using the bipolar encoding.

Signed and Unsigned Multiplier

For stochastic computing, multiplications are realized by AND gates or XNOR gates for the unipolar and bipolar encoding, respectively, implied by basic probability theory [1]. Figures 2.2 and 2.3 show how the unsigned and signed multiplication operations are implemented for uncorrelated stochastic bit streams. The XNOR-based multiplier realizes bipolar multiplications because 1 and 0 represent +1 and -1, respectively, in the bipolar representation. The four cases in bipolar multiplication are listed in Table 2.1.

Table 2.1: Stochastic and conventional 1-bit bipolar multiplication

Conventional Representation	**Stochastic Representation**
$(+1)\cdot(+1)=(+1)$	$'1'\,\mathrm{XNOR}\,'1'='1'$
$(+1)\cdot(-1)=(-1)$	$'1'\,\mathrm{XNOR}\,'0'='0'$
$(-1)\cdot(+1)=(-1)$	$'0'\,\mathrm{XNOR}\,'1'='0'$
$(-1)\cdot(-1)=(+1)$	$'0'\,\mathrm{XNOR}\,'0'='1'$

Weighted Adder

A stochastic adder can be implemented with a multiplexer in Figures 2.2 and 2.3 [1]. The unweighted multiplexer-based adder has a selecting signal that encodes 0.5. Such a signal is often a bit stream directly coming from the output of an LFSR. The selecting signal $S3$ produces the weight by encoding a probability. If the selecting signal is suitably chosen, a weighted adder can be implemented in Figures 2.2 and 2.3.

If data inputs $X1$ and $X2$ are both encoded as unipolar (or bipolar) stochastic sequences $S1$ and $S2$, then the output $S4$ of a two-input multiplexer will also be a unipolar (bipolar) stochastic sequence encoding Y. However, the multiplexer selecting signal $S3$ must be encoded as a unipolar sequence. Consider a stochastic implementation of the weighted sum

$$Y = A1 \cdot X1 + A2 \cdot X2, \tag{2.2}$$

where $A1$ and $A2$ are positive weights whose sum is one (i.e., $A1+A2=1$ and $A1, A2 \geq 0$). Data inputs $X1$ and $X2$ are encoded as stochastic sequences $S1$ and $S2$. For the multiplexer, the select input is controlled by unipolar sequence $S3$ that encodes $A1$, which means that the select input is 1 (switching $S1$ to $S4$) with probability $A1$, i.e.,

$$P(S3) = A1 = 1 - A2. \tag{2.3}$$

The multiplexer output $S4$ will be 1 with probability $P(S4)$, i.e.,

$$P(S4) = A1 \cdot P(S1) + A2 \cdot P(S2). \tag{2.4}$$

First consider the case of unipolar data inputs. By definition the numbers encoded by sequences $S1$, $S2$ and $S4$ are exactly $P(S1) = X1$, $P(S2) = X2$ and $P(S4) = Y$, respectively. Thus the computed output value Y will be the desired weighted sum $A1 \cdot X1 + A2 \cdot X2$.

Now consider the case of bipolar data inputs. By definition, the numbers encoded by $S1$, $S2$ and $S4$ are exactly $(X1+1)/2$, $(X2+1)/2$ and $(Y+1)/2$, respectively. Thus the output sequence $S4$ will be 1 with probability $P(S4)$, i.e.,

$$P(S4) = (A1 \cdot X1 + A2 \cdot X2 + A1 + A2)/2 = (A1 \cdot X1 + A2 \cdot X2 + 1)/2. \tag{2.5}$$

From the definition of the bipolar encoding, sequence $S4$ encodes the number

$$Y = 2 \cdot P(S4) - 1 = A1 \cdot X1 + A2 \cdot X2, \tag{2.6}$$

which is again the desired weighted sum. The same argument can be generalized to weighted sums containing $N \geq 2$ inputs implemented with N-input multiplexers, where the weights $A1, A2, ..., AN$ sum up to 1.

Absolute Value of Subtraction

Interestingly, an XOR gate can be used to realize the absolute value of a subtraction [11], provided that there is an appropriate correlation between the two parallel input sequences. This might happen when two sequences are generated using a shared LFSR and the same initial seed. Figures 2.2 and 2.3 show how the absolute-valued subtraction $S3 = |S1 - S2|$ is implemented for correlated stochastic bit streams. If S1 and S2 are two statistically independent sequences containing N_s bits, respectively, then S1 and S2 are related as follows:

$$\sum_{i=0}^{N_s-1} S1\,(i) \cdot S2\,(i) = \frac{1}{N_s} \sum_{i=0}^{N_s-1} S1\,(i) \cdot \sum_{i=0}^{N_s-1} S2\,(i)\,. \tag{2.7}$$

Usually, stochastic computing requires ideal independence of stochastic sequences. To use an XOR gate to implement the absolute value of a subtraction, one should use correlated sequences as inputs. Here correlation is guaranteed by maximizing overlaps of 1's in the sequence, which can be realized by sharing SNGs using identical initial seeds when the two inputs are generated. Ideally, two correlated N_s-bit stochastic sequences $S1$ and $S2$ satisfy

$$\sum_{i=0}^{N_s-1} |S1\,(i) - S2\,(i)\,| = |\sum_{i=0}^{N_s-1} S1\,(i) - \sum_{i=0}^{N_s-1} S2\,(i)\,|. \tag{2.8}$$

$S1(i)$ and $S2(i)$ being either 0 or 1, are the i^{th} bits in the stochastic sequences $S1$ and $S2$, respectively.

2.2.2 Sequential Stochastic Elements

Combinational stochastic elements can implement polynomial arithmetic using Bernstein polynomials [7]. Many other arithmetic functions can be implemented using stochastic sequential elements. For example, a stochastic exponential function, sequential implementations of a stochastic *tanh* function, and a stochastic linear gain function are discussed in [10]. For example, the state transition diagram of the stochastic *tanh* function is shown in Figure 2.4. This is essentially a saturating up-down counter with the sign bit as the output. The state machine starts at the central state and is always reset to that state before every new *tanh* calculation.

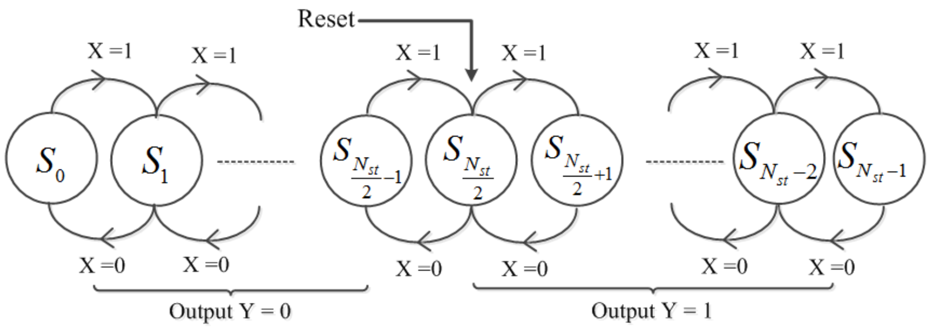


Figure 2.4: A state transition diagram of stochastic *tanh*(): N_{st} denotes the number of states in the FSM [5].

As comparators can be useful in many applications (e.g. the distance calculation and sorting operations required in VQ [20]), we chose to investigate the architecture of a stochastic comparator as a representative sequential stochastic elements.

Ideally the Heaviside step function [21] is required to implement a stochastic comparator. In stochastic computing, the Heaviside step function can be approximated using the stochastic *tanh* function and then realized using a FSM. Let X and Y be the stochastic input and output sequences, respectively. If P_X and P_Y represent the probability of seeing a $'1'$ in X and Y, then P_Y is defined for an approximation to the Heaviside step function as

$$\lim_{N_{st}\to\infty} P_Y = \begin{cases} 0, \text{ for } 0 \leq \mathrm{P_X} < 0.5; \\ 0.5, \text{ for } \mathrm{P_X} = 0.5; \\ 1, \text{ for } 0.5 < \mathrm{P_X} \leq 1. \end{cases} \tag{2.9}$$

where N_{st} is number of states in the FSM and N_{st} is large enough. Essentially, the *tanh* function behaves like a Heaviside function: if the input is below 0.5, the function output goes to 0 and if the input is above 0.5, the output goes to 1. Here $N_{st} = 32$ is used as in [6].

To investigate the influence of sequence length N_s on the switching function, the output of the stochastic *tanh* using a Matlab simulation is plotted in Figure 2.5. The simulations used stochastic sequence lengths $N_s = 256$, 1024 and 4096. 10000 random numbers between 0 and 1 are encoded as N_s-bit stochastic sequences and then used as the inputs of the stochastic *tanh* function. The corresponding outputs of the *tanh* function are then plotted in Figure 2.5. As shown in Figure 2.5, the resulting functions are only rough approximations to the ideal Heaviside function. However, the stochastic *tanh* function becomes an increasingly accurate approximation to the ideal switching function when N_s increases.

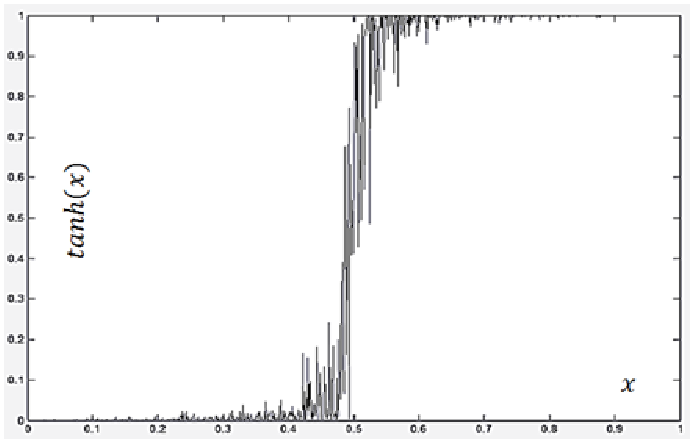
If the input is far from the threshold value 0.5, the behavior of the stochastic *tanh* function is accurately close to that of the Heaviside function. An experiment was set up to investigate how accurately the stochastic *tanh* function approximates the ideal Heaviside function. The goal of the experiment is to find the range of input values between 0 and 1 that map with high probability to the correct output as in the Heaviside function. In the experiment, the stochastic *tanh* function was built based on the state transition diagram in Figure 2.4. The input sequences have $N_s = 1000$ bits, which allows the stochastic encoding to represent the values between 0 and 1 with a minimum step size of 0.001. Thus there are 1000 different input values in total. For each of the 1000 input values, the input is converted to a 1000-bit stochastic sequence to obtain the output of the stochastic *tanh* function. The correct/expected result would be the output of the Heaviside function for the same input. The simulation was repeated 10000 times as the same input can be represented by different stochastic sequences. Then the number of correct/expected results is counted in the 10000 tests. If 99.5% of the 10000 tests produce the expected results, the output is said to have an error rate of 0.5%.

The results of the experiment are shown in Table 2.2. Note that the "Error Rate" refers to the probability of having an unexpected output corresponding to a certain input. The "Accurately Processed Inputs" define the input intervals which can generate correct/expected outputs within the Error Rate requirement. The "Inaccurately Processed Inputs" define the input intervals, which generate outputs with larger probability than the Error Rate to be incorrect. It can be concluded that input values that are far from 0.5 are more likely to produce expected/good results and the stochastic *tanh* function can be used to replace the ideal Heaviside function. As the input approaches 0.5 the output gets more variable and the error rate rises sharply.

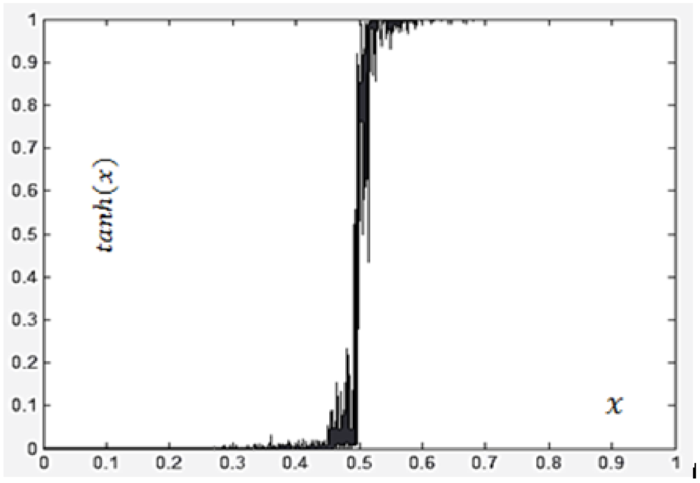
Table 2.2: The accepted input interval under different error rate requirements.

Error Rate	Accurately Processed Inputs	Inaccurately Processed Inputs
0.50%	[0, 0.489] [0.522, 1]	(0.489, 0.522)
0.20%	[0, 0.472] [0.535, 1]	(0.472, 0.535)
0.10%	[0, 0.463] [0.551, 1]	(0.463, 0.551)
0.05%	[0, 0.455] [0.560, 1]	(0.455, 0.560)

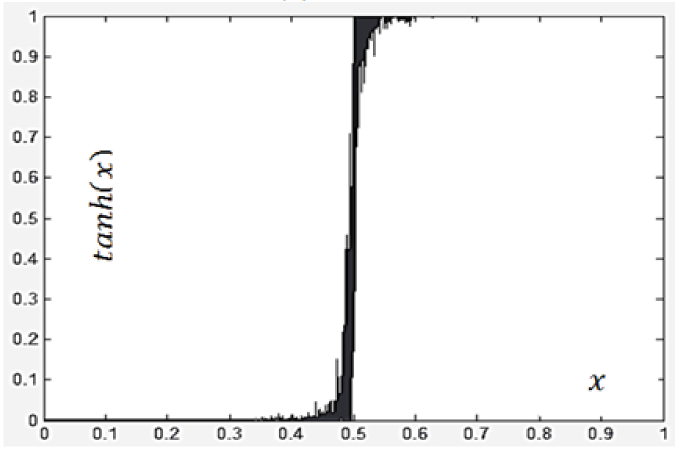
A stochastic comparator can be implemented using the stochastic *tanh* function [10]. In Figure 2.6, the architecture of a stochastic comparator is shown with two stochastic inputs



(a) N_s = 256



(b) N_s =1024



(c) N_s =4096

Figure 2.5: Stochastic *tanh* function for three sequence lengths.

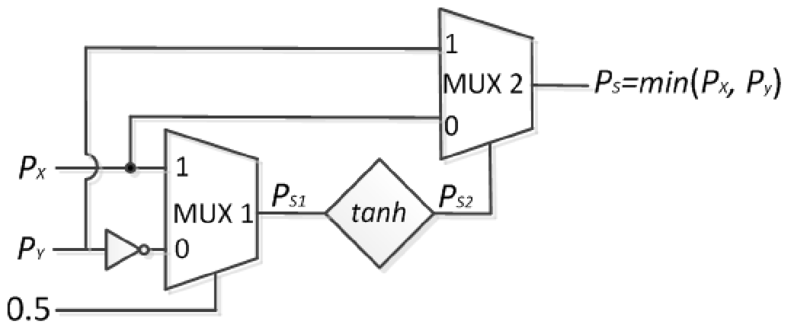


Figure 2.6: Stochastic comparator based on the stochastic *tanh* function [5].

P_X and P_Y, which may be encoded as both unipolar or bipolar sequences. The result of subtracting P_Y from P_X is computed by multiplexer (MUX) 1 and an inverter. The output P_{S1} of MUX 1 goes to the input of the *tanh* block implemented by the FSM shown in Figure 2.4. According to the function of the *tanh* block, the output P_{S2} approaches 1 if $P_X \geq P_Y$ and approaches 0 otherwise. Therefore P_{S2} can be used by MUX 2 to select the smaller of the two primary inputs, P_X and P_Y. The output P_s of the stochastic comparator is a stochastic approximation to $min(P_X, P_Y)$. P_S can be passed on as a data input to the next comparator if needed.

2.2.3 Unipolar and Bipolar Stochastic Number Generators

Stochastic number generators (SNGs) are typically based on pseudo-random bit generators such as LFSRs. For example, to generate the stochastic sequence for a 4-bit unsigned binary number, the SNG in Figure 2.7 is implemented with a 4-bit LFSR [19]. The SNG in Figure 2.7 converts a 4-bit unsigned binary number X to a stochastic number (sequence) of length 16. The all-zeros state must be inserted into the maximum-length (15-state) nonzero state sequence using extra logic shown in Figure 2.9. The SNG takes advantage of weight generation [22]. The bit streams named W3, W2, W1 and W0 represent the weights of 1/2, 1/4, 1/8 and 1/16, respectively. The binary number X is converted bit-by-bit with different weights assigned to them. Therefore, we have

$$\begin{aligned} P(S) =& 1/2 \cdot X[3] + 1/4 \cdot X[2] + 1/8 \cdot X[1] \\ &+ 1/16 \cdot X[0] = ((8 \cdot X[3] + 4 \cdot X[2] + 2 \cdot X[1] + 1 \cdot X[0]))/16 = X/16, \end{aligned} \tag{2.10}$$

where S is the output sequence of the SNG and $P(S)$ is the probability that S represents. Thus S is the stochastic representation of the binary number X.

Signed numbers are represented by bipolar stochastic representations [1]. An N_s-bit stochastic sequence with N_1 1's encodes the probability of $(2 \cdot N_1 - N_s)/N_s$. To design an SNG for signed numbers, consider the mappings of a signed binary number to its stochastic representation. For example, for a 4-bit signed binary number in two's complement, Table 2.3 shows its relationship with the probability that every single bit in the stochastic sequence

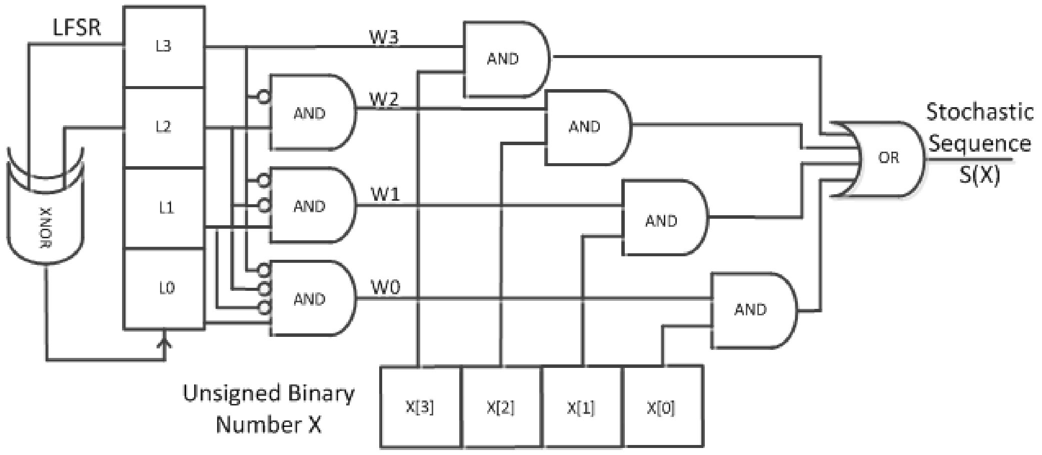


Figure 2.7: Unipolar stochastic number generator [17].

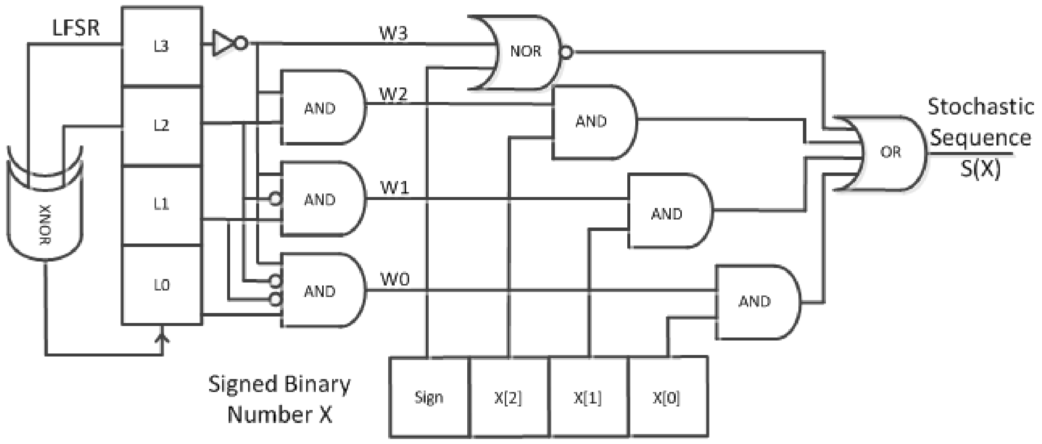


Figure 2.8: Bipolar stochastic number generator.

is $'1'$ and the probability that the number is encoded in the bipolar representation, assuming that the sequence length is 16 bits. This relationship reveals that the stochastic conversion of a signed binary number can be implemented by the SNG for unsigned numbers by simply inverting the sign bit and treating the remaining bits in the signed binary number as for an unsigned number. This SNG design is shown in Figure 2.8. To invert the signal of the sign bit in the 4-bit signed number, a NOR gate is used to replace the AND gate connected to the sign bit and some inverters are combined into one at the output of L3.

2.3 Error Analysis on Stochastic Arithmetic Elements

For conventional binary implementations, quantization errors are the major source of inaccuracy. A real number has to be rounded to the nearest binary number using a fixed number of available bits. In stochastic circuits, however, errors are also caused by the random fluctuation of the stochastic bits. This type of error or noise can propagate through the computation process and degrade the accuracy of the output signal.

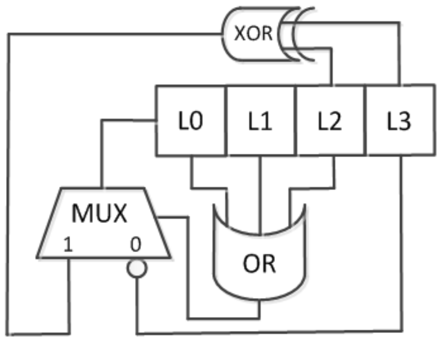


Figure 2.9: Add the all-zero state to a 4-bit LFSR.

Table 2.3: A mapping scheme of unsigned binary numbers and their corresponding stochastic representations.

Decimal	**Signed Binary Number in $2's$ complement**	**Probability of any bit being $'1'$ in the 16-bit sequence**	**Encoded signed probability assuming a bipolar representation**
7	0111	15/16	$(2 \times 15 - 16)/16 = 7/8$
6	0110	14/16	$(2 \times 14 - 16)/16 = 6/8$
5	0101	13/16	$(2 \times 13 - 16)/16 = 5/8$
4	0100	12/16	$(2 \times 12 - 16)/16 = 4/8$
3	0011	11/16	$(2 \times 11 - 16)/16 = 3/8$
2	0010	10/16	$(2 \times 10 - 16)/16 = 2/8$
1	0001	9/16	$(2 \times 9 - 16)/16 = 1/8$
0	0000	8/16	$(2 \times 8 - 16)/16 = 0/8$
-1	1111	7/16	$(2 \times 7 - 16)/16 = -1/8$
-2	1110	6/16	$(2 \times 6 - 16)/16 = -2/8$
-3	1101	5/16	$(2 \times 5 - 16)/16 = -3/8$
-4	1100	4/16	$(2 \times 4 - 16)/16 = -4/8$
-5	1011	3/16	$(2 \times 3 - 16)/16 = -5/8$
-6	1010	2/16	$(2 \times 2 - 16)/16 = -6/8$
-7	1001	1/16	$(2 \times 1 - 16)/16 = -7/8$
-8	1000	0/16	$(2 \times 0 - 16)/16 = -8/8$

In this section, basic error models are introduced. Arithmetic elements can then be analyzed using these error models. Conventional and stochastic multipliers are chosen as an example. The analysis on other arithmetic elements such as the adders, absolute subtractors and SOP is shown in Appendix C.

2.3.1 Error Analysis for Conventional Binary Implementations

When a real number is quantized to become an input to an arithmetic operation, both the rounding and floor functions can be applied. Because the floor function can introduce a biased error distribution into the digital system, only rounding is considered in our experiments. For any computational system with N_b-bit resolution, an N_b-bit normalized binary number x in [0, 1] has to be rounded off to become a normalized binary number x with N_b bits. By normalized we mean that the weights of bits (from the most significant bit to the least significant bit) in an N_b-bit binary number form a geometric series. The weight a_i of the i^{th} binary bit is either 0 or 1. The values of x and x' therefore lie between 0 and 1:

$$x = \sum_{i=1}^{N_b'} a_i \cdot 2^{-i} = \sum_{i=1}^{N_b} a_i \cdot 2^{-i} + \sum_{i=N_b+1}^{N_b'} a_i \cdot 2^{-i} \tag{2.11}$$

$$x' = \begin{cases} \sum\limits_{i=1}^{N_b} a_i \cdot 2^{-i}, a_{N_b+1} = 0, \\ 2^{-N_b} + \sum\limits_{i=1}^{N_b} a_i \cdot 2^{-i}, a_{N_b+1} = 1. \end{cases} \tag{2.12}$$

The quantization error e_q is calculated as

$$e_q = x - x' = \begin{cases} \sum\limits_{i=N_b+2}^{N_b'} a_i \cdot 2^{-i}, a_{N_b+1} = 0, \\ 2^{-N_b} + \sum\limits_{i=1}^{N_b} a_i \cdot 2^{-i}, a_{N_b+1} = 1. \end{cases} \tag{2.13}$$

Let $\mathit{\Delta}$ be the minimum distance between two numbers represented by N_b bits, i.e.,

$$\mathit{\Delta} = 2^{-N_b} \tag{2.14}$$

Then the quantization error e_q is bounded by

$$-\frac{\mathit{\Delta}}{2} \leq e_q < \frac{\mathit{\Delta}}{2} \tag{2.15}$$

The quantization error e_q is usually modeled as a white noise which is independent of the input signal. Therefore,a statistical model is established for the quantization error as illustrated in Figure 2.10. The quantization error e_q is an additive variable to the linear system. The value of e_q is considered to be evenly distributed. The probability density function (PDF) of the quantization error e_q is described as in [23].

$$f(e_q) = \begin{cases} \frac{1}{\mathit{\Delta}}, \ -\frac{a}{2} \leq e_q < \frac{\mathit{\Delta}}{2}; \\ 0, \ otherwise. \end{cases} \tag{2.16}$$

Based on the PDF, the mean and the variance of e_q are given by

$$E(e_q) = \int_{-\Delta/2}^{\Delta/2} e_q \cdot f(e_q)\, de_q = 0. \tag{2.17}$$

$$\begin{aligned}\sigma_q^2 =& E[(e_q - E(e_q))^2] = \int_{-\Delta/2}^{\Delta/2} (e_q - E(e_q)) \cdot f(e_q) de_q \\ =& \int_{-\Delta/2}^{\Delta/2} \frac{1}{a} \cdot {e_q}^2 de_q = \frac{\Delta^2}{12} = \frac{1}{12 \cdot 2^{2N_b}}.\end{aligned} \tag{2.18}$$

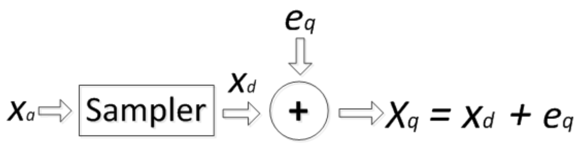


Figure 2.10: A statistical model for the quantization error e_q: X_a is the analog input and X_d is the digital sample from X_a; X_q is the quantized sample with quantization error e_q [19].

Inaccuracies in Conventional Binary Multiplications

A statistical model is developed to evaluate error accumulation and propagation in conventional binary multiplications. Let the multiplier and the multiplicand be represented by H and X as follows.

$$Y = H \cdot X, \tag{2.19}$$

where H and X are numbers with infinite precision. When quantized numbers are processed, the actual result Y' could be biased from the expected result Y, i.e.,

$$\begin{aligned}Y' &= (H + e_H)(X + e_X) \\ &= (HX + Xe_H + He_X + e_H e_X) \\ &= Y + (Xe_H + He_X + e_H e_X),\end{aligned} \tag{2.20}$$

where e_H and e_X are the quantization errors of the coefficient H and the input X, respectively. Both e_H and e_X can be modeled by e_q in Equation 2.13. To calculate the mean of the computational error, $E(Y')$, we must find

$$E(Y') = E(Y) + E(Xe_H + He_X + e_H e_X), \tag{2.21}$$

We first determine the mean of the cross product term $e_H e_X$. Due to [24], if e_H and e_X are independent real-valued continuous random variables with finite expected values, then

$$E(e_H e_X) = E\left(e_H\right) E(e_X) = 0. \tag{2.22}$$

Taking into consideration Equations 2.17, 2.21 and 2.22, we obtain

$$E(Y') - E(Y) = XE(e_H) + HE(e_X) + E(e_{Hi}e_{Xi}) = 0. \tag{2.23}$$

By definition and Equation 2.23, the variance of the output Y' is

$$Var(Y') = E[(Y' - E(Y'))^2] = E[(Y' - Y)^2]. \tag{2.24}$$

On the other hand, the variance of Y' is also given by

$$\begin{aligned} Var\left(Y'\right) &= \left(X^2 Var(e_H) + H^2 Var(e_X) + Var(e_H e_X)\right) \\ &= \left(X^2\sigma_q^2 + H^2\sigma_q^2 + Var(e_H e_X)\right). \end{aligned} \tag{2.25}$$

Because $Var(e_H e_X)$ is of a higher order than the other terms in Equation 2.25, we have

$$max\left\{Var(e_H e_X)\right\} = \frac{\Delta^4}{16} \ll \sigma_q^2\left(X^2 + H^2\right) = \frac{\Delta^2}{12}\left(X^2 + H^2\right). \tag{2.26}$$

Therefore, $Var(e_H e_X)$ in 2.25 can be ignored and by Equation 2.18 we obtain

$$Var(Y') \approx \sigma_q^2(X^2 + H^2) = \frac{\Delta^2}{12}(X^2 + H^2). \tag{2.27}$$

From Equation 2.27, it can be seen that the quantization error is amplified by a multiplicative factor. Thus, $e_{oc}^{(mul)}$ is defined to be the overall error for the conventional binary multiplier, i.e.,

$$e_{oc}^{(mul)} = |Y' - Y|. \tag{2.28}$$

By combining Equations 2.25, 2.27 and 2.28, we obtain the mean of the squared error as

$$E\left({e_{oc}^{(mul)}}^2\right) = E[\left(Y' - Y\right)^2] = Var\left(Y'\right) = \frac{\Delta^2}{12}\left(X^2 + H^2\right) = \frac{1}{12\cdot 2^{2N_b}}\left(X^2 + H^2\right). \tag{2.29}$$

By Equation 2.29, e_c can be estimated as

$$e_{oc}^{(mul)} = |Y' - Y| \approx \sqrt{\frac{\Delta^2}{12}\left(X^2 + H^2\right)} = \frac{\sqrt{3}}{3\cdot 2^{N_b+1}}\sqrt{(X^2 + H^2)}. \tag{2.30}$$

2.3.2 Error Analysis for Stochastic Implementations

Quantization Errors in Stochastic Computing

When a signal is stochastically encoded, the precision is limited by the sequence length N_s. For any N_s-bit sequence used in a stochastic system, the minimum distance between two representable numbers is

$$\Delta_s = 1/N_s. \tag{2.31}$$

Then the quantization error is bounded by

$$-\frac{\Delta_s}{2} < e_{qs} \leq \frac{\Delta_s}{2}. \tag{2.32}$$

Similar to the conventional binary quantization error given in Equation 2.13, the mean and variance of the stochastic quantization error are respectively

$$E(e_{qs}) = 0, \tag{2.33}$$

$$\sigma_{qs}{}^2 = \frac{\Delta_s{}^2}{12} = \frac{1}{12 \cdot N_s{}^2} \tag{2.34}$$

Random Fluctuations in Stochastic Computing

In addition to the quantization error caused by the limited sequence length, stochastic computing also suffers from a fluctuation error because the pseudo random number generator introduces uncertainty into the stochastic representations. The fluctuation error will be denoted by e_{fs}. P''_Y is defined as the final result obtained by stochastic computing [18] and [25] . Due to random fluctuations, P''_Y is different from P_Y. If P_Y is encoded by the stochastic bit stream $y(i)$, where $i = 1, 2, , N_s$ and N_s is the sequence length, then the final output P''_Y is obtained by

$$P_{Y''} = \frac{1}{N_s} \sum_{i=1}^{N_s} y\left(i\right), \tag{2.35}$$

If no stochastic fluctuations existed, there is no difference between P_Y and P''_Y. The stochastic bit stream $y(i)$ is usually a Bernoulli sequence [18]. The mean of the output P''_Y is

$$E(P''_Y) = P_Y. \tag{2.36}$$

The variance of $P_{Y''}$ is given by

$$Var\left[P_{Y''}\right] = Var\left[\frac{1}{N_s} \sum_{i=1}^{N_s} y\left(i\right)\right] = E\left[\left(P_{Y''} - E\left(P_{Y''}\right)\right)^2\right] = E\left[\left(P_{Y''} - P_Y\right)^2\right] \tag{2.37}$$

As $y(i)$ $(i = 1, 2, , N_s)$ is a Bernoulli sequence, we have

$$Var\left[\frac{1}{N_s} \sum_{i=1}^{N_s} y\left(i\right)\right] = \frac{P_Y(1 - P_Y)}{N_s}. \tag{2.38}$$

Let e_{fs} be the fluctuation error defined as

$$e_{fs} = |P''_Y - P_Y|. \tag{2.39}$$

The error is measured using the variance in Equations 2.37 and 2.38, i.e.,

$$E\left[e_{fs}{}^2\right] = E\left[\left(P_{Y''} - P_Y\right)^2\right] = \frac{P_Y(1 - P_Y)}{N_s}. \tag{2.40}$$

Therefore the fluctuation error can be approximated as

$$e_{fs} = \sqrt{\frac{P_Y(1 - P_Y)}{N_s}}. \tag{2.41}$$

Equation 2.41 implies that the fluctuation can be controlled by simply using longer sequences.

Inaccuracies in Stochastic Multipliers

The propagation effect of quantization errors in stochastic computing is similar to that in a conventional multiplier. The effect of quantization errors in stochastic computing can be analyzed by evaluating the output P'_Y. If P_Y is the correct output without errors, we have

$$P_Y = P_H \cdot P_X. \tag{2.42}$$

We further include the quantization effect by adding errors P_{e_H} and P_{e_X} to input P_X, the coefficient P_H and the output P_Y, respectively. Hence the output P'_Y can be calculated by

$$\begin{aligned} P_{Y'} =& (P_H + P_{e_H})(P_X + P_{e_X}) \\ &= (P_H P_X + P_X P_{e_H} + P_H P_{e_X} + P_{e_H} P_{e_X}) \\ &= P_Y + (P_X P_{e_H} + P_H P_{e_X} + P_{e_H} P_{e_X}). \end{aligned} \tag{2.43}$$

P_{e_H} and Pe_X are independent quantization errors that can be modeled by e_{qs} with the properties in Equations 2.32, 2.33 and 2.34. The mean of the stochastic output P'_Y is thus given by

$$E(P_{Y'}) = E(P_Y) + E(P_X P_{e_H} + P_H P_{e_X} + P_{e_H} P_{e_X}) = E(P_Y). \tag{2.44}$$

The variance of the output $P_{Y'}$ can be obtained from Equation 3.25 as

$$\begin{aligned} Var(P_{Y'}) &= \left(P_X^2 Var(P_{e_H}) + P_H^2 Var(P_{e_X}) + Var(P_{e_H} P_{e_X})\right) \\ &= \left(P_X^2 + P_H^2)\sigma_{qs}^2 + Var(P_{e_H} P_{e_X})\right). \end{aligned} \tag{2.45}$$

$Var(P_{e_{H_i}} P_{e_{X_i}})$ in Equation 2.45 can be ignored due to the fact that

$$\max\left\{Var\left(P_{e_{H_i}} P_{e_{X_i}}\right)\right\} = \frac{\mathit{\Delta}_s^4}{16} \ll \frac{\mathit{\Delta}_s^2}{12}\left(P_{Xi}^2 + P_{Hi}^2\right). \tag{2.46}$$

Therefore, Equation 2.45 becomes

$$Var(P_{Y'}) \approx \sigma_{qs}^2\left(P_X^2 + P_H^2\right) = \frac{\Delta_s^2}{12}\left(P_X^2 + P_H^2\right) \tag{2.47}$$

By definition and Equation 2.44, we have

$$Var(P_{Y'}) = E\left[(P_{Y'} - E(P_{Y'}))^2\right] = E\left[(P_{Y'} - P_Y)^2\right]. \tag{2.48}$$

By combining Equations 2.47 and 2.48, we get

$$E\left((P_{Y'} - P_Y)^2\right) = \frac{\Delta_s^2}{12}(P_X^2 + P_H^2). \tag{2.49}$$

The overall quantization error $e_{qs}^{(mul)}$ caused by the quantization effect can be defined as the absolute difference between the calculated result P'_Y and the accurate result P_Y, i.e.,

$$e_{qs}^{(mul)} = |P'_Y - P_Y|. \tag{2.50}$$

From Equations 2.49 and 2.50, the mean of the squared error ${e_{qs}^{(mul)}}^2$ can be evaluated as

$$E\left[{e_{qs}^{(mul)}}^2\right] = E\left[(P_{Y'} - P_Y)^2\right] = \frac{\Delta_s^2}{12}\left(P_X^2 + P_H^2\right). \tag{2.51}$$

Because of Equations 2.31 and 2.51, $e_{qs}^{(mul)}$ can be approximated as

$$e_{qs}^{(mul)} \approx \sqrt{\frac{\Delta_s^2}{12}\left(P_{X_i}^2 + P_{H_i}^2\right)} = \frac{\sqrt{3}}{6N_s}\sqrt{P_X^2 + P_H^2}. \tag{2.52}$$

The stochastic computation can suffer from both quantization errors and fluctuation errors. The overall error $e_{os}^{(mul)}$ in a stochastic filter is defined as the sum of the quantization error and the random fluctuation error, i.e.,

$$e_{os}^{(mul)} = e_{oq}^{(mul)} + e_{fs}^{(mul)}. \tag{2.53}$$

$e_{os}^{(mul)}$ defined in Equation 2.53 can be further written as the sum of the quantization error $e_{qs}^{(mul)}$ in Equation 2.52 and the fluctuation error $e_{fs}^{(mul)}$ in Equation 2.41:

$$e_{os}^{(mul)} = \sqrt{\frac{P_Y(1-P_Y)}{N_s}} + \frac{\sqrt{3}}{6N_s}\sqrt{P_X^2 + P_H^2}. \tag{2.54}$$

2.3.3 Required Sequence Length for Stochastic Computing Elements

The stochastic sequence length is an important parameter as it determines both the computational accuracy and the circuit performance. To determine the minimum required sequence length for stochastic computing elements, the overall error of the conventional binary implementation and the error of the stochastic implementation are compared. In this way, the stochastic sequence length N_s can be represented using functions of the bit resolution N_b.

For the stochastic multiplier, the overall error $e_{os}^{(mul)}$ in Equation 2.54 is used as a consecutive approximation of the stochastic error. To understand the relationship between bit resolution N_b and sequence length N_s, let $e_{os}^{(mul)} = e_{oc}^{(mul)}$, i.e.,

$$\sqrt{\frac{P_Y(1-P_Y)}{N_s}} + \frac{\sqrt{3}}{6N_s}\sqrt{P_X^2 + P_H^2} = \frac{\sqrt{3}}{3 \cdot 2^{N_b+1}}\sqrt{(X^2 + H^2)}. \tag{2.55}$$

Because the first term on the left side of the Equation 2.55 is dominant, the stochastic sequence length N_s is approximately

$$N_s \approx \frac{P_Y(1-P_Y)}{12(X^2 + H^2)} \cdot 2^{2N_b}. \tag{2.56}$$

Considering $\frac{P_Y(1-P_Y)}{12(X^2+H^2)}$ as a constant coefficient in Equation 2.56, we can use the big O notation to estimate the relationship between the necessary stochastic sequence length for a stochastic multiplier to match the performance of the conventional binary multiplier. Therefore, Equation 2.56 can be written as

$$N_s = O\left(2^{2N_b}\right). \tag{2.57}$$

Equation 2.57 shows that the sequence length N_s grows exponentially as the binary resolution N_b increases.

2.4 Performance Evaluation on Stochastic Arithmetic Elements

In this section, the area, power, delay, energy per operation and throughput per area are explored with regard to the stochastic circuits that were discussed in the previous section. These important metrics are then compared with conventional binary designs implemented using regular approaches. Various sequence lengths for stochastic computing are investigated to match the corresponding conventional binary designs.

2.4.1 Required Sequence Length for Stochastic Computing Elements Using the Simulation Method

Due to the binomial distribution and the potential correlation between implemented stochastic sequences, stochastic computing is inherently inaccurate. Noise in stochastic sequences can be higher compared to binary representations where the inherent inaccuracy is caused by quantization errors. This noise can also be amplified after a stochastic computing element. To fairly compare the efficiency of the stochastic approach and the conventional binary approach, they must perform identically in terms of accuracy. The sequence length N_s is the most important metric in stochastic computing as it affects the trade-off between accuracy and efficiency.

In this section, conventional binary implementations of the basic arithmetic elements are discussed for resolutions ranging from 3 bits to 16 bits. The stochastic arithmetic elements are then implemented using different sizes of SNGs and hence different sequence lengths. Accuracy is measured by RMSE for the bit resolution N_b, which can be defined as

$$RMSE(N_b) = \frac{1}{2^{N_b}}\sqrt{\frac{1}{N_t}\sum_{i=1}^{N_t}(\hat{Y}_i - Y_i)^2}, \tag{2.58}$$

where N_b is the bit resolution and N_t is the number of trials. $\hat{Y}_i$ and Y_i are the computed result and the true result of the i^{th} trial, respectively. Those conventional and stochastic implementations with same or similar accuracy will be paired and their circuit performances compared below. For each and every arithmetic element, 100 random sets of inputs (i.e. $N_t = 100$) are observed and the RMSE for each bit resolution is calculated. For the conventional binary approach, this process is repeated for increasing bit resolutions from 3 bits to 16 bits ($N_b = 3, 4, ..., 16$). For the stochastic approach, this process is repeated for increasing sequence lengths N_s, where $N_s = 2^{N_{LFSR}}$ and $N_{LFSR} = 3, 4, ...$, until a matching accuracy is found for the target conventional binary implementation.

Table 2.4 shows the resulting matching sequence lengths for increasing bit resolutions for addition, multiplication and absolute subtraction. The sequence length reported in Table 2.4 will then be used in the evaluation of circuit performance.

Table 2.4: RMSE comparison of the conventional binary and stochastic implementations represented by $\mathrm{RMSE_B}$ and $\mathrm{RMSE_S}$, respectively, using various sequence lengths N_s and different bit resolutions N_b .

	Unsigned Multiplier (Unipolar Stochastic Encoding)			Signed Multiplier (Bipolar Stochastic Encoding)		
N_b (Bits)	$RMSE_B$ (%)	$RMSE_S$ (%)	N_s (Bits)	$RMSE_B$ (%)	$RMSE_S$ (%)	N_s (Bits)
3	6.167	6.656	64	6.348	6.477	64
4	3.496	3.342	256	3.402	3.290	256
5	1.533	1.585	512	1.596	1.608	512
6	8.956×10^{-1}	7.688×10^{-1}	1,024	9.315×10^{-1}	8.025×10^{-1}	1,024
7	4.088×10^{-1}	4.807×10^{-1}	2,048	3.910×10^{-1}	4.585×10^{-1}	2,048
8	2.358×10^{-1}	2.171×10^{-1}	4,096	2.256×10^{-1}	2.180×10^{-1}	4,096
9	1.171×10^{-1}	1.052×10^{-1}	8,192	1.136×10^{-1}	1.084×10^{-1}	8,192
10	5.813×10^{-2}	5.097×10^{-2}	32,768	5.731×10^{-2}	5.024×10^{-2}	32,768
11	2.528×10^{-2}	2.995×10^{-2}	262,144	2.483×10^{-2}	2.858×10^{-2}	262,144
12	1.258×10^{-2}	1.391×10^{-2}	524,288	1.318×10^{-2}	1.373×10^{-2}	524,288
13	6.246×10^{-3}	6.841×10^{-3}	2,097,152	6.421×10^{-3}	6.532×10^{-3}	2,097,152
14	3.434×10^{-3}	3.485×10^{-3}	8,388,608	3.540×10^{-3}	3.365×10^{-3}	8,388,608
15	1.653×10^{-3}	1.667×10^{-3}	33,554,432	1.658×10^{-3}	1.649×10^{-3}	33,554,432
16	9.321×10^{-4}	7.704×10^{-4}	134,217,728	9.019×10^{-4}	7.745×10^{-4}	134,217,728

2.4.2 Circuit Performance Comparison

A standard application-specific integrated circuit (ASIC) design flow was used to build both conventional binary circuits and stochastic circuits for adders, multipliers and absolute subtractors. For bit resolutions $N_b = 3, 4, ..., 16$, resistor-transistor logic (RTL)-level designs were established for conventional binary circuits. The generated netlists were synthesized using Synopsys Design Compiler (syn_vF-2011.09-SP4) with STMicroelectronics 28 nm technology to yield reports on area, power and delay. Those metrics are required to calculate EPO and TPA, which are useful general metrics for comparing the conventional binary implementations and the stochastic implementations. Note that there are limitations using Synopsys to estimate power as the activity factor is likely lower than the average activity factor produced by real input values in stochastic computing. Therefore, the power consumption of stochastic circuits may be underestimated.

After the original data are collected from reports using the Synopsys Design Compiler, measurements of throughput per area and energy per operation are calculated. Take the stochastic multiplier using 64-bit sequences as an example. Assume that the minimum clock period is 5 ns, that the area is 0.8 um^2 and the power is 40 mW including both static and dynamic power. The calculations proceed as follows.

1. Throughput: one symbol per $5\,ns = 0.2\,Gb/s$
2. Throughput per area is $\frac{0.2\,Gb/s}{0.945\,mm^2} = 0.705\,Gb/(s \cdot mm2)$
3. Time per operation is $16 \times 4 \times 6\,ns = 384\,ns$
4. Energy per operation is $42.718 \times 384\,ns = 16.4 \times 10^{-9}\,J$

Unsigned Multipliers (Unipolar Stochastic Encoding)

Stochastic multipliers are usually implemented using AND gates for the unipolar encoding or XNOR gates for the bipolar encoding. Unsigned multiplication is considered first, as shown in Figure 2.11. Auxiliary circuits such as SNGs and counters are first included and then excluded to compare with the corresponding conventional binary multiplication in terms of circuit performance. Tables 2.5, 2.6 and 2.7 report the area, power and minimum clock period comparisons for the conventional binary multiplier and the stochastic multiplier, respectively. Then Tables 2.9 and 2.8 show the EPO and TPA comparisons for the binary unsigned multiplier and the stochastic unipolar multiplier. The measurements include stochastic multipliers with or without auxiliary circuits such as SNGs and counters.

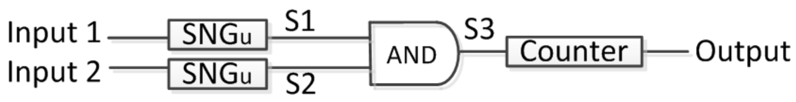


Figure 2.11: Schematic of the stochastic unipolar multiplier: $S3 = S1 \cdot S2$ ($S1$ and $S2$ are uncorrelated unipolar sequences).

The area comparison is shown in Table 2.5. The area of the stochastic unipolar multiplier with auxiliary circuits is much smaller than that of the conventional binary multipliers. However, this advantage in terms of area cost for the stochastic multiplier becomes less significant when the bit resolution increases. The stochastic multiplier without auxiliary circuits is much more efficient than the conventional binary multipliers in terms of area cost. Because the stochastic multiplier is implemented by an AND gate in the unipolar encoding (or an XNOR gate in the bipolar encoding), the area does not change as the bit resolution varies. Similar conclusions can be drawn for power comparisons in Table 2.5. Another important metric is the minimum clock period which the circuit can run at. As the circuits are fully pipelined, the longest stage delay in the pipeline corresponds to the minimum clock period. In Table 2.7, the stochastic multipliers can run at a very small clock period as expected. Although the advantage in terms of clock period for stochastic multiplier is indeed significant, the long sequence required for stochastic computing makes the stochastic multiplier less competitive.

Area, power and minimum clock period are three important measurements that reflect the performance of a circuit. However, circuit performance is a multi-dimensional problem where these three measurements must be integrated. To fairly compare the stochastic approach and the conventional binary approach, the EPO and TPA should both be used. Tables 2.9 and 2.8 list the EPOs and TPAs of the conventional binary multiplier and the stochastic multiplier. The stochastic multiplier with auxiliary circuits does not have any advantage over the conventional binary multiplier in terms of TPA and EPO. The EPO ratios of the stochastic multiplier without auxiliary circuits over the conventional binary multiplier are smaller than 1 for 4 bit resolutions and below. The stochastic multiplier without auxiliary circuits is only competitive in terms of TPAs for 6-bit precision. For higher resolutions, the stochastic multiplier underperforms the conventional binary multiplier as it suffers from long delays caused by required stochastic sequences (see the last columns in Tables 2.9 and 2.8). To sum up, the stochastic multiplier with auxiliary circuits is inefficient in terms of TPA and EPO. However, stochastic multiplication may become efficient for low bit resolutions in a large system where many multipliers are cascaded. In this way, the cost of auxiliary circuits such as SNGs and counters would become a less significant proportional overhead.

Table 2.5: Area report of binary unsigned multipliers (B), stochastic unipolar multipliers including auxiliary circuits (S1) and stochastic unipolar multipliers excluding auxiliary circuits (S2). N_b represents the bit resolution.

N_b (bits)	B (um^2)	S1 (um^2)	Ratio: S1/ B	S2 (um^2)	Ratio: S2/ B
3	5.946	0.741	0.125	0.047	7.904×10^{-3}
4	8.524	1.212	0.142	0.047	5.514×10^{-3}
5	11.480	1.821	0.159	0.047	4.094×10^{-3}
6	16.650	2.887	0.173	0.047	2.823×10^{-3}
7	25.375	4.852	0.191	0.047	1.852×10^{-3}
8	34.824	7.161	0.206	0.047	1.350×10^{-3}
9	44.028	9.733	0.221	0.047	1.068×10^{-3}
10	56.030	12.901	0.230	0.047	8.388×10^{-4}
11	69.113	17.245	0.250	0.047	6.800×10^{-4}
12	82.956	22.700	0.274	0.047	5.666×10^{-4}
13	95.914	26.921	0.281	0.047	4.900×10^{-4}
14	109.074	32.185	0.295	0.047	4.309×10^{-4}
15	121.530	39.316	0.324	0.047	3.867×10^{-4}
16	136.224	46.587	0.342	0.047	3.450×10^{-4}

Signed Multipliers (Bipolar Stochastic Encoding)

Similarly, signed multiplications can be analyzed by comparing the conventional binary implementation and the stochastic implementation (see Figure 2.12) using XNOR gates and the bipolar encoding. The unipolar encoding can be converted to bipolar encodings using a linear function in Equation 2.1 and vice versa. Hence, similar results can be obtained as the unsigned multiplication. Again, the area, power and minimum clock period are shown in Tables 2.10, 2.11 and 2.12 for the binary signed multiplier and the stochastic bipolar multiplier, respectively.

In Tables 2.14 and 2.13, it can be seen that the stochastic signed multipliers with aux-

Table 2.6: Power report of binary unsigned multipliers (B), stochastic unipolar multipliers including auxiliary circuits (S1) and stochastic unipolar multipliers excluding auxiliary circuits (S2) at minimum clock period. N_b represents the bit resolution.

N_b (bits)	B (mW)	S1 (mW)	S2 (mW)
3	0.476	0.378	4.280×10^{-3}
4	0.510	0.505	4.280×10^{-3}
5	0.542	0.630	4.280×10^{-3}
6	0.559	0.827	4.280×10^{-3}
7	0.567	0.850	4.280×10^{-3}
8	0.580	0.892	4.280×10^{-3}
9	0.585	1.070	4.280×10^{-3}
10	0.602	1.120	4.280×10^{-3}
11	0.627	1.190	4.280×10^{-3}
12	0.645	1.270	4.280×10^{-3}
13	0.662	1.420	4.280×10^{-3}
14	0.677	1.460	4.280×10^{-3}
15	0.715	1.590	4.280×10^{-3}
16	0.746	1.691	4.280×10^{-3}

Table 2.7: Minimum clock period report of binary unsigned multipliers (B), stochastic unipolar multipliers including auxiliary circuits (S1) and stochastic unipolar multipliers excluding auxiliary circuits (S2). N_b represents the bit resolution.

N_b (bits)	B (ps)	S1 (ps)	S2 (ps)
3	470	140	140
4	490	140	140
5	500	150	140
6	500	150	140
7	510	150	140
8	530	160	140
9	540	160	140
10	560	160	140
11	570	160	140
12	580	160	140
13	600	170	140
14	600	170	140
15	600	170	140
16	610	170	140

Table 2.8: EPO report of binary unsigned multipliers (B), stochastic unipolar multipliers including auxiliary circuits (S1) and stochastic unipolar multipliers excluding auxiliary circuits (S2). N_b represents the bit resolutions and N_s represents sequence length.

N_b (bits)	B ($nJ/Operation$)	S1 ($nJ/Operation$)	Ratio: S1/ B	S2 ($nJ/Operation$)	Ratio: S2/ B	N_s (bits)
3	223.9	$3.387\times10^{+3}$	15.1	$3.835\times10^{+1}$	0.2	64
4	249.9	$1.810\times10^{+4}$	72.4	$1.534\times10^{+2}$	0.6	256
5	270.9	$4.838\times10^{+4}$	178.6	$3.068\times10^{+2}$	1.1	512
6	279.3	$1.270\times10^{+5}$	454.8	$6.136\times10^{+2}$	2.2	1,024
7	289.2	$2.611\times10^{+5}$	903.0	$1.227\times10^{+3}$	4.2	2,048
8	307.2	$5.846\times10^{+5}$	1902.7	$2.454\times10^{+3}$	8.0	4,096
9	316.0	$1.402\times10^{+6}$	4438.1	$4.909\times10^{+3}$	15.5	8,192
10	336.9	$5.872\times10^{+6}$	17429.8	$1.963\times10^{+4}$	58.3	32,768
11	357.3	$4.991\times10^{+7}$	139702.1	$1.571\times10^{+5}$	439.7	262,144
12	374.1	$1.065\times10^{+8}$	284760.0	$3.142\times10^{+5}$	839.7	524,288
13	397.0	$5.063\times10^{+8}$	1275323.7	$1.257\times10^{+6}$	3165.6	2,097,152
14	406.1	$2.082\times10^{+9}$	5127197.9	$5.026\times10^{+6}$	12378.0	8,388,608
15	429.0	$9.070\times10^{+9}$	21140455.9	$2.011\times10^{+7}$	46864.1	33,554,432
16	454.9	$3.858\times10^{+10}$	84821985.4	$8.042\times10^{+7}$	176802.2	134,217,728

Table 2.9: TPA report of binary unsigned multipliers (B), stochastic unipolar multipliers including auxiliary circuits (S1) and stochastic unipolar multipliers excluding auxiliary circuits (S2). N_b represents the bit resolutions and N_s represents sequence length.

N_b (bits)	B ($(ns \cdot um^2)^{-1}$)	S1 ($(ns \cdot um^2)^{-1}$)	Ratio: S1/ B	S2 ($(ns \cdot um^2)^{-1}$)	Ratio: S2/ B	N_s (bits)
3	0.358	1.506×10^{-1}	4.209×10^{-1}	2.375	6.636	64
4	0.239	2.302×10^{-2}	9.615×10^{-2}	5.937×10^{-1}	2.480	256
5	0.174	7.150×10^{-3}	4.104×10^{-2}	2.968×10^{-1}	1.704	512
6	0.120	2.255×10^{-3}	1.877×10^{-2}	1.484×10^{-1}	1.236	1,024
7	0.077	6.709×10^{-4}	8.682×10^{-3}	7.421×10^{-2}	9.603×10^{-1}	2,048
8	0.054	2.131×10^{-4}	3.933×10^{-3}	3.710×10^{-2}	6.848×10^{-1}	4,096
9	0.042	7.839×10^{-5}	1.864×10^{-3}	1.855×10^{-2}	4.411×10^{-1}	8,192
10	0.032	1.478×10^{-5}	4.639×10^{-4}	4.638×10^{-3}	1.455×10^{-1}	32,768
11	0.025	1.383×10^{-6}	5.446×10^{-5}	5.797×10^{-4}	2.284×10^{-2}	262,144
12	0.021	5.252×10^{-7}	2.527×10^{-5}	2.899×10^{-4}	1.395×10^{-2}	524,288
13	0.017	1.042×10^{-7}	5.996×10^{-6}	7.247×10^{-5}	4.170×10^{-3}	2,097,152
14	0.015	2.179×10^{-8}	1.426×10^{-6}	1.812×10^{-5}	1.186×10^{-3}	8,388,608
15	0.014	4.459×10^{-9}	3.251×10^{-7}	4.529×10^{-6}	3.303×10^{-4}	33,554,432
16	0.012	9.408×10^{-10}	7.817×10^{-8}	1.132×10^{-6}	9.409×10^{-5}	134,217,728

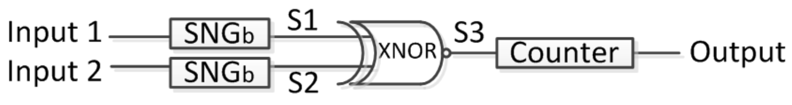


Figure 2.12: Schematic of the stochastic bipolar multiplier: $S3 = S1 \cdot S2$ ($S1$ and $S2$ are uncorrelated bipolar sequences).

iliary circuits underperform the conventional binary multiplier in terms of TPA and EPO. However, the stochastic signed multiplier without auxiliary circuits can be competitive in terms of EPO for 4 bit resolutions and below. In addition, the TPAs of stochastic bipolar multipliers without auxiliary circuits are larger than those of the binary signed multipliers for 6 bit resolutions and below.

Table 2.10: Area report of binary signed multipliers (B), stochastic bipolar multipliers including auxiliary circuits (S1) and stochastic bipolar multipliers excluding auxiliary circuits (S2). N_b represents bit resolutions.

N_b (bits)	B (um^2)	S1 (um^2)	Ratio: S1/ B	S2 (um^2)	Ratio: S2/ B
3	6.051	0.747	0.123	0.059	9.750×10^{-3}
4	8.577	1.217	0.142	0.059	6.879×10^{-3}
5	11.515	1.833	0.159	0.059	5.124×10^{-3}
6	16.939	2.933	0.173	0.059	3.483×10^{-3}
7	25.389	4.891	0.193	0.059	2.324×10^{-3}
8	34.835	7.162	0.206	0.059	1.694×10^{-3}
9	44.631	9.822	0.220	0.059	1.322×10^{-3}
10	56.813	13.128	0.231	0.059	1.038×10^{-3}
11	70.449	17.413	0.247	0.059	8.375×10^{-4}
12	83.395	22.757	0.273	0.059	7.075×10^{-4}
13	96.314	26.972	0.280	0.059	6.126×10^{-4}
14	109.230	32.488	0.297	0.059	5.401×10^{-4}
15	123.705	40.096	0.324	0.059	4.769×10^{-4}
16	138.529	47.411	0.342	0.059	4.259×10^{-4}

2.5 Discussions and Summary

There are strategies that one might consider to overcome the limitations of the stochastic method while preserving most of its strengths. For example, the low cost of stochastic multipliers allows us to extend the stochastic approach by encoding one binary number as two or more parallel stochastic bit streams. This would help speed up the throughput, but not the throughput per area. The energy per operation would not change. We cannot gain better performance in terms of TPA or EPO, but stochastic computing makes it easy for us to get a desired balance between throughput and hardware cost.

Hybrid binary-stochastic schemes can be another idea to improve the stochastic approach. Analogous to floating point numbers, a real number can be represented with an exponent in binary plus a mantissa in the stochastic representation. A hybrid multiplier is consisted of two components. A stochastic multiplier is used to calculate the product

Table 2.11: Power report of binary signed multipliers (B), stochastic bipolar multipliers including auxiliary circuits (S1) and stochastic bipolar multipliers excluding auxiliary circuits (S2) at minimum clock period. N_b represents the bit resolution.

N_b (bits)	B (mW)	S1 (mW)	S2 (mW)
3	0.477	0.382	4.410×10^{-3}
4	0.511	0.515	4.410×10^{-3}
5	0.543	0.634	4.410×10^{-3}
6	0.565	0.829	4.410×10^{-3}
7	0.570	0.862	4.410×10^{-3}
8	0.589	0.909	4.410×10^{-3}
9	0.586	1.079	4.410×10^{-3}
10	0.608	1.139	4.410×10^{-3}
11	0.629	1.204	4.410×10^{-3}
12	0.656	1.289	4.410×10^{-3}
13	0.670	1.432	4.410×10^{-3}
14	0.678	1.465	4.410×10^{-3}
15	0.725	1.612	4.410×10^{-3}
16	0.753	1.711	4.410×10^{-3}

Table 2.12: Minimum clock period report of binary signed multipliers (B), stochastic bipolar multipliers including auxiliary circuits (S1) and stochastic bipolar multipliers excluding auxiliary circuits (S2). N_b represents the bit resolution.

N_b (bits)	B (ps)	S1 (ps)	S2 (ps)
3	0.47	0.14	0.14
4	0.49	0.14	0.14
5	0.50	0.15	0.14
6	0.50	0.15	0.14
7	0.51	0.15	0.14
8	0.53	0.16	0.14
9	0.54	0.16	0.14
10	0.56	0.16	0.14
11	0.57	0.16	0.14
12	0.58	0.16	0.14
13	0.60	0.17	0.14
14	0.60	0.17	0.14
15	0.60	0.17	0.14
16	0.61	0.17	0.14

Table 2.13: EPO report of binary signed multipliers (B), stochastic bipolar multipliers including auxiliary circuits (S1) and stochastic bipolar multipliers excluding auxiliary circuits (S2). N_b represents the bit resolution and N_s represents the sequence length.

N_b (bits)	B ($nJ/Operation$)	S1 ($nJ/Operation$)	Ratio: S1/ B	S2 ($nJ/Operation$)	Ratio: S2/ B	N_s (bits)
3	224.1	$3.422\times10^{+3}$	15.268	$3.951\times10^{+1}$	0.18	64
4	250.6	$1.845\times10^{+4}$	73.625	$1.581\times10^{+2}$	0.63	256
5	271.6	$4.870\times10^{+4}$	179.289	$3.161\times10^{+2}$	1.16	512
6	282.7	$1.273\times10^{+5}$	450.247	$6.322\times10^{+2}$	2.24	1,024
7	290.9	$2.649\times10^{+5}$	910.579	$1.264\times10^{+3}$	4.35	2,048
8	312.4	$5.956\times10^{+5}$	1906.420	$2.529\times10^{+3}$	8.09	4,096
9	316.6	$1.414\times10^{+6}$	4465.183	$5.058\times10^{+3}$	16.0	8,192
10	340.5	$5.970\times10^{+6}$	17535.7	$2.023\times10^{+4}$	59.4	32,768
11	358.6	$5.050\times10^{+7}$	140855.9	$1.618\times10^{+5}$	451.4	262,144
12	380.4	$1.081\times10^{+8}$	284154.2	$3.237\times10^{+5}$	850.8	524,288
13	402.1	$5.106\times10^{+8}$	1269936.4	$1.295\times10^{+6}$	3220.2	2,097,152
14	407.0	$2.089\times10^{+9}$	5132614.5	$5.179\times10^{+6}$	12725.6	8,388,608
15	435.0	$9.194\times10^{+9}$	21135653.1	$2.072\times10^{+7}$	47623.4	33,554,432
16	459.3	$3.905\times10^{+10}$	85013787.9	$8.287\times10^{+7}$	180399.8	134,217,728

Table 2.14: TPA report of binary signed multipliers (B), stochastic bipolar multipliers including auxiliary circuits (S1) and stochastic bipolar multipliers excluding auxiliary circuits (S2). N_b represents the bit resolution and N_s represents the sequence length.

N_b (bits)	B ($(ns\cdot um^2)^{-1}$)	S1 ($(ns\cdot um^2)^{-1}$)	Ratio: S1/ B	S2 ($(ns\cdot um^2)^{-1}$)	Ratio: S2/ B	N_s (bits)
3	0.352	1.494×10^{-1}	4.248×10^{-1}	1.892	5.380	64
4	0.238	2.293×10^{-2}	9.637×10^{-2}	4.729×10^{-1}	1.987	256
5	0.174	7.102×10^{-3}	4.089×10^{-2}	2.365×10^{-1}	1.361	512
6	0.118	2.220×10^{-3}	1.880×10^{-2}	1.182×10^{-1}	1.001	1,024
7	0.077	6.655×10^{-4}	8.617×10^{-3}	5.911×10^{-2}	7.654×10^{-1}	2,048
8	0.054	2.130×10^{-4}	3.933×10^{-3}	2.956×10^{-2}	5.457×10^{-1}	4,096
9	0.041	7.768×10^{-5}	1.872×10^{-3}	1.478×10^{-2}	3.562×10^{-1}	8,192
10	0.031	1.453×10^{-5}	4.622×10^{-4}	3.695×10^{-3}	1.175×10^{-1}	32,768
11	0.025	1.369×10^{-6}	5.498×10^{-5}	4.618×10^{-4}	1.855×10^{-2}	262,144
12	0.021	5.238×10^{-7}	2.534×10^{-5}	2.309×10^{-4}	1.117×10^{-2}	524,288
13	0.017	1.040×10^{-7}	6.010×10^{-6}	5.773×10^{-5}	3.336×10^{-3}	2,097,152
14	0.015	2.158×10^{-8}	1.415×10^{-6}	1.443×10^{-5}	9.458×10^{-4}	8,388,608
15	0.013	4.372×10^{-9}	3.245×10^{-7}	3.608×10^{-6}	2.678×10^{-4}	33,554,432
16	0.012	9.244×10^{-10}	7.812×10^{-8}	9.020×10^{-7}	7.622×10^{-5}	134,217,728

of the two mantissas. A binary adder is used to calculate the sum of the two exponents. When one of the two operands is much larger or smaller than the other one, the hybrid multiplier can be potentially more efficient compared to the stochastic multiplier because it would require less stochastic bits to represent only the mantissas. However, the hybrid binary-stochastic scheme is only applicable to multiplications where the computation of exponents and mantissas can be completely separated. To perform addition, an adjustment must be made to equalize the exponents of operands. However, this can not be done as efficiently as in multiplication.

In this chapter, several combinational and sequential stochastic elements are introduced and investigated. Simulation-based experiments show the required sequence lengths for a corresponding series of bit resolutions. Stochastic multipliers are compared with their conventional binary counterparts with respect to circuit performance. Note that in the evaluation, auxiliary circuits are first included and then excluded. The results indicate that a stochastic system may be cost-efficient if the proportion of the auxiliary circuits can be reduced. This will be further investigated in Chapters 3 and 4 where real-world applications like FIR filters and VQ encoders are implemented and evaluated. Other stochastic elements such as adders and absolute subtractors are also investigated in Appendix C. A slightly more complicated SOP function is also implemented and compared to its conventional binary implementation using similar strategies in Appendix C.

Chapter 3

Stochastic Circuit Design and Evaluation of FIR Filters

FIR filters are key elements in DSP due to their linear phase-frequency response. In this chapter, the problem of implementing FIR filters are considered using the stochastic approach. Novel stochastic FIR filter designs based on multiplexers are proposed and compared to conventional binary designs implemented by synthesizing high-level very-high-speed-integrated-circuit hardware description language (VHDL) designs using the Synopsys Design Compiler with a 28-nm standard cell library. Silicon area, power and maximum clock frequency are obtained using estimated results from the Synopsys Design Compiler to evaluate the TPA and the EPO. For equivalent filtering performance, the stochastic FIR filters underperform in terms of TPA and EPO compared to the conventional binary design, although the stochastic design shows more graceful degradation in performance with a significant reduction in energy consumption. A detailed analysis is performed to evaluate the accuracy of stochastic FIR filters and to determine the required stochastic sequence length. The fault-tolerance of the stochastic design is compared with that of the binary circuit using TMR. The stochastic designs are more reliable than the conventional binary design and its TMR implementation with unreliable voters, but they are less reliable than the binary TMR implementation when the voters are fault-free.

In this chapter, three different stochastic FIR filter designs are investigated. The conventional weighted average (CWA) design exploits basic stochastic arithmetic elements such as the XNOR gate for multiplication and the multiplexer for addition. Stochastic FIR filters based on the HWA and MWA designs are proposed by using multiplexers as weighted adders. Different strategies are considered for generating the filter coefficients. In the HWA design, the filter coefficients or weights are given by repeating inputs to the multiplexer. Finally, the MWA design leverages the fact that every input signal is selected with a certain weight determined by the selection inputs to a multiplexer. The MWA design explicitly uses additional multiplexers to generate filter coefficients as the weights to a weighted adder. In all three designs, the weight of an input signal is the probability of selecting that input signal. That probability is then encoded in the frequency at which the corresponding combination of the selecting signals occurs in the bit streams. It is shown that both HWA- and MWA-based FIR filters have improved performance in terms of area, power and speed, compared to the CWA design.

Different resolutions are considered to determine the threshold (or break-even point) with regards to TPA and EPO that defines the competitive resolution range for stochastic circuits. A detailed analysis is performed to evaluate the accuracy of the binary and stochastic filters. The analytical results are then used to determine the minimum stochastic sequence length that is required to ensure that the performance of the stochastic filter matches that of a conventional filter. 3-bit to 16-bit FIR filters using both the stochastic and binary approaches are implemented initially. Then the minimum required stochastic sequence length that enables the stochastic circuit to filter signals as accurately as the binary one is determined empirically. The general metrics of throughput per area (TPA) and energy per operation (EPO) are used to characterize and compare the performance of the stochastic and conventional circuits. A detailed comparison is provided with respect to the fault tolerance of the HWA- and MWA-based stochastic filters. The conventional binary filter and its fault-tolerant TMR implementation are considered in the comparison.

3.1 Architectures of FIR Filters

FIR filters are frequently used in signal processing applications. Specifically the FIR filter implements a sum of products over a sliding window of the $N_t \geq 1$ most recent input samples, as specified in Equation 3.1. The fixed filter coefficients $H[i]$ give the finite impulse response of the filter.

$$Y\left[n\right] = \sum_{i=0}^{N_f - 1} H\left[i\right] X\left[n-i\right] \tag{3.1}$$

The hardware implementation of an FIR filter consists of adders and multipliers as well as delay units, which are typically implemented as D flip flops. In a typical pipelined FIR filter, the output is summed up over multiple clock cycles along the delay pipeline to produce the output $Y[n]$ (see Figure 3.1). A major challenge with stochastic designs is that they can be slow and expensive with many D flip-flops required to provide storage before every arithmetic block [13]. The stochastic sequence must be at least as long as N_b, where N_b is the binary bit resolution. Obtaining the delayed version of the input it takes at least $2N_b$ clock cycles to while only 1 clock cycle is necessary if the stochastic sequence can be transmitted in parallel.

To avoid the latency, the stochastic filter structure shown in Figure 3.2 is proposed. R1 to R4 are four registers for storing the stochastic sequences representing $X[n-3]$, $X[n-2]$, $X[n-1]$ and $X[n]$. The N_s-bit stochastic sequence $S(X[n])$ represents the N_b-bit binary number $X[n]$ where $X[n]$ has been normalized to lie between 0 and 1. R1 to R4 are N_s-bit registers. The blocks labeled SNG represents stochastic number generator, which convert binary inputs to stochastic output sequences. SNGs are frequently implemented using

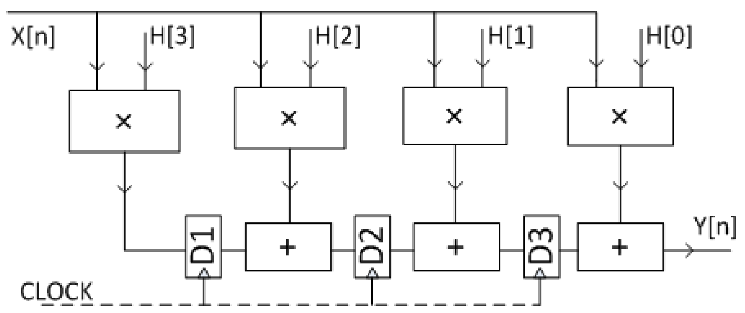


Figure 3.1: A 4-tap FIR filter design (i.e., $N_f = 4$ in 3.1)

LFSRs (Linear Feedback Shift Registers). The clock signal for the D flip-flops is called the sample clock of frequency f_D. The clock signal for the SNGs is called the stochastic clock at frequency f_S, where $f_S = N_s \cdot f_D$. At every stochastic clock cycle, one new bit of the stochastic sequence is generated according to the value of the input binary number $X[n]$. Each new bit from the SNG is saved in R1 as the next bit of the sequence $S(X[n])$. The sample clock is updated once for every N_s stochastic clock cycles when $S(X[n])$ is generated completely and stored in R1. Meanwhile, the other three registers R2, R3 and R4 are also updated by copying the stochastic sequences from the neighboring registers to their left. In this structure, the output sequence $S(Y[n])$ is obtained every N_s stochastic clock cycles.

Another solution to the relatively long latency and large storage cost is to move the input signal through the D flip-flops before they are converted into stochastic bit streams (see Figure 3.3). The $X[n-1]$, $X[n-2]$ and $X[n-3]$ signals after the multi-bit D flip-flops are the delayed versions of the binary input $X[n]$. At every stochastic clock cycle there will be one stochastic bit generated by the each of the SNGs which represent the four binary numbers $X[n]$, $X[n-1]$, $X[n-2]$ and $X[n-3]$. These stochastic bits and the stochastic coefficients are then combined serially to present the final output. After N_s stochastic clock cycles, or one sample clock cycle, an N_s-bit stochastic sequence $S(Y[n])$ is fully generated representing the outcome of the calculation. In the meantime, $X[n]$ is updated with a new binary input while $X[n-1]$, $X[n-2]$ and $X[n-3]$ are updated by copying the binary values from the neighboring D flip-flops to their left.

The two approaches described above both reduce the expense when generating the delayed versions of the input. The approach in Figure 3.2 requires one SNG module (based on an N_b-bit LFSR) and four N_s-bit registers while the one in Figure 3.3 requires four expensive SNG modules (based on four N_b-bit LFSRs) but only three N_b-bit registers. Roughly, there are $(4N_s+N_b)$ one-bit registers in the first approach and $7N_b$ one-bit registers in the second approach. Because usually the length of a stochastic bit stream $N_s \geq 2^{N_b}$, we have $(4N_s + N_b) \geq 4 \cdot 2^{N_b} + N_b$. When $N_b > 2$, we have $(4N_s + N_b) > 7N_b$. Based on this rough analysis, the second approach costs less than the first one. For example, a typical value for N_b is 16 and by using an oversampling factor of 4 we have $N_s = 4 \cdot 2^{16} = 262144$. For the first approach the cost is approximately $(4N_s + N_b) = 1048592$ while for the second

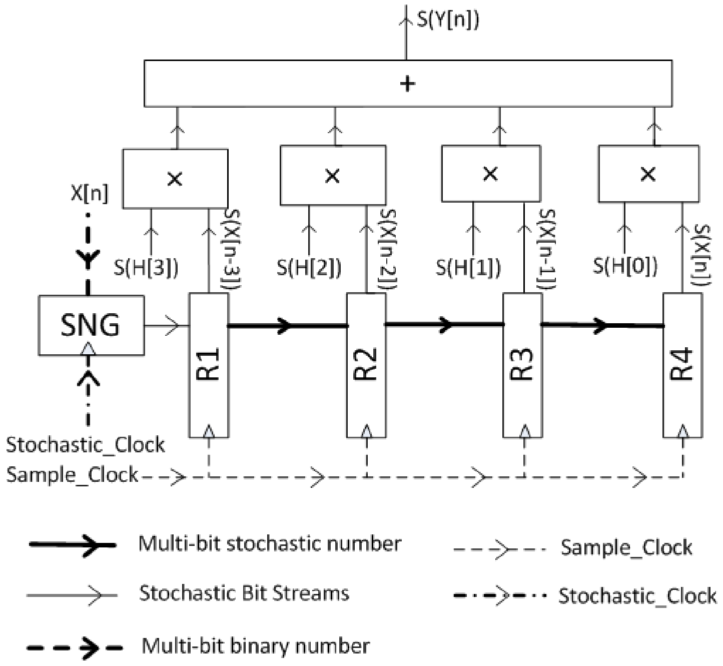


Figure 3.2: Using one-time conversion and stochastic storages: R1,R2,R3 and R4 are N_s-bit registers to store the stochastic sequence representing $X[n]$ which is a N_b-bit binary number

approach it is roughly $7N_b = 112$. Therefore, a significant lower cost can be achieved for the second approach. The approach in Figure 3.3 is thus selected for further investigation.

3.2 Stochastic FIR Filter Designs

3.2.1 Conventional Weighted Average (CWA) Design

The CWA design is built using conventional stochastic arithmetic elements as a basis for comparison with two novel stochastic designs introduced later. At the core of an N_f-tap FIR filter is an N_f-input weighted average function. For a 16-tap FIR filter, this function is implemented using stochastic logic, see Figure 3.4. The MUX is used as a simple adder. The XNOR gates implement bipolar multiplications provided that the two input sequences, i.e., the input sequence and the corresponding coefficient sequence, are statistically independent [19]. Two N_s-bit bipolar stochastic sequences S1 and S2 are said to be independent if

$$P(\overline{S1 \oplus S2}) = P(S1) \bullet P(S2), \tag{3.2}$$

where P(S1) and P(S2) denote the probabilities encoded by S1 and S2, respectively. $(\overline{S1 \oplus S2})$ denotes the sequence produced by an XNOR gate with S1 and S2 as input sequences. Note that the selecting signals are unipolar sequences encoding the probability of 0.5. In Figure 3.4, all the numbers to data inputs are converted by using bipolar SNGs (denoted by $\mathrm{SNG_b}$) and all the numbers to selecting inputs are converted using unipolar SNGs (denoted

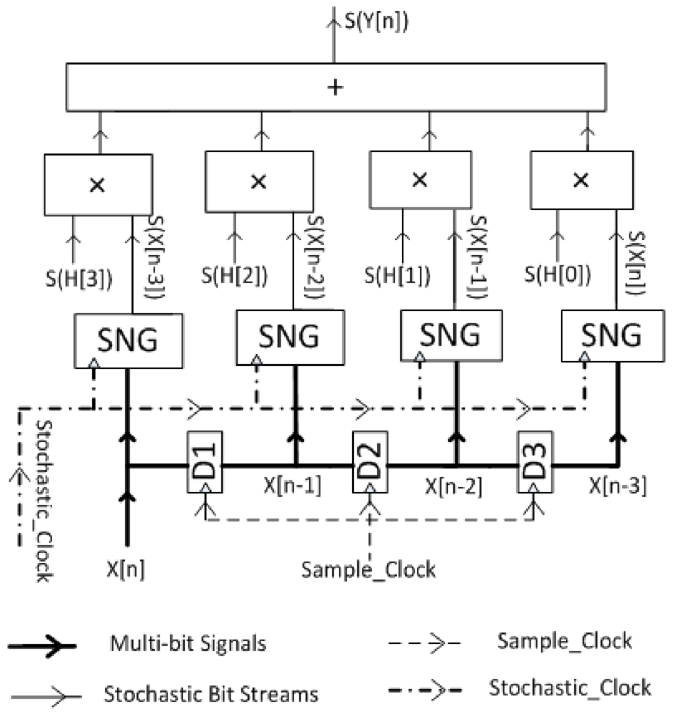


Figure 3.3: Using multiple conversions: D1,D2 and D3 are N_b-bit flip flops to get the delayed versions of $X[n]$, which is a N_b-bit binary number.

by $\mathrm{SNG_u}$).

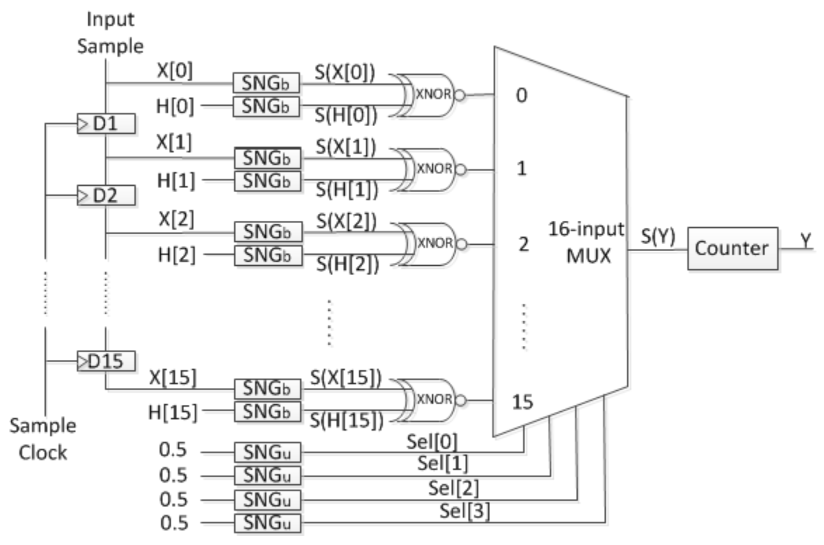


Figure 3.4: The 16-tap stochastic FIR filter using the conventional stochastic design.

3.2.2 Hard-wired Weighted Average (HWA) Design

In the CWA design, the SNGs cannot be shared, due to the requirement of signal independence. In the hard-wired weighted average (HWA) design, however, the absolute values of the coefficients can be implemented by assigning unbiased stochastic sequences to the selecting inputs of the multiplexer. In an unbiased stochastic sequence, the probabilities of each bit being '1' and '0' are the same, i.e., 0.5. The probability is then the same for selecting each of the inputs. However, a particular data input can be given more weight in the multiplexer output by connecting the input to multiple multiplexer inputs. Note that the signs of the coefficients can be implemented by XOR gates at the data inputs of the multiplexer. XOR gates help invert the corresponding input when the coefficient is negative. When the coefficients are positive, the XOR gates become buffers.

In Figure 3.5, for example, Wires 8 to 15 are associated with the same input $S(X[4])$, where $S(X[4])$ is the stochastic bit stream encoding the value of $X[4]$. Thus the probability of selecting the input $S(X[4])$ is 8/16 or 1/2, which means the coefficient of $X[4]$ is either 1/2 or -1/2. Similarly, all the other coefficients can be weighted by repeating inputs appropriately. The weighted average function in Equation 3.3 requires a multiplexer with four selecting inputs. It can be implemented by a 16-input multiplexer with combined data inputs as in Figure 3.5.

$$\begin{aligned} Y =& sign(A[0]) \cdot \frac{1}{16} \cdot X[0] + sign(A[1]) \cdot \frac{1}{16} \cdot X[1] \\ &+ sign(A[2]) \cdot \frac{1}{8} \cdot X[2] + sign(A[3]) \cdot \frac{1}{4} \cdot X[3] + sign(A[4]) \cdot \frac{1}{2} \cdot X[4]. \end{aligned} \tag{3.3}$$

The HWA design potentially improves the CWA design in that the SNGs for the weights can be removed. In general, to implement the N_f-tap FIR filter in Equation 3.1 using an N_b-bit resolution, the major steps are as follows:

1) Convert the floating point coefficients $H[i]$ in Equation 3.1 to fixed point N_b-bit binary numbers $A[i]$, where $i = 0, 1, ..., N_f - 1$.

2) Calculate the sum of all the absolute values of the coefficients A = $\sum_{i=0}^{N_f-1}$ |A [i]| where $A[i]$ is an N_b-bit binary number, for $i = 0, 1, ..., N_f - 1$.

3) The multiplexer has 2^{N_m} data inputs and N_m selecting inputs, where N_m is determined by $N_m = \lceil \log_2 A \rceil$ and $\lceil x \rceil$ is the ceiling function. Each selecting input is an unbiased stochastic sequence encoding the probability of 0.5.

4) The number of inputs to be combined is given by $|A[i]|$ for input X[i] ($i = 0, 1, ..., N_f - 1$). The sign of the coefficient $A[i]$ ($i = 0, 1, ..., N_f - 1$) is one of the inputs of the corresponding XOR gate.

5) Use a synthesis tool to optimize the design.

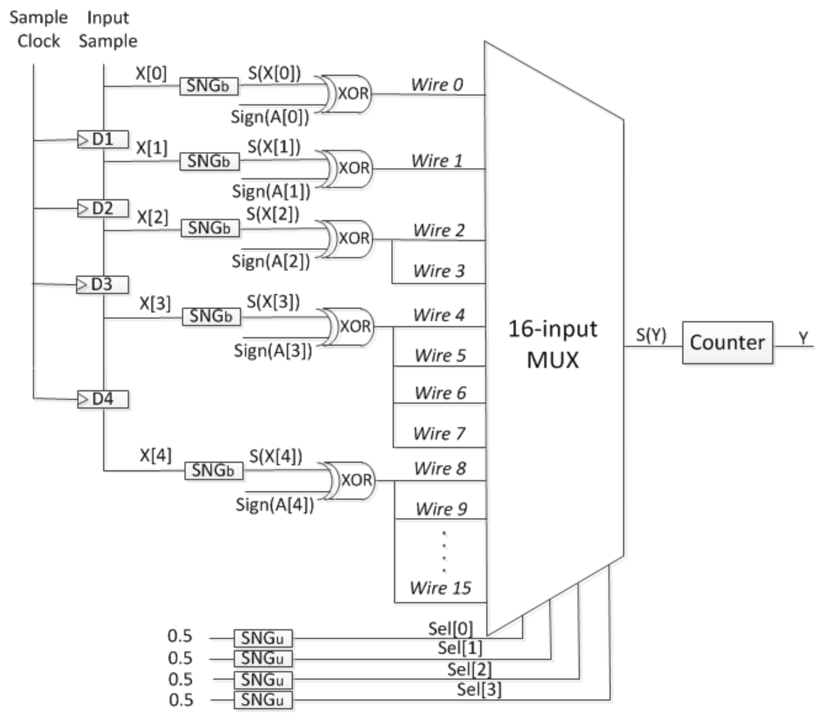


Figure 3.5: The hard-wired weighted average design of a 16-tap FIR filter.

3.2.3 Multiple-stage Weighted Average (MWA) Design

In general, the function of a weighted adder is given by Equation 3.4

$$\mathrm{Y} = \alpha \sum_{i=0}^{N_f-1} \mathrm{A}\left[\mathrm{i}\right] \mathrm{X}\left[\mathrm{i}\right], \tag{3.4}$$

where $\alpha = 1/\sum_{i=0}^{N_f-1} \mathrm{A}\left[\mathrm{i}\right]$. To implement stochastic weighted average using multiplexers, the weights are generated from the absolute values of the coefficients $A[i]$ $(i = 0, 1, , N_f)$. Therefore, (9) can be changed to Equation 3.5

$$\mathrm{Y} = \alpha \sum_{i=0}^{N_f-1} \left|\mathrm{A}\left[\mathrm{i}\right]\right| \cdot \mathrm{sign}(\mathrm{A}\left[\mathrm{i}\right]) \mathrm{X}\left[\mathrm{i}\right], \tag{3.5}$$

where $\alpha = 1/\sum_{i=0}^{N_f-1} \mathrm{A}\left[\mathrm{i}\right]$. To implement the weighted adder for an N_f-tap FIR filter, $\lceil \log_2 N_f \rceil$ selection signals are required. As an example, consider the design of a 4-tap FIR filter. In this case, $A[0]$, $A[1]$, $A[2]$ and $A[3]$ are the conventional binary coefficients determined by the filter specification. The sign of a coefficient is implemented using an XOR gate at the data input of the weighted adder. If the coefficient is positive, the XOR gate becomes transparent without changing the input value. If the coefficient is negative, the XOR gate acts like an inverter to invert the corresponding input value. The two selection signals are determined by the weights or filter coefficients, and they are used to select one of the four multiplexer inputs (see Figure 3.6). The probability that each combination appears (e.g. Sel[0] = 0, Sel[1] = 1) is the normalized coefficient for that input (e.g.$|A[1]| \cdot \alpha$). The stochastic representation of a number must lie within the interval [0, 1], so the coefficients are already normalized by the factor α in Equations 3.4 and 3.5. The relationship between the specified coefficients and the probabilities of selecting the corresponding inputs (see Figure 3.6) are given by

$$P\{Sel[0] = 0\} = (|A[0]| + |A[1]|) \cdot \alpha, \tag{3.6}$$

$$P\{Sel[0] = 1\} = 1 - (|A[0]| + |A[1]|) \cdot \alpha, \tag{3.7}$$

$$P\{Sel[1] = 0|Sel[0] = 0\} = |A[0]|/(|A[0]| + |A[1]|), \tag{3.8}$$

$$P\{Sel[1] = 1|Sel[0] = 0\} = |A[1]|/(|A[0]| + |A[1]|), \tag{3.9}$$

$$P\{Sel[1] = 0|Sel[0] = 1\} = |A[2]|/(|A[2]| + |A[3]|), \tag{3.10}$$

$$P\{Sel[1] = 1|Sel[0] = 1\} = |A[3]|/(|A[2]| + |A[3]|). \tag{3.11}$$

Here, $P\{Sel[X] = Y\}$ denotes the probability that the select signal Sel[X] ($X = 0$ or 1) is Y ($Y = 0$ or 1). $P\{Sel[X_1] = Y_1|Sel[X_2] = Y_2\}$ denotes the probability of the select signal bit Sel[X_1] ($X_1 = 0$ or 1) being Y_1 ($Y_1 = 0$ or 1) under the condition that the select signal bit Sel[X_2] ($X_2 = 0$ or 1) is Y_2 ($Y_2 = 0$ or 1). Stochastic sequences can then be generated to represent the corresponding select signals. Equations 3.6 3.11 can be implemented using the weight generator (WG) in Figure 3.6. Then the weighted addition can be realized using the weighted adder (WA) in Figure 3.6 with the selecting signals provided by instances of the WG in Figure 3.6.

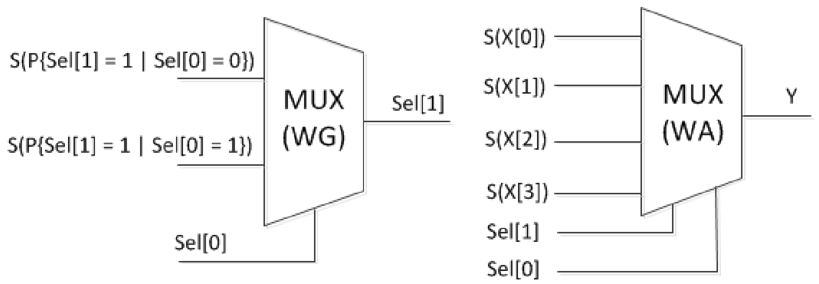
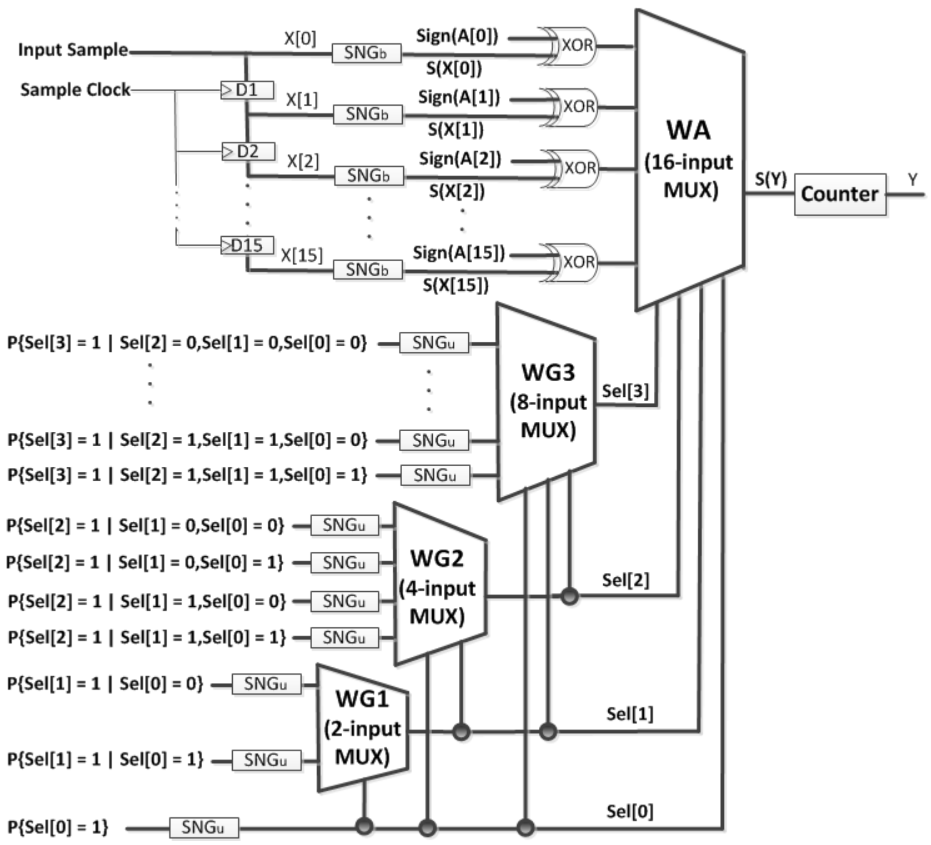


Figure 3.6: A 4-term sum of products implemented using (a) a stochastic Weight Generator (WG) and (b) a stochastic Weighted Adder (WA).

The overall schematic of a 16-tap FIR filter using the proposed weighted average structure is shown in Figure 3.7. Three multiplexers (WG1, WG2 and WG3) are needed as weight generators for four selecting signals. For the selecting signals, Sel[0] comes directly from a SNG that encodes its corresponding binary value. The value of Sel[0] is determined by coefficients $A[0]$ and $A[1]$. Sel[1] is the output of the 2-input MUX whose selecting signal is Sel[0]. Signals Sel[2] and Sel[3] are generated similarly. Therefore there are four different sizes of multiplexers in the core of the stochastic FIR filter design. The 16-input MUX (WA) implements the sum of products as a weighted average while the other three multiplexers implement weight generators. Note that the bipolar SNGs are used at the data inputs of the weighted adder, and unipolar SNGs are used elsewhere.


Input Sample
Sample Clock
X[0]
SNGb
Sign(A[0])
S(X[0])
XOR
D1
X[1]
Sign(A[1])
S(X[1])
D2
X[2]
Sign(A[2])
S(X[2])
D15
X[15]
Sign(A[15])
S(X[15])
WA (16-input MUX)
S(Y)
Counter
Y
P{Sel[3] = 1 | Sel[2] = 0,Sel[1] = 0,Sel[0] = 0}
P{Sel[3] = 1 | Sel[2] = 1,Sel[1] = 1,Sel[0] = 0}
P{Sel[3] = 1 | Sel[2] = 1,Sel[1] = 1,Sel[0] = 1}
SNGu
WG3 (8-input MUX)
Sel[3]
P{Sel[2] = 1 | Sel[1] = 0,Sel[0] = 0}
P{Sel[2] = 1 | Sel[1] = 0,Sel[0] = 1}
P{Sel[2] = 1 | Sel[1] = 1,Sel[0] = 0}
P{Sel[2] = 1 | Sel[1] = 1,Sel[0] = 1}
WG2 (4-input MUX)
Sel[2]
P{Sel[1] = 1 | Sel[0] = 0}
P{Sel[1] = 1 | Sel[0] = 1}
WG1 (2-input MUX)
Sel[1]
P{Sel[0] = 1}
Sel[0]
Figure 3.7: The 16-tap FIR filter implemented with the MWA design.

3.3 Performance Evaluation of the Conventional Binary and the Proposed Stochastic FIR Filters

A low-pass FIR filter is considered to help evaluate the proposed stochastic and binary filter designs. Detailed specifications of the low-pass filter are given in Table 3.1. The filter was designed using a Hamming window, with the embedded Matlab function $fir1()$. The number of taps is 267, i.e., $N_f = 267$. The coefficients of the filter are obtained to

Table 3.1: Low-pass FIR filter specifications

Specifications	Values
Cut-off frequency f_c	100 Hz
The width of the transition band BW	30 Hz
Minimum stop-band attenuation	-50 dB
Maximum peak-to-peak pass-band ripple	0.1 dB

meet the specifications in Table 3.1. The binary filter operates by converting the floating point numbers to fixed point numbers at different resolutions. The stochastic filters are built using the HWA design and the MWA design. The sequence length N_s is given by $N_s = 2^{N_{LFSR}}$, where N_{LFSR} is the number of bits in an LFSR. Various sequence lengths have been investigated for different resolutions to compare with the conventional binary design. In this experiment, N_{LFSR} is varied from 3 up to 30 (i.e, $N_{LFSR} = 3, 4, ..., 30$) to determine the sequence length. The resolution N_b ranges from 3 bits to 16 bits. The magnitude responses of the filters are then investigated. PR and SA are used to evaluate the performance of the binary design for various resolutions and the stochastic designs for different sequence lengths. Here PR and SA are defined as the maximum magnitude response of a frequency in the passband and stopband, respectively. In this experiment, the passband, transition band and stopband are given as, respectively. The results are shown in Table 3.

In Table 3.2, the PR and SA are shown for different bit resolutions and sequence lengths for the MWA-based stochastic filter and the HWA-based stochastic filter. The stochastic FIR filters suffer from both quantization errors and random fluctuations, but they show gradually-improving performance as the sequence length increases. For each bit resolution, the minimum sequence length N_s is found to match the performance of the stochastic FIR filters to that of the conventional N_b-bit binary FIR filter. The performance matching multiplier (PMM) is then calculated by $PMM = N_s/2^{N_b}$. For example, for the 8-bit binary FIR filter, the HWA-based stochastic FIR filter using sequences with at least 8192 bits has smaller or similar passband ripples and stopband attenuations. The performance matching multiplier is thus $PMM = 8192/2^8 = 32$. Therefore, the HWA-based stochastic implementation using 8192-bit sequences is considered as the best case to compare with the 8-bit binary conventional implementation. The same strategy is applied to determine the proper sequence lengths for various resolutions for the MWA-based stochastic filter.

The minimum resolution to achieve the filter specifications in Table 3.1 is 13 bits for the binary design. The attenuation in the stopband is -51.4476 dB and the passband ripple is 0.0392 dB. The magnitude responses of the stochastic and binary filters are plotted in Figures 3.8, 3.9 and 3.10 for 13-bit resolution. The HWA-based stochastic design with the minimum sequence length suffers from a maximum passband ripple of 0.2098 dB, as shown

Table 3.2: Performance of the FIR filters using the conventional binary approach, the stochastic MWA approach and the stochastic HWA approach. N_b: bit resolution, PR: passband ripple and SA: stopband attenuation.

	CB		**MWA**				**HWA**			
R (Bits)	PR (dB)	SA (dB)	Ns(bits)	PR (dB)	SA (dB)	PMM	Ns(bits)	PR (dB)	SA (dB)	PMM
3	3.0776	-3.9121	64	2.5327	-7.0345	8	64	2.5826	-7.1452	8
4	4.7882	-9.9327	256	1.3941	-11.3365	16	256	1.4147	-11.4942	16
5	2.8602	-13.1133	512	1.3016	-14.5083	16	512	1.3247	-14.7042	16
6	1.4623	-17.7985	1,024	0.7843	-17.6721	16	1,024	0.8017	-17.9031	16
7	0.7215	-24.0151	2,048	0.5391	-20.3922	16	2,048	0.5522	-20.653	16
8	0.3913	-27.634	8,192	0.2017	-27.2137	32	8,192	0.2098	-27.5353	32
9	0.1795	-31.1319	16,384	0.2835	-31.6085	32	16,384	0.2826	-31.9565	32
10	0.1026	-36.3678	65,536	0.0733	-35.6365	64	65,536	0.0715	-36.0254	64
11	0.0517	-45.582	524,288	0.0488	-44.1428	256	524,288	0.0507	-44.603	256
12	0.0324	-49.9427	1,048,576	0.0419	-47.8491	256	1,048,576	0.0373	-48.3225	256
13	0.0392	-51.4476	4,194,304	0.0342	-51.0634	512	4,194,304	0.0348	-51.5738	512
14	0.035	-54.5942	16,777,216	0.037	-54.971	1,024	16,777,216	0.0369	-55.5106	1,024
15	0.036	-55.9094	134,217,728	0.0327	-54.3603	4,096	134,217,728	0.0351	-54.8941	4,096
16	0.0364	-54.9557	536,870,912	0.0404	-54.7923	8,192	536,870,912	0.0364	-55.3241	8,192

in Figure 3.9. The stopband attenuation for the HWA-based FIR filter is only -30.5392 dB (Figure 3.9). To match the performance of the binary filter, the PMM has to be increased to 512 (Figure 3.10). The PR and SA are 0.0348 dB and -51.5738 dB, respectively, for the HWA-based stochastic filter in this case.Similarly conclusions can be drawn for the MWA-based FIR filter in Table 3.2.

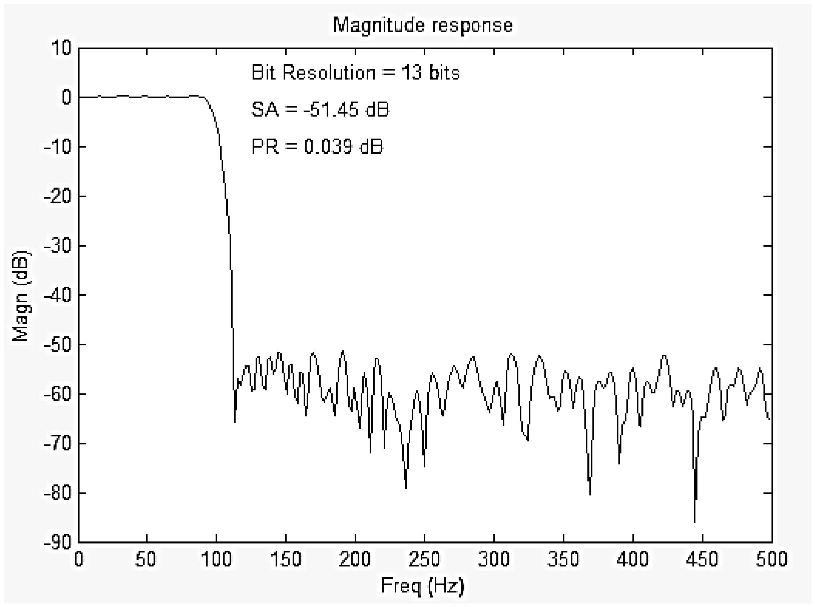


Figure 3.8: Magnitude responses of 13-bit Conventional binary FIR filter.

These results are consistent with the theory in [8] that a rather long stochastic bit stream is required to achieve the same RMSE. The necessary stochastic bit stream length is found to be $L = 2^{2N_b+1}$ (N_b is the binary bit resolution) due to the intrinsic fluctuation errors and

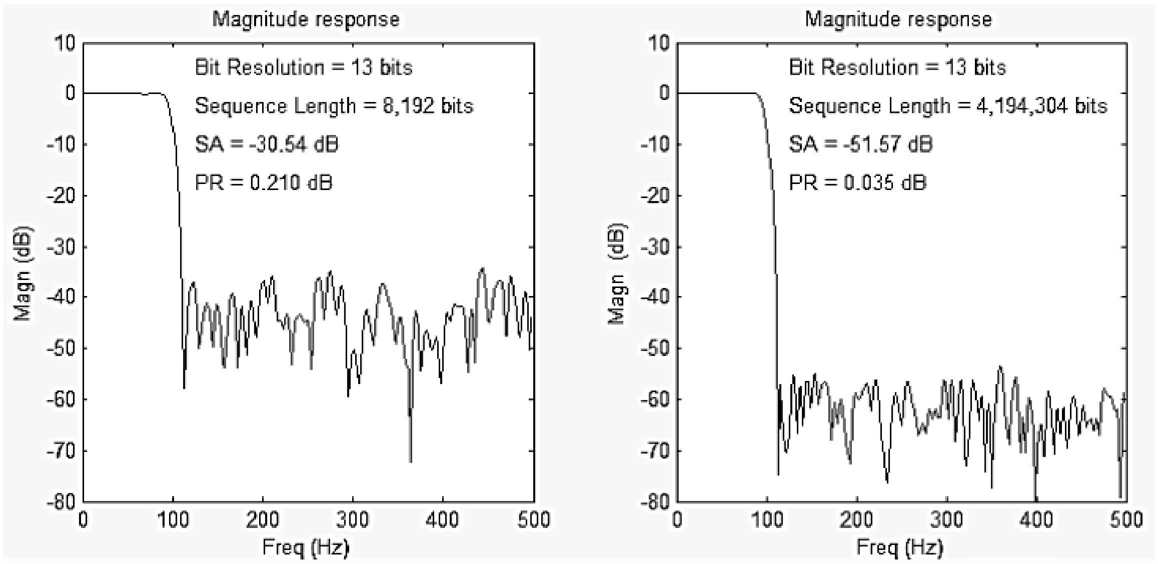


Figure 3.9: Magnitude responses of 13-bit stochastic HWA-based FIR filter without and with performance matching ($PMM = 512$).

the SNR decrease caused by multiple stages [8] and [9]. Such a long stochastic sequence means a relatively long computing time.

However, a shorter sequence length can be used for the HWA-based stochastic filter to obtain a degraded but possibly still acceptable performance. For example, if the PMM is 32 instead of 512, the SA becomes -42.3512 dB and the PR is 0.0593 dB (see Figure 3.11). However, the time and energy per computed output is reduced to only 1/16 of the previous result. Similarly, the MWA-based stochastic FIR filter also benefits from the property of graceful degradation in performance (see Figure 3.12). In contrast, an 11-bit conventional binary implementation of the filter shows similarly degraded performances with an SA of -45.5820 dB and a PR of 0.0517 dB (see Figure 3.13), however with very little saving in energy consumption. This is discussed in more detail next.

3.4 Simulation Results

For hardware performance comparison, the Synopsys Design Compiler [26] was used to synthesize a high-level design in VHDL into a standard cell ASIC design. Metrics such as silicon area, power consumption and delay were obtained.

The FIR filters specified in Table 3.1 were simulated for various resolutions from 3 bits to 16 bits. In Table 3.4, the circuit performance is compared with respect to silicon area, power consumption and delay. Although the core of the stochastic circuit is implemented using XNOR gates and multiplexers as adders and multipliers, the interfacing circuits require a relatively large number of SNGs and counters, especially for FIR filters with a large number of taps. The hardware cost of binary circuits grows faster than that of the stochastic circuits,

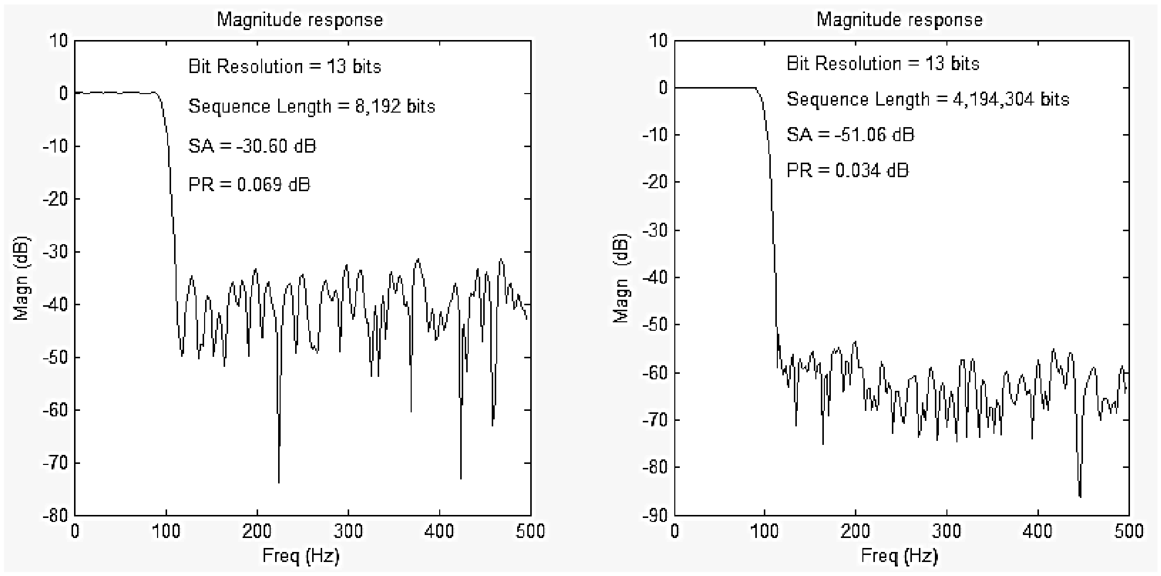


Figure 3.10: Magnitude responses of 13-bit stochastic MWA-based FIR filter without and with performance matching ($PMM = 512$).

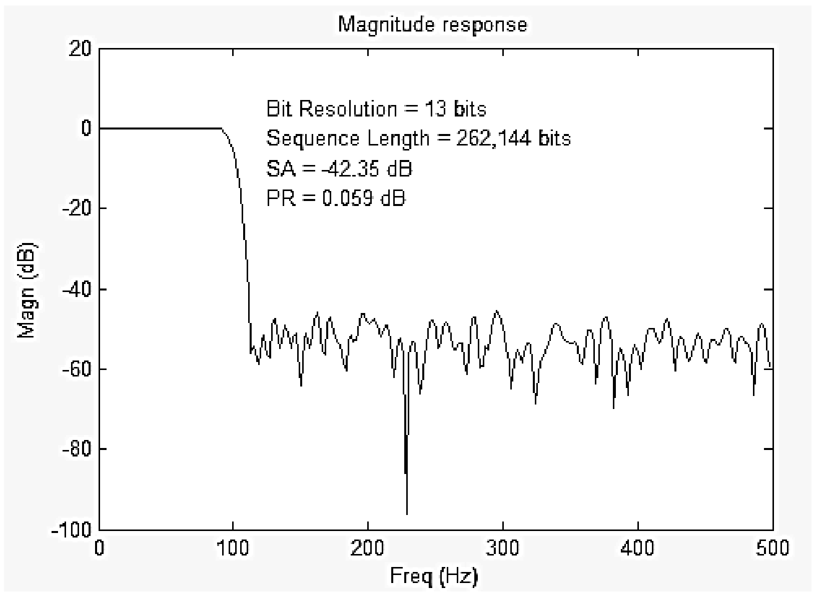


Figure 3.11: Magnitude responses of lower-quality 11-bit conventional binary FIR filters.

so for a larger resolution, the stochastic circuits become increasingly advantageous over a binary design. The auxiliary circuits such as the SNGs and counters, however, make this advantage of stochastic circuits less significant. Although stochastic logic gates such as multiplexers and XNOR gates are inexpensive, the auxiliary circuits used for conversions can be costly in terms of silicon area and power. In fact, in a 12-bit HWA-based stochastic FIR filter, it has been found that 80.3% of the silicon area comes from the stochastic number generators and counters.

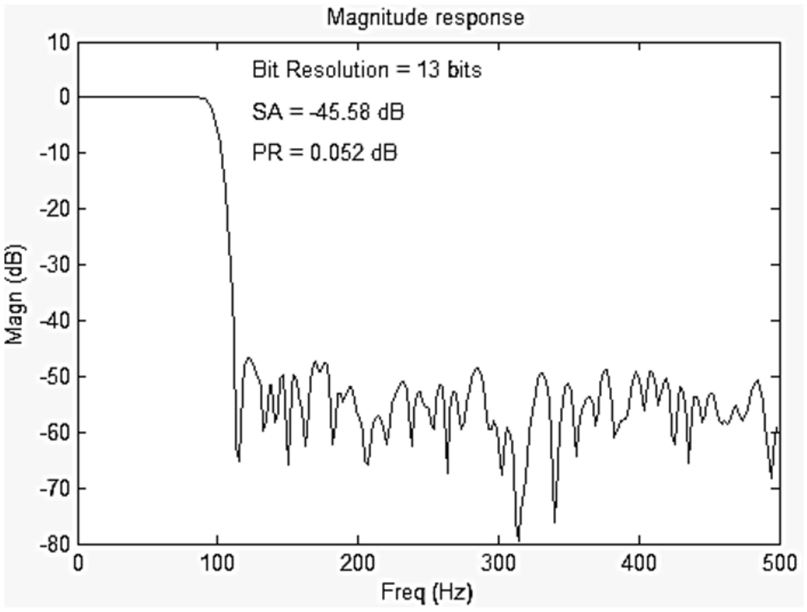


Figure 3.12: Magnitude responses of lower-quality 13-bit stochastic HWA-based FIR filters with $PMM = 32$.

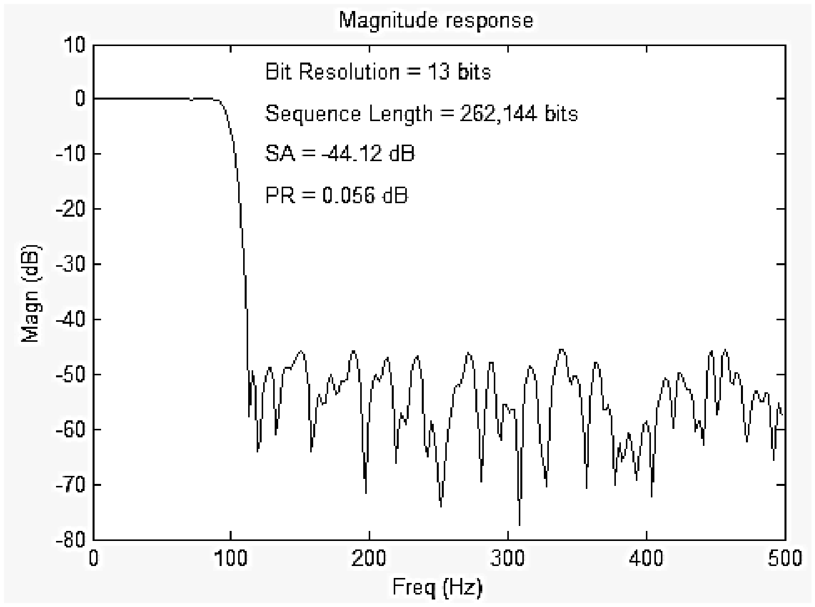


Figure 3.13: Magnitude responses of lower-quality 13-bit stochastic MWA-based FIR filters with $PMM = 32$.

Note that both the binary and stochastic circuits have been optimized for maximum throughput by adding pipeline registers as determined by the Synopsys synthesis tool. The area could be over-estimated for this reason. Long latency has been a major challenge for stochastic circuits. Adopting a faster clock is a potential way to reduce this latency. With the help of timing analysis, the clock can be pushed to the limit according to the slack time. The results are shown in Table 3.4. The required stochastic sequence length is given by $2^{N_b} \cdot PMM$, where N_b ($N_b = 3, 4, ..., 16$) is the binary resolution and PMM is the performance matching multiplier. The reported power consumptions are estimated at the fastest clocks for each of the resolutions.

There are techniques that discussed other possibilities of avoiding the auxiliary circuits such as SNGs. For example, the authors in [27] proposed to generate random bits from analog signals using sigma-delta modulation. We therefore decided to present the circuit performance when auxiliary circuits are excluded. Only the core of the stochastic circuits is considered without the auxiliary circuits such as SNGs and counters. Area, power and minimum clock period results are reported in Table 3.3. The core of the CWA-based and MWA-based circuits for the 267-tap FIR filter does not change for various bit resolutions. The HWA-based implementation becomes more complex as the bit resolution increases. This is because the HWA-based structure depends on the coefficients of the filter. In Table 3.3, it can be seen that the stochastic circuits use less hardware area and consume less power compared to the conventional binary implementation as expected. The advantage of stochastic circuits becomes more significant as the bit resolution increases.

Table 3.3: Comparison of hardware cost, power consumption and minimum clock period for conventional binary (CB), CWA-based, MWA-based and HWA-based stochastic FIR filters without any auxiliary circuits such as SNGs and counters.

N_b **(Bits)**	**Area (um^2)**				**Power (mW)**				**Min Clock Period (ps)**			
	CB	**CWA**	**MWA**	**HWA**	**CB**	**CWA**	**MWA**	**HWA**	**CB**	**CWA**	**MWA**	**HWA**
3	12757	8719	8747	2565	24.24	15.7	15.5	4	390	220	210	170
4	21262	8719	8747	3096	35.5	15.7	15.5	5.8	410	220	210	180
5	31893	8719	8747	3749	47.86	15.7	15.5	6.6	420	220	210	180
6	44650	8719	8747	4399	61.22	15.7	15.5	7.5	440	220	210	180
7	59533	8719	8747	4931	75.49	15.7	15.5	9.8	470	220	210	200
8	76543	8719	8747	5461	90.59	15.7	15.5	11.3	490	220	210	200
9	95679	8719	8747	5980	106.49	15.7	15.5	12.9	490	220	210	210
10	116943	8719	8747	6620	123.12	15.7	15.5	14.4	500	220	210	210
11	140330	8719	8747	7287	140.45	15.7	15.5	15.7	510	220	210	210
12	165843	8719	8747	8669	138.45	15.7	15.5	17	540	220	210	220
13	193482	8719	8747	8947	158.45	15.7	15.5	18.1	550	220	210	220
14	223269	8719	8747	9602	177.1	15.7	15.5	19.8	570	220	210	230
15	255141	8719	8747	10155	196.36	15.7	15.5	20.8	590	220	210	230
16	289160	8719	8747	10964	226.64	15.7	15.5	21.3	640	220	210	230

It can be seen that the stochastic circuits are more compact and they consume less energy per clock cycle than conventional implementations. However, they suffer from the long latency caused by the required stochastic sequences, which makes the total EPO less competitive. EPO is obtained as the product of power and the time required for performing one operation. TPA is further considered as the number of operations per circuit area in a unit time. When computing the TPA and the EPO, the stochastic FIR filter must work as effectively as the binary conventional FIR filter. The effectiveness is measured using the performance metrics PR and SA in Table 3.2, so the sequence lengths in Table 3.2 must be used, which makes the results even less competitive at high resolutions. In fact, the stochastic approach is no longer competitive in terms of TPA and EPO when long sequences have to be used in a stochastic implementation.

Table 3.4: Comparison of hardware cost, power consumption and minimum clock period for conventional binary and HWA-based stochastic FIR filters.

	Area ()				**Power (mW)**				**Min Clock Period (ps)**			
R (Bits)	**CB**	**CWA**	**MWA**	**HWA**	**CB**	**CWA**	**MWA**	**HWA**	**CB**	**CWA**	**MWA**	**HWA**
3	12757	14944	12972	12855	24.24	20.35	19.89	19.82	390	370	310	340
4	21262	15415	15815	15525	35.5	29.79	29.35	29.05	410	370	310	350
5	31893	21023	18948	18752	47.86	40.17	38.21	38.02	420	400	320	360
6	44650	26841	22331	21998	61.22	51.39	47.56	47.96	440	400	320	360
7	59533	27989	24801	24710	75.49	63.36	59.31	59.06	470	410	340	390
8	76543	27690	27782	27410	90.59	76.04	71.68	71.55	490	410	360	400
9	95679	33545	30796	29986	106.49	89.38	84.82	84.66	490	420	380	410
10	116943	36248	33655	33122	123.12	103.3	98.05	97.23	500	430	380	410
11	140330	38343	36690	36510	140.45	117.9	109.2	109	510	470	390	420
12	165843	44999	43761	43536	138.45	133.1	127.5	125.2	540	470	400	440
13	193482	46574	45921	44910	158.45	148.7	141.5	140.7	550	470	400	440
14	223269	50162	48774	48067	177.1	164.8	159.8	159.2	570	490	410	450
15	255141	53746	51991	50829	196.36	181.5	176.1	174.8	590	490	420	460
16	289160	57322	55039	54898	226.64	198.6	192.8	189.6	640	500	440	460

When the auxiliary circuits, such as stochastic number generators and counters, are shared in a large circuit, their cost may be acceptably small compared to the core stochastic circuit. As the two proposed stochastic designs have similar performance, the stochastic HWA-based circuit is used to compare with the conventional binary circuit. Figures 3.14 and 3.15 show plots of EPO and TPA, respectively, for the binary and stochastic HWA-based circuits (with and without the auxiliary circuits). For stochastic implementations, the sequence length is the factor that dominates the overall circuit performance in terms of TPA and EPO. As the required sequence lengths are the same for both the HWA-based and MWA-based implementations, the circuit performances of the two designs are also similar. Therefore only the TPA and EPO for the stochastic HWA-based circuit are shown. The y-axes in both plots are the base-10 logarithms of the original metrics. The x-axes are the bit resolutions from 3 bits to 16 bits. The two figures show that the stochastic approach is not competitive in terms of the EPO and TPA. The higher the resolution, the less competitive the stochastic implementation becomes. This is caused by the required sequence length, which grows exponentially with the bit resolution. When the auxiliary circuits are not considered, the stochastic circuit shows a better performance. In particular, it performs better in terms of the TPA than the binary design for resolutions below 5 bits.

Although the stochastic designs suffer from long latencies, their performance degrades gracefully as the energy is reduced. Take the 13-bit designs as an example. As shown in Table 3.5, a lower-quality HWA-based stochastic filter is implemented using 262,144 bits (compared to 4,194,304 bits required by an HWA-based filter that matches the performance

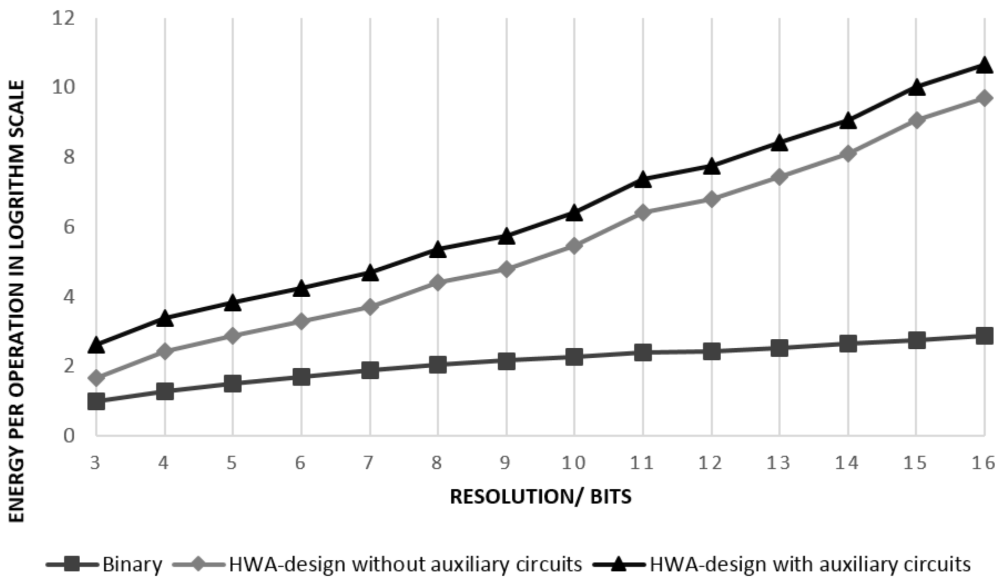


Figure 3.14: EPO comparison: Stochastic HWA design (with/without auxiliary circuits) and the binary design.

of a 13-bit conventional binary filter). An 11-bit conventional binary filter shows similarly degraded performance. The SA of both the lower-quality filters is 6 dB higher than that of the good-quality filters. As the HWA-based stochastic filter only requires 1/16 of the original sequence length, the EPO is just 6.25% of that for the good-quality HWA-based filter. Similarly, the EPO of the lower quality MWA-based stochastic filter is only 6.32% of that for the good-quality MWA-based filter. However, the 11-bit binary filter consumes 82.19% of the energy per operation compared to the 13-bit binary filter. The EPOs of these lower-quality filters are shown in Table 3.5.

Table 3.5: Energy savings with lower-quality implementations for conventional binary (CB), MWA-based and HWA-based FIR filters.

Implementations	CB	MWA	HWA
EPO of Higher Quality Filter (pJ)	87.15	22397583360	253700027
EPO of Lower Quality Filter (pJ)	71.63	1415848960	15856251
Energy Saving (%)	17.81	93.68	93.75

3.5 Error Analysis

3.5.1 Error Analysis for Conventional Binary FIR Filters

A statistical model is developed to evaluate error accumulation and propagation in the computation. Because an FIR filter is a linear system, errors added to the FIR filters are propagated to the next arithmetic operation with similar characteristics. To start the

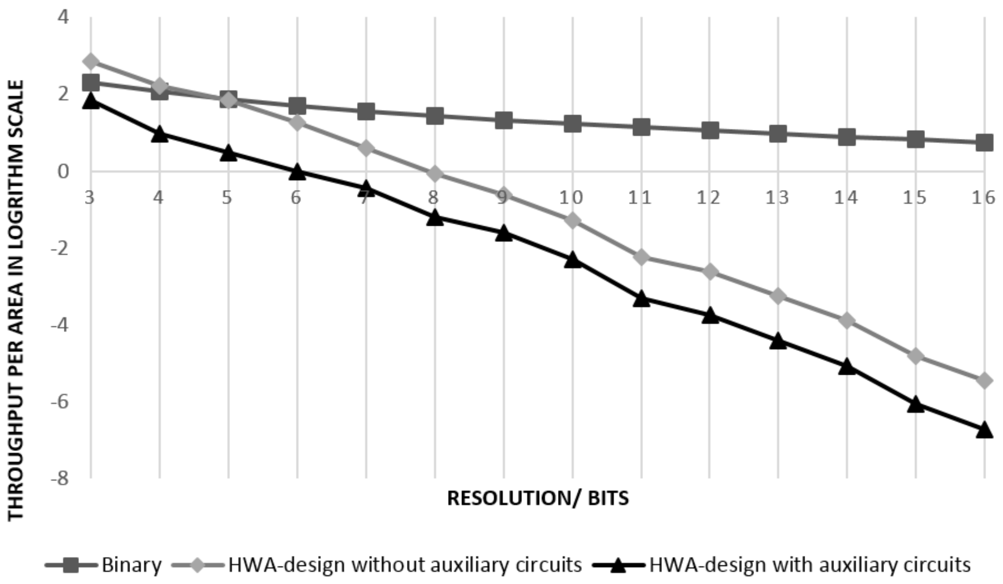


Figure 3.15: TPA comparison: Stochastic HWA design (with/without auxiliary circuits) and the binary design.

analysis, the original/theoretical FIR filter in 3.1 is simplified to

$$Y = \sum_{i=0}^{N_f-1} H_i X_i, \tag{3.12}$$

where H_i and X_i are numbers with infinite precision. By quantization, the actual result Y' is biased from the expected result Y, i.e.,

$$\begin{aligned} Y' &= \sum_{i=0}^{N_f-1} (H_i + e_{Hi})(X_i + e_{Xi}) = \sum_{i=0}^{N_f-1} (H_i X_i + X_i e_{Hi} + H_i e_{Xi} + e_{Hi} e_{Xi}) \\ &= Y + \sum_{i=0}^{N_f-1} (X_i e_{Hi} + H_i e_{Xi} + e_{Hi} e_{Xi}), \end{aligned} \tag{3.13}$$

where e_{Hi} and e_{Xi} are quantization errors of the coefficient Hi and the input Xi. Both e_{Hi} and e_{Xi} can be modeled by e_q in 2.13. To calculate the mean of the computational error, $E(Y')$, i.e.,

$$E(Y') = E(Y) + E(\sum_{i=0}^{N_f-1} (X_i e_{Hi} + H_i e_{Xi} + e_{Hi} e_{Xi})). \tag{3.14}$$

We first determine the mean of the cross product term $e_{Hi} e_{Xi}$. Due to [24], if e_{Hi} and e_{Xi} are independent real-valued continuous random variables with finite expected values, then

$$E(e_{Hi} e_{Xi}) = E\left(e_{Hi}\right) E(e_{Xi}) = 0. \tag{3.15}$$

Taking into consideration Equations 2.17, 3.14 and 3.15, we obtain

$$E(Y') - E(Y) = \sum_{i=0}^{N_f-1} \left(X_i E(e_{Hi}) + H_i E(e_{Xi}) + E(e_{Hi} e_{Xi})\right) = 0. \tag{3.16}$$

By definition and Equation 3.16, the variance of the output Y is

$$Var(Y') = E[(Y' - E(Y'))^2] = E[(Y' - Y)^2]. \tag{3.17}$$

On the other hand, the variance of Y' is also given by

$$\begin{aligned} Var\left(Y'\right) &= \sum_{i=0}^{N_f-1} \left(X_i^2 Var(e_{Hi}) + H_i^2 Var(e_{Xi}) + Var(e_{Hi} e_{Xi})\right) \\ &= \sum_{i=0}^{N_f-1} \left(X_i^2 \sigma_q^2 + H_i^2 \sigma_q^2 + Var(e_{Hi} e_{Xi})\right). \end{aligned} \tag{3.18}$$

Because $Var(e_{Hi} e_{Xi})$ is of a higher order than the other terms in Equation 3.18, we have

$$max\left\{Var(e_{Hi} e_{Xi})\right\} = \frac{\Delta^4}{16} \ll \sum_{i=0}^{N_f-1} \sigma_q^2 \left(X_i^2 + H_i^2\right) = \sum_{i=0}^{N_f-1} \frac{\Delta^2}{12} \left(X_i^2 + H_i^2\right). \tag{3.19}$$

Therefore, $Var(e_{Hi} e_{Xi})$ in 3.18 can be ignored and by Equation 2.18 we obtain

$$Var(Y') \approx \sum_{i=0}^{N_f-1} \sigma_q^2 (X_i^2 + H_i^2) = \frac{\Delta^2}{12} \sum_{i=0}^{N_f-1} (X_i^2 + H_i^2). \tag{3.20}$$

From Equation 3.20, it can be seen that the quantization error is amplified by a multiplicative factor. Then, e_c is defined as the overall error for the conventional binary filter, i.e.,

$$e_c = |Y' - Y|. \tag{3.21}$$

By combining Equations 3.18, 3.20 and 3.21, we obtain the mean of the squared error as

$$\begin{aligned} E\left(e_c^2\right) &= E\left(\left(Y' - Y\right)^2\right) = Var\left(Y'\right) \\ &= \frac{\Delta^2}{12} \sum_{i=0}^{N_f-1} \left(X_i^2 + H_i^2\right) = \frac{1}{12 \cdot 2^{2N_b}} \sum_{i=0}^{N_f-1} \left(X_i^2 + H_i^2\right). \end{aligned} \tag{3.22}$$

By Equation 3.22, e_c can be estimated as

$$\begin{aligned} e_c = |Y' - Y| &\approx \sqrt{\frac{\Delta^2}{12} \sum_{i=0}^{N_f-1} \left(X_i^2 + H_i^2\right)} \\ &= \frac{\Delta}{2\sqrt{3}} \sqrt{\sum_{i=0}^{N_f-1} \left(X_i^2 + H_i^2\right)} = \frac{\sqrt{3}}{3 \cdot 2^{N_b+1}} \sqrt{\sum_{i=0}^{N_f-1} \left(X_i^2 + H_i^2\right)}. \end{aligned} \tag{3.23}$$

3.5.2 Error Analysis for Stochastic FIR Filters

Quantization effect in stochastic computing

The propagation effect of quantization errors in stochastic computing is similar to that in conventional FIR filters. The effect of quantization errors in stochastic computing can be analyzed by evaluating the output P'_Y. If P_Y is the correct output without errors, we have

$$P_Y = \sum_{i=0}^{N_f-1} P_{Hi} P_{Xi}. \tag{3.24}$$

We further include the quantization effect by adding errors $P_{e_{H_i}}$ and $P_{e_{X_i}}$ to input P_{Xi}, the coefficient P_{Hi} and the output P_Y, respectively, where $i = 0, 1, ..., N_f - 1$. Hence the output P'_Y can be calculated by

$$\begin{aligned} P_{Y'} &= \sum_{i=0}^{N_f-1} (P_{Hi} + P_{e_{H_i}})(P_{Xi} + P_{e_{X_i}}) \\ &= \sum_{i=0}^{N_f-1} (P_{Hi}P_{Xi} + P_{Xi}P_{e_{H_i}} + P_{Hi}P_{e_{X_i}} + P_{e_{H_i}}P_{e_{X_i}}) \\ &= P_Y + \sum_{i=0}^{N_f-1} (P_{Xi}P_{e_{H_i}} + P_{Hi}P_{e_{X_i}} + P_{e_{H_i}}P_{e_{X_i}}). \end{aligned} \tag{3.25}$$

$P_{e_{H_i}}$ and $P_{e_{X_i}}$ are independent quantization errors that can be modeled by e_{qs} with the properties in Equations 2.32, 2.33 and 2.34. The mean of the stochastic output P'_Y is thus given by

$$E\left(P_{Y'}\right) = E\left(P_Y\right) + E\left(\sum_{i=0}^{N_f-1} \left(P_{Xi}P_{e_{H_i}} + P_{Hi}P_{e_{X_i}} + P_{e_{H_i}}P_{e_{X_i}}\right)\right) = E\left(P_Y\right). \tag{3.26}$$

The variance of the output $P_{Y'}$ can be obtained from Equation 3.25 as

$$\begin{aligned} Var\left(P_{Y'}\right) &= \sum_{i=0}^{N_f-1} \left(P_{Xi}^2 Var(P_{e_{H_i}}) + P_{Hi}^2 Var(P_{e_{X_i}}) + Var(P_{e_{H_i}}P_{e_{X_i}})\right) \\ &= \sum_{i=0}^{N_f-1} \left((P_{Xi}^2 + P_{Hi}^2)\sigma_{qs}^2 + Var(P_{e_{H_i}}P_{e_{X_i}})\right). \end{aligned} \tag{3.27}$$

$Var(P_{e_{H_i}}P_{e_{X_i}})$ in Equation 3.27 can be ignored due to the fact that

$$\max\left\{Var\left(P_{e_{H_i}}P_{e_{X_i}}\right)\right\} = \frac{\varDelta_s^4}{16} \ll \sum_{i=0}^{N_f-1} \sigma_{qs}^2\left(P_{Xi}^2 + P_{Hi}^2\right) = \sum_{i=0}^{N_f-1} \frac{\varDelta_s^2}{12}\left(P_{Xi}^2 + P_{Hi}^2\right). \tag{3.28}$$

Therefore, Equation 3.27 becomes

$$Var\left(P_{Y'}\right) \approx \sigma_{qs}^2 \left[\sum_{i=0}^{N_f-1} \left(P_{Xi}^2 + P_{Hi}^2\right)\right] = \frac{\Delta_s^2}{12}\left[\sum_{i=0}^{N_f-1} \left(P_{Xi}^2 + P_{Hi}^2\right)\right] \tag{3.29}$$

By definition and 3.26, we have

$$Var\left(P_{Y'}\right) = E\left[\left(P_{Y'} - E\left(P_{Y'}\right)\right)^2\right] = E\left[\left(P_{Y'} - P_Y\right)^2\right]. \tag{3.30}$$

By combining Equations 3.29 and 3.30, we get

$$E\left[\left(P_{Y'} - P_Y\right)^2\right] = \frac{\Delta_s^2}{12}[\sum_{i=0}^{N_f-1} \left(P_{Xi}^2 + P_{Hi}^2\right)]. \tag{3.31}$$

The overall quantization error e_{oq} caused by the quantization effect can be defined as the absolute difference between the calculated result P'_Y and the accurate result P_Y, i.e.,

$$e_o q = |P'_Y - P_Y|. \tag{3.32}$$

From Equations 3.31 and 3.32, the mean of the squared error ${e_{oq}}^2$ can be evaluated as

$$E\left[e_{oq}^2\right] = E\left[\left(P_{Y'} - P_Y\right)^2\right] = \frac{\Delta_s^2}{12}\left[\sum_{i=0}^{N_f-1} \left(P_{Xi}^2 + P_{Hi}^2\right)\right]. \tag{3.33}$$

Because of Equations 2.31 and 3.33, e_{oq} can be approximated as

$$e_{oq} \approx \sqrt{\frac{\Delta_s^2}{12}\left[\sum_{i=0}^{N_f-1} \left(P_{Xi}^2 + P_{Hi}^2\right)\right]} = \frac{\sqrt{3}}{6N_s}\sqrt{\sum_{i=0}^{N_f-1} \left(P_{Xi}^2 + P_{Hi}^2\right)} \tag{3.34}$$

Inaccuracies in the computation of stochastic FIR filters

As discussed in Chapter 2, the stochastic computation can suffer from both quantization errors and fluctuation errors. The overall error e_{os} in a stochastic filter is defined as the sum of the quantization error and the random fluctuation error, i.e.,

$$e_{os} = e_{oq} + e_{fs}. \tag{3.35}$$

Therefore, e_{os} can be further written as the sum of e_{oq} in Equation 3.34 and e_{fs} in Equation 2.41, i.e.

$$e_{os} = \sqrt{\frac{P_Y(1-P_Y)}{N_s}} + \frac{\sqrt{3}}{6N_s}\sqrt{\sum_{i=0}^{N_f-1} \left(P_{Xi}^2 + P_{Hi}^2\right)}. \tag{3.36}$$

It can be seen that the result in Equation 3.36 is similar to the Equation 2.54 except the coefficients of the two terms. This is because we applied the same error models to analyze the stochastic implementations. The first term is dominant which estimates the random fluctuation error.

Stochastic Sequence Length Estimate by Error Analysis

The stochastic sequence length is an important parameter as it determines both the computational accuracy and the circuit performance. To determine the sequence length for stochastic FIR filters, Equation 3.23 is considered to evaluate the overall error of the conventional binary circuit. For the stochastic circuit, the overall stochastic error e_{os} in Equation 3.36 is used as an approximation of the stochastic error. To understand the relationship between bit resolution N_b and sequence length N_s, let $e_{os} = e_c$, i.e.,

$$\sqrt{\frac{P_Y(1-P_Y)}{N_s}} + \frac{\sqrt{3}}{6N_s}\sqrt{\sum_{i=0}^{N_f-1}\left(P_{X_i}^2 + P_{H_i}^2\right)} = \frac{\sqrt{3}}{3 \cdot 2^{N_b+1}}\sqrt{\sum_{i=0}^{N_f-1}\left(X_i^2 + H_i^2\right)}. \tag{3.37}$$

Hence the stochastic sequence length N_s is approximately

$$N_s \approx \frac{12P_Y(1-P_Y)}{\sum_{i=0}^{N_f-1}\left(X_i^2 + H_i^2\right)} \cdot 2^{2N_b}. \tag{3.38}$$

Let α represent the coefficient $\frac{12P_Y(1-P_Y)}{\sum_{i=0}^{N_f-1}\left(X_i^2+H_i^2\right)}$ in Equation 3.38. We then have

$$N_s \approx \alpha \cdot 2^{2N_b}. \tag{3.39}$$

Because α is a constant coefficient, we can use the big O notation to estimate the relationship between the necessary stochastic sequence length for a stochastic FIR filter to match the performance of the conventional binary FIR filter. Therefore, Equation 3.39 can be written as

$$N_s = O\left(2^{2N_b}\right). \tag{3.40}$$

Equation 3.40 shows that the sequence length N_s grows exponentially as the binary resolution N_b increases. This is consistent with the stochastic sequences required in our experiments (see Table 3.2). Clearly, this fact has an adverse effect to any stochastic implementation. However, if a degraded performance in accuracy is acceptable, an exponential reduction would result in the required sequence length. This reduction in sequence length subsequently means a substantial reduction in energy consumption, thus achieving a significant improvement in performance metrics such as the EPO. For example, if the performance of a stochastic filter equivalent to that of an 8-bit binary filter is acceptable for an ideal 12-bit filter, the required sequence length, as per Equation 3.38, would be only 1/256 of the length required for matching the performance of the 12-bit filter. This would reduce the EPO of the stochastic circuit by a factor of 255/256. However, the conventional binary implementation would only result in a much smaller energy reduction, as shown in Table 3.5 (albeit for a different example).

3.6 Fault Tolerance Analysis and Simulation

Although errors are inevitable in the quantization process or caused by the inherent random fluctuation, stochastic computing has been known to be intrinsically fault-tolerant. In this section,the fault tolerance of both the conventional binary and stochastic designs is considered by taking into account soft errors.

3.6.1 Fault-tolerance Analysis

We first discuss the different behaviors of the conventional binary and stochastic circuits using the bit-flip error model in [10] and [18]. Consider a normalized N_b-bit binary number with value B:

$$B = x_1 \cdot 2^{-1} + x_2 \cdot 2^{-2} + ... + x_{N_b} \cdot 2^{-N_b}, \tag{3.41}$$

where x_i is the bit with weight 2^{-i} $(i = 0, 1, ..., N_b - 1)$. Let R_i be a random variable to indicate if an error occurs or not, i.e., if R_i is 1, an error occurs, so bit i flips. Further let ε be the error probability, i.e.,

$$P\left(R_i = 1\right) = \varepsilon. \tag{3.42}$$

Affected by possible bit flips, the normalized binary number becomes

$$B' = \sum_{i=1}^{N_b} \mathrm{x}'_i \cdot 2^{-i} = \sum_{i=1}^{N_b} [R_i\left(1 - \mathrm{x}_i\right) + \mathrm{x}_i\left(1 - R_i\right)] \cdot 2^{-i}. \tag{3.43}$$

The error for the conventional binary approach is defined as

$$e_c^{(i)} = B' - B, \tag{3.44}$$

where a superscript (i) is used to indicate that $e_c^{(i)}$ is to denote the error caused by error injection. It has been shown in [10] that the error $e_c^{(i)}$ has the following mean value and variance:

$$E[e_c^{(i)}] \approx (1 - 2B)\varepsilon; \tag{3.45}$$

$$Var[e_c^{(i)}] \approx \frac{1}{3}(1 - \varepsilon)\varepsilon. \tag{3.46}$$

Since the stochastic implementation does not necessarily use the minimum length for an N_b-bit binary number, the mean value and variance for the stochastic method are re-computed for comparison. The stochastic number S is the encoded value of the N_b-bit binary number B. As a PMM is used, the actual sequence length is $N_s = PMM \cdot 2^{N_b}$. For implementation convenience usually PMM is also a number in the form of $2^{N'_b}$ $(N'_b = 0, 1, 2, ...)$. Hence the N_s-bit stochastic sequence can be generated using a $(N_b + N'_b)$-bit LFSR. A stochastic bit stream $\{y_1, y_2, ..., y_{N_s}\}$ is produced to encode a normalized N_b-bit binary number in [0, 1]. Therefore the stochastic number S can be calculated by

$$S = \frac{1}{N_s} \sum_{i=1}^{N} y_i = \frac{1}{PMM \cdot 2^{N_b}} \sum_{i=1}^{N_s} y_i, \tag{3.47}$$

where N_s is the stochastic sequence length.

For a stochastic circuit, S_i $(i = 1, 2, ..., N_s)$ is defined as a random variable to indicate if an error occurs or not in a stochastic bit stream, i.e., if S_i is 1, an error occurs, so bit i flips. The error injection rate is also considered to be ε, i.e.,

$$P(S_i = 1) = \varepsilon. \tag{3.48}$$

$S_1, S_2, ..., S_{N_s}$ are statistically independent, so

$$E[S_i] = \varepsilon. \tag{3.49}$$

$$Var[S_i] = (1-\varepsilon)\varepsilon. \tag{3.50}$$

A bit affected by a possible error is denoted by y_i', thus

$$y_i' = S_i\left(1 - x_i\right) + x_i\left(1 - S_i\right). \tag{3.51}$$

The stochastic number then becomes

$$S' = \frac{1}{N_s}\sum_{i=1}^{N} \mathrm{y}'_i = \frac{1}{N_s}\sum_{i=1}^{N_s}[S_i\left(1 - \mathrm{y}_i\right) + \mathrm{y}_i\left(1 - S_i\right)]. \tag{3.52}$$

Hence, the stochastic error $e_s^{(i)}$ is determined by

$$e_s^{(i)} = S' - S = \frac{1}{N_s}\sum_{i=1}^{N_s} S_i\left(1 - 2\mathrm{y}_i\right). \tag{3.53}$$

By Equations 3.47, 3.49 and 3.53, the mean of error $e_s^{(i)}$ is given by

$$E\left[S_i\right] = \frac{1}{PMM \cdot 2^{N_b}}\sum_{i=1}^{N_s}\left(1 - 2\mathrm{y}_i\right)\varepsilon = \left(1 - 2S\right)\varepsilon. \tag{3.54}$$

With Equations 3.50 and 3.53, the variance of error $e_s^{(i)}$ is obtained as

$$Var\left[S_i\right] = \frac{1}{N_s^2}\sum_{i=1}^{N_s}\left(1 - 2\mathrm{y}_i\right)^2\left(1 - \varepsilon\right)\varepsilon = \frac{1}{PMM \cdot 2^{N_b}}\left(1 - \varepsilon\right)\varepsilon. \tag{3.55}$$

To investigate how injected errors affect the function to implement FIR filters, it is assumed that the injected errors $e_c^{(i)}$ and $e_s^{(i)}$ follow additive Gaussian distributions.

For the conventional binary FIR filter defined in Equation 3.12 without injected errors, the erroneous output $Y^{(i)}$ is given by

$$Y^{(i)} = \sum_{i=0}^{N_f - 1}(H_i + e_{Hi}^{(i)})(X_i + e_{Xi}^{(i)}), \tag{3.56}$$

where $e_{Hi}^{(i)}$ and $e_{Xi}^{(i)}$ are independent errors modeled by $e_{oc}^{(i)}$ with the mean and variance given in Equations 3.45 and 3.46, respectively. The overall error due to error injection for the conventional binary FIR filters $e_{oc}^{(i)}$ is then

$$e_{oc}^{(i)} = Y^{(i)} - Y. \tag{3.57}$$

When the error injection rate is small, the mean and variance of the overall error $e_{oc}^{(i)}$ are given by (approximately)

$$E\left[e_{oc}^{(i)}\right] \approx \sum_{i=0}^{N_f-1} [(H_i + X_i - 4H_iX_i)\varepsilon + (1-2H_i)(1-2X_i)\varepsilon^2]; \tag{3.58}$$

$$Var\left[e_{oc}^{(i)}\right] \approx \frac{\varepsilon}{9} \sum_{i=0}^{N_f-1} \left\{3\left(H_i^2 + X_i^2\right) + [1 - 3\left(H_i^2 + X_i^2\right)]\varepsilon\right\}. \tag{3.59}$$

How Equations 3.58 and 3.59 are derived is shown in detail in the appendix. Similarly, the overall error due to error injection for the stochastic FIR filters $e_{oc}^{(i)}$ is given by

$$e_{os}^{(i)} = P_{Y^{(i)}} - P_Y. \tag{3.60}$$

Its mean and variance are given by

$$E\left[e_{os}^{(i)}\right] = \sum_{i=0}^{N_f-1} [(P_{Hi} + P_{Xi} - 4P_{Hi}P_{Xi})\varepsilon + (1-2P_{Hi})(1-2P_{Xi})\varepsilon^2]; \tag{3.61}$$

$$Var\left[e_{os}^{(i)}\right] \approx \frac{\varepsilon}{N_s^2} \sum_{i=0}^{N_f-1} \left\{N_s\left(H_i^2 + X_i^2\right) + \left[1 - N_s\left(P_{Xi}^2 + P_{Hi}^2\right)\right]\varepsilon\right\}. \tag{3.62}$$

$E[e_{oc}^{(i)}]$ and $E[e_{os}^{(i)}]$ in Equations 3.58 and 3.61 show that the mean output error depends on both the inputs and the coefficients. The mean of the stochastic error $E[e_{os}^{(i)}]$ is identical to the mean of the conventional binary error $E[e_{oc}^{(i)}]$ for the same filter function (thus with the same inputs and coefficients).

To compare the variances of the binary error and the stochastic error, all the inputs and coefficients in Equations 3.59 and 3.62 are assumed to be 0.5. We then obtain

$$Var\left[e_{oc}^{(i)}\right] \approx \frac{5N_f}{18}\varepsilon - \frac{N_f}{6}\varepsilon^2, \tag{3.63}$$

$$Var\left[e_{os}^{(i)}\right] \approx \left(\frac{N_f}{N_s^2} + \frac{N_f}{2N_s}\right)\varepsilon - \frac{N_f}{2N_s}\varepsilon^2. \tag{3.64}$$

For any $N_s \geq 3$, the variances in Equations 3.63 and 3.64 satisfy

$$Var\left[e_{oc}^{(i)}\right] > Var\left[e_{os}^{(i)}\right]. \tag{3.65}$$

Due to the factor of N_s, the stochastic method results in a smaller variance. The variation of the error for the stochastic implementation $e_{os}^{(i)}$ is inversely proportional to the sequence length squared, ${N_s}^2$. When using $N_s = PMM \cdot 2^{N_b}$ $(PMM = 2^0, 2^1, 2^2, ...)$ bits in the stochastic encoding of an N_b-bit binary number, the variance of the stochastic error can be reduced by increasing the PMM. In the conventional binary approach, however, it is more difficult to obtain a smaller variance as it lacks the tuning parameter PMM in the stochastic approach.

3.6.2 Fault-tolerance Simulation

Simulations are further performed to evaluate the reliability of the binary and stochastic circuits. To measure the reliability of a design, the average absolute error (AAE) is defined as

$$AAE = \frac{1}{M} \cdot \frac{1}{2^{2N_b+4}} \sum_{i=0}^{M-1} |X_i - X_i'|, \tag{3.66}$$

where X_i and X_i' are the expected correct output and the actual output, respectively, M is the number of simulations, and the factor $\frac{1}{2^{2N_b+4}}$ is taken as a constant coefficient so that the AAEs are between 0 and 1 ($N_b = 13$ here). The AAE indicates how seriously the injected error affects the correct output.

The AAE for the conventional binary 13-bit low-pass FIR filter with 267 taps is investigated, as well as the stochastic MWA design and HWA design using a sequence length of 4,194,304 bits (from Table 3.2) under various injected error rates. In addition, redundant copies of the binary circuit can be used to achieve a better fault tolerance, for example, in the form of TMR. The TMR implementations of the binary circuit with unreliable and fault-free voters are further considered. The stochastic computational models in [18] are used to facilitate our fault-tolerance analysis. XOR gates are used to inject errors into the circuit. The majority voters in the TMR circuits are considered bitwise rather than word-wise.

Table 3.6 shows the comparison of AAEs obtained from 200 simulations with a sequence length of 100,000 bits. The results with error injection are compared with those without error injection, thus the AAEs for the stochastic circuits are 0 when the injected error rate is 0. It can be seen that the AAE increases as the injected error rate increases. The conventional binary circuit is not as fault-tolerant as the stochastic circuits, which is consistent with the analysis. When one bit in a binary circuit flips, it can cause a serious error if the erroneous bit is among the most significant bit (MSB)s. However, all bits in a stochastic sequence have the same weight, so the effect of a single bit flip is insignificant in a relatively long stochastic sequence. The binary TMR circuit with unreliable voters has an improved reliability, but it is still not as reliable as the stochastic approaches. However, the binary TMR circuit with reliable voters becomes more fault-tolerant than the stochastic circuits.

Table 3.6: Average absolute error of the stochastic and binary circuits with and without redundancy at various injected error rates. The results are obtained from 200 simulations using sequences of 100,000 bits.

	Average Absolute Error (%)				
Injected Error Rate (%)	MWA	HWA	Binary	Binary TMR	
				Error-free Voter	Unreliable Voter
0	0	0	0	0	0
0.1	0.063	0.065	1.507	0.004	0.126
0.2	0.121	0.136	2.325	0.009	0.225
0.5	0.339	0.326	3.29	0.035	0.581
1	0.592	0.574	5.209	0.111	1.203
2	1.476	1.226	6.368	0.198	2.382
5	3.009	2.948	10.942	0.337	5.794
10	5.472	5.696	21.477	1.123	12.05

3.7 Summary

In this chapter, a stochastic HWA design and a stochastic MWA design are proposed for implementing FIR filters. The HWA design takes advantage of simply repeating the input wires of a multiplexer to implement the weights of different data inputs, while the MWA design uses multiplexers to generate the required filter coefficients or weights. The proposed stochastic designs show an improved performance, a smaller circuit area and lower power consumption, compared with the conventional stochastic design. The MWA design with multiple stages is not as competitive as the HWA design, but it can be more easily reconfigured by re-programming the weight generators. This task is not easy for the HWA design that uses repeated inputs to implement the filter coefficients.

Compared to binary FIR filter circuits, the proposed stochastic designs have a significant advantage in circuit area, especially at higher resolutions. With respect to the performance metrics of throughput per area and energy per operation, however, the stochastic design does not show any advantages over its binary counterpart. This is primarily due to the significant latency in stochastic computing because long stochastic sequences must be used to achieve the same filtering performance as a binary circuit. With a shorter stochastic sequence, however, the stochastic circuit shows a graceful degradation in performance compared to the binary design. The features of a stochastic circuit are investigated in detail by both analysis and simulation.

A binary TMR circuit using error-free voters is shown to be more reliable than the stochastic design. Due to its intrinsic fault tolerance, however, the proposed stochastic design shows significant advantages in reliability over the conventional binary design and its TMR implementation when the voters are subject to errors. These results suggested that other SOP based circuits could also benefit from the stochastic implementation.

Chapter 4

Stochastic Circuit Design and Evaluation of Vector Quantization

4.1 Background

Vector quantization based compression algorithms are useful in that the amount of stored and transmitted data can be reduced with a readily adjusted trade-off between compression ratio and implementation size. These features are important in multimedia processing and communications, such as for voice and image compression. VQ is a lossy data compression method, and the loss in the original information must be kept as low as possible. In VQ information loss can be reduced by simply using a larger suitably designed codebook. The resulting extra codebook search time can be minimized by using more parallel computational elements [28].

VQ is a useful method in speech recognition. In [29], VQ is employed to characterize a speaker's voice. The minimum-distance entry in a codebook of speakers is used to recognize the identity of an arbitrary speaker. VQ also has applications in image compression coding. Several major issues were discussed in [20] such as the edge degradation issue. One of the potential solutions is to use a dynamic codebook whose contents can be updated so that the codevectors can match the partial image to be encoded.

In this chapter, the possibility of using stochastic computing is explored to realize VQ. To implement VQ based on the L^1-norm and squared L^2-norm errors, a few basic computational elements are required such as multiplication and addition. For other error measures such as the L^p-norm or p^{th}-law ($p \geq 3$), the Bernstein polynomial method is introduced to efficiently implement high-order polynomials. Stochastic computing has feature that it is able to conveniently provide progressive quality. Typically, a longer sequence offers greater representational accuracy. In the VQ compression algorithm, shorter sequences are used to provide faster computation but with less accurate results. To measure important performance characteristics, both stochastic and binary VQ designs are implemented. Starting with the requirements of a vector quantization implementation, stochastic computing using both combinational and sequential logic is introduced. Synthesis reports from the Synopsys design compiler (version:) are discussed as well as the simulation results in an image processing case study. The competitive sequence lengths are determined and the circuit performances are compared.

4.2 Methodology

VQ is a lossy digital compression technique. First the source data is partitioned into equal-length vectors [30]. Each vector is then replaced with the index of the closest matching codevector that is contained in a given codebook. This encoding process is shown in Figure 4.1(a). Note that each input vector can be represented more compactly using the index of the closest codevector. The indexes are converted back to the corresponding codevectors during decompression, which is the decoding process shown in Figure 4.1(b). The principle in generating a codebook of a given size is to minimize the expected error distances to the input vectors over the expected input data domain. The size of the codebook (i.e., number of codevectors) determines the trade-off between the accuracy of the representation and the transmission bit rate of codevector indexes.

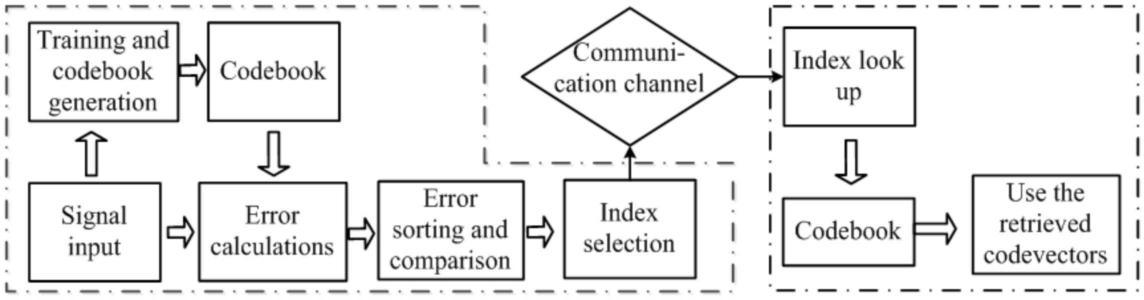


Figure 4.1: The block diagram for the (a) encoding and (b) decoding process in Vector Quantization.

4.2.1 Codebook Generation

In 1980's, Linde, Buzo and Gray (LBG) proposed a VQ codebook generation algorithm [28]. Using the generated codebook, source or input vectors can be encoded based on a training sequence. The use of a training sequence makes it possible to generate a codebook with reduced computational cost. Although other efficient codebook generation approaches have been developed, the LBG algorithm was selected to generate our codebooks because of its efficiency. The VQ encoding process is as follows:

1) A set of N_x source vectors $\{\mathbf{X_1}, \mathbf{X_2}, ..., \mathbf{X_{N_x}}\}$ is to be compressed.

2) A codebook with N_c codevectors generated previously as

$$C = \{\mathbf{C_1}, \mathbf{C_2}, ..., \mathbf{C_{N_c}}\}. \tag{4.1}$$

3) The codevector $\mathbf{C_i}$ ($i = 1, 2, ..., N_c$) in the codebook that is the nearest to each of the source vectors based on errors E_i ($i = 1, 2, ..., N_c$) must be found. If function f maps the

source vector $\mathbf{X}$ to its nearest codevector $\mathbf{C_i}$, we have

$$f(\mathbf{X}) = \mathbf{C_i},\ \mathrm{if}\ \mathrm{E_i} \leq \mathrm{E'_i},\ \forall \mathrm{i'} = 1, 2, ..., \mathrm{N_c}. \tag{4.2}$$

By L^1 norm, squared L^2 norm and p^{th}-law ($p \geq 3$), E_i is defined in Equations 4.3, 4.4 and 4.5, respectively:

$$L^1\, norm : E_i = |\mathbf{X} - \mathbf{C_i}|, \tag{4.3}$$

$$Squared\, L^2\, norm : E_i = |\mathbf{X} - \mathbf{C_i}|^2, \tag{4.4}$$

$$p^{th} - law : E_i = |\mathbf{X} - \mathbf{C_i}|^p, \tag{4.5}$$

where $i = 1, 2, ..., N_c$ are the indexes of the codevectors.

4) Compression is obtained by mapping the N_x source vectors to the N_x corresponding indexes of the closest codevectors.

5) A compressed approximation to the N_x source vectors is constructed from the compressed representation by replacing the indexes with the corresponding codevectors.

4.2.2 Error Calculation in the Encoding Process

As shown in Figure 4.1, the pre-defined codebook is used to encode every source vector $\mathbf{X}$. The distance between $\mathbf{X}$ and each of the codevectors in the codebook must be calculated. The index of the codevector that is the closest to $\mathbf{X}$ is determined and then used to encode $\mathbf{X}$. A source vector $\mathbf{X}$ is consisted of N_e elements as follows.

$$\mathbf{X} = \{X_0, X_1, ..., X_{N_e - 1}\}. \tag{4.6}$$

There are two common ways to define the required distance metric. The L^1-norm and squared error distance calculation formulas are shown below in Equations 4.7 and 4.8. Let N_c be the number of codevectors in the codebook, or the number of possible error distances that must be compared. N_e is the number of elements in a vector (any codevector or input vector $\mathbf{X}$). Index i identifies the different codebook entries and hence error distances and index j identifies the elements in the vectors.

$$L^1\, norm : E_i = \sum_{j=0}^{N_e - 1} |X_j - C_{ij}|,\ i = 1, 2, \ldots, N_c. \tag{4.7}$$

$$Squared\, L^2\, norm : E_i = \sum_{j=0}^{N_e - 1} (X_j - C_{ij})^2,\ i = 1, 2, \ldots, N_c. \tag{4.8}$$

By expanding the squared error in Equation 4.8, it turns out to be

$$E_i = \sum_{j=0}^{N_e - 1} ({X_j}^2 - 2X_jC_{ij} + {C_{ij}}^2),\ i = 1, 2, \ldots, N_c. \tag{4.9}$$

As the input vector X is constant during the comparison (i.e. $\sum\limits_{j=0}^{N_e-1} X_j^2$ does not change for all the N_c codevectors), ${X_j}^2$ in Equation 4.9 can be ignored and only the other two terms are needed in calculation. That is, we simply need to calculate and compare the result E_i' in Equation 4.10.

$$E_i = \sum_{j=0}^{N_e-1} (C_{ij}^2 - 2C_{ij}X_j),\ i = 1, 2, \ldots, N_c. \tag{4.10}$$

The L^p-norm error and its p^{th} power (or p^{th}-law) error are also regular error measures and they are defined as

$$E_i = (\sum_{j=0}^{N_e-1} |X_j - C_{ij}|^p)^{1/p},\ i = 1, 2, \ldots, N_c, \tag{4.11}$$

$$E_i = \sum_{j=0}^{N_e-1} |X_j - C_{ij}|^p,\ i = 1, 2, \ldots, N_c, \tag{4.12}$$

where p is an integer and $p \geq 3$ [28]. The error measure in Equation 4.11 is more widely used since it is a distance that satisfies the triangle inequality:

$$d\,(\mathbf{X}, \mathbf{Y}) + d(\mathbf{Y}, \mathbf{Z}) \geq d(\mathbf{X}, \mathbf{Z}),\ \text{for any}\ \mathbf{Y}. \tag{4.13}$$

Here $d(\mathbf{X}, \mathbf{Y})$ is the distance between two vectors $\mathbf{X}$ and $\mathbf{Y}$. This property makes it easy to bound the overall error. However, $f(x) = x^{1/p}$ is a monotonic increasing function for $p \geq 3$. If two p^{th}-law errors satisfy $E_i \geq E_j$, then $E_i^{1/p} \geq E_j^{1/p}$, where $E_i^{1/p}$ and $E_j^{1/p}$ are the L^p-norm errors in Equation 4.11. Errors measured by p^{th}-law in Equation 4.12 are therefore considered for computational convenience.

With the errors computed, the next step is to compare and find the minimum error distance E_{min} using Equations 4.7, 4.10 and 4.12. If $E_i = E_{min}$, index i is then used as the compressed encoding of the input vector $\mathbf{X}$.

4.3 Proposed Vector Quantization Circuit Design

4.3.1 Overall System Architecture

The VQ system can be abstracted as in Figure 4.2. As an example for performance evaluation and comparison, consider an image of 300 pixels by 300 pixels. Each of the four-by-four square blocks is considered a vector while the pixel values are the elements in the vector. Figure 4.3 illustrates how the 16-element input vectors are formed. There are thus $300 \times 300/6 = 5625$ four-by-four pixel blocks in this image. Suppose we have 256 codevectors in the codebook. Both the binary and stochastic implementations of VQ use errors calculated based on the L^1 norm, squared L^2 norm and p^{th}-law in the system architecture in Figure 4.7. In addition to the gates for error calculation and comparison, the total hardware cost must also include memory cost because the iteratively updated errors are stored in indexed arrays.

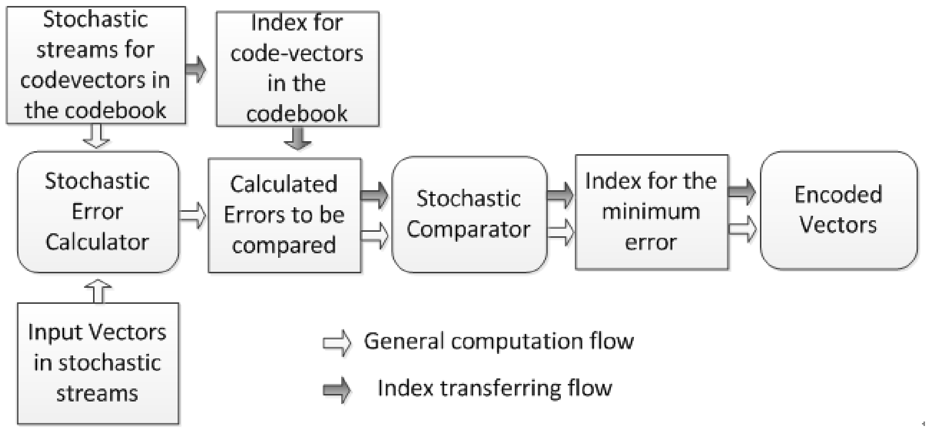

Stochastic streams for codevectors in the codebook
Index for code-vectors in the codebook
Stochastic Error Calculator
Calculated Errors to be compared
Stochastic Comparator
Index for the minimum error
Encoded Vectors
Input Vectors in stochastic streams
General computation flow
Index transferring flow

Figure 4.2: Data flow in vector quantization encoding process.

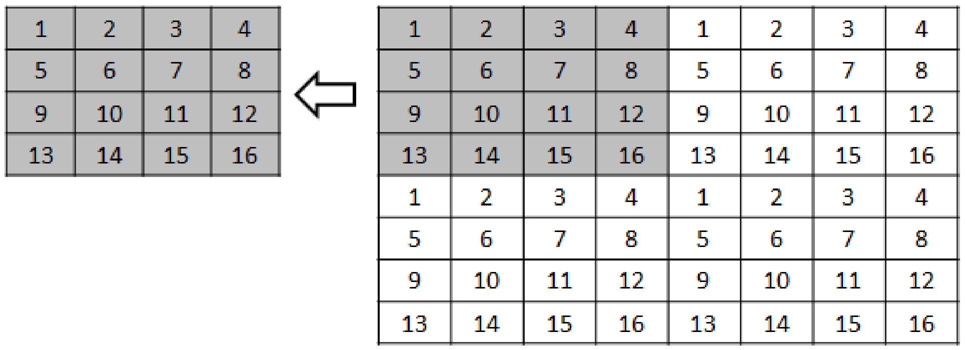


Figure 4.3: An input vector has 16 entries, from 1 to 16 in the left block, forming a macro-pixel. In the right block, the array of 8×8 pixels can be divided into 4 macro-pixels.

4.3.2 Polynomial Arithmetic Synthesized Using Bernstein Polynomials

Although various operations can be implemented by combinational and sequential digital circuits, a general synthesis approach for stochastic logic is needed to implement an arbitrary single-variable polynomial. In [7], this problem is solved by converting a polynomial to a series of Bernstein basis polynomials. An encoder is needed to implement the Bernstein basis polynomials. The coefficients are then associated with the corresponding basis polynomials by a multiplexer.

To implement the absolute value of an arbitrary polynomial, we can follow a standard design flow as below [7].

1) Convert the polynomial to a linear combination of Bernstein basis polynomials with Bernstein coefficients. Suppose that there is an n-order polynomial

$$y = \left| \sum_{k=0}^{n} a_k \cdot x^k \right|. \tag{4.14}$$

The n-order polynomial inside the absolute value operation in Equation 4.12 can be converted into a Bernstein polynomial with $(n+1)$ Bernstein basis functions, i.e.,

$$y = \left| \sum_{k=0}^{n} b_k \cdot B_{n,k} \right|, \tag{4.15}$$

where b_k $(k = 0, 1, 2, ..., n)$ is a Bernstein coefficient, and $B_{n,k}$ is a basis polynomials. They can be calculated by

$$b_k = \sum_{j=0}^{k} \frac{\begin{pmatrix} k \\ j \end{pmatrix}}{\begin{pmatrix} n \\ j \end{pmatrix}} \cdot a_k, \tag{4.16}$$

$$B_{n,k} = \begin{pmatrix} n \\ k \end{pmatrix} x^k (1-x)^{n-k}, \tag{4.17}$$

where $k = 0, 1, 2, ..., n$.

2) Normalize the Bernstein coefficients so that they can be implemented using stochastic logic. The input x and coefficients b_k $(k = 0, 1, 2, ..., n)$ are converted to stochastic sequences.

3) Properly assign input wires of a multiplexer for each basis polynomial and assign the Bernstein coefficients with combined inputs. Here we take a 4-term Bernstein polynomial as an example. We assume that the input x is a normalized positive number, which can be converted using unipolar stochastic number generators (SNG_u) as in Figure 2.7. The Bernstein coefficient b_k $(k = 0, 1, 2, ..., n)$ is encoded as a bipolar stochastic sequences using the bipolar stochastic number generator (SNG_b). The polynomial can be written as

$$y = \left| b_0 \begin{pmatrix} 3 \\ 0 \end{pmatrix} (1-x)^3 + b_1 \begin{pmatrix} 3 \\ 1 \end{pmatrix} x(1-x)^2 + b_2 \begin{pmatrix} 3 \\ 2 \end{pmatrix} x^2 (1-x) + b_3 \begin{pmatrix} 3 \\ 3 \end{pmatrix} x^3 \right|. \tag{4.18}$$

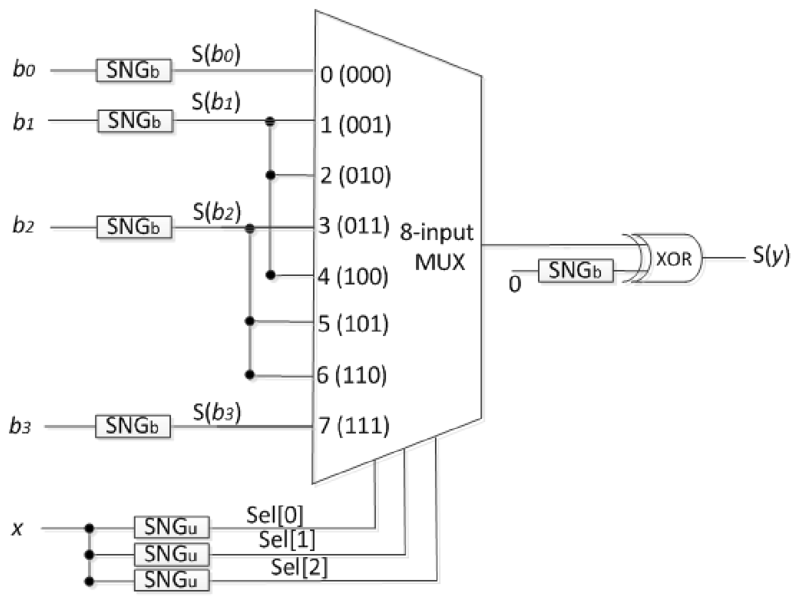


Figure 4.4: Architecture of the 4-term Bernstein polynomial defined in Equation 4.18 using stochastic logic.

In Figure 4.4, the input x is encoded using three different unipolar SNGs as three uncorrelated stochastic sequences which then become the three selecting signals of the 8-input multiplexer. The data inputs of the multiplexer are indexed with binary numbers from $(000)_2$ to $(111)_2$. The coefficient b_k is then connected to the data inputs whose binary index has k $1'$s. For instance, data inputs indexed by $(001)_2$, $(010)_2$ and $(100)_2$ are all connected to coefficient b_1 (see Figure 4.4). Therefore, the probability of selecting coefficient b_1 as the output of the multiplexer is $\begin{pmatrix} 3 \\ 1 \end{pmatrix} x(1-x)^2$. If this strategy is applied to the other coefficients, the architecture in Figure 4.4 implements the Bernstein polynomial in Equation 4.18. The architecture in Figure 4.4 can be easily scaled for other problems by using larger multiplexers. Generally, for the Bernstein polynomial with $(n+1)$ terms in Equation 4.15, a 2^n-input multiplexer with combined data inputs is needed.

To implement the absolute value function at the output of the multiplexer, we take advantage of the XOR gates shown in Figures 2.2 and 2.3. One of the inputs of the XOR gate is the output of the multiplexer while the other one is a correlated bipolar stochastic sequence encoding 0. Here, correlated sequences are referred to as two sequences that have the maximum overlapped 1's. Ideally, two correlated N_s-bit stochastic sequences S1 and S2 satisfy

$$\sum_{i=1}^{N_s} |S1_i - S2_i| = |\sum_{i=1}^{N_s} S1_i - \sum_{i=1}^{N_s} S2_i|. \tag{4.19}$$

$S1_i$ and $S2_i$, being either 0 or 1, are the i^{th} bits in the stochastic sequences $S1$ and $S2$, respectively. To guarantee correlation, we use the same SNGs and the same initial seeds to encode all of the Bernstein coefficients as well as 0, which is the second input of the XOR gate. Output $S(y)$ is thus the absolute value of the Bernstein polynomial and it can now be treated as a unipolar stochastic sequence encoding numbers in the range between 0 and 1.

4.3.3 Detailed design for stochastic VQ

Stochastic VQ Implementation Using L^1-norm Errors

In the L^1-norm error calculation, Equation 4.7 needs to be implemented. In Figure 4.5, an example is shown with 16 elements in a vector, i.e. $N_e = 16$. $X[i]$ and $H[i]$ represent the i^{th} element in the input vector and one of the $N_c = 256$ codevectors, respectively. Both $X[i]$ and $H[i]$ are encoded as stochastic sequences from their previous 8-bit binary values for a grey-scale image. This can be done using SNGs, as described in [19], and the computation is based on stochastic unipolar representations. Therefore in Figure 4.5, the $\mathrm{SNG_u}$ (see Figure 2.7) is used to denote the unipolar (regular) stochastic number generators. $S(Y)$ is the stochastic output, which must be stored prior to being converted back to a binary number at the final stage using a counter. In our stochastic VQ design, however, this conversion is not needed as the final result of the encoding process is a binary index which has been embedded in the stochastic sequences. The XOR gates are used to implement the absolute subtractions in stochastic computing with correlated stochastic sequences, where correlated sequences have the maximum overlap of 1's [7]. This can be implemented by sharing the same LFSR for SNGs at the inputs of the XOR gates. For the inputs of the different XOR gates and selecting inputs of the multiplexer, however, we generate sequences with different LFSRs and initial seeds for SNGs in order to reduce the correlation. Then the results are added up by the 16-input multiplexer whose selecting signals Sel[0] to Sel[3] are four independent stochastic sequences encoding 0.5.

256 copies of the circuit in Figure 4.5 are implemented, which is the L^1-norm error calculator, so that the 256 errors can be computed in parallel. The next step is to compare the 256 errors with a tree-structured comparator to find the minimum one. Note that the architecture in Figures 4.6 and 4.7 implements a 16-error comparison. The squares represent the error calculators. The triangles represent the stochastic comparators implemented in Figure 2.6. A stochastic comparator has two inputs and the smaller one of them is selected as the output. To use this tree structure, it must be extended to a larger scale to compare 256 errors at one time. Meanwhile, all the indexes of the codevectors should be stored and delivered so that the one with the minimum error can be identified.

Stochastic VQ Implementation Using L^2-norm Errors

In the squared L^2-norm error calculation, the function to be implemented is the square function and multiplications shown in Equation 4.10. In Figure 4.8, an example is shown where the number of elements in a vector is 16 ($N_e = 16$) using both traditional stochastic arithmetic elements (i.e. XOR gates to implement the negative value of a bipolar multi-

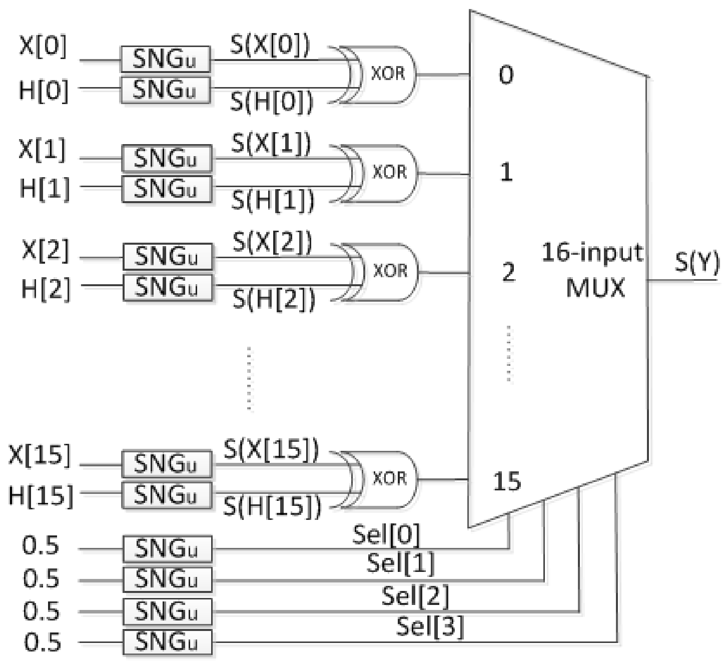


Figure 4.5: Architecture of the L^1-norm error calculator.

plication) and the Bernstein polynomial method. $X[i]$ and $H[i]$ represent the i^{th} element in the input vector and one of the 256 codevectors ($N_c = 256$), respectively. Both $X[i]$ and $H[i]$ are stochastic sequences encoded from their previous 8-bit binary values. As the simplified squared L^2-norm error calculator in Equation 4.10 could be negative, bipolar SNGs (see Figure 2.8) are used in Figures 4.8 and 4.9 (denoted by SNG_b). $S(Y)$ is the stochastic output. Note that the selecting inputs are still sequences generated by unipolar SNGs (denoted by SNG_u in Figures 4.8 and 4.9). Additions are also implemented using multiplexers of various sizes.

For the implementation using traditional stochastic arithmetic elements in Figure 4.8, the upper 16 inputs are bipolar stochastic sequences $H[0]^2, H[1]^2, ..., H[15]^2$, encoding the squares of the coefficients. The bottom 16 XOR gates are used to calculate additive inverse of the coefficients multiplied by primary inputs $X[0], X[1], X[15]$. A 32-input multiplexer is then used to sum up all the squared coefficients $H[0], H[1], ..., H[15]$ from its upper 16 inputs and subtract the products of the 16 pairs of primary inputs and coefficients from its bottom 16 inputs. The selecting signals Sel[0] to Sel[3] are four independent stochastic sequences encoding 0.5. The selecting signal Sel[4] is used to implement the constant coefficient '2' before the cross products. Therefore, a sequence encoding 2/3 is generated. In this way, the probability of selecting the top 16 primary inputs from 0 to 15 is 1/3, and the probability of selecting the bottom 16 primary inputs (from inputs 16 to 31) is 2/3.

For the implementation using the Bernstein polynomial method in Figure 4.9, Equation

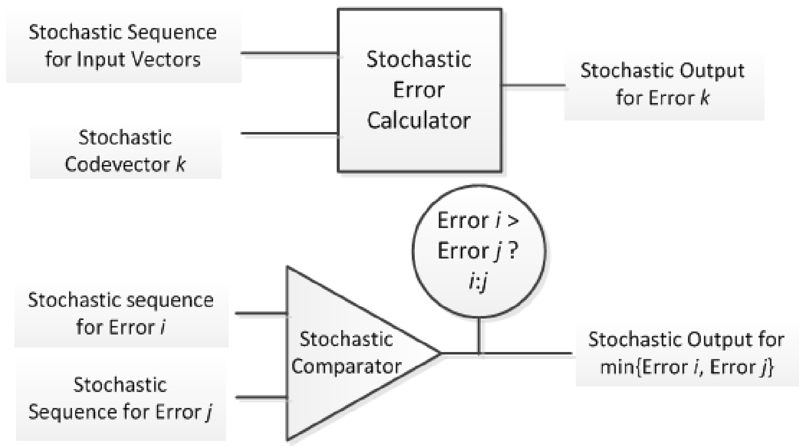

Stochastic Sequence for Input Vectors
Stochastic Codevector k
Stochastic Error Calculator
Stochastic Output for Error k
Error i > Error j ? i:j
Stochastic sequence for Error i
Stochastic Sequence for Error j
Stochastic Comparator
Stochastic Output for min{Error i, Error j}

Figure 4.6: Stochastic elements for VQ.

4.10 is converted to Equation 4.20 first into the form of Bernstein polynomials.

$$E_i' = \sum_{j=0}^{N_e-1} [b_{0,ij} X_{ij} + b_{1,ij}(1 - X_{ij})], i = 1, 2, \ldots, N_c, \tag{4.20}$$

where $b_{0,ij} = C_{ij}^2 - 2C_{ij}$ and $b_{1,ij} = C_{ij}^2$ are pre-computed Bernstein coefficients. For each term indexed by j ($j = 0, 1, 2, ..., N_e - 1$) in Equation 4.20, a 2-input multiplexer is needed. There are a total of 16 two-input multiplexers ($N_e = 16$). A 16-input multiplexer is then used to sum up the outputs of all the two-input multiplexers. The new coefficients $b_{0,ij}$ and $b_{1,ij}$ are encoded by independent bipolar stochastic number generators (denoted by $\mathrm{SNG_b}$) while the primary inputs X_{ij} is encoded by independent unipolar stochastic number generators (denoted by $\mathrm{SNG_u}$). The selecting signals Sel[0] to Sel[3] are four independent stochastic sequences encoding 0.5, which are also unipolar stochastic sequences. The outputs of the squared L^2-norm error calculators in Figures 4.8 and 4.9 are then passed on to the parallel comparison tree in Figure 4.7.

To compare the two implementations shown respectively in Figures 4.8 and 4.9, the circuits for the squared error calculators are designed and synthesized with the Synopsys Design Compiler tool [26]. The resulting synthesis report provides the silicon area, the power and the minimum clock period (see Table 4.1). It is clear that the Bernstein polynomial method shows a slightly better performance, so the implementation in Figure 4.9 is selected as the squared L^2-norm error calculator. Although the advantage over the traditional stochastic arithmetic elements is not so significant for the squared L^2-norm error calculation, the Bernstein polynomial method becomes more favorable for L^p-norm or p^{th}-law ($p \geq 3$) error calculations. This is primarily because the higher-order terms can be more efficiently implemented using the Bernstein polynomial method in that less stochastic number generators are required.

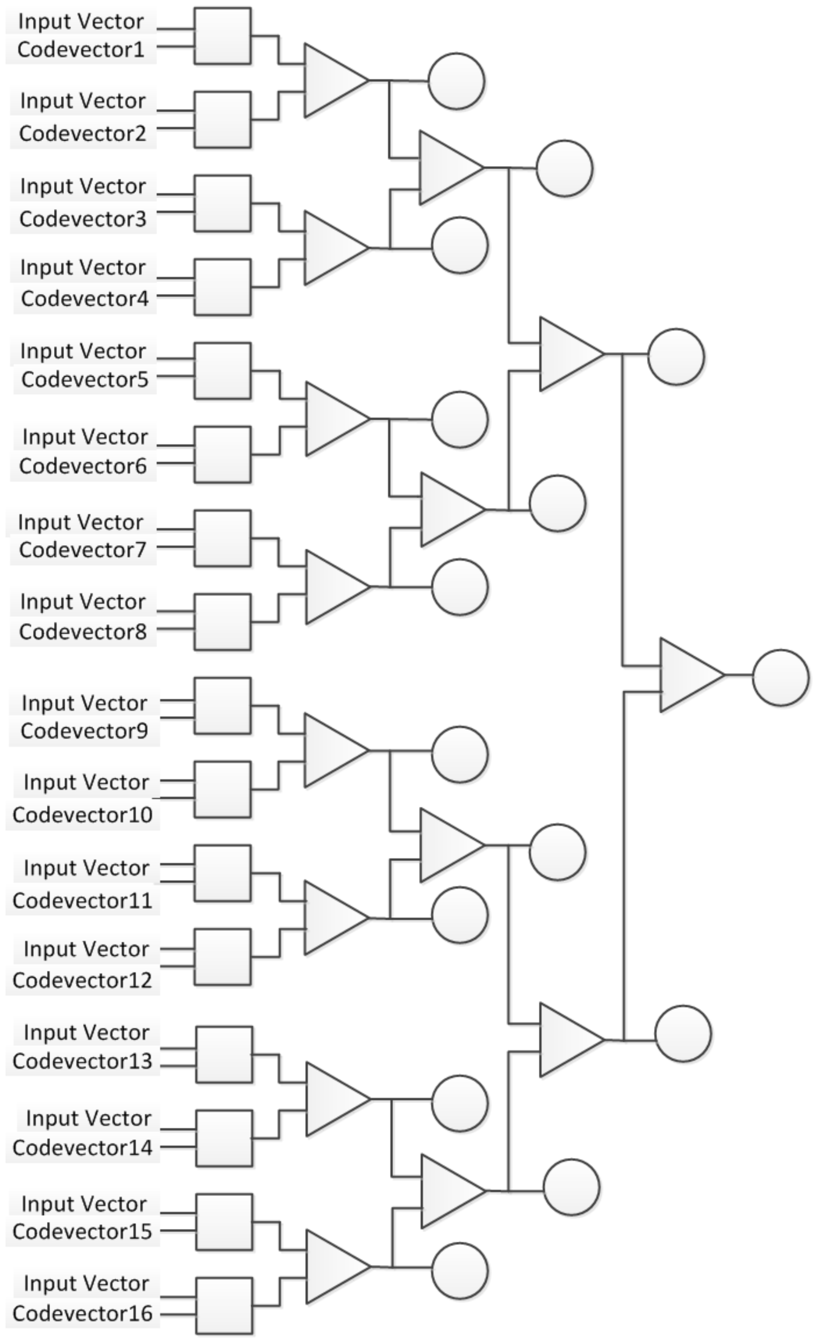


Figure 4.7: Stochastic comparison tree: a square represents an error calculator, a circle represents a comparison result and a triangle represents a stochastic comparator (see Figure 4.6). Note that the codevector with the smallest error appears at the output as the comparison result.

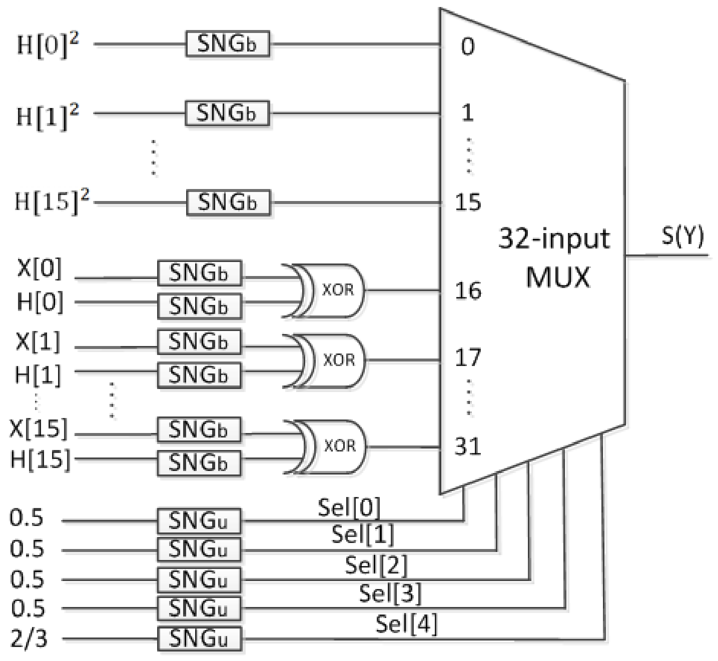


Figure 4.8: Architecture of the squared L^2-norm error calculator using traditional stochastic arithmetic elements.

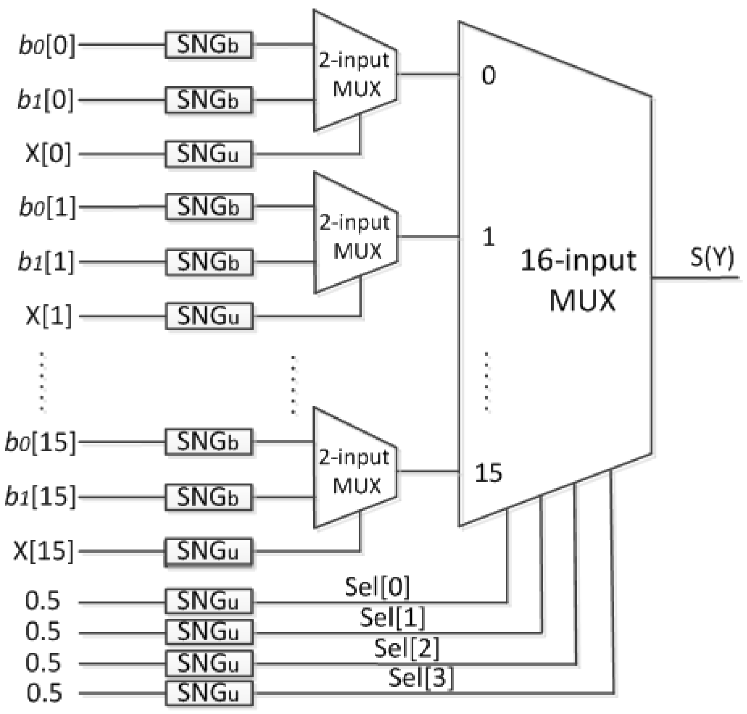


Figure 4.9: Architecture of the squared L^2-norm error calculator using the Bernstein polynomial method.

Table 4.1: Circuit performance of the squared error calculators defined in Equations 4.10 and 4.20 using (a) traditional stochastic arithmetic elements and (b) the Bernstein polynomial method $N_e = 16$).

Area (um2)			Power (uW) @ Min Clock Period			Minimum Clock Period (ns)		
(a)	(b)	Ratio: (a)/(b)	(a)	(b)	Ratio: (a)/(b)	(a)	(b)	Ratio: (a)/(b)
5.246	5.129	1.02	8.74	8.36	1.05	0.05	0.05	1

Stochastic VQ Implementation Using p^{th}-law Errors

As discussed above, the Bernstein polynomial method is selected for error calculations of the p^{th}-law ($p \geq 3$) in Equation 4.12 to achieve efficiency. Based on the Bernstein polynomial calculator in Figure 4.4, the overall architecture of the p^{th}-law error calculator is shown in Figure 4.10, where $p = 3$ and $N_e = 16$ in our example. $X[i]$ represents the i^{th} element in the input vector. The Bernstein coefficients $b0[i]$, $b1[i]$, $b2[i]$ and $b3[i]$ are calculated using $b_k = \sum_{j=0}^{k} \frac{\begin{pmatrix} k \\ j \end{pmatrix}}{\begin{pmatrix} 3 \\ j \end{pmatrix}} \cdot a_k$, where a_k is the k^{th}-order coefficient of the error polynomial in Equation 4.12 without the absolute value function and $k = 0, 1, 2, 3$. For an input vector $\mathbf{X}$ with N_e elements, the input $X[i]$ ($i = 0, 1, ..., N_e - 1$) is always positive as it is an 8-bit binary value encoding a grey scale pixel. It is then converted into unipolar stochastic sequences. The Bernstein coefficients can be positive or negative, so in Figure 4.10 bipolar SNGs are used in Bernstein polynomial calculators (in Figure 4.4). $S(Y[i])$ is the stochastic output of the absolute value of the Bernstein polynomial for input $X[i]$, where the Bernstein polynomial has $(p+1)$ terms for errors measured by the p^{th}-law. In general, a p^{th}-law error is a sum of N_e Bernstein polynomials. Therefore,all the outputs of N_e Bernstein polynomials should be added up using an N_e-input multiplexer. The output of the error calculator $S(Y)$ in Figure 4.10 is a stochastic sequence to be compared with other errors in the stochastic comparison tree shown in Figure 4.7.

4.3.4 Index Storage and Delivery

A source vector is encoded by the index of the codevector that produces the minimum error among all the calculated ones. The comparison result comes naturally as stochastic streams that represent probabilities instead of deterministic Boolean values. Therefore the comparison streams have to be converted to binary numbers by counters, which would add cost. To avoid this problem, the index is embedded in the last few bits of the stochastic sequences as a binary-encoded value, as shown in Figure 4.11. Initially a error calculator is used to obtain the stochastic sequence for the error of the k^{th} codevector at the input port, and then it is necessary to label this stochastic error with index k. Thus the last few bits in the stochastic error sequence are replaced (the last six bits that are shaded in the example in Figure 4.11) with the binary number that represents k (the grey bits in Figure 4.11). The rest of the bits (the bits represented by the white squares in Figure 4.11) in the bit stream are left unchanged.

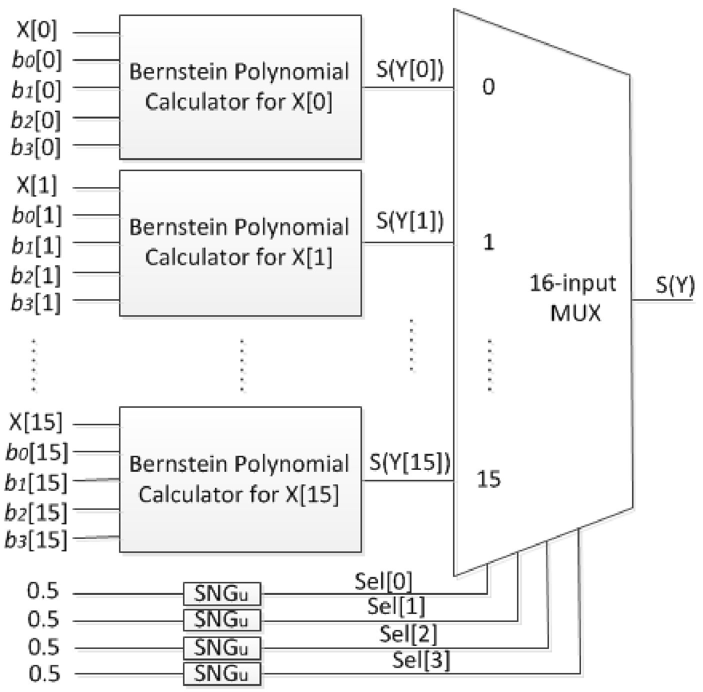


Figure 4.10: Architecture of the p^{th}-law error calculator for $p = 3$ using the Bernstein polynomial calculator in Figure 4.4.

If the sequences are long enough, giving up the last few bits will have little effect on the stochastic value. For most of the codewords, the stochastic comparator will rapidly converge to select one of the inputs as the output after an initial period of instability. So one can rely the index being delivered correctly especially when the sequence is long enough. Only one counter is prepared at the last stage to extract the index from the stochastic sequence. Registers for index storage and counters used as the stochastic-to-binary converter for every comparator are saved to reduce hardware cost. A shorter delay also results as no extra time is needed to process the index, which is extracted easily from the output bit stream.

4.3.5 Error Analysis

A mathematical analysis is given to show the validity of the index storage and delivery method. Assume that the last M bits are used to store the index in an N_s-bit stochastic sequence (see Figure 4.12). The index of the smaller inputs between P_x and P_y must be safely passed on through the stochastic comparator shown in Figure 2.6. This requires that the last M bits in the stochastic sequence encoding P_{S2} correctly indicate the result of comparing stochastic numbers P_x and P_y. As P_{S2} is the output of the stochastic *tanh* function, we consider the state transition diagram of the stochastic *tanh* function where N_{st} is the

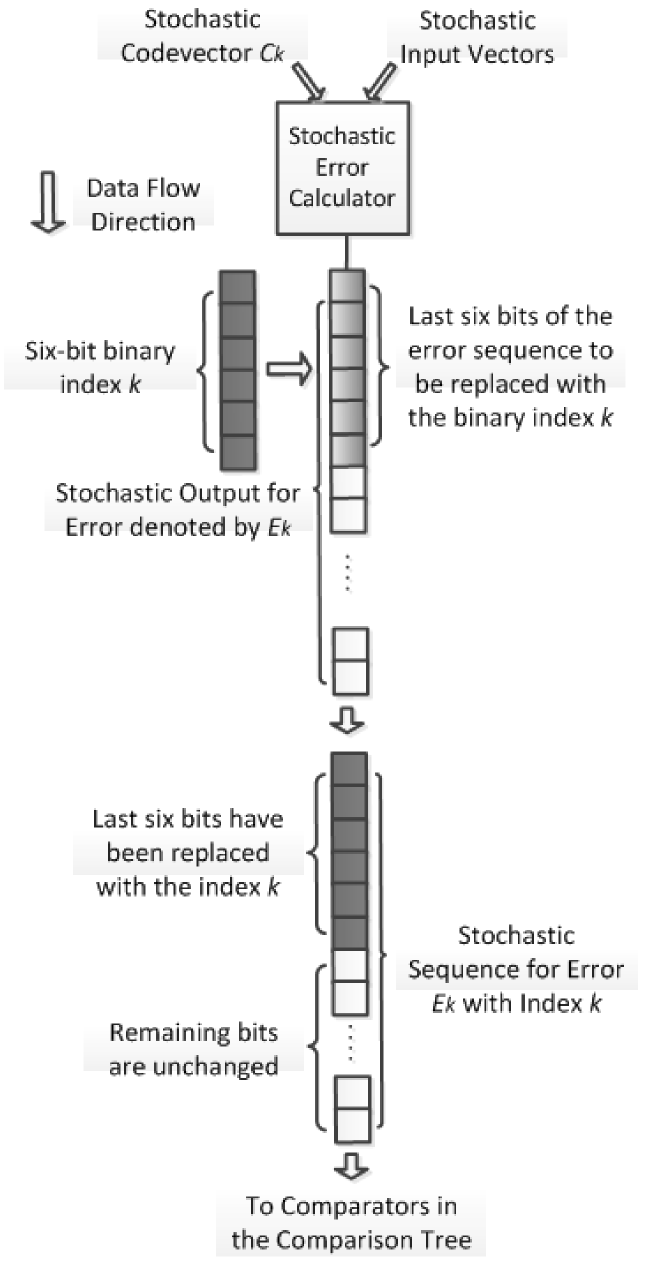

Stochastic Codevector C_k
Stochastic Input Vectors
Stochastic Error Calculator
Data Flow Direction
Six-bit binary index k
Last six bits of the error sequence to be replaced with the binary index k
Stochastic Output for Error denoted by E_k
Last six bits have been replaced with the index k
Stochastic Sequence for Error E_k with Index k
Remaining bits are unchanged
To Comparators in the Comparison Tree

Figure 4.11: Embedding a 6-bit binary index into the stochastic error bit stream.

number of states in the FSM (see Figure 2.4) and $N_s >> M$ (see Figure 4.12). The goal is to ensure with high probability that the last M bits in the stochastic sequence encoding P_{S2} in Figure 2.6 are stable at 1 (or 0) for the comparison result $P_x \geq P_y$ (or $P_x \leq P_y$). Now the conditional probability $P\{Last\ M\ bits\ in\ P_{S2}\ are\ all\ 1's \mid P_x \geq P_y\}$ should be calculated, which would be similar to calculating $P\{Last\ M\ bits\ in\ P_{S2}\ are\ all\ 0's \mid P_x \leq P_y\}$.

To focus on the index embedded in the last M bits of a stochastic sequence, the state transitions after $(N_s - M)$ transitions are considered in the diagram (see Figure 2.4). The remaining M states produce the last M bits of the stochastic sequence P_{S2}, which selects the M-bit index of the smaller one between P_x and P_y. Suppose that the current state is denoted by CS_i, where the subscript i ($i = 0, 1, ..., M-1$) corresponds to the i^{th} index storage bit. We consider all possible values of P_{S1}, which is the input of the *tanh* function, to analyze the output P_{S2}. The computation steps are as follows.

1) It can be seen that $S_{\frac{N_{st}}{2}}$ is the central state in the state transition diagram in Fig. 16. $S_{\frac{N_{st}}{2}+k}$ represents the k^{th} state to the right of $S_{\frac{N_{st}}{2}}$, where k ($k = 0, 1, 2, ...$) is an integer index. $S_{\frac{N_{st}}{2}+k}$ ($k \geq M$) are considered "safe" initial states because the next M transitions will remain in the right half of the state transition diagram no matter what the inputs (M-bit embedded index shown in Fig. 16) are. Hence, if $CS_0 = S_{\frac{N_{st}}{2}+k}$ and ($k \geq M$), the last M bits in P_{s2} will be held at '1'. Assume that the two errors P_x and P_y encoded by the stochastic sequences are evenly distributed between 0 and 1. The probability that $CS_0 = S_{\frac{N_{st}}{2}+k}$ and ($k \geq M$) is

$$P_1 = 1 - \frac{2M}{N_s} + \frac{M^2}{N_s^2} > \left(1 - \frac{2M}{N_s}\right), \tag{4.21}$$

which is proved in detail in the appendix.

2) If $CS_0 = S_{\frac{N_{st}}{2}+k}$ and ($0 \leq k < M$), the next state CS_i ($i = 0, 1, ..., M-1$) will possibly cross the boundary from outputting 1's to outputting 0's, so that it fails to hold the value '1'. Let the probability of not crossing the boundary be

$$P_2 = \sum_{k=0}^{M-1} P_{2,k}, \tag{4.22}$$

where $P_{2,k}$ denotes the probability that the output of CS_i is held at '1' for any $i \in 0, 1, , M-1$ provided that $CS_0 = S_{\frac{N_{st}}{2}+k}$ ($0 \leq k < M$). According to its definition, $P_{2,k}$ can be obtained by

$$P_{2,k} = P\{CS_0 = S_{\frac{N_{st}}{2}+k}\} \cdot P\{The\ output\ of\ CS_i\ is\ held\ at\ 1 \mid CS_0 = S_{\frac{N_{st}}{2}+k}\}, \tag{4.23}$$

Where $k = 0, 1, ..., M-1$. According to Equation B.8 in the appendix, the probability that the initial state is $S_{\frac{N_{st}}{2}+k}$ can be calculated as

$$P\{CS_0 = S_{\frac{N_{st}}{2}+k}\} \approx \frac{2}{N_s}, \tag{4.24}$$

for any k ($k = 0, 1, ..., M-1$).

It is rather complicated to calculate the probability $P\{The\ output\ of\ CS_i\ is\ held\ at\ 1|\ CS_0 = S_{\frac{N_{st}}{2}+k}\}$ for every value of k ($k = 0, 1, ..., M-1$). Instead, we can derive a lower bound of the probability by simplifying the problem. If the state always transitions to its right neighbor until it reaches the nearest "safe" state $S_{\frac{N_{st}}{2}+M}$, it is guaranteed that the output of the state machine is held at '1'. However, this assumption is too pessimistic as there are many other cases where the state transitions back and forth, but still produces outputs of 1's. We assume that the last M bits of the stochastic sequence P_{s1} (the input of the state machine) have the same probability of being '1' or '0'. Therefore, the probability that the state transitions to its left or right is 1/2. As it takes $(M-k)$ steps to reach the nearest "safe" state $S_{\frac{N_{st}}{2}+M}$ from the initial state $S_{\frac{N_{st}}{2}+k}$, the probability that the output of the state machine is held at '1' must be greater than $(\frac{1}{2})^{M-k}$, i.e.

$$P\{The\ output\ of\ CS_i\ is\ held\ at\ 1|\ CS_0 = S_{\frac{N_{st}}{2}+k}\} > (\frac{1}{2})^{M-k}. \tag{4.25}$$

Therefore, the lower bound of the probability $P_{2,k}$ defined in Equation 4.23 is determined as

$$P_{2,k} > \frac{2}{N_s} \cdot (\frac{1}{2})^{M-k}. \tag{4.26}$$

Then we can further obtain a lower bound of P_2 in Equation 4.22 by

$$P_2 > \frac{2}{N_s} \sum_{k=0}^{M-1} (\frac{1}{2})^{M-k} \approx \frac{1}{N_s}. \tag{4.27}$$

3) Considering 1) and 2), we obtain the probability that the last M bits are held at '1' as

$$P = P_1 + P_2 > 1 - \frac{2M}{N_s} + \frac{1}{N_s} = 1 - \frac{2M-1}{N_s}. \tag{4.28}$$

4) In this case, $N_s = 2048$ and $M = 8$ are selected as a typical setting. Therefore, the probability in Equation 4.28 can be calculated as

$$P > 1 - \frac{2M-1}{N_s} \approx 99.27\%. \tag{4.29}$$

The result shows that the accuracy is above 99.27% despite the approximation used to estimate the probability. This means that we can safely use the index storage method and count on it for simpler, faster and reliable computation.

4.4 Simulation and Discussion

4.4.1 Required Sequence Length

The loss in image quality caused by the compression can be measured as the total power in the error between the original image and the image reconstructed from the output of a

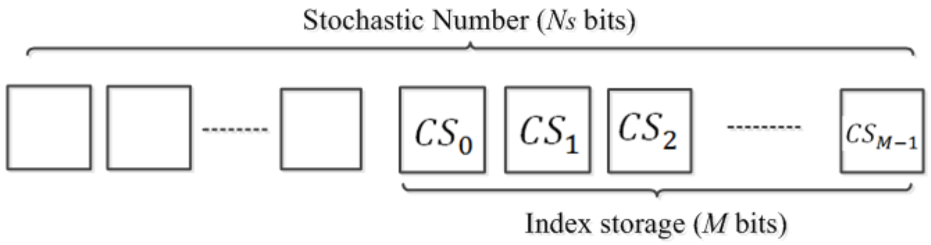


Figure 4.12: The embedded index in stochastic sequences and state transitions in *tanh* function.

VQ encoder. Assume that the image contains N_p pixels. Let the pixel values in the original image be denoted by $P_{Oi}\,(i = 0, 1, ..., N_p - 1)$, while the pixel values of the reconstructed image after compression are denoted by $P_{Ci}\,(i = 0, 1, , N_p - 1)$. The APE of the loss of quality is defined as

$$APE = \sqrt{\frac{1}{N_p}\sum_{i=0}^{N_p-1}(P_{Oi} - P_{Ci})^2}, \tag{4.30}$$

where $N_p = 90,000$.

Table 4.2: The APE values at different sequence lengths for the stochastic VQ.

Sequence Length (bits)	256	512	1024	2048	4096	9192
L^1 norm	30.4	19.9	10.9	7.3	7.1	7.1
squared L^2 norm	25.7	18.6	9.5	6.0	6.0	6.0
3^{rd}-law	25.1	17.2	8.8	6.0	5.9	5.9

Table 4.3: The APE values at different bit resolutions for the conventional binary VQ.

Resolution (Bits)	4	6	7	8	9	10
L1 norm	33.1	18.3	11.0	7.8	7.8	7.8
squared L2 norm	26.7	16.9	7.2	6.0	6.1	6.0
3^{rd}-law	24.6	14.7	6.2	5.8	5.7	5.7

For an image with 90,000 pixels (i.e. $N_p = 90,000$), stochastic VQ is implemented using the L^1 norm, squared L^2 norm and 3^{rd}-law error calculations, respectively, assuming the overall architecture in Figure 4.7. Various sequence lengths are investigated and the corresponding APEs are compared in Table 4.2. The APE decreases as the stochastic sequence length grows, as expected. Conventional binary implementations of VQ with the same experimental parameters are also simulated for comparison. The APE values are reported in Table 4.3 as a comparison with the stochastic results in Table 4.2. It can be seen that the 8-bit resolution results in low APE values and that higher bit resolutions do not improve the APE significantly. Compared with the results in Table 4.2, the stochastic implementation using 2048 bits subjectively matches the 8-bit binary conventional implementation as they show a similar APE performance. Similarly, roughly equivalent performance is found between a 6-bit binary design and a stochastic design that uses 512-bit sequences. It is then decided to compare the circuit performances of the implementations that show a similar performance in terms of accuracy.

The sequence length implies an output latency that limits the performance of stochastic implementations. Vector quantization, however, is already a lossy data compression method. We can in some cases accept quality deterioration to reduce the latency. In fact, the quality of the compression relies heavily on the comparison results of the errors. Hence, the accurate ranking of the errors is more important than the values of the errors. Finally, in streaming media applications, latency is often not an issue as it only affects the initial delay.

4.4.2 Functional Simulation Using Matlab

Stochastic vector quantization using different measures of errors were simulated using Matlab. The classic Lena image in Figure 4.13 was used as the input source and the LBG algorithm was used to generate a codebook. Figures 4.14, 4.15 and 4.16 show the stochastic VQ simulation results using L^1 norm, squared L^2 norm and 3^{rd}-law errors, respectively. The input is a 300×300 pixel grey-scale image. Each pixel is represented by an 8-bit binary number. After using stochastic vector quantization to compress the original image, the image is re-constructed using codebook look-up and displayed for visual assessment. The image has 5625 input vectors, and each vector comprises 16 unsigned 8-bit pixel values. 2048 bits are used in a stochastic representation, so it takes 2048 clock cycles to finish one round of calculation. The stochastic representation of 2048 bits can be generated by an 11-bit LFSR using similar circuits shown in Figures 2.7 and 2.8. To encode the 5625 input vectors in a fully-parallel architecture, a total of 5625 independent processor units are required and one unit includes 256 error calculators and a 256-input comparison tree.

The output images in Figures 4.14,4.15 and 4.16 illustrate the progressive quality feature of stochastic computing. The reconstructed image after stochastic compression for 256^{th}, 512^{th}, 1024^{th} and 2048^{th} clock cycles are shown for the three error measures. However the reconstructed output images are vague and only show a rough outline of the original image after compression using 256 clock cycles. The reconstructed image becomes a clearer and more accurate reproduction as the stochastic encoding time increases. Because 11-bit LFSRs are used to generate the stochastic sequences, the sequences repeat every 2048 cycles. Thus the image quality stops improving after 2048 clock cycles.

4.4.3 Circuit Performances

Following the results in Tables 4.2 and 4.3, three pairs of implementations are compared: (a) an 8-bit binary implementation with the stochastic implementation using 2048-bit sequences, (b) a 7-bit binary implementation with the stochastic implementation using 1024-

Figure 4.13: The classic grey-scale Lena image is used in the experiments.

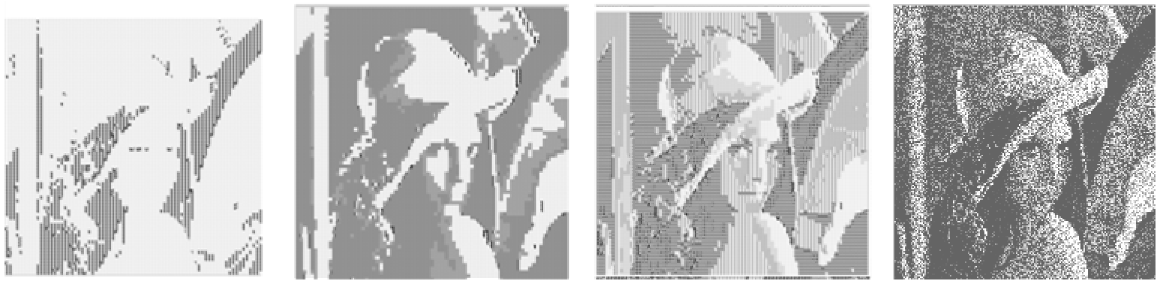


Figure 4.14: The progressive improvement of image quality using L^1-norm stochastic VQ after 256, 512, 1024 and 2f048 clock cycles.

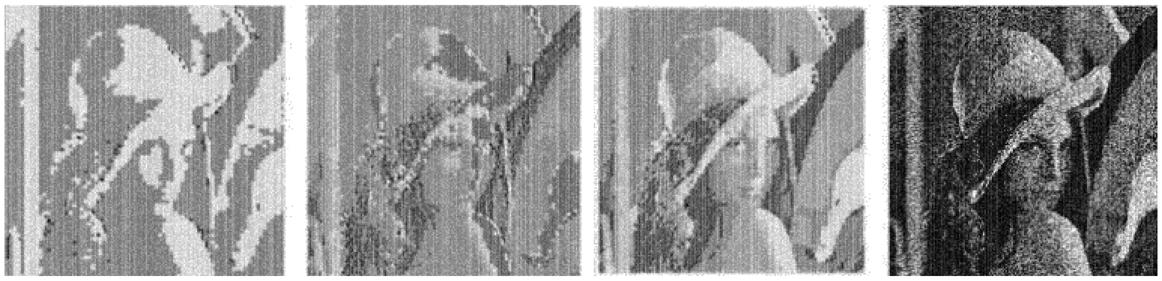


Figure 4.15: The progressive improvement of image quality using squared L^2-norm stochastic VQ after 256, 512, 1024 and 2048 clock cycles.

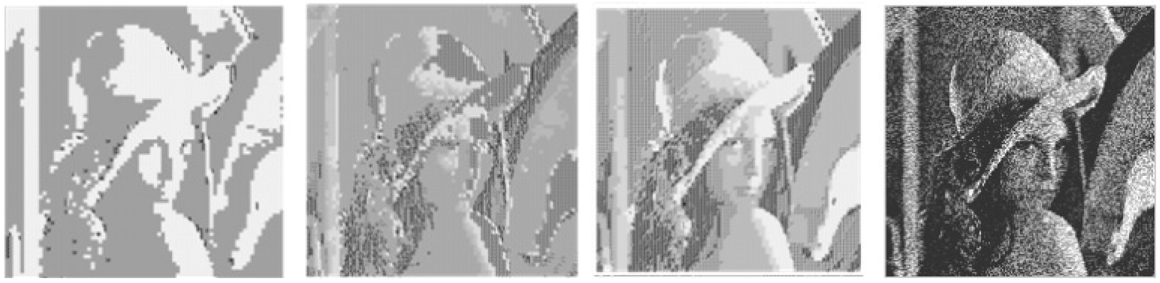


Figure 4.16: The progressive improvement of image quality using the 3^{rd}-law stochastic VQ after 256, 512, 1024 and 2048 clock cycles.

bit sequences and (c) a lower quality processing implementation using the 6-bit binary and the 512-bit stochastic designs. The hardware area, power consumption and delay comparisons are shown in Tables 4.4, 4.5 and 4.6 for L^1-norm, squared L^2-norm and 3^{rd}-law implementations, respectively. The stochastic circuits are built according to the architecture in Figure 4.7. The designs of the error calculators are shown in Figures 4.5, 4.9 and 4.10. By using the Synopsys design compiler the fastest clock is obtained that still meets the timing requirements. Then the power consumption and silicon area are obtained for the fastest clock frequency. Note that the auxiliary circuits such as stochastic number generators (implemented by LFSRs) and counters are all included.

As shown in Tables 4.4, 4.5 and 4.6, the stochastic circuits have significantly lower hardware cost. Stochastic implementations only cost roughly 1% of the hardware of binary implementations. This also leads to savings in power consumption. The time required for an encoding operation is also an important measurement to calculate the total energy, and it is determined by the product of the clock period and the stochastic sequence length. Because the structure of stochastic circuits is simpler, a shorter critical path delay is expected. This is reflected in the columns showing that the minimum stochastic clock periods are smaller than the minimum binary clock periods.

The TPA and the EPO are two important metrics. In Table 4.4 for the L^1-norm, the stochastic approach shows significant advantages over the binary approach in terms of the area cost, power consumption and delay. When long sequences such as 2048 bits are considered, the ratio of stochastic over binary energy per operation is about 5.43, and the ratio of the throughputs per area is approximately 0.38. Therefore, the stochastic approach using 2048-bit sequences underperforms the conventional binary approach using 8-bit resolution. However, if some loss in quality is acceptable in the application, the stochastic implementation using 512-bit sequences shows only 2.35 times the energy cost per operation and 2.60 times throughput per area in only 1.5% the total area compared to a 6-bit binary implementation. It can be seen that the stochastic implementation using 1024-bit sequences shows similar performance compared to the 7-bit binary implementation in terms of TPA. The stochastic VQ is thus not competitive for 7-bit or higher bit resolutions in terms of the TPA performance.

In Tables 4.5 and 4.6, the stochastic VQ implementations for the squared L^2-norm and the 3^{rd}-law errors are compared with the conventional binary implementations. In general, the squared L^2-norm and the 3^{rd}-law implementations use more hardware and consume more energy as the computational complexity increases from the L^1-norm implementation. However, the implementation areas for the stochastic implementations are still less than 1.5% of the binary designs. The TPAs of the 3^{rd}rd-law VQ implementations are much smaller than those of L^1-norm and squared L^2-norm VQ implementations. However, the squared L^2-norm and the 3^{rd}-law benefit from more accurate results compared with the L^1 norm. For the same stochastic sequence length, the reconstructed images using the squared

L^2-norm and the 3^{rd}-law have higher fidelity as they show smaller average penalized error (APE) than that using the L^1 norm, which is shown in Table 4.2. The 3^{rd}-law VQ takes advantage of the Bernstein polynomial method to build the error calculator based on high order polynomials. It shows the best compression quality compared with the other two implementations.

When high-quality (8-bit binary and 2048-bit stochastic) VQ implementations are considered, the EPO ratios of the stochastic implementation over conventional binary implementation are 9.67 and 17.85 for the squared L^2-norm and the 3^{rd}-law errors, respectively. For lower-quality (6-bit binary and 512-bit stochastic) VQ implementations, the EPO ratios become 2.20 and 3.85 for the squared L^2-norm and the 3^{rd}-law errors, respectively. With respective to the EPO, therefore, the stochastic implementations are not competitive due to the required long sequences.

For high-quality VQ compressions (using 2048-bit stochastic and 8-bit conventional binary implementations), the stochastic implementations are not advantageous over the conventional binary implementations in terms of TPA. When a lower-quality compression is acceptable (using 512-bit stochastic and 6-bit conventional binary implementations), however, the TPA ratios of the stochastic implementation over the conventional binary implementation are 2.02 and 1.16 for the squared L^2-norm and the 3^{rd}-law errors, respectively. The stochastic approach using shorter sequences loses some accuracy but saves more in terms of hardware cost and power consumption. By comparing the 1024-bit stochastic and 7-bit conventional binary VQ implementations, it can be seen that the stochastic approach could be competitive for resolutions below 7 bits in terms of TPA.

Stochastic VQ has the flexibility to easily adapt to poor communication channels where lower compression quality is preferred. Using L^1 norm, for example, two input images can be compressed using the 2048-bit stochastic VQ implementation and achieve the same compression quality as the 1024-bit stochastic VQ implementation. In this way, only the 2048-bit stochastic VQ implementation is needed instead of two copies of the 1024-bit stochastic VQ circuit, further saving 32.3% of the hardware area.

Table 4.4: Circuit performance of L^1-norm vector quantization with three compression qualities: (a) 8-bit binary (B) vs. 2048-bit stochastic (S), (b) 7-bit binary (B) vs. 1024-bit stochastic (S) and (c) 6-bit binary (B) vs. 512-bit stochastic (S).

	Area (μm^2)			Power (mW) @ Min Clock Period			Minimum Clock Period (ns)		
	B	S	Ratio: S/B	B	S	Ratio: S/B	B	S	Ratio: S/B
(a)	93294	1358	0.015	107.56	3.22	0.03	2.28	0.2	0.09
(b)	86231	1003	0.012	81.3	2.86	0.04	2.26	0.2	0.09
(c)	79177	641	0.008	50.09	2.47	0.05	2.23	0.21	0.09

	Energy per Operation (pJ/Operation)			Throughput per Area ($1/(\mu m^2 \cdot s)$)			Required Sequence Length (bits)		
	B	S	Ratio: S/B	B	S	Ratio: S/B	B	S	Ratio: S/B
(a)	243	1320	5.43	4743	1797	0.38	N/A	2048	N/A
(b)	184	586	3.19	5131	4869	0.95	N/A	1024	N/A
(c)	113	266	2.35	5588	14519	2.6	N/A	512	N/A

Table 4.5: Circuit performance of squared L^2-norm vector quantization with three compression qualities: (a) 8-bit binary (B) vs. 2048-bit stochastic (S), (b) 7-bit binary (B) vs. 1024-bit stochastic (S) and (c) 6-bit binary (B) vs. 512-bit stochastic (S).

	Area (μm^2)			Power (mW) @ Min Clock Period			Minimum Clock Period (ns)		
	B	S	Ratio: S/B	B	S	Ratio: S/B	B	S	Ratio: S/B
(a)	113847	1588	0.014	61.48	3.17	0.052	2.29	0.21	0.09
(b)	99631	1264	0.013	56.3	2.79	0.05	2.26	0.2	0.09
(c)	89992	972	0.011	50.09	2.39	0.048	2.23	0.2	0.09
	Energy per Operation (pJ/Operation)			Throughput per Area ($1/(\mu m^2 \cdot s)$)			Required Sequence Length (bits)		
	B	S	Ratio: S/B	B	S	Ratio: S/B	B	S	Ratio: S/B
(a)	141	136	9.67	3836	1464	0.38	N/A	2048	N/A
(b)	127	572	4.49	4441	3864	0.87	N/A	1024	N/A
(c)	111	244	2.2	4983	10047	2.02	N/A	512	N/A

Table 4.6: Circuit performance of 3^{rd}-law vector quantization with three compression qualities: (a) 8-bit binary (B) vs. 2048-bit stochastic (S), (b) 7-bit binary (B) vs. 1024-bit stochastic (S) and (c) 6-bit binary (B) vs. 512-bit stochastic (S).

	Area (μm^2)			Power (mW) @ Min Clock Period			Minimum Clock Period (ns)		
	B	S	Ratio: S/B	B	S	Ratio: S/B	B	S	Ratio: S/B
(a)	256261	3772	0.014	183.2	12.86	0.07	4.43	0.55	0.12
(b)	247439	3481	0.014	179.6	11.71	0.07	2.26	0.2	0.09
(c)	238261	3251	0.013	175.7	10.72	0.06	4.38	0.54	0.12
	Energy per Operation (pJ/Operation)			Throughput per Area ($1/(\mu m^2 \cdot s)$)			Required Sequence Length (bits)		
	B	S	Ratio: S/B	B	S	Ratio: S/B	B	S	Ratio: S/B
(a)	811	14485	17.85	881	235	0.27	N/A	2048	N/A
(b)	406	2398	5.91	1788	1403	0.78	N/A	1024	N/A
(c)	769	2963	3.85	958	1112	1.16	N/A	512	N/A

4.5 Summary

In this chapter, stochastic circuits are designed to implement the L^1-norm, squared L^2-norm and p^{th}-law ($p = 3$ is used as an example)-based vector quantization (VQ). Finite state machine-based stochastic arithmetic elements and the Bernstein polynomials are used to build error calculators and stochastic comparison trees. By embedding the codevector indexes into the last few bits of the stochastic error sequences, costly counters are saved to reduce hardware cost. Various sequence lengths are considered in the stochastic vector quantization and the compression quality was assessed using average penalized error (APE) for a grey-scale image.

Implementations using a codebook of 256 codevectors with similar compression qualities are then compared with respect to APE: (a) an 8-bit binary implementation with the stochastic implementation using 2048-bit sequences, (b) a 7-bit binary implementation with the stochastic implementation using 1024-bit sequences and (c) a lower quality processing implementation using the 6-bit binary and the 512-bit stochastic designs. Due to the compact stochastic arithmetic elements and an efficient index storage approach, the area advantage of the stochastic VQ implementations is significant. The implementation areas for the stochastic circuits are no more than 1.5% of the fully parallel binary implementations. Our results show that the stochastic VQ underperforms the conventional binary VQ in terms of energy per operation. However, the stochastic VQ can be efficient in terms of TPA when the implementation (c) with an acceptable lower quality is considered. For the three error measures, the TPA of the 512-bit stochastic implementation is shown to be 1.16, 2.02 and 2.60 times as large as that of the 6-bit binary implementations with a similar compression quality. It is found that the 7-bit binary implementation and 1024-bit stochastic implementation shows higher performance in terms of TPA, especially for the L^1-norm error. Thus, stochastic VQ will not be competitive for bit resolutions higher than 7 bits.

Applications can benefit from stochastic VQ where the extra energy and the higher latency are not significant factors. The constant transmission latency of stochastic computing may not be a problem for many streaming media applications. The stochastic design can be interesting because a small silicon area can encode many multimedia streams in the same area as only one conventional binary encoder/decoder. Moreover, a stochastic VQ implementation for high compression quality can be easily applied to obtain lower compression quality when encoding multiple input images. Compared to lower-quality stochastic VQ implementations, the area cost can further be significantly reduced.

The inherent progressive quality of the stochastic VQ design is a potentially useful feature. The stochastic VQ encoder is tested on the grey-scale image for different stochastic sequence lengths and hence different run-times, which indicate different encoding qualities. To have a better accuracy, one simply waits for more clock cycles during the computation. The number of clock cycles can be readily reduced to save power at the cost of a lower image quality.

Chapter 5

Conclusions and Future Work

5.1 Conclusions of This Thesis

In this thesis, we investigated the use of stochastic arithmetic elements. Stochastic circuits are known for their smaller area compared to corresponding conventional binary circuits. However, the cost due to auxiliary circuits must be considered such as SNGs and counters that are used to convert binary numbers to stochastic bit streams and vice versa. Besides, stochastic computing may suffer from random fluctuations and its inherent variation must be taken into account. The experimental results have proved that stochastic arithmetic elements without the auxiliary circuits can be advantageous in terms of area and power consumption. However, stochastic computing is not competitive in terms of TPA or EPO for bit resolutions above 4 bits if the auxiliary circuits are included. A SOP function is used as an example to show how stochastic computing performs if various stochastic arithmetic elements are combined to form a larger computing system. A computing system for real-world applications, however, usually consists of many individual arithmetic operations. FIR filters and VQ encoders are examples of such computing systems and they are selected as two applications for a detailed comparative study.

A stochastic HWA design and a MWA design are proposed to implement stochastic FIR filters. The proposed stochastic designs show an improved performance, a smaller circuit area and lower power consumption, compared to the conventional stochastic design. Compared to binary FIR filter circuits, the proposed stochastic designs have a significant advantage in circuit area, especially at higher resolutions. With respect to the performance metrics of TPA and EPO, the stochastic design does not show any advantages over its binary counterpart when the auxiliary circuits are included. This is because long stochastic sequences must be used to achieve the same filtering performance as a binary circuit. Therefore, stochastic computing can be very time-consuming. A shorter stochastic sequence, however, can be used to trade off some accuracy. The stochastic circuit shows a graceful degradation in performance compared to the binary design as stochastic computing can save a lot more energy than the conventional binary approach when their performance reductions are identical.

FIR filters are used as an example for stochastic error analysis. Random fluctuations and quantization errors are considered in FIR filter functions. The relation between the required sequence length and the bit resolution can be obtained. This approach can be extended for other stochastic applications to determine the sequence length that are required.

The stochastic computing is also used to implement VQ based on the L^1-norm, squared L^2-norm and p^{th}-law ($p = 3$ is taken as an example). Non-combinational stochastic methods such as the FSM-based arithmetic elements and the Bernstein polynomial approach are used to build error calculators and stochastic comparison trees. By embedding the corresponding indexes into the last few bits of stochastic errors, costly registers are avoided to keep and deliver the indexes. Various sequence lengths are considered in the stochastic vector quantization and the compression quality was assessed using APE for a grey-scale image. Although the stochastic VQ is not competitive against the conventional binary VQ in terms of TPA or EPO, it can be efficient in terms of TPA with an acceptable lower quality. The fixed transmission latency of stochastic is thus not a problem for many streaming media applications. The stochastic design can be competitive because a small silicon area could encode many multimedia streams in the same area required by only one conventional binary encoder/decoder. Moreover, a stochastic VQ implementation for high compression can be easily applied to obtain lower compression quality when encoding multiple input images. Compared to lower-quality stochastic VQ implementations, the area cost can be significantly reduced. The inherent progressive quality of the stochastic VQ design is another potentially useful feature. It has been demonstrated that the stochastic VQ can encode the grey-scale image for different stochastic sequence lengths and hence different run-times, which correspond to different encoding qualities. Better accuracy can easily be achieved by running more clock cycles, while power can be also saved simply by using part of the stochastic sequence at the cost of a lower image quality.

According to the experimental and analytical results in this thesis, stochastic computing may not be recommended for those applications where accuracy is the priority. Sequence length in stochastic computing can be very long to match the performance of a corresponding conventional binary implementation. However, stochastic computing can be efficient where approximation is allowed or even necessary such as in a lossy data compression. Careful observations and designs should be adopted so that auxiliary circuits can be avoided as much as possible to save hardware cost. Features of stochastic computing such as the graceful degradation can not be achieved using the conventional binary approach. Performance in terms of accuracy can be improved gradually by simply waiting for more clock cycles, which is another useful and unique feature of stochastic computing. Note that this thesis is based on current CMOS technologies. There have been an increasing interests in novel techniques such as the spintronic memristors [31], which could be in favor of the stochastic approach. This thesis provides an example of investigating the issues in stochastic computing. Similar study on next-generation devices such as spintronic memristors can be an interesting topic for future work.

5.2 Recommendations for Future Work

Although some issues in terms of efficiency are discussed in this thesis for stochastic computing, more aspects in this field should be explored and fully understood. Further research is recommended as follows.

- In this thesis, power consumption is roughly evaluated using the Synopsys Design Compiler. Power estimations using activities from simulations with real data are needed to obtain results with better accuracy.

- The functions of FSM-based stochastic elements are usually affected by the number of states N_{st}. To explore this relationship, The appropriate number of states N_{st} can be written as a function of the sequence length N_s.

- Stochastic computing can be application-dependent. Real-world applications need to be implemented and evaluated to determine the feasibility of designing stochastic computing systems. The negative results for stochastic FIR filters and the more promising results for stochastic VQ encoders indicate that stochastic computing can be useful but should be carefully evaluated and tuned for specific systematic and environmental variables. The success of stochastic computing in lower quality VQ compression demonstrates that applications with tolerance of performance loss may benefit from the stochastic approach.

- Stochastic computing is inherently based on probabilities and statistical theory. One possible direction may be to apply stochastic computing to deal with mathematical problems, e.g. machine learning algorithms, artificial intelligence, etc.

- There have been great efforts in the research of stochastic number representations and generations. Remarkable ideas have been proposed to improve the accuracy in stochastic representation. However, stochastic number generation is still the bottleneck that restricts the efficiency of an overall stochastic computing system. The problem of designing SNGs to compromise accuracy and hardware cost may be another good viewpoint of research.

- The stochastic sequences of the inputs to a multiplexer can be correlated when the selection signals are statistically independent. Hence the LFSR can be shared for these SNGs. A poor pseudo-random number generator (e.g. a digital counter) can replace some of the LFSRs to reduce hardware cost. When a circuit has multiple layers of stochastic processing, however, the correlation in inputs will lead to biased or inaccurate results.

Bibliography

[1] B. Gaines, "Stochastic computing systems," in *Advances in information systems science.* Springer, 1969, pp. 37–172.

[2] M. Pedram, "Design technologies for low power vlsi," *Encyclopedia of Computer Science and Technology*, vol. 36, pp. 73–96, 1997.

[3] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI.* ACM, 2015, pp. 343–348.

[4] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Proceedings of the conference on Design, Automation & Test in Europe.* European Design and Automation Association, 2014, pp. 95:1–95:4.

[5] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Test Symposium (ETS), 2013 18th IEEE European.* IEEE, 2013, pp. 1–6.

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

[7] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *45th ACM/IEEE Design Automation Conference.* IEEE, 2008, pp. 648–653.

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

[9] ——, "The influence of spatial and transient circuit variations on energy and accuracy in stochastic computing circuits," in *International workshop on designing with uncertainty: opportunities and challenges*, 2014.

[10] 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*, vol. 22, no. 3, pp. 449–462, 2014.

[11] A. Alaghi, C. Li, and J. Hayes, "Stochastic circuits for real-time image-processing applications," in *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–6.

[12] P. Li, "Analysis, design, and logic synthesis of finite-state machine-based stochastic computing," Ph.D. dissertation, 2013.

[13] Y. N. Chang and K. K. Parhi, "Architectures for digital filters using stochastic computing," in *International Conference on Acoustic, Speech and Signal Processing (ICASSP).* IEEE, 2013, pp. 2697–2701.

[14] A. Alaghi and J. P. Hayes, “On the functions realized by stochastic computing circuits,” *Great Lakes Symposium on VLSI*, 2015.

[15] Y. Ding, Y. Wu, and W. Qian, “Generating multiple correlated probabilities for mux-based stochastic computing architecture,” in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design.* IEEE Press, 2014, pp. 519–526.

[16] B. Moons and M. Verhelst, “Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 4, pp. 475–486, Dec 2014.

[17] P. Zhu, J. Han, L. Liu, and M. Zuo, “A stochastic approach for the analysis of fault trees with priority and gates,” *IEEE Transactions on Reliability*, vol. 63, no. 2, pp. 480–494, June 2014.

[18] J. Han, H. Chen, J. Liang, P. Zhu, and F. Lombardi, “A stochastic computational approach for accurate and efficient reliability evaluation,” *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1336–1350, June 2014.

[19] A. Alaghi and J. P. Hayes, “Survey of stochastic computing,” *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s, pp. 92:1–92:19, 2013.

[20] N. M. Nasrabadi and R. A. King, “Image coding using vector quantization: A review,” *IEEE Transactions on Communications*, vol. 36, no. 8, pp. 957–971, 1988.

[21] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions: with formulas, graphs, and mathematical tables.* Courier Corporation, 1964, no. 55.

[22] J. A. Waicukauski, E. Lindbloom, E. B. Eichelberger, and O. P. Forlenza, “A method for generating weighted random test pattern,” *IBM Journal of Research and Development*, vol. 33, no. 2, pp. 149–161, 1989.

[23] B. Widrow and I. Kollár, “Quantization noise: Roundoff error in digital computation,” *Signal Processing, Control, and Communications*, pp. 485–528, 2008.

[24] C. M. Grinstead and J. L. Snell, *Introduction to probability.* American Mathematical Soc., 1997.

[25] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, “An architecture for fault-tolerant computation with stochastic logic,” *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.

[26] D. W. Knapp, *Behavioral synthesis: digital system design using the synopsys behavioral compiler.* Prentice-Hall, Inc., 1996.

[27] N. Saraf, K. Bazargan, D. J. Lilja, and M. D. Riedel, “Iir filters using stochastic arithmetic,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014.* IEEE, 2014, pp. 1–6.

[28] Y. Linde, A. Buzo, and R. M. Gray, “An algorithm for vector quantizer design,” *IEEE Transactions on Communications*, vol. 28, no. 1, pp. 84–95, 1980.

[29] F. K. Soong, A. E. Rosenberg, B.-H. Juang, and L. R. Rabiner, “Report: A vector quantization approach to speaker recognition,” *AT&T technical journal*, vol. 66, no. 2, pp. 14–26, 1987.

[30] J. Makhoul, S. Roucos, and H. Gish, “Vector quantization in speech coding,” *Proceedings of the IEEE*, vol. 73, no. 11, pp. 1551–1588, 1985.

[31] X. Wang, Y. Chen, H. Xi, H. Li, and D. Dimitrov, “Spintronic memristor through spin-torque-induced magnetization motion,” *Electron Device Letters, IEEE*, vol. 30, no. 3, pp. 294–297, 2009.

[32] F. G. Ashby, "Multivariate probability distributions," *Multidimensional models of perception and cognition*, pp. 1–34, 1992.

Appendix A

Proof of Equations 3.58 and 3.59

In Appendix A, it is proved that Equations 3.58 and 3.59 give the mean and variance of the overall error in the conventional binary filter circuit.

To investigate how the injected errors affect the output of the FIR filters, we assume that the correct output of the conventional binary FIR filter Y is given by Equation 3.12, where H_i and X_i are the inputs and filter coefficients without injected error. With error injection, the output $Y^{(i)}$ given in Equation 3.56 is evaluated by

$$\begin{aligned} Y^{(i)} &= \sum_{i=0}^{N_f-1} (H_i + e_{Hi}^{(i)})(X_i + e_{Xi}^{(i)}) \\ &= \sum_{i=0}^{N_f-1} (H_i X_i + X_i e_{Hi}^{(i)} + H_i e_{Xi}^{(i)} + e_{Hi}^{(i)} e_{Xi}^{(i)}) \\ &= Y + \sum_{i=0}^{N_f-1} (X_i e_{Hi}^{(i)} + H_i e_{Xi}^{(i)} + e_{Hi}^{(i)} e_{Xi}^{(i)}), \end{aligned} \tag{A.1}$$

where $e_{H_i}^{(i)}$ and $e_{X_i}^{(i)}$ are statistically independent errors of Gaussian distribution for the coefficient H_i and the input X_i with mean and variance given by Equations 3.45 and 3.46 respectively. Then we have

$$E\left(e_{Hi}^{(i)}\right) \approx (1 - 2H_i)\varepsilon; \tag{A.2}$$

$$E\left(e_{Xi}^{(i)}\right) \approx (1 - 2X_i)\varepsilon; \tag{A.3}$$

$$Var\left[e_{Xi}^{(i)}\right] = Var[e_{Hi}^{(i)}] \approx \frac{1}{3}(1 - \varepsilon)\varepsilon. \tag{A.4}$$

The overall error due to error injection for the conventional binary FIR filter $e_{oc}^{(i)}$ is given by Equation 3.57. The mean of the overall error $e_{oc}^{(i)}$ is given by

$$\begin{aligned} \mathrm{E}\left(e_{oc}^{(i)}\right) &= E\left(Y' - Y\right) \\ &= E\left(\sum_{i=0}^{N_f-1} (X_i e_{Hi}^{(i)} + H_i e_{Xi}^{(i)} + e_{Hi}^{(i)} e_{Xi}^{(i)})\right) \\ &= \sum_{i=0}^{N_f-1} [X_i E\left(e_{Hi}^{(i)}\right) + H_i E\left(e_{Xi}^{(i)}\right) + E\left(e_{Hi}^{(i)} e_{Xi}^{(i)}\right)]. \end{aligned} \tag{A.5}$$

Since $e_{H_i}^{(i)}$ and $e_{X_i}^{(i)}$ are statistically independent, the mean of their product is [24]

$$E\left(e_{Hi}^{(i)}e_{Xi}^{(i)}\right) = E\left(e_{Hi}^{(i)}\right)E\left(e_{Xi}^{(i)}\right) \approx (1-2H_i)(1-2X_i)\varepsilon^2. \tag{A.6}$$

By Equations A.2, A.3 and A.4, it is easy to show that the mean of the overall error of the conventional binary circuit due to error injection is given by 3.58. The variance of the overall error $e_{oc}^{(i)}$ is given by

$$\begin{aligned} Var\left[e_{oc}^{(i)}\right] &= Var\left(\sum_{i=0}^{N_f-1}(X_i e_{Hi}^{(i)} + H_i e_{Xi}^{(i)} + e_{Hi}^{(i)}e_{Xi}^{(i)})\right) \\ &= \sum_{i=0}^{N_f-1}[X_i^2 Var(e_{Hi}^{(i)}) + H_i^2 Var(e_{Xi}^{(i)}) + Var(e_{Hi}^{(i)}e_{Xi}^{(i)})]. \end{aligned} \tag{A.7}$$

The variance of the product of the two independent variables $e_{Hi}^{(i)}$ and $e_{Xi}^{(i)}$ can be calculated by

$$\begin{aligned} Var\left(e_{Hi}^{(i)}e_{Xi}^{(i)}\right) &= Var\left(e_{Hi}^{(i)}\right)Var\left(e_{Xi}^{(i)}\right) \\ &\quad - Var\left(e_{Hi}^{(i)}\right)E^2\left(e_{Xi}^{(i)}\right) - Var\left(e_{Xi}^{(i)}\right)E^2\left(e_{Hi}^{(i)}\right). \end{aligned} \tag{A.8}$$

By Equations A.2, A.3 and A.6, Equation A.8 can be further written as

$$Var\left(e_{Hi}^{(i)}e_{Xi}^{(i)}\right) = \frac{1}{9}\varepsilon^2(1-\varepsilon)^2 - \frac{1}{3}\varepsilon^3(1-\varepsilon)\left[(1-2H_i)^2 + (1-2X_i)^2\right]. \tag{A.9}$$

Due to Equations A.4 and A.9, Equation A.7 becomes

$$Var\left[e_{oc}^{(i)}\right] = \sum_{i=0}^{N_f-1}\left\{\frac{1}{3}(H_i^2+X_i^2)\varepsilon(1-\varepsilon) + \frac{1}{9}\varepsilon^2(1-\varepsilon)^2 - \frac{1}{3}\varepsilon^3(1-\varepsilon)\left[(1-2H_i)^2+(1-2X_i)^2\right]\right\}. \tag{A.10}$$

When the injected error rate ε is small, ε^k for $k \geq 3$ in Equation A.10 can be ignored. This immediately leads to the variance given in Equation 3.59.

Appendix B

Proof of Equations 4.21

In Appendix B, Equation 4.21 is proved by calculating P_1 as the probability that $CS_0 = S_{\frac{N_s}{2}+k}$ and $k \geq M$ (see Figure 4.12). Equation 4.24 can be proved using the same method. Assume that the stochastic errors E_i $(i = 1, 2, ..., N_c)$ are evenly distributed between 0 and 1. The PDF of E_i is

$$f(e) = \begin{cases} 1, 0 \leq e \leq 1; \\ 0, otherwise. \end{cases} \tag{B.1}$$

Let $D_{i,j}$ denote the difference between two independent errors $E_i \geq E_j$, where $i, j \in 1, 2, ..., N_c$ and $i \neq j$.

$$D_{i,j} = E_i - E_j. \tag{B.2}$$

Then the problem becomes how to find the probability that $D_{i,j} \geq M/N_s$, where M is the number of storage bits and N_s is the total length of a stochastic sequence. To solve this problem, we consider the distribution of $D_{i,j}$. The PDF of $D_{i,j}$ is given by the convolution of the PDFs of E_i and $-E_j$ [32], i.e.,

$$f_{D_{i,j}}(x) = f_{E_i+(-E_j)}(x) = \int_{-\infty}^{\infty} f(e, x-e)\, de = \int_{-\infty}^{\infty} f_{E_i}(e) f_{-E_j}(x-e) de. \tag{B.3}$$

According to Equation B.1, we know

$$f_{E_i}(e) = \begin{cases} 1, 0 \leq e \leq 1; \\ 0, otherwise. \end{cases} \tag{B.4}$$

$$f_{-E_j}(e) = \begin{cases} 1, -1 \leq e \leq 0; \\ 0, otherwise. \end{cases} \tag{B.5}$$

Substituting Equations B.4 and B.5 into B.3, the PDF of $D_{i,j}$ is given by

$$f_{D_{i,j}}(x) = \begin{cases} x+1, -1 \leq x \leq 0; \\ 1-x, 0 < x \leq 1; \\ 0, otherwise. \end{cases} \tag{B.6}$$

Therefore, P_1 is given by

$$P_1 = \frac{P\left\{D_{i,j} \geq \frac{M}{N_s}\right\}}{P\left\{D_{i,j} \geq 0\right\}} = \frac{\int_{M/N_s}^{1} f_{D_{i,j}}(x)\, dx}{\int_0^1 f_{D_{i,j}}(x)\, dx} = 1 - \frac{2M}{N_s} + \frac{M^2}{N_s^2} \tag{B.7}$$

which is the same as Equation 4.21. Similarly, we can calculate $P_{2,k}$ as

$$\begin{aligned} P_{2,k} = P\{CS_0 = S_{\frac{N_{st}}{2}+k}\} &= \frac{P\left\{D_{i,j} \geq \frac{k}{N_s}\right\}}{P\left\{D_{i,j} \geq 0\right\}} - \frac{P\left\{D_{i,j} \geq \frac{k+1}{N_s}\right\}}{P\left\{D_{i,j} \geq 0\right\}} \\ &= (1 - \frac{2k}{N_s} + \frac{k^2}{N_s^2}) - (1 - \frac{2(k+1)}{N_s} + \frac{(k+1)^2}{N_s^2}) \approx \frac{2}{N_s}, \end{aligned} \tag{B.8}$$

which is the same as Equation 4.24.

Appendix C

Evaluations of Stochastic Adders and Absolute Subtractors

In Chapter 2, we investigated conventional binary and stochastic multipliers by comparing their circuit performance. In this appendix chapter, we go on to evaluate adders, absolute subtractors and the SOP operation using the similar approach.

C.1 Performance Evaluation on Stochastic Adders and Absolute Subtractors

In this section, various sequence lengths for stochastic computing are investigated to match the corresponding conventional binary designs. We then explore the circuit performance of stochastic adders and absolute subtractors.

C.1.1 Required Sequence Length for Stochastic Computing Elements Using the Simulation Method

To investigate the required sequence length for stochastic adders or absolute subtractors, conventional binary implementations of these basic arithmetic elements are discussed for resolutions ranging from 3 bits to 16 bits. The stochastic arithmetic elements are then implemented using different sizes of SNGs and hence different sequence lengths. Accuracy is measured by RMSE for the bit resolution N_b.

Table C.1 shows the resulting matching sequence lengths for increasing bit resolutions for addition and absolute subtraction. The sequence length reported in Table C.1 will then be used in the evaluation of circuit performance.

C.1.2 Circuit Performance Comparison

Adder

Multiplexers (shown in Figure C.1) can be used as scaled adders for both unipolar and bipolar encodings if the selecting input is encoding 0.5 in the unipolar encoding. In fact, weighted addition can be implemented when the selecting input stochastically encodes the desired weight. Stochastic circuit performances are investigated with or without auxiliary circuits such as SNGs and counters.

In Table C.2, it can be seen that the stochastic adders with auxiliary circuits are larger than the corresponding conventional binary adders for 4 bits and above. As expected, the stochastic adder without auxiliary circuits is much smaller than the conventional binary adders. This advantage becomes more significant for higher resolutions. Similarly, stochastic adders consume more power than the corresponding conventional binary adders if the

Table C.1: RMSE comparison of the conventional binary and stochastic implementations represented by $\mathrm{RMSE_B}$ and $\mathrm{RMSE_S}$, respectively, using various sequence lengths N_s and different bit resolutions N_b .

	Adder			Absolute Subtractor		
N_b (Bits)	$RMSE_B$ (%)	$RMSE_S$ (%)	N_s (Bits)	$RMSE_B$ (%)	$RMSE_S$ (%)	N_s (Bits)
3	3.757	3.371	16	3.220	3.332	16
4	1.999	1.719	32	1.596	1.939	32
5	1.060	9.738×10^{-1}	64	9.471×10^{-1}	8.634×10^{-1}	64
6	5.405×10^{-1}	4.646×10^{-1}	128	5.124×10^{-1}	4.928×10^{-1}	128
7	2.381×10^{-1}	2.228×10^{-1}	256	2.683×10^{-1}	2.492×10^{-1}	256
8	1.031×10^{-1}	1.301×10^{-1}	512	1.027×10^{-1}	1.269×10^{-1}	512
9	5.174×10^{-2}	6.026×10^{-2}	1,024	5.994×10^{-2}	5.763×10^{-2}	1,024
10	2.693×10^{-2}	2.978×10^{-2}	2,048	2.900×10^{-2}	2.523×10^{-2}	2,048
11	1.631×10^{-2}	1.669×10^{-2}	4,096	1.227×10^{-2}	1.333×10^{-2}	4,096
12	6.724×10^{-3}	6.801×10^{-3}	8,192	6.927×10^{-3}	8.333×10^{-3}	8,192
13	4.046×10^{-3}	3.976×10^{-3}	16,384	3.250×10^{-3}	3.238×10^{-3}	16,384
14	1.675×10^{-3}	1.986×10^{-3}	32,768	2.011×10^{-3}	2.030×10^{-3}	32,768
15	1.047×10^{-3}	8.790×10^{-4}	65,536	8.579×10^{-4}	9.272×10^{-4}	65,536
16	4.349×10^{-4}	4.681×10^{-4}	131,072	4.621×10^{-4}	5.335×10^{-4}	131,072

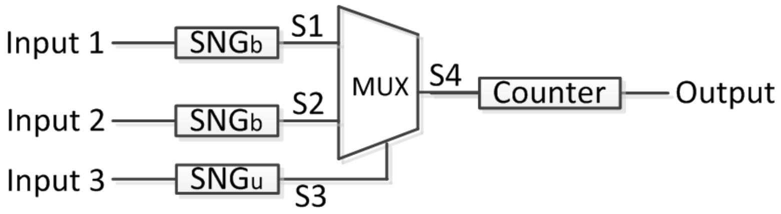


Figure C.1: Schematic of the stochastic adder: $S4 = 0.5 \cdot (S1 + S2)$ ($S1$ and $S2$ are uncorrelated bipolar sequences and S3 is a unipolar stochastic sequence encoding 0.5).

auxiliary circuits are included. The core of a stochastic adder, which is a multiplexer, outperforms the conventional binary adder in terms of power consumption. Table C.4 shows the minimum clock period of both the stochastic and binary circuits. The minimum clock period of the stochastic adder is consistently smaller than the conventional binary adder.

In Tables C.6 and C.5, the EPOs and TPAs of the conventional binary multiplier and the stochastic multiplier are compared. Again the stochastic adder including the auxiliary circuits is not competitive. The recommended bit resolutions for stochastic adders without auxiliary circuits are 4 bits and below as stochastic adders outperform the conventional binary adders in terms of TPA. Note that the required sequence lengths are shown in the last columns in Tables C.6 and C.5.

Absolute Subtractor

Absolute subtraction is a useful arithmetic operation in image processing, especially for the edge-detection application [6]. XOR gates can be directly used as stochastic absolute subtractors using correlated sequences. Both unipolar and bipolar sequences can be applied. The schematic of the stochastic absolute subtractor is shown in Figure C.2.

Measurements such as area, power and minimum clock period are compared in Tables C.7, C.8 and C.9. Similar conclusions can be drawn for the stochastic absolute subtractor as the stochastic adder. However, the stochastic subtractor also shows more promising

Table C.2: Area report of binary adders (B), stochastic adders including auxiliary circuits (S1) and stochastic adders excluding auxiliary circuits (S2). N_b represents the bit resolution.

N_b (bits)	B (um^2)	S1 (um^2)	Ratio: S1/ B	S2 (um^2)	Ratio: S2/ B
3	1.982	1.001	0.505	0.197	0.391
4	2.131	2.140	1.004	0.197	0.197
5	2.296	2.419	1.054	0.197	0.187
6	2.775	3.598	1.297	0.197	0.152
7	3.625	4.830	1.332	0.197	0.148
8	4.353	5.898	1.355	0.197	0.146
9	4.892	6.953	1.421	0.197	0.139
10	5.603	8.196	1.463	0.197	0.135
11	6.283	9.527	1.516	0.197	0.130
12	6.913	10.976	1.588	0.197	0.124
13	7.378	12.162	1.648	0.197	0.120
14	7.791	13.479	1.730	0.197	0.114
15	8.102	14.537	1.794	0.197	0.110
16	8.514	16.879	1.982	0.197	0.100

Table C.3: Power report of binary adders (B), stochastic adders including auxiliary circuits (S1) and stochastic adders excluding auxiliary circuits (S2) at minimum clock period. N_b represents the bit resolution.

N_b (bits)	B (mW)	S1 (mW)	S2 (mW)
3	0.060	0.382	0.013
4	0.061	0.508	0.013
5	0.065	0.633	0.013
6	0.069	0.832	0.013
7	0.073	0.855	0.013
8	0.076	0.894	0.013
9	0.079	1.075	0.013
10	0.082	1.123	0.013
11	0.085	1.195	0.013
12	0.090	1.276	0.013
13	0.093	1.423	0.013
14	0.098	1.562	0.013
15	0.105	1.694	0.013
16	0.110	1.804	0.013

Table C.4: Minimum clock period report of binary adders (B), stochastic adders including auxiliary circuits (S1) and stochastic adders excluding auxiliary circuits (S2). N_b represents the bit resolution.

N_b (bits)	B (ps)	S1 (ps)	S2 (ps)
3	630	210	190
4	630	210	190
5	630	210	190
6	650	210	190
7	660	220	190
8	670	220	190
9	680	220	190
10	690	220	190
11	700	220	190
12	720	220	190
13	730	230	190
14	750	230	190
15	750	230	190
16	760	240	190

Table C.5: EPO report of binary adders (B), stochastic adders including auxiliary circuits (S1) and stochastic adders excluding auxiliary circuits (S2). N_b represents the bit resolution and N_s represents sequence length.

N_b (bits)	B ($nJ/Operation$)	S1 ($nJ/Operation$)	Ratio: S1/ B	S2 ($nJ/Operation$)	Ratio: S2/ B	N_s (bits)
3	3.778×10^{-2}	1.284	34.0	4.101×10^{-2}	1.1	16
4	3.842×10^{-2}	3.412	88.8	8.202×10^{-2}	2.1	32
5	4.073×10^{-2}	8.514	209.1	1.640×10^{-1}	4.0	64
6	4.461×10^{-2}	$2.237\times10^{+1}$	501.5	3.281×10^{-1}	7.4	128
7	4.786×10^{-2}	$4.814\times10^{+1}$	1006.0	6.562×10^{-1}	13.7	256
8	5.096×10^{-2}	$1.007\times10^{+2}$	1976.6	1.312	25.8	512
9	5.339×10^{-2}	$2.421\times10^{+2}$	4535.0	2.625	49.2	1,024
10	5.630×10^{-2}	$5.058\times10^{+2}$	8984.1	5.249	93.2	2,048
11	5.918×10^{-2}	$1.077\times10^{+3}$	18192.7	$1.050\times10^{+1}$	177.4	4,096
12	6.477×10^{-2}	$2.299\times10^{+3}$	35494.1	$2.100\times10^{+1}$	324.2	8,192
13	6.827×10^{-2}	$5.363\times10^{+3}$	78558.7	$4.199\times10^{+1}$	615.2	16,384
14	7.374×10^{-2}	$1.102\times10^{+4}$	149462.2	$8.399\times10^{+1}$	1139.0	32,768
15	7.864×10^{-2}	$2.403\times10^{+4}$	305556.2	$1.680\times10^{+2}$	2136.1	65,536
16	8.365×10^{-2}	$5.330\times10^{+4}$	637203.5	$3.360\times10^{+2}$	4016.1	131,072

Table C.6: TPA report of binary adders (B), stochastic adders including auxiliary circuits (S1) and stochastic adders excluding auxiliary circuits (S2). N_b represents the bit resolution and N_s represents sequence length.

N_b (bits)	B $((ns \cdot um^2)^{-1})$	S1 $((ns \cdot um^2)^{-1})$	Ratio: S1/ B	S2 $((ns \cdot um^2)^{-1})$	Ratio: S2/ B	N_s (bits)
3	0.795	2.973×10^{-1}	3.739×10^{-1}	1.666	2.095	16
4	0.749	6.954×10^{-2}	9.279×10^{-2}	8.330×10^{-1}	1.112	32
5	0.696	3.076×10^{-2}	4.422×10^{-2}	4.165×10^{-1}	5.988×10^{-1}	64
6	0.554	1.034×10^{-2}	1.866×10^{-2}	2.082×10^{-1}	3.758×10^{-1}	128
7	0.419	3.676×10^{-3}	8.772×10^{-3}	1.041×10^{-1}	2.484×10^{-1}	256
8	0.342	1.505×10^{-3}	4.397×10^{-3}	5.206×10^{-2}	1.521×10^{-1}	512
9	0.301	6.385×10^{-4}	2.124×10^{-3}	2.603×10^{-2}	8.659×10^{-2}	1,024
10	0.259	2.708×10^{-4}	1.047×10^{-3}	1.301×10^{-2}	5.032×10^{-2}	2,048
11	0.228	1.165×10^{-4}	5.111×10^{-4}	6.507×10^{-3}	2.855×10^{-2}	4,096
12	0.201	5.055×10^{-5}	2.510×10^{-4}	3.254×10^{-3}	1.615×10^{-2}	8,192
13	0.185	2.182×10^{-5}	1.177×10^{-4}	1.627×10^{-3}	8.777×10^{-3}	16,384
14	0.170	9.844×10^{-6}	5.784×10^{-5}	8.134×10^{-4}	4.780×10^{-3}	32,768
15	0.165	4.564×10^{-6}	2.768×10^{-5}	4.067×10^{-4}	2.467×10^{-3}	65,536
16	0.154	1.883×10^{-6}	1.224×10^{-5}	2.034×10^{-4}	1.322×10^{-3}	131,072

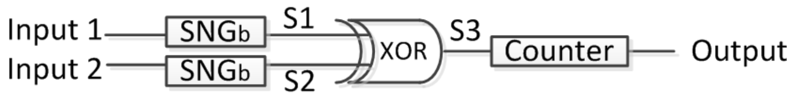


Figure C.2: Schematic of the stochastic absolute subtractor: $S3 = |S1 - S2|$ ($S1$ and $S2$ are correlated sequences).

performance with respect to TPA (see Table C.11). For 4-bit resolution, the TPA of the stochastic absolute subtractor is 1.37 times that of its conventional binary counterpart. In contrast, the TPA of the stochastic adder is only 1.11 times of that of the binary adder. The EPO reported in Table C.10 shows that the stochastic absolute subtractor consumes more energy than the conventional binary implementation even if for 3-bit resolution.

C.2 A Study on the Sum-of-Products Function Using Basic Stochastic Arithmetic Elements

In the previous sections and Chapter 2, basic stochastic arithmetic elements are investigated with respect to accuracy and cost efficiency. To further explore the feasibility of stochastic circuits using multiple basic arithmetic elements, a SOP function $Y = H[0]\cdot X[0]+H[1]\cdot X[1]$ is chosen to be discussed in this section. $H[0]$, $X[0]$, $H[1]$ and $X[1]$ are four signed numbers as primary inputs and Y is the output. The binary SOP can be implemented by multipliers and an adder as shown in Figure C.3(a). For the stochastic SOP, a straightforward implementation using XNOR gates (as bipolar multipliers) and a multiplexer (as an adder) is shown in Figure C.3(b). Bipolar SNGs are used to encode the primary inputs as $S(H[0])$, $S(X[0])$, $S(H[1])$ and $S(X[1])$. A counter is used to decode the stochastic sequence $S(Y)$. Note that similar results can be obtained if the primary inputs $H[0]$, $X[0]$, $H[1]$ and $X[1]$ are unsigned numbers and the unipolar encoding is used in the stochastic SOP implementation.

Stochastic and conventional binary circuits for a SOP function are built to investigate the area, power, delay and calculate the EPO and TPA. Again, the required minimum sequence lengths for the stochastic SOP implementation are determined by matching the RMSE of the corresponding conventional binary designs.

Table C.7: Area report of binary absolute subtractors (B), stochastic absolute subtractors including auxiliary circuits (S1) and stochastic absolute subtractors excluding auxiliary circuits (S2). N_b represents the bit resolution.

N_b (bits)	B (um^2)	S1 (um^2)	Ratio: S1/ B	S2 (um^2)	Ratio: S2/ B
3	2.235	1.002	0.449	0.201	0.448
4	2.548	2.148	0.843	0.201	0.238
5	3.148	2.420	0.769	0.201	0.261
6	3.245	3.603	1.110	0.201	0.181
7	4.496	4.835	1.075	0.201	0.187
8	4.393	5.903	1.344	0.201	0.150
9	4.980	6.962	1.398	0.201	0.144
10	5.640	8.196	1.453	0.201	0.138
11	6.295	9.532	1.514	0.201	0.133
12	7.029	10.979	1.562	0.201	0.129
13	8.223	12.163	1.479	0.201	0.136
14	8.021	13.484	1.681	0.201	0.120
15	8.728	14.538	1.666	0.201	0.121
16	9.415	16.883	1.793	0.201	0.112

Table C.8: Power report of binary absolute subtractors (B), stochastic absolute subtractors including auxiliary circuits (S1) and stochastic absolute subtractors excluding auxiliary circuits (S2) at minimum clock period. N_b represents the bit resolution.

N_b (bits)	B (mW)	S1 (mW)	S2 (mW)
3	0.064	0.383	0.015
4	0.062	0.518	0.015
5	0.070	0.642	0.015
6	0.076	0.841	0.015
7	0.074	0.863	0.015
8	0.085	0.899	0.015
9	0.083	1.083	0.015
10	0.089	1.125	0.015
11	0.093	1.202	0.015
12	0.098	1.285	0.015
13	0.099	1.433	0.015
14	0.099	1.463	0.015
15	0.111	1.602	0.015
16	0.112	1.703	0.015

Table C.9: Minimum clock period report of binary absolute subtractors (B), stochastic absolute subtractors including auxiliary circuits (S1) and stochastic absolute subtractors excluding auxiliary circuits (S2). N_b represents the bit resolution.

N_b (bits)	B (ps)	S1 (ps)	S2 (ps)
3	650	200	190
4	660	210	190
5	670	210	190
6	680	210	190
7	690	210	190
8	700	220	190
9	710	220	190
10	720	220	190
11	730	220	190
12	750	230	190
13	750	230	190
14	760	230	190
15	760	230	190
16	770	240	190

Table C.10: EPO report of binary absolute subtractors (B), stochastic absolute subtractors including auxiliary circuits (S1) and stochastic absolute subtractors excluding auxiliary circuits (S2). N_b represents the bit resolution and N_s represents sequence length.

N_b (bits)	B ($nJ/Operation$)	S1 ($nJ/Operation$)	Ratio: S1/ B	S2 ($nJ/Operation$)	Ratio: S2/ B	N_s (bits)
3	4.145×10^{-2}	1.226	29.6	4.451×10^{-2}	1.1	16
4	4.050×10^{-2}	3.478	85.9	8.901×10^{-2}	2.2	32
5	4.702×10^{-2}	8.635	183.6	1.780×10^{-1}	3.8	64
6	5.202×10^{-2}	$2.259\times10^{+1}$	434.4	3.560×10^{-1}	6.8	128
7	5.104×10^{-2}	$4.637\times10^{+1}$	908.5	7.121×10^{-1}	14.0	256
8	5.908×10^{-2}	$1.012\times10^{+2}$	1713.6	1.424	24.1	512
9	5.860×10^{-2}	$2.440\times10^{+2}$	4164.8	2.848	48.6	1,024
10	6.386×10^{-2}	$5.069\times10^{+2}$	7937.3	5.697	89.2	2,048
11	6.795×10^{-2}	$1.083\times10^{+3}$	15939.7	$1.139\times10^{+1}$	167.7	4,096
12	7.427×10^{-2}	$2.421\times10^{+3}$	32596.5	$2.279\times10^{+1}$	306.8	8,192
13	7.437×10^{-2}	$5.399\times10^{+3}$	72602.6	$4.557\times10^{+1}$	612.8	16,384
14	7.580×10^{-2}	$1.102\times10^{+4}$	145433.9	$9.115\times10^{+1}$	1202.5	32,768
15	8.484×10^{-2}	$2.414\times10^{+4}$	284573.7	$1.823\times10^{+2}$	2148.8	65,536
16	8.617×10^{-2}	$5.358\times10^{+4}$	621856.9	$3.646\times10^{+2}$	4231.1	131,072

Table C.11: TPA report of binary absolute subtractors (B), stochastic absolute subtractors including auxiliary circuits (S1) and stochastic absolute subtractors excluding auxiliary circuits (S2). N_b represents the bit resolution and N_s represents sequence length.

N_b (bits)	B $((ns \cdot um^2)^{-1})$	S1 $((ns \cdot um^2)^{-1})$	Ratio: S1/ B	S2 $((ns \cdot um^2)^{-1})$	Ratio: S2/ B	N_s (bits)
3	0.688	3.118×10^{-1}	4.531×10^{-1}	1.637	2.378	16
4	0.596	6.928×10^{-2}	1.162×10^{-1}	8.183×10^{-1}	1.372	32
5	0.473	3.074×10^{-2}	6.495×10^{-2}	4.091×10^{-1}	8.644×10^{-1}	64
6	0.453	1.033×10^{-2}	2.279×10^{-2}	2.046×10^{-1}	4.514×10^{-1}	128
7	0.322	3.847×10^{-3}	1.194×10^{-2}	1.023×10^{-1}	3.173×10^{-1}	256
8	0.326	1.504×10^{-3}	4.614×10^{-3}	5.114×10^{-2}	1.569×10^{-1}	512
9	0.283	6.376×10^{-4}	2.255×10^{-3}	2.557×10^{-2}	9.042×10^{-2}	1,024
10	0.247	2.708×10^{-4}	1.097×10^{-3}	1.279×10^{-2}	5.179×10^{-2}	2,048
11	0.217	1.164×10^{-4}	5.359×10^{-4}	6.393×10^{-3}	2.943×10^{-2}	4,096
12	0.189	4.834×10^{-5}	2.563×10^{-4}	3.196×10^{-3}	1.695×10^{-2}	8,192
13	0.162	2.182×10^{-5}	1.343×10^{-4}	1.598×10^{-3}	9.838×10^{-3}	16,384
14	0.163	9.840×10^{-6}	6.026×10^{-5}	7.991×10^{-4}	4.894×10^{-3}	32,768
15	0.150	4.563×10^{-6}	3.041×10^{-5}	3.995×10^{-4}	2.663×10^{-3}	65,536
16	0.138	1.883×10^{-6}	1.365×10^{-5}	1.998×10^{-4}	1.448×10^{-3}	131,072

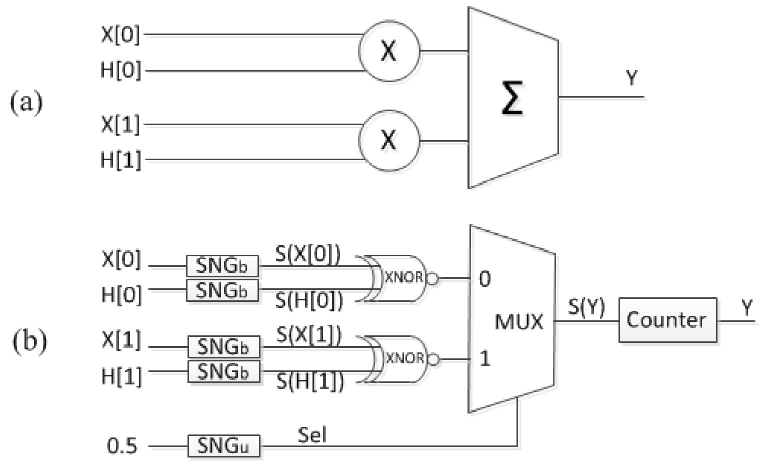


Figure C.3: Architectures of (a) conventional binary and (b) stochastic SOP implementations

C.2.1 Required Sequence Length for a Sum-of-Products Function

The RMSE method is again used as a strategy to determine the required sequence length. Conventional binary implementations of the SOP function are considered for increasing resolutions from 3 bits to 16 bits. Various sequence lengths are then used to compare the stochastic SOP implementation with the conventional binary implementation. Accuracy is measured by RMSE in Equation 2.58. Table C.12 shows the matching sequence lengths for increasing bit resolutions. The sequence length reported in Table C.12 will then be used in the evaluation of circuit performances.

C.2.2 Circuit Performance Comparison

For bit resolutions $N_b = 3, 4, ..., 16$, conventional binary circuits for SOP functions are built and evaluated. The stochastic SOP function can be implemented using two AND gates for unipolar multiplications and a multiplexer for summation. A standard ASIC design flow is used and the generated netlists are synthesized using Synopsys Design Compiler to yield reports on area, power and delay. EPO and TPA can then be calculated to compare the

Table C.12: RMSE comparison of binary and stochastic SOP implementations represented by RMSE_B and RMSE_S, respectively, using various sequence lengths N_s and different bit resolutions N_b.

R (Bits)	RMSE_B (%)	RMSE_S (%)	N_s (Bits)
3	13.31	13.50	128
4	6.741	6.569	512
5	3.114	3.064	1,024
6	1.861	1.548	2,048
7	8.075×10^{-1}	9.217×10^{-1}	4,096
8	4.459×10^{-1}	4.230×10^{-1}	8,192
9	2.173×10^{-1}	2.252×10^{-1}	16,384
10	1.153×10^{-1}	1.006×10^{-1}	65,536
11	4.815×10^{-2}	5.591×10^{-2}	524,288
12	2.744×10^{-2}	2.807×10^{-2}	1,048,576
13	1.347×10^{-2}	1.358×10^{-2}	4,194,304
14	7.106×10^{-3}	6.537×10^{-3}	16,777,216
15	3.358×10^{-3}	3.359×10^{-3}	67,108,864
16	1.790×10^{-3}	1.522×10^{-3}	268,435,456

conventional binary implementations and the stochastic implementations. Note that the stochastic implementations with or without auxiliary circuits such as SNGs and counters are considered.

The area comparison is shown in Table C.13. The stochastic SOP uses a smaller area as expected. The advantage becomes significant for increasing bit resolutions. In terms of power consumptions shown in Table C.14, however, the stochastic implementation is inefficient compared to the conventional binary implementation when the auxiliary circuits are included. In Table C.15, the stochastic SOP can run at a smaller clock period compared to the conventional binary implementation.

Fair comparisons should be based on metrics such as the TPA and the EPO. The stochastic SOP consumes more energy than the conventional binary SOP at all bit resolutions as shown in Table C.16. It is therefore not recommended to use stochastic computing for the SOP function if there is a tight budget on energy supply. From Table C.17, it can be seen that the stochastic SOP with auxiliary circuits underperforms the conventional binary SOP in terms of TPA. The stochastic SOP can only be competitive for 3-bit precisions when the auxiliary circuits are not included.

C.2.3 A Discussion on Various Scales of SOP Circuit Performance

The stochastic SOP implementation above is really a sum of two products. The results can be extended to estimate the performance of general SOP implementations with more inputs. From the previous results and discussions, circuits built using the stochastic approach suffer from poor efficiency because of the required auxiliary circuits such as SNGs. Stochastic circuits can only be competitive for some bit precisions when the auxiliary circuits are not included. Therefore, the proportion of the auxiliary circuits in a computing system is the key factor that determines if a stochastic implementation can achieve an advantage over the corresponding conventional binary implementation. If the general SOP implementation requires more auxiliary circuits such as SNGs, the performance of the stochastic circuits will be degraded compared to the previous sum-of-two-products implementation.

Table C.13: Area reports for the binary SOP function (B), the stochastic SOP function including auxiliary circuits (S1) and the stochastic SOP function excluding auxiliary circuits (S2). N_b represents bit resolutions.

N_b (bits)	B (um^2)	S1 (um^2)	Ratio: S1/ B	S2 (um^2)	Ratio: S2/ B
3	14.135	1.873	0.133	0.291	2.062×10^{-2}
4	19.824	2.313	0.117	0.291	1.470×10^{-2}
5	26.238	2.629	0.100	0.291	1.111×10^{-2}
6	36.265	3.272	0.090	0.291	8.037×10^{-3}
7	54.867	4.522	0.082	0.291	5.312×10^{-3}
8	74.770	5.468	0.073	0.291	3.898×10^{-3}
9	93.079	6.605	0.071	0.291	3.131×10^{-3}
10	118.434	7.937	0.067	0.291	2.461×10^{-3}
11	145.304	8.620	0.059	0.291	2.006×10^{-3}
12	172.830	9.524	0.055	0.291	1.686×10^{-3}
13	199.443	10.618	0.053	0.291	1.461×10^{-3}
14	226.170	11.413	0.050	0.291	1.289×10^{-3}
15	251.927	12.142	0.048	0.291	1.157×10^{-3}
16	281.855	13.274	0.047	0.291	1.034×10^{-3}

Table C.14: Power report of binary SOP function (B), stochastic SOP function including auxiliary circuits (S1) and stochastic SOP function excluding auxiliary circuits (S2) at minimum clock period. N_b represents bit resolutions.

N_b (bits)	B (mW)	S1 (mW)	S2 (mW)
3	0.476	1.138	0.022
4	0.510	1.518	0.022
5	0.542	1.893	0.022
6	0.559	2.486	0.022
7	0.567	2.555	0.022
8	0.580	2.678	0.022
9	0.585	3.215	0.022
10	0.602	3.363	0.022
11	0.627	3.575	0.022
12	0.645	3.816	0.022
13	0.662	4.263	0.022
14	0.677	4.382	0.022
15	0.715	4.774	0.022
16	0.746	5.076	0.022

Table C.15: Minimum clock period report of binary SOP function (B), stochastic SOP function including auxiliary circuits (S1) and stochastic SOP function excluding auxiliary circuits (S2). N_b represents bit resolutions.

N_b (bits)	B (ps)	S1 (ps)	S2 (ps)
3	820	290	230
4	830	300	230
5	840	300	230
6	850	300	230
7	860	300	230
8	870	310	230
9	880	310	230
10	890	310	230
11	900	310	230
12	920	320	230
13	920	320	230
14	930	320	230
15	930	320	230
16	940	330	230

Table C.16: EPO report of binary SOP (B), stochastic SOP including auxiliary circuits (S1) and stochastic SOP excluding auxiliary circuits (S2). N_b represents the bit resolution and N_s represents sequence length.

N_b (bits)	B ($nJ/Operation$)	S1 ($nJ/Operation$)	Ratio: S1/ B	S2 ($nJ/Operation$)	Ratio: S2/ B	N_s (bits)
3	0.391	$4.225\times10^{+1}$	$1.081\times10^{+2}$	6.492×10^{-1}	1.662	128
4	0.422	$2.331\times10^{+2}$	$5.519\times10^{+2}$	2.597	6.147	512
5	0.456	$5.817\times10^{+2}$	$1.276\times10^{+3}$	5.193	$1.140\times10^{+1}$	1,024
6	0.475	$1.528\times10^{+3}$	$3.217\times10^{+3}$	$1.039\times10^{+1}$	$2.187\times10^{+1}$	2,048
7	0.488	$3.139\times10^{+3}$	$6.438\times10^{+3}$	$2.077\times10^{+1}$	$4.260\times10^{+1}$	4,096
8	0.503	$6.801\times10^{+3}$	$1.351\times10^{+4}$	$4.155\times10^{+1}$	$8.253\times10^{+1}$	8,192
9	0.515	$1.633\times10^{+4}$	$3.171\times10^{+4}$	$8.309\times10^{+1}$	$1.614\times10^{+2}$	16,384
10	0.534	$6.832\times10^{+4}$	$1.279\times10^{+5}$	$3.324\times10^{+2}$	$6.221\times10^{+2}$	65,536
11	0.565	$5.810\times10^{+5}$	$1.029\times10^{+6}$	$2.659\times10^{+3}$	$4.707\times10^{+3}$	524,288
12	0.596	$1.280\times10^{+6}$	$2.148\times10^{+6}$	$5.318\times10^{+3}$	$8.920\times10^{+3}$	1,048,576
13	0.608	$5.722\times10^{+6}$	$9.415\times10^{+6}$	$2.127\times10^{+4}$	$3.500\times10^{+4}$	4,194,304
14	0.632	$2.353\times10^{+7}$	$3.724\times10^{+7}$	$8.509\times10^{+4}$	$1.347\times10^{+5}$	16,777,216
15	0.667	$1.025\times10^{+8}$	$1.536\times10^{+8}$	$3.403\times10^{+5}$	$5.099\times10^{+5}$	67,108,864
16	0.701	$4.497\times10^{+8}$	$6.415\times10^{+8}$	$1.361\times10^{+6}$	$1.942\times10^{+6}$	268,435,456

Table C.17: TPA report of binary SOP (B), stochastic SOP including auxiliary circuits (S1) and stochastic SOP excluding auxiliary circuits (S2). N_b represents the bit resolution and N_s represents sequence length.

N_b (bits)	B $((ns \cdot um^2)^{-1})$	S1 $((ns \cdot um^2)^{-1})$	Ratio: S1/ B	S2 $((ns \cdot um^2)^{-1})$	Ratio: S2/ B	N_s (bits)
3	8.625×10^{-2}	1.438×10^{-2}	1.668×10^{-1}	1.165×10^{-1}	1.351	128
4	6.091×10^{-2}	2.815×10^{-3}	4.621×10^{-2}	2.914×10^{-2}	4.783×10^{-1}	512
5	4.531×10^{-2}	1.238×10^{-3}	2.732×10^{-2}	1.457×10^{-2}	3.215×10^{-1}	1,024
6	3.244×10^{-2}	4.974×10^{-4}	1.533×10^{-2}	7.284×10^{-3}	2.245×10^{-1}	2,048
7	2.119×10^{-2}	1.800×10^{-4}	8.492×10^{-3}	3.642×10^{-3}	1.718×10^{-1}	4,096
8	1.540×10^{-2}	7.202×10^{-5}	4.676×10^{-3}	1.821×10^{-3}	1.182×10^{-1}	8,192
9	1.221×10^{-2}	2.981×10^{-5}	2.442×10^{-3}	9.105×10^{-4}	7.458×10^{-2}	16,384
10	9.507×10^{-3}	6.202×10^{-6}	6.523×10^{-4}	2.276×10^{-4}	2.394×10^{-2}	65,536
11	7.637×10^{-3}	7.138×10^{-7}	9.347×10^{-5}	2.845×10^{-5}	3.726×10^{-3}	524,288
12	6.261×10^{-3}	3.129×10^{-7}	4.998×10^{-5}	1.423×10^{-5}	2.272×10^{-3}	1,048,576
13	5.458×10^{-3}	7.017×10^{-8}	1.286×10^{-5}	3.557×10^{-6}	6.516×10^{-4}	4,194,304
14	4.737×10^{-3}	1.632×10^{-8}	3.446×10^{-6}	8.892×10^{-7}	1.877×10^{-4}	16,777,216
15	4.252×10^{-3}	3.835×10^{-9}	9.019×10^{-7}	2.223×10^{-7}	5.227×10^{-5}	67,108,864
16	3.774×10^{-3}	8.505×10^{-10}	2.253×10^{-7}	5.557×10^{-8}	1.472×10^{-5}	268,435,456

Assuming that a general SOP function can be written as

$$Y = \sum_{i=1}^{2^{N_p}} A_i \cdot X_i, \tag{C.1}$$

where 2^{N_p} is the number of products in the SOP function for computational convenience. The hardware used in the SOP implementations is estimated in Table C.18. In the sum-of-two-products implementation, two XNOR gates and one 2-input multiplexer are used to build the core circuit. Meanwhile five SNGs and one counter are required. For a more general case where the sum-of-2^{N_p}-products is implemented, the required hardware can be approximated by 2^{N_p} XNOR gates and $(2^{N_p} - 1)$ 2-input multiplexers. As many as $(2^{N_p+1}+N_p)$ SNGs are needed to convert the inputs of the XNOR gates and the multiplexers. The number of units required in the core circuit is calculated by adding up the number of XNOR gates and the number of multiplexers. Similarly, the number of units required in the auxiliary circuit is considered as the sum of the number of XNOR gatesSNGs and one representing the counter required to convert the stochastic sequence back to a binary number. The ratio of the auxiliary circuits over the core circuits is thus $6/3 = 2$ for the sum-of-two-products scenario. This number is greater than 2 for the sum-of-2^{N_p}-products implementation, indicating that the proportion of the auxiliary circuits increases as the number of the products grows. This rough estimation shows that the stochastic approach is still not competitive versus conventional binary for a general SOP function with 2^{N_p} pairs of products.

Table C.18: Hardware usage estimation of the stochastic SOP implementation. The number of circuit units provide simplified measures of cost.

	Core Circuit (C)		Auxiliary Circuit (A)		Ratio: A/C
SOP implementations	XNOR	2-input MUX	SNG	Counter	
Sum of two Products	2	1	5	1	$6/3 = 2$
Sum of 2^{N_p} Products	2^{N_p}	$2^{N_p} - 1$	$2^{N_p+1} + N_p$	1	> 2