Energy Efficient Stochastic Computing with Low-Discrepancy Sequences

A Dissertation

Presented to the

Graduate Faculty of the

Department of Computer Sience and Engineering

University of Louisiana at Lafayette

In Partial Fulfillment of the

Requirements for the Degree

Doctor of Philosophy

By

Sina Asadi

Summer 2023

Energy Efficient Stochastic Computing with Low-Discrepancy Sequences

Sina Asadi

APPROVED:

M. Hassan Najafi, Chair
Assistant Professor of Computer Science

Nian-Feng Tzeng
Professor of Computer Science

Magdy Bayoumi
Professor of Electrical and Computer Engineering

Ashok Kumar
Associate Professor of Computer Science

Mary Farmer-Kaiser
Dean of the Graduate School

Dedicated to my family.

Acknowledgments

I would like to sincerely thank my supervisor Dr. M. Hassan Najafi for always being supportive and a better friend to me. Without his guides and supports, I would not be able to make it and found my true potentials. His solid background, academic integrity, and hard-working personality inspired me to take steps toward my research. I would definitely remember each and every single discussion that we had so far and use them as a shining light in each direction and path of my life.

I want to express my deepest gratitude to my amazing family, for always being for me and bringing me the best possible and peaceful life I could ever imagine, for your company and encouragement, for the hope that lifted me up, and for all the emotional support. I owe you everything I have and have done so far and you are always in my heart and mind.

Table of Contents

List of Tables

List of Figures

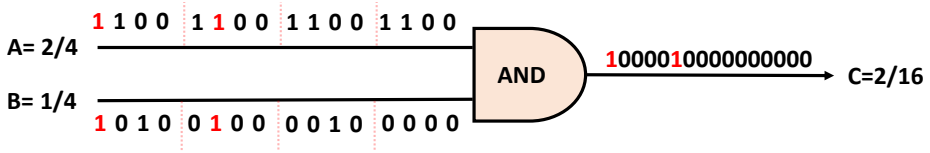
Chapter 1: Introduction

1.1 Introduction to Stochastic Computing

Stochastic computing (SC) [15, 4] is an unconventional computing paradigm offering low-cost and noise-tolerant solutions for a wide range of arithmetic operations from multiplication [33, 37] to division [10, 39, 11], square root [27], scaled addition, subtraction, minimum and maximum value functions [2, 32], trigonometric, logarithmic, and exponential function [31, 16]. SC treats data as probabilities presented by streams of random bits. This unconventional method of representing data leads to extremely simple computation circuits for complex arithmetic operations. Orders of magnitude saving in the hardware costs compared to the conventional binary radix designs are common with SC [4]. A single `AND` gate, for example, can perform multiplication in the stochastic domain (see Figure1.1).

Recently, SC has been utilized in important applications such as image processing and different Neural Network architectures. The paradigm has shown significant reductions in hardware area and power cost. However, there exist some important challenges in current SC designs, which we aim to address in this dissertation. Our goal is to advance the SC paradigm to the best possible extent. In this

Figure 1.1. Example of multiplication using stochastic bit-streams and `AND` gate.



chapter, we discuss the motivations behind the main contributions of this dissertation.

1.2 Motivation

Growing developments in electronic devices have brought up many challenges in energy consumption, area occupancy, and performance. In 1965, Gordon E. Moore observed a constant growth rate for manufacturing semiconductors in terms of the density of components per integrated circuit. This has become a self-fulfilling prophecy known as Moore's law [35]. This rule has been preserved for many years, and many scientists and manufacturers have contributed to maintaining it. Dennard's scaling law [12] states that, as transistors get smaller, their power density stays constant. However, this rule has come to an end due to voltage supply limits. The leakage power has increased more rapidly proportional to voltage drop, leading to heating up the chip, further increasing static power and threatening thermal runaway. This fact solidifies the demand for energy-efficient architectures to keep pace with today's emerging technologies. At the same time, newfound applications increasingly require fast and highly parallel computational components that put more stress on computing technologies, further restraining the growth rate. Emerging design technologies are being employed to fill the gap between chip manufacturing and application requirements.

SC is a re-emerging computational model offering power- and area-efficient hardware designs for different applications to push the market toward maintaining the predicted growth rate. In contrast to the radix-2 number system, well-known as binary, the inputs of this unconventional computing paradigm are bit-streams. All bits in a bit-stream have the same weight regardless of their positions. This makes the data

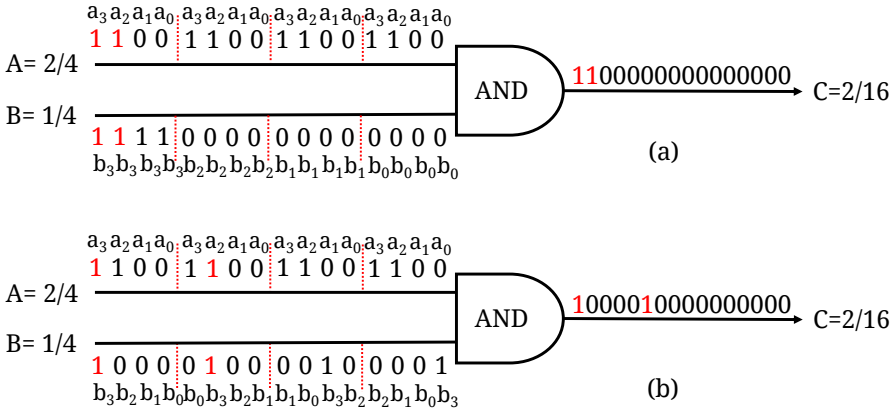
representation significantly error tolerant, therefore, well suited for noisy environments such as speech recognition systems where different types of noise affect the target voice or in image or video processing where sensors gather data in various conditions (i.e., different brightness). A single bit flip in a most significant bit of a binary number considerably diverts the computation result. However, with SC, one or a few bit flips have minimal impact on the computation accuracy.

1.2.1 Motivation for Context-Aware Bit-Stream Generation

SC designs consistently achieve 50× to 100× reduction in gate count compared to conventional binary radix designs [23]. For instance, multiplication, as a common but complex operation used in many applications, can be performed using standard `AND` gate in the stochastic domain. Input data in the [0,1] interval is represented using uniformly distributed random (i.e., interleaved) or unary (i.e., first all '1's followed by all '0's) bit-streams. The ratio of the number of ones to the length of the bit-stream determines the bit-stream value in this paradigm. For example, 10100, 01010, and 1111000000 are all stochastic bit-streams representing 0.4. While this unconventional representation of data is not compact compared to the weighted binary representation, it ensures the computation against soft errors (i.e., bit flips). Multiple bit-flips in a long bit-stream produce small and uniform deviations from the nominal value [33].

The inaccuracy of processing bit-streams was the main issue with the conventional SC designs. Random fluctuations in generating bit-streams and correlation between bit-streams led to computations that were only correct approximately. Some

Figure 1.2. Examples of deterministic bit-stream-based multiplication using (a) clock division (b) rotation method [17].



deterministic methods for processing bit-streams were introduced recently to produce completely accurate results with SC circuits. Relatively prime bit-stream lengths [26], clock dividing bit-streams, and rotation of bit-streams [17] are the three recently proposed methods that guarantee deterministic and accurate processing with stochastic logic. These methods were initially proposed based on unary bit-streams [17]. Figure 1.2 exemplifies the clock division- and the rotation-based multiplication of data using unary bit-streams. Najafi and Lilja [29] enhanced the performance of the three deterministic methods by replacing unary bit-streams with pseudo-random bit-streams. More recently, deterministic methods based on low-discrepancy (LD) bit-streams are also proposed [30, 25, 3]. The results produced by all these deterministic methods are completely accurate, the same as the results from conventional binary design.

A common property to all these deterministic methods is that for operations such as multiplication that require independent input bit-streams [33] producing exact (i.e., completely accurate) results requires generating and processing bit-streams for

$2^{m \times n}$ clock cycles, where m is the number of inputs and n is the precision of input data. For example, processing two 8-bit precision numbers requires generating 2^{16}-bit bit-streams in 2^{16} cycles. Obviously, the processing time increases exponentially by increasing the number of inputs and the precision of data. This latency quickly becomes unacceptable for any application. Long latency further translates to high energy consumption ($energy = power \times time$). The long latency and the high energy consumption make the current deterministic designs of SC inefficient for applications that expect high accuracy.

The first and most costly step in processing data using deterministic systems of SC is to convert the data from conventional binary to bit-stream representation. The conventional bit-stream generators used in these systems generate bit-streams regardless of the data value. Whether the input is 128/256 or 13/256, the same length bit-stream (e.g., 256-bit) is generated. 13/256 is a real number with the data-width (i.e., precision) of eight bits. It then requires a bit-stream of at least 256 bits to be precisely represented. However, 128/256 (=1/2) is a real number with the data-width of one bit. It can therefore be represented precisely using a short bit-stream of only 2 bits (i.e., 10 or 01). This dissertation proposes some context-aware methods for generating stochastic bit-streams. The proposed methods significantly reduce the latency of the deterministic methods. We propose a control unit to determine the bit-width of input data and dynamically adjust the system to generate the bit-streams with the minimum required length. The proposed bit-stream generators reduce the number of operation cycles up to 86% compared to the current bit-stream generators.

1.2.2 Motivation for Low-Cost FSM-based Bit-Stream Generation

Low-discrepancy (LD) bit-streams such as Halton-[3] and Sobol-based [24] bit-streams were proposed recently to improve the accuracy and reduce the processing time of SC. 1's and 0's are uniformly spaced in these bit-streams. Random fluctuations are removed from bit-stream generation, and deterministic and accurate bit-streams are generated. Progressive precision of Sobol sequences [24], in particular, has made them popular for LD bit-stream generation [30, 6]. The first 2^n numbers of any Sobol sequence include all possible n-bit precision values in the [0,1] interval. This allows 2^n-bit Sobol-based bit-streams to precisely represent any n-bit precision value.

For generating an LD bit-stream, a quasi-random number (e.g., a Sobol number from a Sobol sequence) is compared to a target number using a binary comparator. The output of this comparison generates one bit of the LD bit-stream in each cycle. Quasi-random number generators, however, are costly. A 4-, 8-, and 16-bit precision Sobol sequence generator takes 2.8×, 4.7×, and 9.1×, respectively, more hardware area cost than the same-precision pseudo-random number generator [28]. The high hardware cost limits the potential benefits and the scalability of the conventional comparator-based LD bit-stream generator [4][28]. Sim et al. [37] recently introduced an FSM-based LD bit-stream generator to convert data from binary to LD bit-stream. Sim's generator selects x_{n-i} or the $(n-i)^{th}$ bit of the binary input X first at cycle 2^{i-1} and thereafter every 2^i cycles. The hardware cost of Sim's generator is considerably lower than that of the comparator-based LD generator [37]. The challenge is that Sim's generator can only generate *one fixed LD pattern*. Hence, it cannot be used in SC

designs in which multiple independent LD bit-streams are needed. This includes multi-input multipliers, scaled adders [4], and the ReSC-based designs such as the Gamma correction circuit [33], to name a few. These designs still have to employ conventional comparator-based LD generators.

We propose a low-cost FSM-based LD bit-stream generator that supports the generation of any number of LD patterns. We develop an algorithm to implement different LD patterns based on different Sobol sequences [24]. The proposed generator is able to generate any number of independent bit-streams. SC systems implemented based on the proposed generator are able to produce completely accurate results, the same as the results from the conventional binary counterparts.

1.2.3 Motivation for Low-Discrepancy Correlation Manipulation Circuits

In stochastic systems, both correlated and uncorrelated bit-streams are needed subject to the desired function. Two bit-streams are *positively correlated* when all 1's in the two bit-streams are completely overlapped, i.e., they are exactly in the same bit positions. An important part of a stochastic system is the data conversion unit that converts the input data from positional binary to bit-stream representation, where bit-streams correlation is managed. While this method is common for controlling the correlation between the inputs of the stochastic system at the first stage of computations, it cannot be used in the next stages of the system as the data are already in the bit-stream form. A naive method for manipulating correlation in the intermediate stages is to convert the bit-streams back to the positional binary format

and then re-generate them with the desired correlation. But this approach incurs significant area and power overheads. Developing low-cost correlation manipulation circuits for in-stream (i.e., without re-generating bit-streams) and managing of correlation between bit-streams is an ongoing research.

The mentioned issue has motivated us to strive for an in-stream-based approach that can prepare the bit-streams for any arithmetic operations despite the level of correlation between input bit-streams. Our approach, called CORLD, proposes an in stream correlator that can increase positive correlation between stochastic bit-streams while generating low-discrepancy bit-streams. We further develop a decorrelator circuit that can make input bit-streams uncorrelated with negligible impact on their value.

1.3 Dissertation Summary

This dissertation aims to develop novel solutions for some on-going and important challenges in designing SC systems. In particular, we target the state-of-the-art low-discrepancy methods of SC and address their design challenges from bit-stream generation to correlation manipulation. Chapter 2 starts with the basic definitions of SC such as stochastic numbers (SN) and different methods of bit-stream generation from conventional models to novel approaches. Progressive precision is one of the important properties of SC which will be discussed in Chapter 2 besides the background on SC correlation. Chapter 3 to chapter 5 present the main contributions of this dissertation. The challenges of the most recent and relevant works will be analyzed to demonstrate the determined track of our methods. We compare our proposed design approaches with the relevant and state-of-the-art methods evaluating

their performance and implementation costs.

Chapter 2: Stochastic Computing Basics

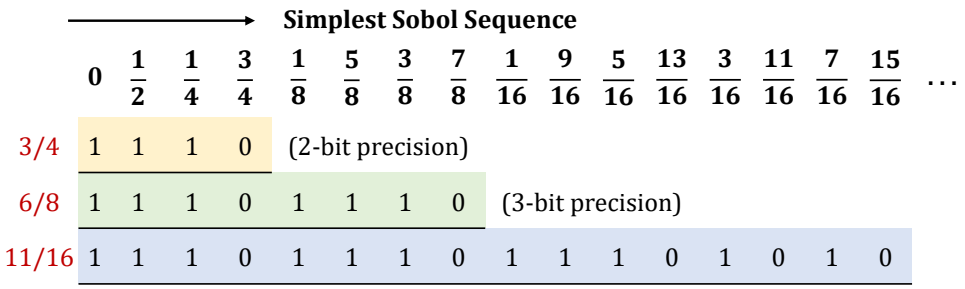
A brief introduction to Stochastic Computing (SC) has been provided in the previous chapter. In this chapter, the Stochastic Number (SN) format or bit-stream is thoroughly explained, and different generation techniques are shown. We also introduce the SC progressive precision property and its effect on the latency and energy consumption, and the SC correlation characteristic and SC functions' requirements concerning the correlation between bit-streams to achieve higher accuracy.

2.1 Stochastic Number format

SN is a stream of bits (1s or 0s), irrespective of their positions, showing the expected probability value based on the number of 1s embedded in the bit-stream. Figure 2.1 shows examples of some SNs with different probabilities. They are produced by the simplest Sobol sequence generator. The first 16 numbers of the simplest Sobol sequence and examples of LD bit-streams are generated in this example. A '1' is generated in the bit-stream if the Sobol number is *less* than the target number. For example, the probability of 0.75 is shown with a 4-bit long bit-stream, having three 1s. Although the distribution of 1s can profoundly affect any SC operation's result, it does not have an impact on an SN representation. Mentioned unipolar SN is limited to the unit interval of [0,1]; therefore, any operation requires inputs beyond this interval, and the inputs must be scaled in advance.

The SN representation can also be extended to several further formats, such as signed numbers. For negative numbers, SC bipolar format has been introduced as follows: if the value X in unipolar format is mapped to Y, which is in bipolar format,

Figure 2.1. Examples of LD bit-streams with different precisions generated using the simplest Sobol sequence.

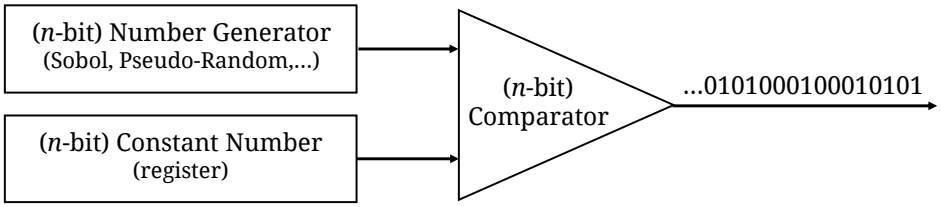


the formula is Y = 2X - 1. In other words, a bit-stream with only 0 bits has a value of -1 in bipolar representation ((Y = -1) = 2(X = 0) - 1). Many alternative data formats have been proposed for SC, nevertheless, arithmetic functions in each format might come with certain SC circuits. While in unipolar format, an AND gate performs a multiplication operation, XNOR gate on the other hand processes the multiplication in bipolar format.

2.2 Stochastic Number Generator Components

To process data with the stochastic systems, we need first to convert the data from binary-radix to bit-stream representation. Figure 2.2 shows the conventional structure of a binary-to-stochastic converter (a.k.a. stochastic number generator (SNG)). A number source is compared to a constant number (based on the target input data), and the output of the comparison produces one bit of the bit-stream in each cycle. A one is generated at the output of the comparator if "*Number Source*" $<$ "*Constant Number*". The number generator determines the type of the produced bit-stream. As noted earlier, the value of a bit-stream is dependent on the number of 1s, while the type of representation is different. A pseudo-random number source such as LFSR generates a

Figure 2.2. Conventional Binary-to-Stochastic Converter (BSC)



type of bit-streams that might not be at a consistent pace. An n-bit counter can also be a number source starting from zero ending in 2^n that generates a series of bits (1s followed by zeros) called Unary bit-streams. On the other hand, Low Discrepancy (LD) sequences such as the Sobol sequence produce deterministic bit-streams that increases the accuracy of SC operations due to their inherent characteristics. Quasi-random (or low discrepancy) sequences are sequences for which the convergence to the uniform distribution on [0; 1)s occurs rapidly [38].

2.3 Progressive Precision property

In SC, accuracy is defined as the level of bit-stream conformity with the actual value. If the accuracy of representation increases with the length of the bit-stream, it is called progressive precision. This property is clearly shown in figure 2.1. If the actual probability is 11/16, as the precision of the data, or the length of the bit-stream, increases, the accuracy gets closer to the actual value.

This property plays an important role in reducing the number of cycles if an application is resilient to some degree of inaccuracy. LD bit-streams have the best progressive precision behavior among all types of SNs so that within 8 clock cycles the required accuracy is provided in some cases.

2.4 Stochastic Correlation

The correlation between two bit-streams X and Y is quantified using the SC Correlation (SCC) as defined in [2]:

$$SCC(X,Y) = \begin{cases} \dfrac{ad - bc}{N \times min(a+b, a+c) - (a+b)(a+c))} & ad > bc \\ \dfrac{ad - bc}{(a+b)(a+c) - N \times max(a-d, 0)} & else \end{cases}$$

In this formula, a is the number of bit positions where both bit-streams (X and Y) are 1, b is the number of bit positions where X is 1, and Y is 0, c is the number of bit positions where X is 0 and Y is 1, and d is the number of bit positions where both bit-streams are 0. An SCC equal to 0 means the bit-streams are completely uncorrelated. SCC values close to +1 or -1 show high positive or negative correlation, respectively. A positive correlation of +1 means that there is a maximal correlation between the two bit-streams.

Chapter 3: Accelerating Deterministic Stochastic Computing with Context-Aware Bit-stream Generator[5][6]

Deterministic approaches to SC were proposed recently to produce completely accurate results with stochastic logic. Real-valued numbers in the [0,1] interval are converted to unary or pseudo-random bit-streams and processed using the relatively prime bit-stream length, clock division, or rotation method. Fast converging deterministic methods based on low-discrepancy bit-streams were also introduced. Long latency is the main issue with all these deterministic methods. To process m n-bit precision numbers, bit-streams of $2^{m\times n}$ bits must be generated. In this dissertation, we propose a context-aware bit-stream generator to improve the performance of deterministic bit-stream processing systems. The proposed design reduces the processing time up to 86% for the cases where completely accurate results are desired. When the application can tolerate some small rates of inaccuracy, orders of magnitude reduction in the latency are achievable. This chapter's material has been published in [6] and [5].

3.1 Deterministic Bit-stream Generator

In this section, we first discuss the current binary-to-stochastic converter (BSC) used in the deterministic SC. We then show the architecture of the state-of-the-art clock division [17], rotation [17], and LD [30] deterministic designs. Finally, we propose our context-aware architectures for these deterministic systems.

3.1.1 Conventional Design

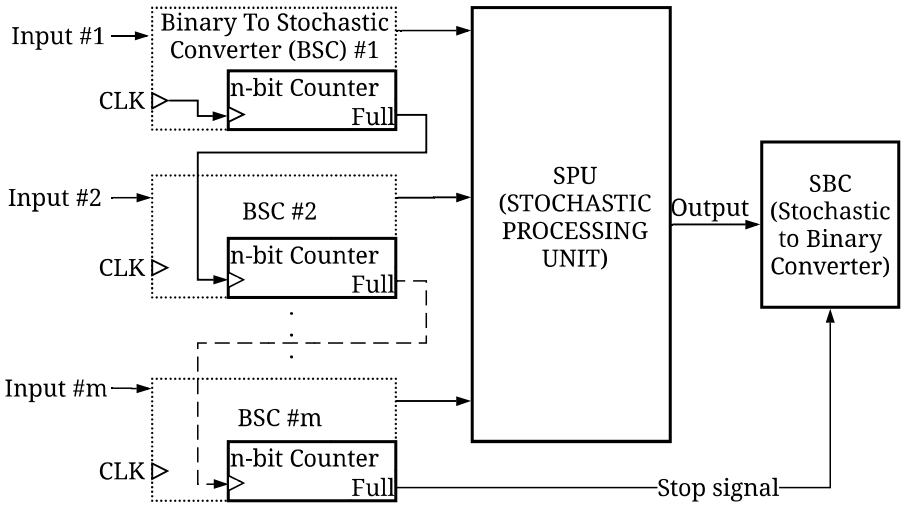
The conventional design of the of a binary-to-stochastic converter is reviewed earlier in chapter 2 which is currently used in the deterministic designs of SC. An

increasing/decreasing number from an up/down counter (for the clock division and rotation methods) or a Sobol number from a Sobol sequence generator (for the LD method) is compared to a constant number (based on the target input data), and the output of comparison produces one bit of the bit-stream in each cycle. A one is generated at the output of the comparator if "*Number Source*" < "*Constant Number*".

Figures 3.1 and 3.2 depict the conventional architecture of the SC system based on the clock division and rotation method. Input data is converted from binary to unary bit-streams using counter-based BSC units. A *stop signal* determines when the result is ready and the system must stop processing. As shown in Figure 3.1, in the clock division design, the *stop signal* is produced at no additional hardware cost by using the same counters used in the converter modules. The rotation-based design, on the other hand, utilizes some additional counters chaining the overflow (full) output of each converter's counter to the clock inputs of the next input counters to generate the *inhibit* signals [17]. With this design, the n-bit counter of the i^{th} converter module is inhibited every $2^{n \cdot i}$ cycles.

Sobol sequences are also used to generate and process bit-streams deterministicly [30, 28, 25, 7]. Direct use of different Sobol sequences in converting input data to LD bit-streams can guarantee deterministic and accurate processing of bit-streams [30]. The output converges to the expected result faster than the unary bit-stream-based designs. However, the number of processing cycles to produce completely accurate results is the same as the unary designs. The down-side is the high cost of generating Sobol sequences [30]. Figure 3.4 demonstrates the architecture of a

Figure 3.1. Conventional architecture of a deterministic system based on the "clock division" method of [17].

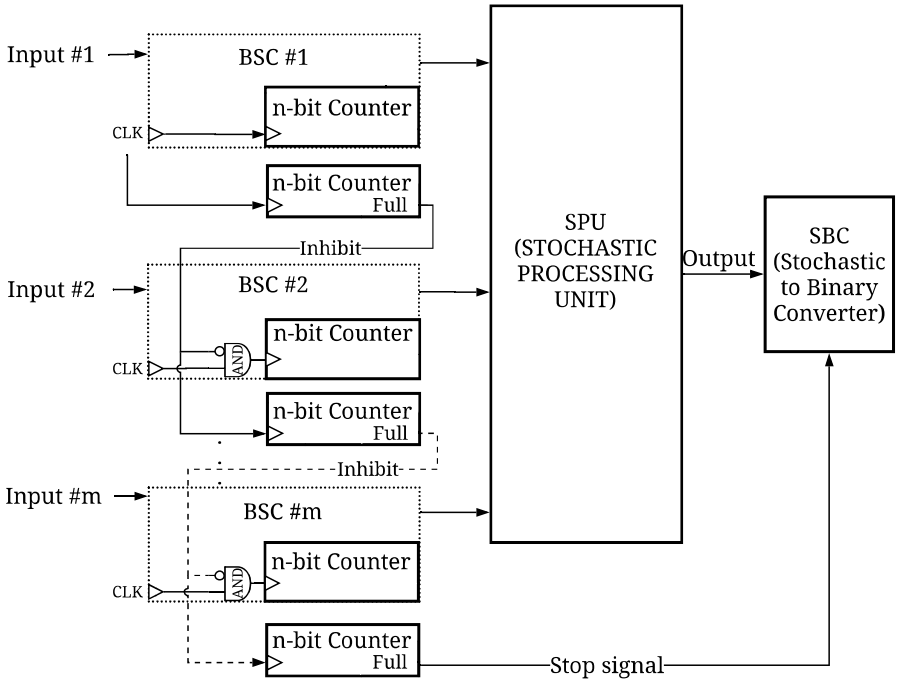


deterministic system based on LD Sobol bit-streams. Each BSC unit compares the input data to a Sobol number from a Sobol number generator. As elaborated in [30], an $m \times n$-bit counter is shared between all convertor modules.

The *full* output of this counter serves as the required *stop signal* of the system.

In all these deterministic systems, m independent inputs are converted from binary to bit-stream representation using m converter modules. Each system runs for $2^{m \times n}$ cycles to produce completely accurate result. The processing is stopped by sending a *stop signal* to the stochastic-to-binary converter (SBC) unit. The stop signal turns to "1" when the system operates for exactly $2^{m \times n}$ cycles.

Figure 3.2. Conventional architecture of a deterministic system based on the "rotation" method of [17].



3.1.2 Proposed Context-Aware Design

3.1.2.1 Structure for Completely Accurate Computation

The conventional architectures discussed in Section 3.1.1 , generate bit-streams regardless of the value of the input data. Multiplying 10/16 (1010 in binary radix) and 8/16 (1000 in binary radix) using these architectures takes the same number of cycles as multiplying 9/16 and 15/16. In both cases, the system must run for 2^{4+4}(=256) cycles to produce the exact result. The deterministic methods of SC, however, can produce the exact result of multiplying 10/16 and 8/16 in only 2^{3+1} (=16) cycles if effectively change 10/16 to 5/8 and 8/16 to 1/2, and represent each one using its minimum required bit-stream length (i.e., 8 and 2 bits, respectively). For this set of

Figure 3.3. The structure of a 4-bit Modified Counter

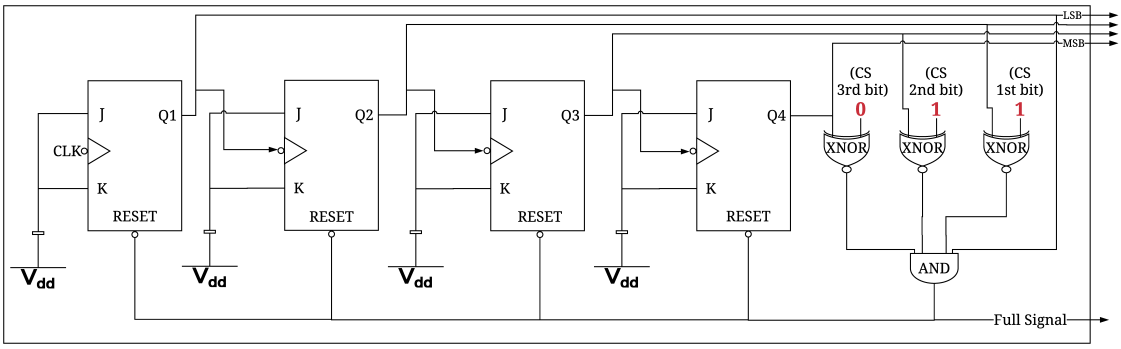
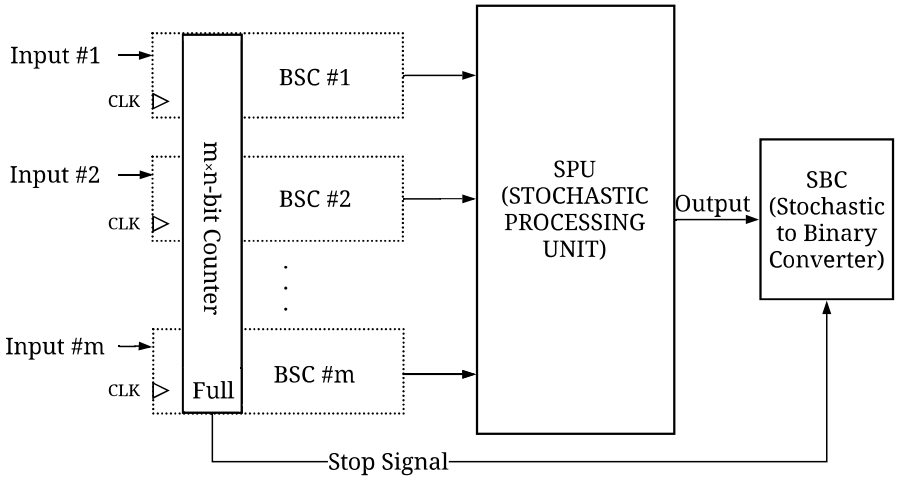


Figure 3.4. Conventional architecture of a deterministic system based on LD Sobol sequences [30].



inputs running the system for more than 16 clock cycles wastes the time and more importantly the energy resources of the system.

Figure 3.5 shows our proposed context-aware bit-stream generator for the clock division- and rotation-based designs. The regular counter of the BSC unit shown in Figure 2.2 is replaced with a modified counter *(MC)*. A control unit (CU) is also added to determine the data-width. The control unit reads the input data from the constant register and determines the minimum data-width to precisely represent input data.

Figure 3.5. Proposed bit-stream generator with Control Unit (CU) and Modified Counter (MC)

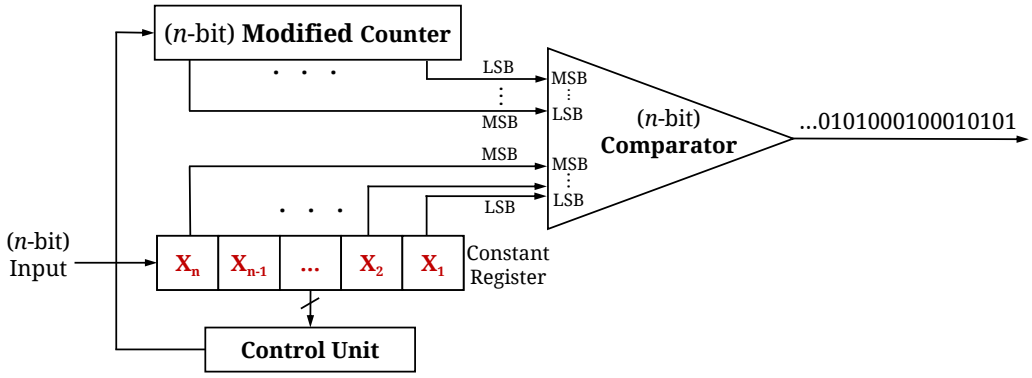


Figure 3.6 shows how *CU* produces the control signals for A=10/16 (1010 in binary). 10/16 is equivalent to 5/8 and has a data-width of 3 bits. *CU* sends 011 to the *MC*, forcing it to work as a 3-bit counter (counting from 0 to 7) instead of a 4-bit counter. Figure 3.3 shows the structure of a 4-bit *MC*. The output bits of the *MC* are connected to the comparator in *reverse* order. Considering the fact that the simplest Sobol sequence can be generated by simply reversing the output bits of a counter [25], by connecting the output bits of the *MC* to the comparator in reverse order the effective bits of the constant register (e.g., 101 in 1010 and 1 in 1000) is compared to a new Sobol number in each cycle. This results in converting the input binary data to a fast converging LD bit-stream instead of a unary bit-stream (by comparing to the output bits of a counter in normal order), at no additional hardware cost. LD bit-streams are preferred to unary bit-streams as they enjoy the progressive precision of random bit-streams [30]. Figure 3.7 shows the structure of an n-bit *CU* for the proposed bit-stream generator.

Figure 3.8 depicts our proposed architecture for a "clock division"-based

Figure 3.6. A 4-bit *CU* generating control signals for 10/16

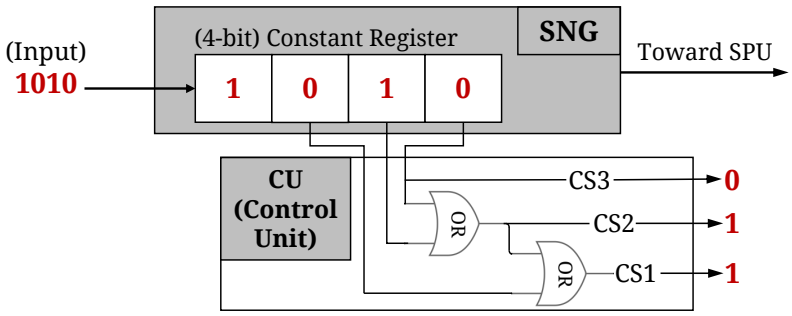
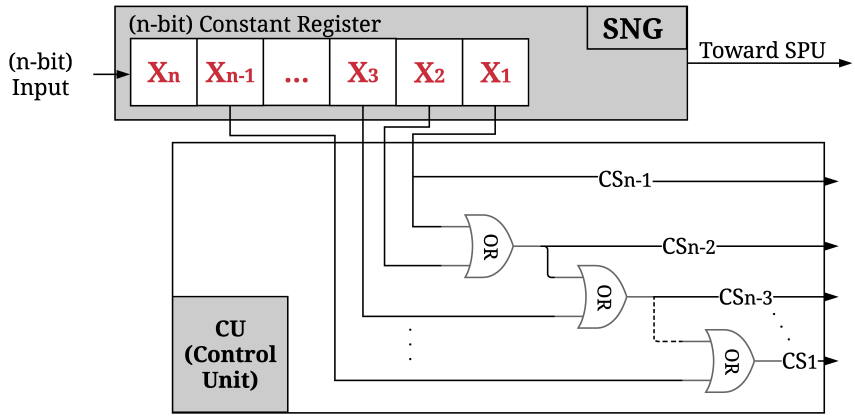


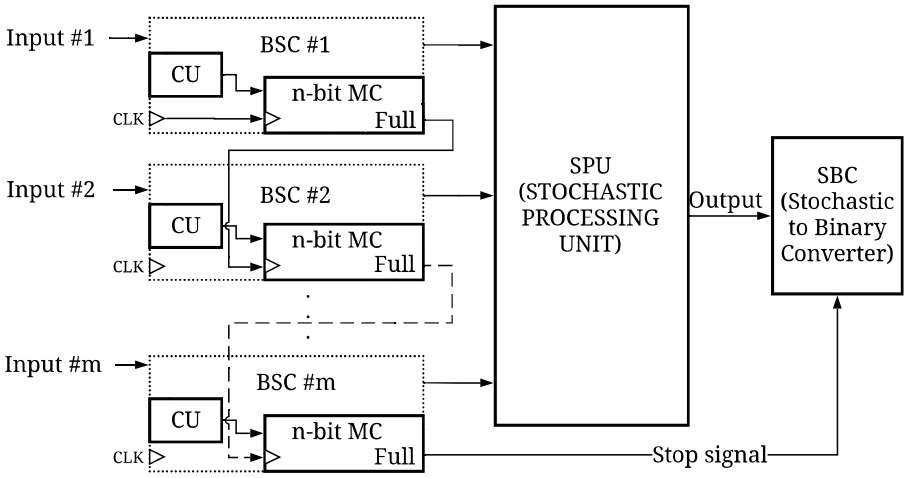
Figure 3.7. The structure of an n-bit *CU*



deterministic design. The system operates for $2^{Q_1+Q_2+...+Q_m}$ cycles (instead of $2^{m\times n}$ cycles), where Q_i is the data-width of input i and m is the number of independent inputs in the system. The conventional architecture of a "rotation"-based system uses regular counters to inhibit the converters at the powers of the operands (i.e., bit-streams) lengths. Figure 3.9 shows our proposed architecture for a "rotation"-based system. The proposed design similarly replaces the regular counters with *MC* units. The *inhibit* and *stop* signals are generated depending on the bit-width determined by the *CU* units.

We also develop a context-aware architecture for the Sobol-based deterministic

Figure 3.8. Proposed architecture for a "clock division"-based deterministic system.



system. The first 2^n bits of a Sobol-based bit-stream can precisely represent any n-bit precision data [30]. Due to this property of Sobol-based bit-streams, no change in the structure of the converter modules is necessary to adapt the system to input data. Only the *stop signal* must be set accordingly. Figure 3.10 presents our proposed architecture. *CU* receives the input data from the constant register and sends some control signals to the *MC* unit. Figure 3.11 demonstrates an example of deterministic multiplication using the conventional and the proposed LD design. As can be seen, shorter bit-streams are generated and processed by stopping the operation at the point needed to get the accurate result.

3.1.2.2 Structure for Error Tolerant Applications

In binary radix representation, least significant bits (LSBs) have less impact on the accuracy of computation than most significant bits (MSBs). This impact is further

Figure 3.9. Proposed architecture for a "rotation"-based deterministic system.

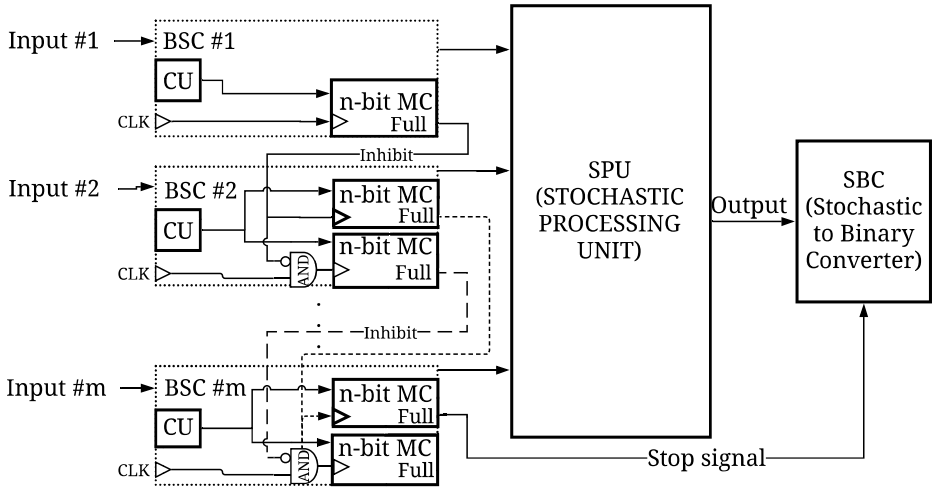
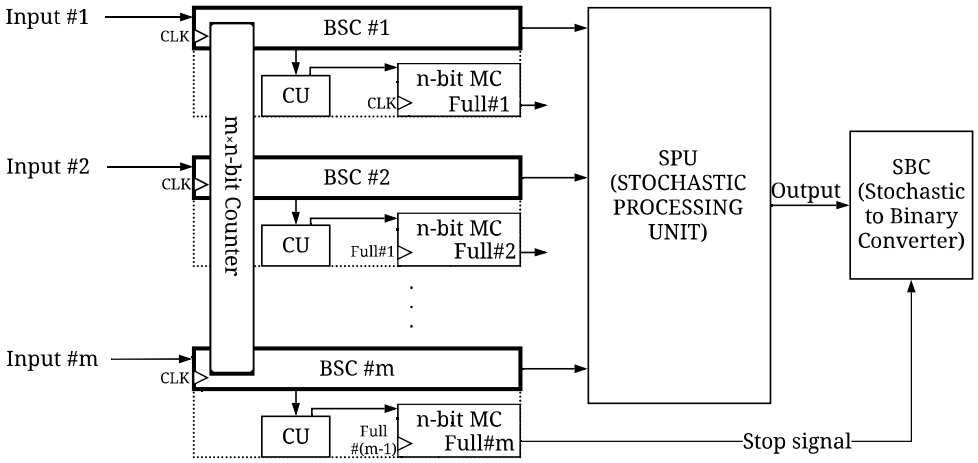


Figure 3.10. Proposed architecture for a "Sobol"-based deterministic system.



decreased when the data bit-width increases. The difference between two 8-bit precision numbers A=10000000 and B=10000001 is only on the LSB. A represents 0.5 while B represents 0.5039. The absolute difference between these two values is only 0.0039, a negligible difference in many applications. If the application can tolerate small rates of

Figure 3.11. An example of Sobol bit-stream-based multiplication using (a) conventional design and (b) proposed design.

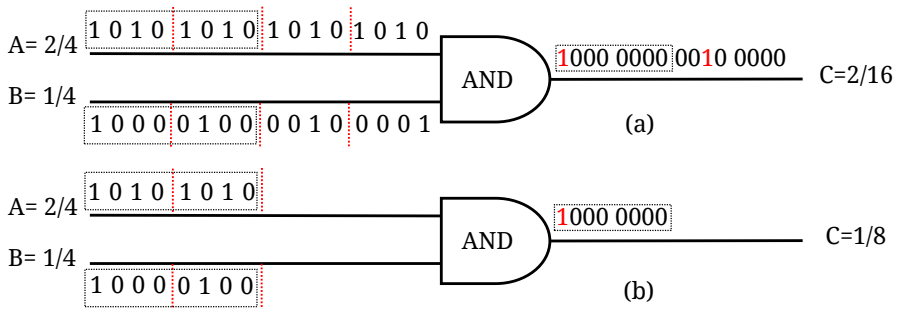
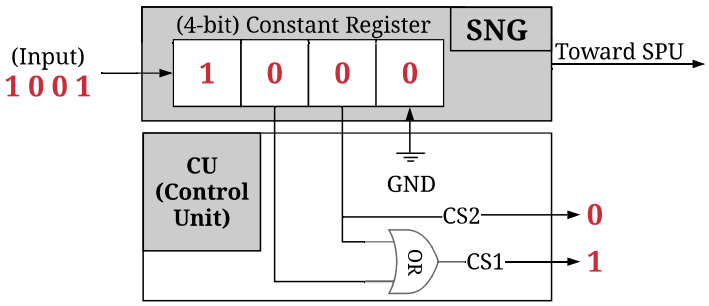


Figure 3.12. Modified 4-bit *CU* for error-tolerant applications



inaccuracy, it is feasible to further decrease the processing time of the deterministic system for many input cases. For example, if setting the LSB of B=1000000$\underline{1}$ to 0, the data can be precisely represented using a stream of only 2 bits rather than a stream of 2^8 bits. This significantly reduces the number of processing cycles at the cost of a negligible accuracy loss. Figure 3.12 shows a modified *CU* for a system that processes 4-bit precision data and can tolerate small rates of inaccuracy. Assume two 4-bit inputs, A=1001 and B=1001, are to be multiplied. With the proposed design, the length of bit-streams and so the number of processing cycles decreases from 2^8 to 2^2 at the cost of 0.06 absolute error.

3.2 Evaluation

3.2.1 Hardware Cost Comparison

The overhead of the proposed designs is some additional logic gates due to adding the *CU* and *MC* units. As shown in Figure 3.7, *CU*s include some standard `OR` gates. *MC*s also contain some additional `XNOR` gates compared to the regular counters. The overhead cost of these units is insignificant compared to the total cost of the system. Table 3.1 shows the overhead of the proposed designs for different input data-widths in terms of the required additional logic gates. For example, for the 16-bit precision design, 14 additional `OR` and 15 additional `XNOR` gates are required for both the clock division- and the Sobol-based system, and 14 additional `OR` and 30 additional `XNOR` gates are required for the rotation-based system.

By accepting inaccuracy in the computation not only the overhead but also the overall area occupancy of the system decreases compared to the conventional deterministic architectures. This additional saving is due to the reduction in the size of the *MC* units. For example, assume the data bit-width is 16 and the application can tolerate up to 2 percent error rate in the result. This allows the system to ignore up to six LSBs (i.e., set six LSBs to 0) for a 2-input multiplier design. The modified design for such an error-tolerant application has an overhead of 8 `OR` and 9 `XNOR` gates (for the clock division and Sobol-based designs) and can work with a 10-bit counter instead of a 16-bit one. For some rates of error the hardware savings from reducing the size of the counter is greater than the overhead cost of additional gates added to the system. In these cases, the proposed design reduces the hardware cost compared to the

Table 3.1. Overhead of the Proposed Designs (CD: Clock Division, Ro: Rotation, So: Sobol)

Input Precision	# of Ignored LSB Bits	# of `OR` Gates	# of `XNOR` Gates		Saved Components	
		CD & Ro & So	CD & So	Ro	CD & So	Ro
4-bit	0 (Exact)	2	3	6	0	
	1	1	2	4	1 Flip Flop, 1 `AND` Gate	2 Flip Flops, 2 `AND` Gates
8-bit	0 (Exact)	6	7	14	0	
	1	5	6	12	1 Flip Flops, 1 `AND` Gates	2 Flip Flops, 2 `AND` Gates
	2	4	5	10	2 Flip Flops, 2 `AND` Gates	4 Flip Flops, 4 `AND` Gates
	3	3	4	8	3 Flip Flops, 3 `AND` Gates	6 Flip Flops, 6 `AND` Gates
	4	2	3	6	4 Flip Flops, 4 `AND` Gates	8 Flip Flops, 8 `AND` Gates
16-bit	0 (Exact)	14	15	30	0	
	1	13	14	28	1 Flip Flop, 1 `AND` Gate	2 Flip Flops, 2 `AND` Gates
	2	12	13	26	2 Flip Flops, 2 `AND` Gates	4 Flip Flops, 4 `AND` Gates
	3	11	12	24	3 Flip Flops, 3 `AND` Gates	6 Flip Flops, 6 `AND` Gates
	4	10	11	22	4 Flip Flops, 4 `AND` Gates	8 Flip Flops, 8 `AND` Gates
	5	9	10	20	5 Flip Flops, 5 `AND` Gates	10 Flip Flops, 10 `AND` Gates
	6	8	9	18	6 Flip Flops, 6 `AND` Gates	12 Flip Flops, 12 `AND` Gates

conventional architecture.

We synthesized the conventional BSC unit (shown in Figure 2.2) and the proposed bit-stream generator (shown in Figure 3.5) using the Synopsis Design Compiler v2018.06 with the 45nm FreePDK gate library. For the conventional BSC unit, regular binary counter is used as the number generator. Table 3.2 compares the hardware area cost and the critical path latency (CP) for three different data

Table 3.2. Hardware Cost Comparison of the BSC unit

Input Precision	Conventional Design		Proposed Design		
	CP (ns)	Area (μm^2)	CP (ns)	Area (μm^2)	% of Area Overhead
4-bit	0.29	123	0.29	158	27
8-bit	0.35	284	0.35	358	25
16-bit	0.41	599	0.41	744	24

precisions. As shown, the proposed bit-stream generator costs up to 27 percent area overhead with the same CP latency. A significant decrease in the number of processing cycles compared to the conventional BSC leads to a considerable reduction in the total latency (CP $\times$ number of processing cycles) and hence in the energy consumption.

3.2.2 Performance Comparison

Table 3.3 reports the performance improvements with the proposed designs. For each case (different number of inputs and different data precision), we report the number of processing cycles required for the conventional design and also the percentage of the reduced number of cycles with the proposed design when exhaustively processing all possible input cases (e.g., 256×256 combinations for the case of processing two 8-bit precision input data). As shown, by increasing the number of inputs and the precision of data, a higher reduction in the number of processing cycles is achieved. Figure 3.13 shows the histogram distribution of the resulting bit-stream lengths for the case of processing two 8-bit precision data using the proposed context-aware designs.

Tables 3.4 and 3.5 demonstrate the performance of the proposed architectures for error-tolerant applications. All three proposed designs achieve significant reductions in the processing time when accepting small rates of error in the computation. For

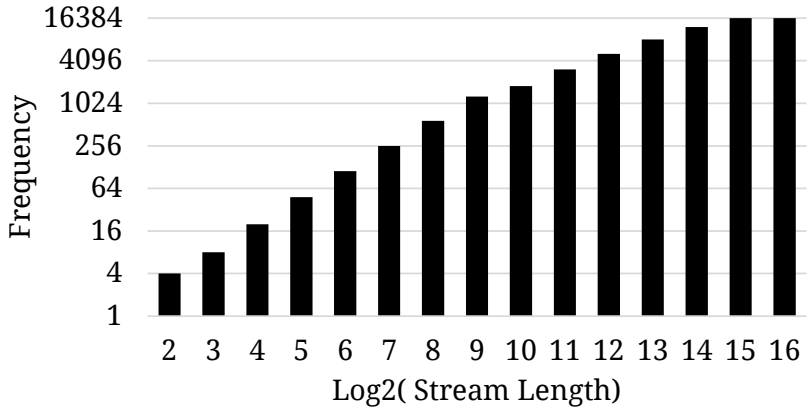
Table 3.3. Performance improvements with the proposed designs.

# of Inputs	Input Precision	Conventional Design # of cycles	Proposed Design Percentage of reduced cycles
2 Inputs	4-bit	2^8	54
	8-bit	2^{16}	55
	16-bit	2^{32}	55
3 Inputs	4-bit	2^{12}	69
	8-bit	2^{24}	70
	16-bit	2^{48}	70
4 Inputs	4-bit	2^{16}	79
	8-bit	2^{32}	80
	16-bit	2^{64}	80
5 Inputs	4-bit	2^{20}	86
	8-bit	2^{40}	86
	16-bit	2^{80}	86

Table 3.4. Performance improvement with the proposed Clk Div and Rotation designs for error tolerant applications. *The error rates are the maximum error for the multiplication operation.

# of Inputs	Input Precision	Conv. Design # of Cycles	# of ignored LSB bits	Proposed Design Latency Reduction	Error Rate* (%)
2 Inputs	4-bit	2^8	1	8×	11.0
	8-bit	2^{16}	1	9×	0.7
			2	35×	2.0
			3	142×	5.0
			4	555×	11.0
	16-bit	2^{32}	1	9×	3E-03
			2	23×	9E-03
			3	75×	2E-02
			4	277×	4E-02
			5	1000×	9E-02
			6	3333×	0.2
3 Inputs	4-bit	2^{12}	1	24×	15.0
	8-bit	2^{24}	1	27×	1.0
			2	100×	3.0
			3	625×	7.0
			4	3333×	16.0
	16-bit	2^{48}	1	27×	4E-03
			2	110×	1E-02
			3	625×	3E-02
			4	3333×	6E-02
			5	34482×	0.1
			6	250000×	0.2

Figure 3.13. Histogram distribution of the resulting bit-stream lengths for the case of processing two 8-bit precision data using the proposed context-aware designs.



example, for the case of multiplying two 16-bit precision inputs in an application with an error tolerance of 0.20 percent, the proposed clock division and rotation designs achieve 3.3×10^4 times reduction in the number of processing cycles, and the proposed Sobol-based design is capable of decreasing the number of processing cycles by 6.6×10^4 times on average compared to the conventional design with a processing time of 2^{32} clock cycles.

3.2.3 Application Case Study: Gamma Correction

To further evaluate the performance of the proposed designs we implemented the Gamma Correction circuit proposed in [33]. Figure 3.14 shows the implemented architecture.This circuit approximates the gamma correction function ($F(x) = X^{0.45}$) by mapping the function into a degree-6 Bernstein polynomial. Six independent bit-streams, all representing the same input value of X, must be generated and summed to select one of the inputs of the multiplexer (MUX) in each cycle. The inputs of the MUX are seven correlated bit-streams corresponding to the Bernstein Coefficients of

Table 3.5. Performance improvement with the proposed Sobol design for error tolerant applications.

Input Precision	Conv. Design # of cycles	Proposed Design Latency Reduction	Error Rate (%)
4-bit	2^8	5.5×	3
		15×	9
8-bit	2^{16}	5.7×	0.2
		19×	0.5
		70×	1.13
		264×	2.93
		1000×	6
16-bit	2^{32}	5.7×	7E-04
		19×	2E-03
		70×	5E-03
		264×	1E-02
		1000×	2E-02
		3333×	4E-02
		16393×	9E-02
		66666×	0.2
		2500000×	0.3

the Gamma Correction function, i.e., $b_0 = 0.0955$, $b_1 = 0.7207$, $b_2 = 0.3476$,, $b_3 = 0.9988$, $b_4 = 0.7017$, $b_5 = 0.9695$, and $b_6 = 0.9939$. Correlated bit-streams can be generated by sharing the same number source between the bit-stream generators. Energy consumption of the Gamma Correction circuit with the proposed and with the conventional bit-stream generator are reported in Table 3.6. We report the energy for processing four different 512×512 input test images. Each pixel of the input test image is converted to six independent bit-streams (i.e., X_1-X_6) using the bit-stream generator and processed by the Gamma Correction stochastic circuit. As shown, the proposed design reduces the energy consumption more than 2×.

Figure 3.14. Gamma Correction Stochastic Circuit [33]

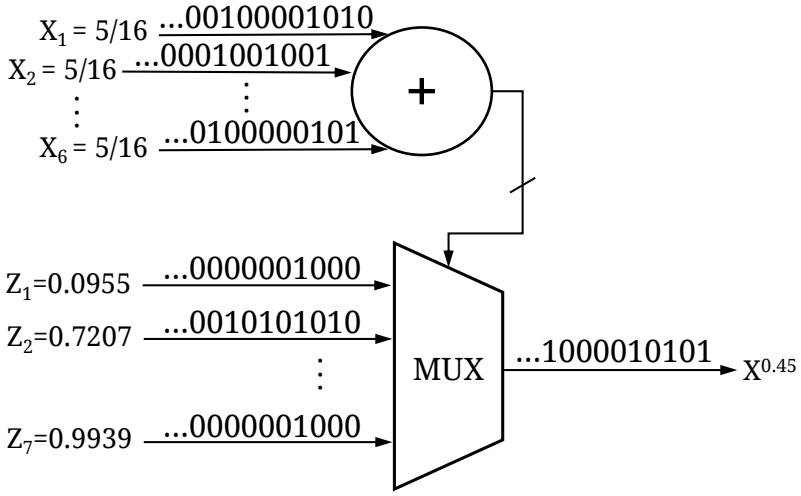


Table 3.6. Energy Consumption of the Gamma Correction Stochastic Circuit

Image Name	Conventional Design		Proposed Design	
	# of Required Cycles	Total Energy (KJ)	Total Energy (KJ)	Percentage of Reduction
Lena	$2^{48}\times512\times512$	23,415	12,399	53
Cameraman			12,355	53
Livingroom			12,460	53
Mandrill			12,358	53

3.3 Conclusion

In this dissertation, we proposed three context-aware architectures to accelerate the three state-of-the-art deterministic methods of SC. The proposed designs employ a control unit to extract the minimum bit-width required to precisely represent each input data. The lengths of bit-streams are reduced to the minimum required lengths to precisely represent each data. The noise-tolerance property of the system is preserved as each bit-flip can only introduce a least significant bit error. We showed that the proposed designs achieve a considerable improvement in the processing time at a reasonable hardware cost overhead. The proposed designs make the deterministic

methods of SC more appealing for applications that expect accurate computation and also for error-tolerant applications.

Chapter 4: A Low-Cost FSM-based Bit-Stream Generator for Low-Discrepancy Stochastic Computing[7][8]

4.1 Proposed LD Bit-Stream Generator

An LD bit-stream generator converts an n-bit precision binary data into a 2^n-bit bit-stream with uniformly spaced 1's and 0's. x_i or the i^{th} bit of binary input X appears in the LD bit-stream exactly 2^i times. We connect the binary input data to the main inputs of an $(n+1)$-to-1 multiplexer (MUX). A 2^n-state FSM is connected to the select input of the MUX to select one of the input bits at any cycle. The FSM controls the order of bit selection and the number of times each input bit is selected. Different LD bit-selection orders are needed to generate independent LD bit-streams. Figure 4.1 depicts the structure of the proposed bit-stream generator for n=3. In what follows, we discuss how the bit selection orders are determined for the proposed bit-stream generator. This chapter's material has been published in [8] and [7].

Prior work [24][28] showed that, among different types of stochastic bit-streams, the Sobol sequence-based bit-streams provide the fastest convergence to the target value. Independence between different Sobol-based LD bit-streams is provided by using

Figure 4.1. Proposed FSM-based LD bit-stream generator for n=3. The FSM selects the input bits based on one of the patterns produced by Algorithm 1.

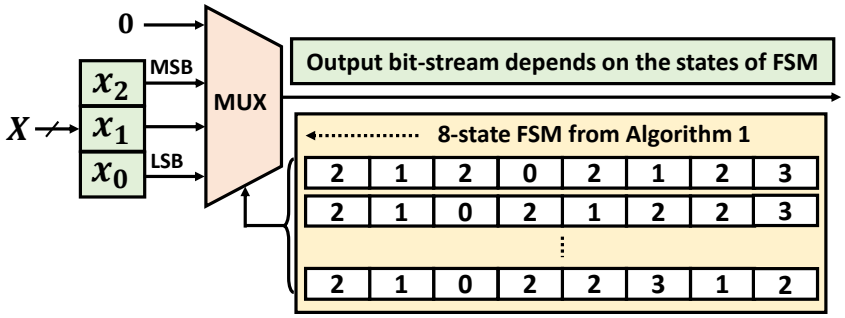


Table 4.1. MAE (%) Comparison of the proposed and the prior stochastic designs when multiplying two 8-bit precision data

Design \Number of Cycles	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}
Comparator-based non-LD [14]	0.05	0.15	0.26	0.39	0.58	0.79	1.20	1.67	2.32	3.32	4.72	6.62
Comparator-based LD [24][30]	0.00	0.0003	0.0013	0.0035	0.009	0.019	0.04	0.09	0.19	0.45	0.92	1.85
Sim's FSM-based LD [37]	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	12.60	18.85	21.99
Proposed FSM-based LD	**0.00**	**0.0003**	**0.0013**	**0.0034**	**0.007**	**0.017**	**0.03**	**0.07**	**0.16**	**0.35**	**0.79**	**1.36**

Algorithm 1: Constructing an FSM based on a Sobol sequence

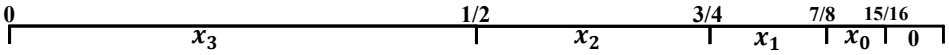
Input: Sobol seq (Sobol-num $[0 : 2^n - 1]$), data-width (n)
Output: A 2^n-state FSM
for $k = 0, 1, \ldots, 2^n - 1$ **do**
  if $0 \leq$ *Sobol-num(k)* $< 1/2$
    FSM output $= n - 1$
  else if $1/2 \leq$ *Sobol-num(k)* $< 3/4$
    FSM output $= n - 2$
  .
  .
  else if $(2^{n-1} - 1)/2^{n-1} \leq$*Sobol-num(k)*$< (2^n - 1)/2^n$
    FSM output $= 0$
  else
    FSM output$= n$

different Sobol sequences in converting input data into bit-stream representation [30]. Here, we propose an algorithm to determine the order of bit selection by the FSM of our bit-stream generator based on the distribution of numbers in the Sobol sequences. An independent LD bit-stream is generated by setting up the FSM using a different Sobol sequence. Note that this step is performed offline, and the structure of the FSM will not change after implementation.

Algorithm 1 demonstrates the procedure. Each Sobol number from a Sobol sequence determines one state of the FSM. Assume S_k is the k^{th} number of the Sobol sequence. If $(2^{m-1} - 1)/2^{m-1} \leq S_k < (2^m - 1)/2^m$ (where $m = 1, 2, .., n$), x_{n-m} or the

Figure 4.2. First 16 numbers of two Sobol sequences and their corresponding selected bits in a 4-bit data based on Algorithm 1.

Sobol Seq. 1	0	1/2	1/4	3/4	1/8	5/8	3/8	7/8	1/16	9/16	5/16	13/16	3/16	11/16	7/16	15/16
Selected bit	x_3	x_2	x_3	x_1	x_3	x_2	x_3	x_0	x_3	x_2	x_3	x_1	x_3	x_2	x_3	0
Sobol Seq. 2	0	1/2	3/4	1/4	5/8	1/8	3/8	7/8	15/16	7/16	3/16	11/16	5/16	13/16	9/16	1/16
Selected bit	x_3	x_2	x_1	x_3	x_2	x_3	x_3	x_0	0	x_3	x_3	x_2	x_3	x_1	x_2	x_3



$(n-m)^{th}$ bit of binary input X should be selected by the FSM. For example, if $1/2 \leq S_k < 3/4$, m is 2, and the $(n-2)^{th}$ bit of the input data should be selected by the FSM. Assume a 4-bit binary data is to be converted to a 16-bit LD bit-stream. Figure 4.2 shows the first 16 numbers of two different Sobol sequences and their corresponding selected bits (i.e., FSM states) based on Algorithm 1.

4.2 Evaluation

4.2.1 Accuracy

We evaluate the accuracy of the proposed bit-stream generator compared to the state-of-the-art LD bit-stream generators (Sim's design and comparator-based design) and also to the conventional comparator-based non-LD (pseudo-random) generator by exhaustively testing the multiplication of two 8-bit precision data. For the non-LD generator, two different 16-bit LFSRs are used as the number sources. For Sim's design, a 256-state design of the FSM-based LD generator of [37] and a unary bit-stream generator (built from a pair of 8-bit counter and comparator) are used as elaborated in [37] to convert the two inputs. The comparator-based LD [24, 30] and the proposed FSM-based generator use the first and the second Sobol sequences from the MATLAB built-in Sobol sequence generator as their LD number sources. Table 4.1 compares the

mean absolute errors (MAEs) of these designs for different bit-stream lengths. We multiply the measured MAE of each design by 100 and report it as a percentage.

Due to random fluctuations in pseudo-random bit-streams and correlation between bit-streams, the non-LD design cannot provide comparable accuracy to LD designs. The output bit-streams produced by the Sim's design has a period of 2^8 cycles and so their accuracy never improves after 2^8 cycles. Sim's design achieves a higher accuracy (i.e., a lower error rate) than the non-LD only when processing bit-streams of 2^8 (or multiples of 2^8) bits. Unary bit-streams suffer from truncation error [28]. Hence, Sim's design converts the second input to a unary bit-stream shows poor results when truncating the bit-streams and processing bit-streams shorter than 2^8 bits (e.g., 2^7 bits). For small bit-stream lengths, both of the non-LD and the LD comparator-based designs show a better performance than Sim's design.

The authors in [30] proved that converting two n-bit precision numbers to two 2^{2n}-bit independent LD bit-streams leads to completely accurate multiplication results when logical-`AND`ing the generated bit-streams. The comparator-based and the proposed FSM-based LD designs implemented here both convert the input data into independent LD bit-streams. Consequently, as reported in Table 4.1, both of these designs are able to produce completely accurate results (zero error rate) when processing for $2^{16}(=2^{2\times 8})$ cycles. They also show low error rates when processing shorter bit-streams. However, as we show in Section 4.2.2 , the proposed design has significantly lower hardware cost for computation precisions not exceeding 12 bits. We discuss that for more precise computations the proposed FSM-based design can be

integrated with the rotation technique of [17] for hardware efficiency.

4.2.2 Hardware Cost

4.2.2.1 Single Bit-stream Generator

The hardware cost of the comparator-based LD generator depends on the required precision (bit-stream length) and is independent of the selected Sobol sequence. To generate an LD bit-stream of 2^n bit length, one n-bit Sobol sequence generator and one n-bit binary comparator are needed. Different Sobol sequences are generated by changing the values of direction vectors in the sequence generator [24]. The hardware cost of the FSM-based generator, however, not only depends on the precision but also depends on the selected LD pattern (i.e., selected Sobol sequence in Algorithm 1) [8]. This is because the structure of the FSM changes with a different bit-selection order. Figure 4.3 compares the hardware area cost of the proposed FSM-based LD generator for different precisions (n=4,8,12) and LD patterns (based on ten different Sobol sequences). We synthesize the designs using the Synopsys Design Compiler v2018.06 with the 45nm FreePDK gate library [1]. As it can be seen, the proposed generator provides on average 56, 55, and 51 percent hardware area saving for 4-, 8-, and 12-bit precision bit-stream generation, respectively, compared to the comparator-based LD generator.

Figure 4.3. Hardware area cost of the proposed LD bit-stream generator for different precisions (4, 8, and 12 bits) and LD patterns (Sobol 1-10).

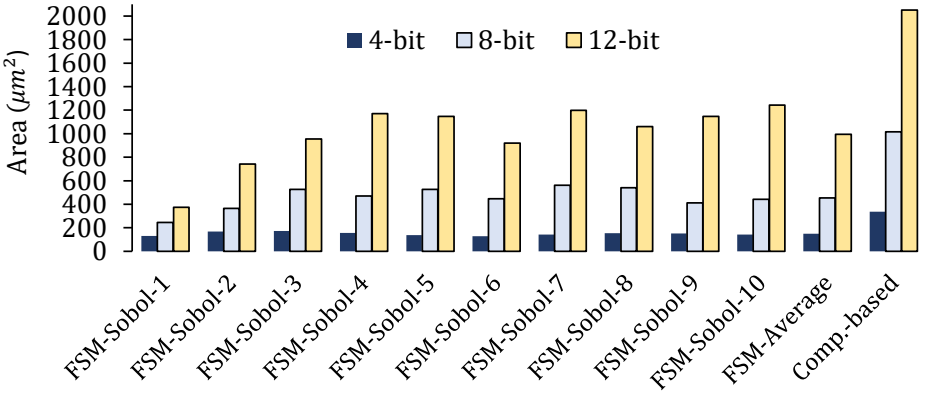


Table 4.2. Hardware area cost (μm^2) of the bit-stream generators for processing different numbers of inputs (i) and data precisions (n).

Design	n=4 i=2	n=4 i=3	n=4 i=4	n=8 i=2	n=8 i=3	n=8 i=4
Comparator-based LD (Limited)	366	627	888	1087	1944	2801
This work - FSM-based LD (Limited)	**188**	**280**	**337**	**451**	**781**	**1026**
Comparator-based LD (Full) [24, 30]	1005	3740	9127	3361	13193	32406
This work - FSM-based LD (Full)	**300**	**746**	Large	Large	Large	Large
Comp-based LD + Rotation (Full) [30]	456	806	1156	1277	2324	3371
This work FSM LD + Rotation (Full)	**353**	**609**	**864**	**751**	**1495**	**2149**

4.2.2.2 Multiple Bit-stream Generators

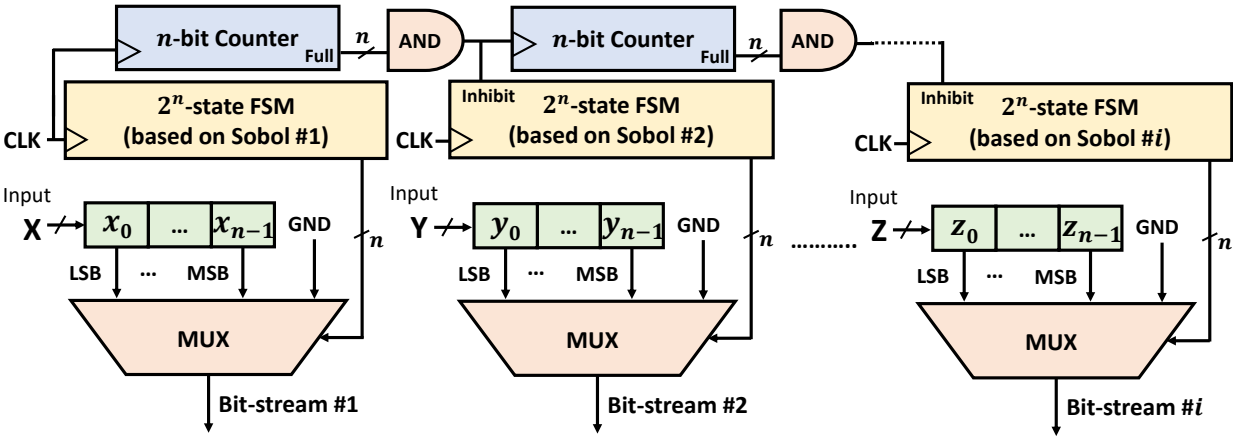
The hardware area cost of the proposed bit-stream generator and the state-of-the-art comparator-based LD generator [24, 30] are compared in Table 4.2 for implementing stochastic multipliers with different number of inputs (i=2, 3, 4) and data precisions (n=4, 8). We implemented three different designs for each case: one limited-precision design that produces n-bit precision output (2^n-bit output bit-stream) and two full-precision designs that produce $(i \times n)$-bit precision output ($2^{i \times n}$-bit output

bit-stream). The proposed bit-stream generator provides significantly lower hardware cost for all limited-precision designs and also for the full-precision designs with at most 12-bit output precision. A weakness of the proposed generator is its high hardware cost for output precisions exceeding 12 bits. This is because producing an $(i \times n)$-bit precision output requires implementing a separate $2^{i \times n}$-state FSM for each input. Although such high output precision (i.e., $\geq$ 12 bits) is rarely needed in today's common applications such as neural networks and image processing, we integrate our design with the rotation method of [17] for high-precision bit-streams.

Figure 4.4 shows the structure of our generator integrated with the rotation method. This integration allows to generate i $2^{i \times n}$-bit bit-streams by using i 2^n-state FSMs (instead of i $2^{i \times n}$-state FSMs). This significantly reduces the hardware cost at no accuracy loss while producing full-precision output [28]. The rotation technique guarantees a full-precision output by rotating the bit-streams by inhibiting or stalling on powers of the stream lengths. The output of the first FSM repeats every 2^n cycles and does not rotate. Other FSMs (FSM #k=2,3,..,i) have a period of 2^n but rotate every $2^{(k-1) \cdot n}$ cycles by inhibiting. As reported in Table 4.2, compared to the rotation-based design of the compared-based generator [30], our rotation-based design provides a lower hardware cost in all cases.

According to Table 4.2, the proposed bit-stream generator provides up to 80% saving in the hardware area cost. Considering the fact that for the same number of processing cycles the proposed generator provides a better or the same level accuracy as the comparator-based one (see Table 4.1), the proposed design achieves more than 80%

Figure 4.4. Integrating the FSM-based LD bit-stream generator and the rotation method of [17] to generate i independent $2^{i\times n}$-bit bit-streams.



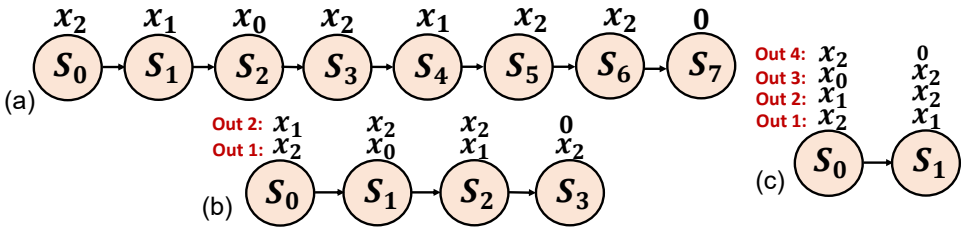
savings in the area-delay product.

4.2.2.3 Parallel Bit-stream Generator

Parallelization has been used to mitigate the long latency of SC at the cost of higher hardware area and power consumption. Liu and Han [25] developed a hardware efficient parallel Sobol sequence generator that can generate multiple Sobol numbers in each clock cycle at the cost of some additional `XOR` gates. For an $M\times$ parallel comparator-based LD bit-stream generator, an $M\times$ parallel Sobol generator and M comparators are needed. This design reduces the number of processing cycles by a factor of M by generating M LD bit-streams of length $2^N/M$ in parallel [25]. A reasonable increase in the hardware cost but $M\times$ reduction in the latency makes the parallel design of the comparator-based LD generator an attractive alternative to the non-parallel implementation.

We also develop a parallel design for the proposed bit-stream generator. In

Figure 4.5. An example of the FSMs for converting a 3-bit precision data to 8-bit LD bit-stream: a) non-parallel with 8 states b) 2× parallel with 4 states c) 4× parallel with 2 states. Each state is processed in one clock cycle.



contrast to the non-parallel design which has 2^N states, the $M\times$ parallel design has $2^N/M$ states. Each FSM state in the non-parallel design selects one bit of the input data. Each state in the $M\times$ parallel design, however, selects *at most* M bits of the binary input to generate M output bits in parallel. Figure 4.5 exemplifies the case of converting a 3-bit data into an 8-bit LD bit-stream. Figure 4.5.a shows the FSM of the non-parallel design. In this design, one input/output bit is selected/generated at any cycle. Conversion with this design takes eight cycles. The FSM of the 2× parallel design is shown in Figure 4.5.b. At each cycle, two output bits are generated, which reduces the number of processing cycles from eight to four. Finally, the FSM of the 4× design, shown in Figure 4.5.c, produces four output bits at any cycle, which reduces the processing time to only two clock cycles.

Table 4.3 compares the hardware cost of the 2×, 4×, and 8× parallel FSM-based LD generator (implemented based on the first five Sobol sequences from the MATLAB Sobol sequence generator) and the state-of-the-art parallel comparator-based LD generator [25] for the case of converting 8-bit data to 2^8-bit bit-stream. As it can be seen, both the non-parallel (1×) and the parallel designs of the FSM-based generator

Table 4.3. Hardware Area (μm^2) and Critical Path Latency (CP) (nS) of the Comparator-based and the FSM-based LD Generator for the case of converting 8-bit precision data

Parallel Design	1×		2×		4×		8×	
	Area	CP	Area	CP	Area	CP	Area	CP
Comparator-based [25]	1013	0.45	1245	0.47	1658	0.47	2484	0.47
FSM-based (Sob.1)	246	0.37	266	0.35	267	0.34	250	0.30
FSM-based (Sob.2)	366	0.39	375	0.42	422	0.41	455	0.39
FSM-based (Sob.3)	526	0.42	504	0.43	512	0.40	485	0.39
FSM-based (Sob.4)	471	0.40	483	0.43	513	0.39	485	0.37
FSM-based (Sob.5)	526	0.42	479	0.41	483	0.41	487	0.39

provide significantly lower hardware cost than their corresponding comparator-based design. The hardware cost saving provided by the FSM-based design increases when increasing the level of parallelism. On average, the 2×, 4×, and 8× design of the FSM-based generator achieves 66, 73, and 82 percent saving, respectively, compared to the corresponding comparator-based generator.

An interesting property of the proposed FSM-based design is that a higher level of parallelism can be achieved with no considerable increase in the hardware cost. In some cases, the area is even reduced with more parallelism. For instance, the non-parallel design of the FSM-based generator implemented based on Sobol sequence 1 takes an area footprint of 246 μm^2 while its 2×, 4×, and 8× parallel designs cost 266 μm^2, 267 μm^2, and 250 μm^2 area, respectively. This happens because by increasing the level of parallelism 1) the number of states decreases and 2) the same input bit is selected for more than one output bit (e.g., x_2 in the FSM of Figure 4.5.c).

4.2.3 Fault-Tolerance

Fault tolerance is one of the attractive properties of SC designs [4, 34]. The processing elements of SC systems inherently tolerate high rates of soft errors (i.e., bit flips) as they process data in the stochastic domain. However, the bit-stream generators that convert binary data to stochastic bit-streams are vulnerable to bit flips as they operate in the binary domain. Here, we compare the fault tolerance of the proposed FSM-based and the comparator-based generator of [24] when converting input data with different precisions (n = 4, 8, and 12 bits) to LD bit-streams with corresponding lengths (2^4, 2^8, and 2^{12} bits). Soft errors are injected by flipping bits in the input and output bits of different components of the bit-stream generator, including the storage array of the Sobol generator for the comparator-based and the states of the FSM for the FSM-based generator. Table 4.4 shows the results. Evidently, increasing the precision reduces the error rate (improves fault tolerance) in both the proposed and the comparator-based generators. This is because longer bit-streams are generated for higher precisions, and longer bit-streams have a higher tolerance to bit flips [33, 22].

The proposed FSM-based generator shows a slightly lower tolerance to soft errors compared to the comparator-based design. This is due to the high sensitivity of FSMs to changing their state caused by bit-flips. As it can be seen in the reported numbers of Table 4.4, the difference between the MAEs of the two LD generators increases when increasing the fault injection rate. However, the hardware cost saving provided by the proposed generator can be exploited to improve its tolerance to soft error by using additional techniques of improving fault tolerance such as the N-modular

Table 4.4. MAE (%) Comparison of Different LD Bit-stream Generators when Injecting Different Rates of Soft Error.

			Injected Soft Error					
LD Bit-Stream Generator	Area (μm^2)	Input Width	1%	2%	5%	10%	20%	30%
Comparator-based	337	4	3.8	5.4	7.5	9.4	13.3	17.5
	1013	8	1.1	1.6	3.0	5.3	10.2	15.1
	2054	12	0.56	1.04	2.5	5.0	10.0	15.0
Comparator-based + 3-MR	1179	4	0.18	0.68	3.0	6.0	9.6	14.0
	3349	8	0.24	0.53	1.02	2.1	5.5	11.3
	6624	12	0.06	0.12	0.38	1.4	5.3	10.6
Proposed FSM-based	165	4	3.9	4.8	7.3	11.0	17.2	21.5
	366	8	1.4	2.6	6.0	10.8	17.7	21.8
	742	12	1.3	2.5	6.0	10.8	17.7	21.7
Proposed FSM-based + 3-MR	587	4	0.11	0.42	2.1	4.9	9.9	16.0
	1266	8	0.14	0.34	1.00	2.9	9.3	16.5
	2377	12	0.05	0.18	0.95	3.5	11.3	18.2
Proposed FSM-based + 5-MR	1051	4	0.006	0.04	0.46	2.3	7.1	13.2
	2252	8	0.005	0.04	0.33	1.1	5.6	13.4
	4090	12	0.004	0.02	0.19	1.1	6.6	15.5

redundancy (N-MR) [18].

For the comparator-based design we evaluate a 3-MR design by implementing three identical copies of the main components of the generator and using majority gates to vote between them. For the FSM-based design we implement a 3-MR and a 5-MR design. Table 4.4 compares the hardware area cost and the MAE of the implemented generators for different noise injection rates. Clearly, implementing the N-MR technique has improved the fault tolerance of the generators. For example, for the case of injecting 1% soft error, the MAEs of the 5-MR implementations of the FSM-based

generator are all below 0.01%, which shows over three orders of magnitude reduction in the error rate compared to the non-redundant implementation. The reported area numbers show that the 5-MR design of the FSM-based generator has a lower hardware cost than the 3-MR design of the comparator-based generator while achieving significantly lower error rates for noise injection rates below 10%. For higher injection rates, the comparator-based generator shows higher tolerance to noise.

The high hardware cost of the Sobol sequence generator in the comparator-based design makes it difficult for the designer to exploit techniques such as N-MR to improve soft error tolerance. However, supported by the area and MAE numbers reported in Table 4.4, the low-cost advantage of the proposed LD generator allows us to use additional techniques to improve the soft error tolerance of the bit-stream generator in the SC system.

4.3 Case Study: Convolution Design

LD bit-streams has been recently used for accurate and energy-efficient design of SC-based convolutional neural networks (CNNs) [13, 37, 19, 21, 36]. To further evaluate the effectiveness of using the proposed bit-stream generator, we compare the cost of LD bit-stream generation in SC design of convolution functions with different sizes of $k \times k$ (k =3, 5, 7, 9, and 11). 8-bit precision data is converted from binary radix to LD bit-streams and fed to the convolution design. In convolution, pairs of input data are first multiplied, and then the results are accumulated. For a higher output accuracy, the state-of-the-art SC convolution designs implement the multiplication operations in the stochastic domain (using `AND` gates) but perform the accumulation in the binary

domain using binary adders [13, 21]. Since the accumulation is performed in the binary domain, the outputs of the multiplication operations do not need to be independent. This permits to reuse of two LD patterns to convert all input data to bit-stream representation. We evaluate three LD bit-stream generation approaches:

A1. Comparator-based: Each input data is compared with a Sobol number using a separate binary comparator. Two different Sobol sequences are needed to provide the two required LD patterns. To minimize the cost of generating the two Sobol sequences, the first sequence is generated by simply reversing the output bits of a binary counter [30]. The second Sobol sequence is generated by implementing the Sobol number generator described in [24]. So, the comparator-based approach consists of one 8-bit binary counter, one 8-bit Sobol generator, and $k \times k$ 8-bit comparators to convert the input data.

A2. Proposed FSM-based: Each input data is connected to the main inputs of a separate 9-to-1 MUX unit. Two 256-state FSMs, each implemented based on a different Sobol sequence, are connected to the select input of the MUX units. The two input data of each multiplication operation are connected to two separate MUX units, while the select input of each MUX is fed with one of the two FSMs. So, the FSM-based design consists of two 256-state FSMs and $k \times k$ 9-to-1 MUX units.

A3. Proposed FSM + One-Hot Encoder: We also implement a third design to further reduce the cost of generating LD bit-streams for applications such as the targeted convolution that a few FSMs (in our case, two) is reused in converting a large number of inputs. In this approach, we need a separate pair of FSM and one-hot

Figure 4.6. The Probability Conversion Circuit (PCC).

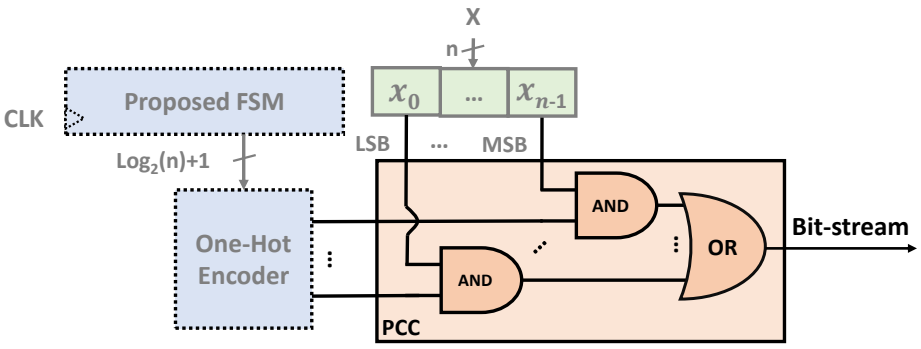
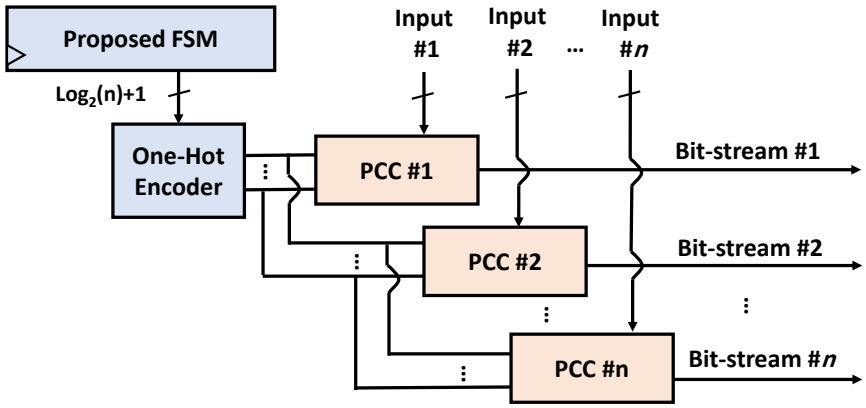


Figure 4.7. Converting a large number of inputs from binary to LD bit-stream representation by sharing one one-hot encoder and one FSM.



encoder for each LD pattern. Converting each input also requires a *Probability Conversion Circuit (PCC)* made of standard `AND` and `OR` gates. Figure 4.6 shows the design of the PCC unit. PCC has a lower hardware cost than MUX. Hence, when converting a large number of inputs, using PCCs instead of MUXs results in a considerable hardware cost saving. Figure 4.7 shows the conversion circuit for converting n input data by sharing one FSM and one one-hot encoder. For the targeted convolution design, two 256-state FSMs (each implemented based on a different Sobol sequence), two one-hot encoders, and $k \times k$ PCCs are needed.

Table 4.5. Synthesis Results of the Bit-stream Generators for the Implemented Convolution designs

LD Bit-stream Generator Method	Conv. Size	Area (μm^2)	Critical Path Latency (nS)	Power@Max Freq. (mW)	Energy per Cycle (pJ)
Comparator-based	3×3	2060	0.47	2.91	1.37
	5×5	4154	0.49	4.70	2.31
	7×7	7405	0.5	7.27	3.63
	9×9	9938	0.52	10.82	5.63
	11×11	14326	0.53	14.69	7.79
Proposed FSM-based	3×3	1455	0.42	1.73	0.73
	5×5	2943	0.44	2.79	1.23
	7×7	5175	0.46	4.22	1.94
	9×9	8151	0.47	6.09	2.86
	11×11	11871	0.47	8.57	4.02
Proposed FSM + One-Hot Encoder	3×3	1183	0.42	1.69	0.71
	5×5	2054	0.44	2.25	0.99
	7×7	3400	0.46	3.22	1.48
	9×9	5133	0.47	4.30	2.02
	11×11	7333	0.47	5.44	2.56

Note that we do not compare with Sim's bit-stream generation approach of [37] as it generates one LD and one unary bit-stream while here we need two independent LD bit-streams. The described bit-stream generation approaches are implemented for different convolution sizes of 3×3, 5×5, 7×7, 9×9, and 11×11 using Verilog HDL and synthesized using the Synopsys Design Compiler v2018.06-SP2 with the 45nm FreePDK library [1]. The synthesis results are reported in Table 4.5. As can be seen, the proposed FSM + one-hot encoder design provides the minimum bit-stream generation cost by reducing the hardware area cost up to 54% compared to the comparator-based design. Critical path latency and power consumption are also

reduced with this approach. Energy consumption is further decreased up to 67% compared to the comparator-based design. This reduction in hardware cost is expected to have a significant impact on the hardware efficiency of the SC-based CNNs such as the ones developed in [21], [37], [19], and [13].

4.4 Conclusion

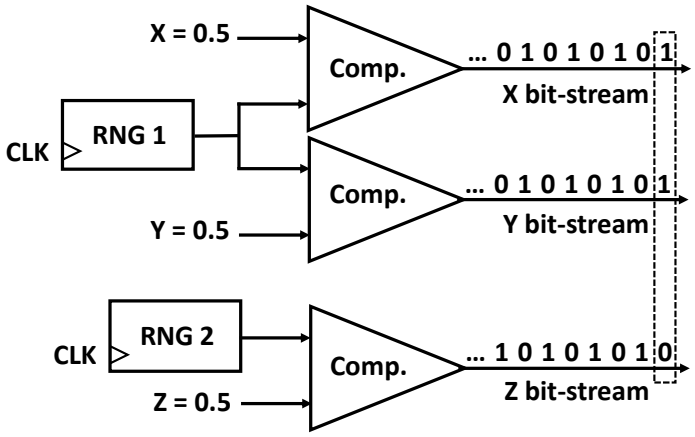
LD bit-streams have shown the best performance among all types of stochastic bit-streams. The state-of-the-art LD bit-stream generators are costly and not efficient for all SC designs. In this dissertation, we proposed a low-cost FSM-based LD bit-stream generator for SC designs that need multiple independent bit-streams. The proposed generator reduces the hardware cost up to 80% while generating accurate bit-streams. The parallel design of our bit-stream generator provides on average 66 percent area saving for the 2× and 82 percent area saving for the 8× parallel implementation compared to the state-of-the-art parallel LD bit-stream generator. We showed that the area saving provided by the proposed generator can be exploited to improve the fault-tolerance of the bit-stream generator, a vulnerable component in the SC systems. For noise injection rates below 10 percent, the 5-MR design of the proposed generator provides orders of magnitude reduction in the error rate at a lower hardware cost than the 3-MR. By evaluating the overhead cost of bit-stream generation for SC convolution design, significant area and energy consumption savings were achieved by using the proposed bit-stream generator. A new design for further cost reduction of the FSM-based LD bit-stream generator was also developed for the case of generating a large number of bit-streams.

Chapter 5: In-Stream Correlation Manipulation for Low-Discrepancy Stochastic Computing[9]

SC treats data as probabilities presented by streams of random bits. The value of an SC bit-stream is determined by the ratio of 1's in the bit-stream, i.e., a bit-stream with P ones and length L represents P/L value. For example, 10011000 with P=3 and L=8 represents 3/8 in the stochastic domain. This unconventional method of representing data leads to extremely simple computation circuits for complex arithmetic operations. For example, multiplication operation can be realized by bit-wise `AND`ing stochastic bit-streams [33]. But this simple method of multiplying data is accurate only if the input bit-streams are statistically *independent* or *uncorrelated*. Bit-wise `AND`ing two correlated bit-streams with high overlap between the position of 1's in the bit-streams gives the minimum of the two bit-streams and not their product. Correlation, hence, plays an important role in correct functionality of SC circuits [20, 2, 32].

Stochastic operations such as minimum using bit-wise `AND`, maximum using bit-wise `OR`, absolute difference using bit-wise `XOR`, and division [10] are examples of operations that need positively correlated inputs. When there is no or minimum overlap between the bit positions of 1's in the bit-streams, the bit-streams are called *negatively correlated*. For example, 101100 and 010011 are two negatively correlated bit-streams. Saturating addition using bit-wise `OR` is an example of a stochastic operation that needs negatively correlated inputs [20]. Operations such as multiplication using bit-wise `AND` and scaled addition using multiplexer need *uncorrelated* inputs. For example, 1010 and 1001 are two uncorrelated bit-streams both

Figure 5.1. Example of generating correlated and uncorrelated bit-streams. X and Y bit-streams are correlated, and both uncorrelated with bit-stream Z due to using the same and different RNGs in generating them.



representing 0.5 value. Bit-wise `AND`ing these two bit-streams gives 1000, the expected value (0.25), from multiplying the two inputs.

The common approach for implementing a converter unit is by using a binary comparator and a RNG. In each clock cycle, the input data in binary format is compared with a random number from the RNG. A 1 is produced at the output of the comparator if the random number is smaller than the input data. Correlation between input bit-streams can be controlled at this stage when the bit-streams are generated. Sharing the same RNG between different data conversion units results in generating correlated bit-streams. Using different RNG, on the other hand, leads to generating uncorrelated bit-streams. Figure 5.1 shows this using an example circuit.

Lee et al. [20] developed a novel synchronizer and desynchronizer circuit for increasing positive and negative correlation, respectively, between two bit-streams. Figure 5.2 depicts the synchronizer (correlator) circuit of [20] with depth size = 1.

Figure 5.2. Synchronizer circuit of [20] with depth = 1.

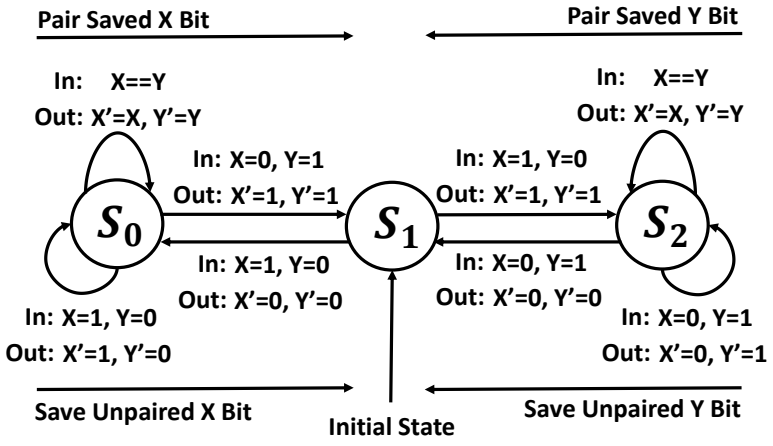
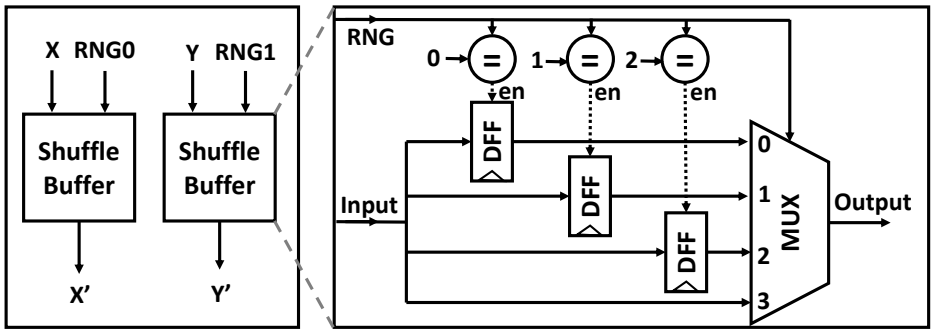


Figure 5.3. Decorrelator circuit of [20] with depth = 4.



Given two stochastic bit-streams X and Y, the synchronizer produces two bit-streams that are more positively correlated. They also developed a novel decorrelator circuit (Figure 5.3) to reduce correlation between two SC bit-streams. At each cycle, the decorrelator circuit either passes the current input bit or stores it in a shuffle buffer and emits a previously stored bit by scrambling the stored bits using an RNG. They use these circuits to propose improved SC maximum, minimum, and saturating adder designs. Wu and Miguel [39] also developed an in-stream correlation-based division and square root circuit by introducing a skewed synchronizer.

This work proposes an improved correlator that can increase positive correlation

between stochastic bit-streams. We focus on positive correlation and leave negative correlation for our future work, as positive correlation is more common and needed by more SC operations. We further develop a decorrelator circuit that can make input bit-streams uncorrelated with negligible impact on their value. An important advantage of our proposed circuits is that the output bit-streams have LD distribution [8]. With LD distribution, 1's and 0's are uniformly spaced. Random fluctuations are removed from bit-streams, and the bit-streams converge to target values significantly faster than the true- or pseudo-random bit-streams [24, 3]. This significantly improves the accuracy and reduce the processing time of SC operations. In summary, the main contributions of this work are as follows:

- A novel correlator and decorrelator circuit for in-stream correlation manipulation of stochastic bit-streams with no significant impact on the data values.
- The proposed circuit's output high-quality LD bit-streams irrespective of the distribution of the input bit-streams.
- The proposed circuits can manipulate the correlation of any number of inputs.
- The accuracy, performance, and hardware cost are all improved compared to those of the SoA correlation manipulation techniques.
- A FSM-based correlator circuit for SC division.
- Significant improvement in the accuracy and hardware cost of SC sorting and median filtering circuits.

This chapter's material has been published in [9]. The rest of the chapter is organized as follows: Section 5.1 demonstrates the proposed LD correlator. Section 5.2 introduces our LD decorrelator. Section 5.3 discusses our correlator circuit for SC division. Section 6.1 evaluates the accuracy and the hardware cost of the proposed techniques. Section 5.5 evaluates the proposed techniques in two application case studies. Finally, Section 6.2 concludes the work.

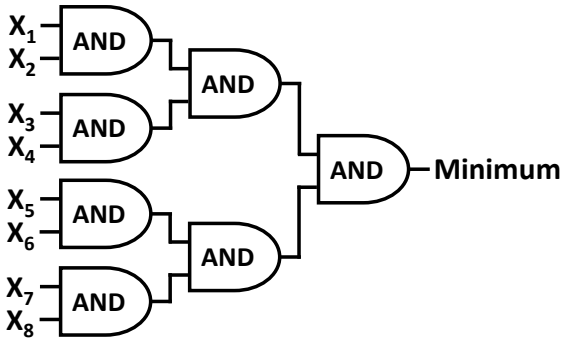
5.1 Proposed Low-Discrepancy Correlator

As we discussed, the common approach for generating correlated bit-streams is to use the same RNG in the data conversion units when generating bit-streams. When correlated bit-streams with LD distribution are desired the same Sobol [24] or Halton [3] number generator can be shared between the data conversion units to generate LD bit-streams that have maximum overlap between the bit positions of 1's, i.e., LD bit-streams with maximum positive correlation. But this approach is only applicable to the input stage of the SC system where the input data are converted from binary to bit-stream representation.

Consider a simple SC circuit, consisting of seven `AND` gates, that finds the minimum value between eight input numbers (Figure 5.4).

The data are split into four pairs of two numbers, each pair connected to one `AND` gate. Each `AND` gate finds the minimum of its two inputs. An advantage of the common approach of generating correlated bit-streams is that because correlated input bit-streams are directly generated, the output bit-streams from operations such as minimum using `AND` or maximum using `OR` are also correlated and so can directly be fed

Figure 5.4. 8-input SC Minimum Circuit



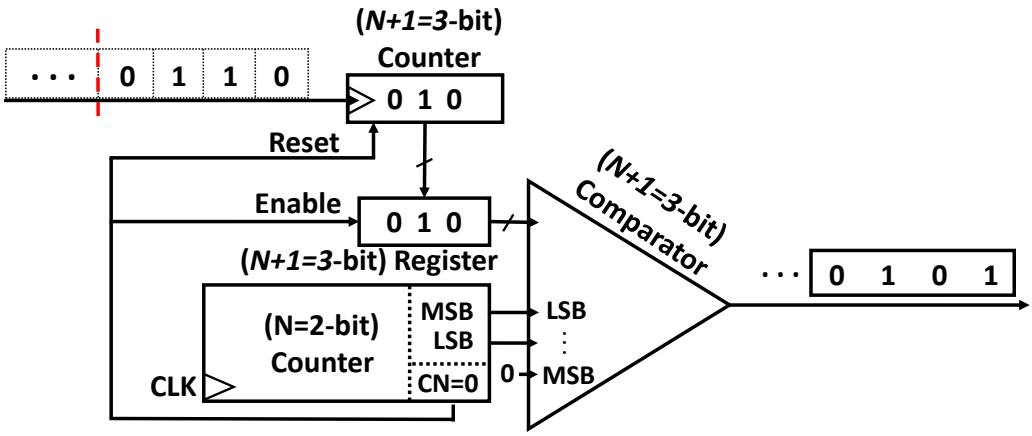
to the next stages (if correlated bit-streams are needed) with no need for any correlation manipulation. So, with this approach, the minimum of the four produced output bit-streams are found by connecting them to three `AND` gates in two more stages as shown in Figure 5.4. But the conventional approach can be used in this design example when the inputs are in the binary format and we have control over bit-stream generation.

The question is, what if the inputs to the SC system are already in the bit-stream form and they are uncorrelated, or the correlation level is unknown. A solution is to convert the bit-streams back to binary format and then re-generate them by sharing the same RNG. But this approach costs a significant latency, area, and power overhead. The in-stream synchronizer proposed in [20] can be used in such designs to increase the positive correlation between input bit-streams with no need for re-generating them. While the synchronizer can manipulate correlation at a lower cost compared to the bit-stream re-generation method, it has two weaknesses: 1) it lacks the advantage of the conventional approach; the output bit-streams from operations such as minimum and maximum operation are not necessarily correlated. An additional

synchronizer is needed to make any two output bit-streams correlated. This will lead to a significant correlation manipulation cost for SC systems with multiple stages of computation. 2) the bit-streams produced by the synchronizer are random and so suffer from random fluctuations [4]. Even in cases where the inputs to the synthesizer have LD distribution, there is no guarantee that the synchronizer provides the same uniform distribution in its output bit-streams.

Here we propose an in-stream correlator called CORLD-C that addresses the weaknesses of the SoA synchronizer technique. Figure 5.5 shows our proposed correlator with a FS of 2. The proposed correlator processes the input bit-stream in segments of 2^{FS} bits and converts the value of each segment into an LD bit-stream. CORLD-C consists of two counters (an input-controlled counter and an independent counter), a register, and a binary comparator. In the correlator of Figure 5.5, FS is 2, indicating that the number of bits that can be taken in each segment is 4. The circuit counts the number of 1's in each 4-bit segment of the bit-stream, stores the value in a register, and then restarts. While the top counter processes the next segment, concurrently, the bottom part converts the counted segment to bit-stream using of the bottom counter and comparator. A counter is a Sobol sequence generator if reversing its output bits [24]. So the output bits of the bottom counter are connected to the comparator in reverse order to compare the value stored in the register with a new Sobol number in each cycle. The comparator produces one bit of the final bit-stream in each cycle. Since there are at most 4 ones in each 4-bit segment, the proposed circuit should be able to count from 0 to 4 and then restart. Therefore, when the bottom

Figure 5.5. Proposed Correlator (CORLD-C) with FS=2

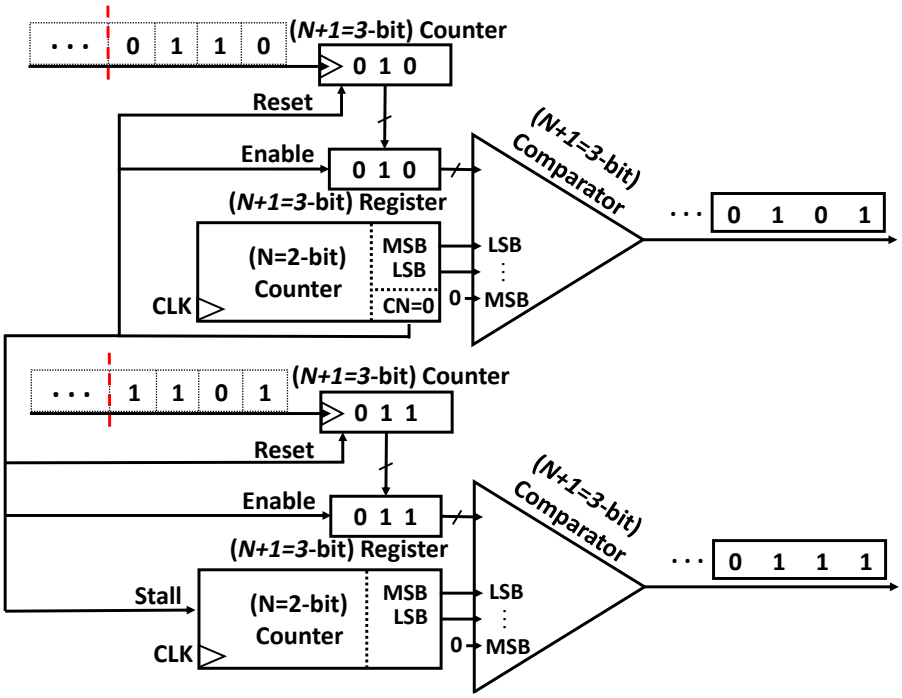


counter reaches its first state (CN=0), an *Enable* signal is set to copy the value of the top counter into the register, and then the top counter restarts. It should be noted that both *Enable* and *Reset* signals are clock edge sensitive; *Enable* is sensitive to the positive edge, and *Reset* changes with the negative edge of the clock signal. The output bit-streams produced by the proposed correlator are all LD and correlated as they are all manipulated based on the same Sobol sequence.

5.2 Proposed Low-Discrepancy Decorrelator

Conventionally, different RNG are used in the data conversion units to generate uncorrelated (independent) stochastic bit-streams. By comparing the input data with Sobol numbers from different Sobol sequences, the data can be converted to independent LD bit-streams [24, 7]. But similar to the discussion on correlation, this method can only be used and is efficient if we have control over the data conversion units. If the input bit-streams are already generated, e.g., they are the outputs from other stochastic circuits, and our SC system requires uncorrelated input bit-streams, in-stream decorrelator circuits are needed to minimize correlation between input

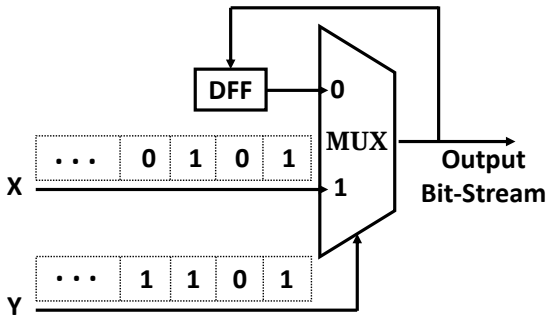
Figure 5.6. Proposed Decorrelator (CORLD-D) with FS=2



bit-streams and guarantee correct functionality of the system.

Figure 5.6 shows our proposed in-stream decorrelator, CORLD-D, with $FS = 2$. The proposed circuit consists of binary counters, registers, and comparators, and is capable of converting two bit-streams, whether they are correlated or not, into two independent LD bit-streams. The input bit-streams are segmented into sub-parts of 2^{FS} bits and processed by the decorrelator circuit to produce independent bit-streams. The first bit-stream is manipulated using the same approach utilized in CORLD-C. The second bit-stream is manipulated using a similar approach, but with this difference that the bottom counter, which is connected to the comparator, is halted for one clock cycle after each round of counting the bits. This is inspired by the rotation technique of [28].

Figure 5.7. CORDIV SC Division Circuit [10]



5.3 Correlation and SC Division

Before evaluating the proposed correlator and decorrelator, we discuss the SoA correlation-based SC division circuits, CORDIV [10] and ISCBDIV [39], and propose a correlator circuit to improve the performance of the CORDIV design.

Chen and Hayes developed a novel SC division circuit, called CORDIV [10], that exploits the correlation between input bit-streams to realize division operation. The architecture of CORDIV is shown in Figure 5.7. The result of X/Y falls outside [0,1] – the accepted range of numbers in SC – if $X > Y$. So for the SC division circuit, it is assumed that the divisor Y is greater than the dividend X. CORDIV requires that the X and Y bit-streams are positively correlated. Our simulations show that the MAE rate of the division operation when connecting two 2^8-bit correlated LD bit-streams (generated by sharing the simplest Sobol sequence as the source of the random number in the data conversion unit) to CORDIV is no less than 2.79% (see the last column of Table 5.1).

A weakness of CORDIV is that it needs an expensive organization to regenerate

bit-streams to guarantee correlation between them [10]. Inspired by CORDIV, Wu and Miguel developed an in-stream correlation-based division technique called ISCBDIV [39]. Figure 5.8 shows the architecture of ISCBDIV. ISCBDIV uses a skewed synchronizer (SS) to correlate the input bit-streams for the CORDIV kernel. Two bit-streams are fed into the SS unit regardless of their correlation. The SS unit leverages the assumption that the divisor is larger than the dividend, and reorders the dividend bit-stream based on the divisor. The 1's in the dividend bit-stream are recorded when the divisor bit is 0, and then the saved 1's are paired with the upcoming 1's in the divisor bit-stream. The D-Flip Flop of CORDIV is replaced with a 2-bit shift register (SR). Some extra logic gates are also used to generate a single-bit random number (RN) which is connected to the select input of the multiplexer (MUX). The MAE rates of ISCBDIV are reported in Table 5.1.

In this section, we propose an FSM-based correlator that improves the accuracy of the SC division when the bit-streams are connected to the CORDIV circuit. The proposed correlator receives data in binary format and generates correlated bit-streams. At the cost of a down-counter and an `AND` gate, and by using the SoA FSM-based LD bit-stream generator [8], an LD bit-stream (divisor) correlated with another LD bit-stream (dividend) is generated. The proposed correlator is shown in Figure 5.9. A similar assumption here is that $X < Y$. The MAE rates of the proposed FSM-based correlator when connected to CORDIV are reported in Table 5.1. As can be seen, the proposed correlator achieves the minimum MAE rate among the SoA SC division techniques.

Figure 5.8. ISCBDIV SC Division Circuit[39]

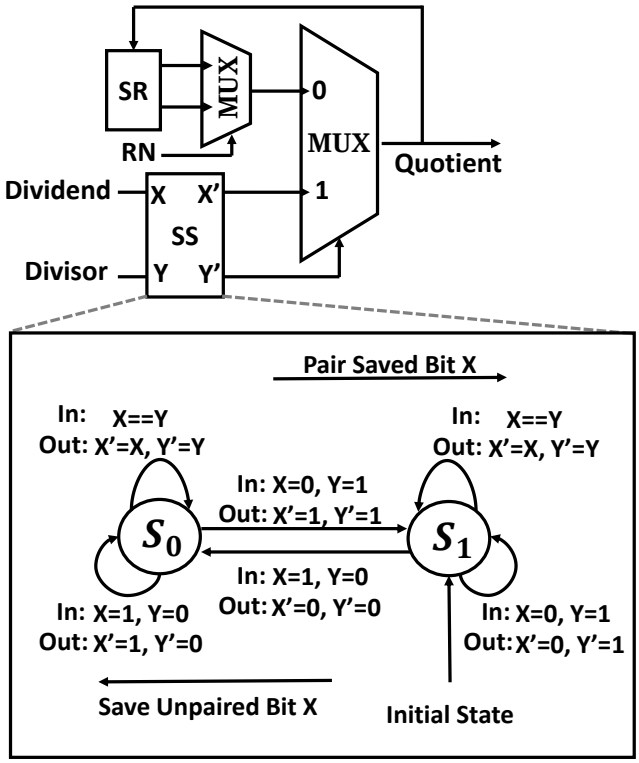
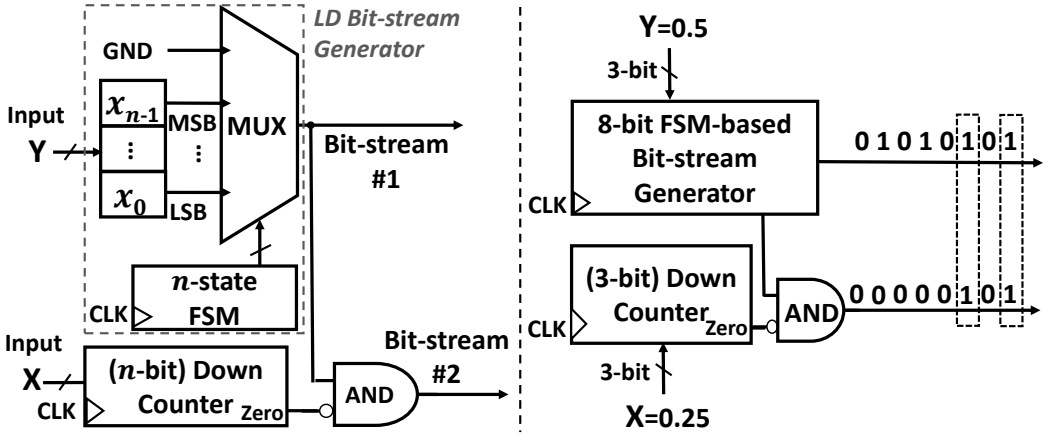


Figure 5.9. (left) Proposed FSM-based correlator for SC division (right) An example for n=3.



5.4 Evaluation

In this section, we evaluate the accuracy and the hardware cost of the proposed correlator and decorrelator circuit compared to the SoA correlation manipulation techniques.

5.4.1 Accuracy Comparison

5.4.1.1 Correlator

The proposed correlator is evaluated and compared with the SoA synchronizer technique [20] for minimum (bit-wise `AND`), maximum (bit-wise `OR`), and absolute difference (bit-wise `XOR`) function. All these functions require correlated input bit-streams for correct functionality. As the input, we first use two uncorrelated LD bit-streams with different lengths of 2^N bits where N is the input bit-width = 6, 7, and 8. MAE rates between the expected and the produced results are provided in Table 5.1 for different FS ranging from 2 to 5 for the proposed correlator, and different *depths* ranging from 1 to 5 for the synchronizer technique.

The correlation between two bit-streams X and Y is quantified using the SC Correlation (SCC) as defined in [2]:

$$SCC(X,Y) = \begin{cases} \dfrac{ad - bc}{N \times min(a+b, a+c) - (a+b)(a+c))} & ad > bc \\ \dfrac{ad - bc}{(a+b)(a+c) - N \times max(a-d, 0)} & else \end{cases}$$

In this formula, a is the number of bit positions where both bit-streams (X and Y) are

Table 5.1. Accuracy Evaluation of the Proposed Correlator with LD Input Bit-streams of 2^N bits (N=Input Width).

Input Width	Input SCC	Method	Depth /FS	Output SCC	Minimum Function MAE (%)	Maximum Function MAE (%)	ABS-Diff Function MAE (%)	Division Function MAE (%)	Division Method	MAE (%)	CORDIV MAE (%) Correlated Sobol 1 Based Inputs
6	1.58	Synchronizer [20]	1	99.48	0.1	1.37	1.47	7.13	Proposed FSM-based Correlator + CORDIV	0.95	2.82
			2	99.97	0	2.83	2.83	9.25			
			3	99.97	0	4.31	4.31	11.36			
			4	99.97	0	5.74	5.74	13.43			
			5	99.97	0	7.11	7.11	15.41			
		CORLD-C	2	93.59	0.51	0.51	1.03	3.39	ISCBDIV (SS Depth=FS)	3.15	
			3	98.29	0.12	0.12	0.24	2.84		3.1	
			4	99.65	0.02	0.02	0.04	2.82		2.98	
			5	100	0	0	0	2.82		2.96	
7	1.73	Synchronizer [20]	1	99.63	0.08	0.62	0.72	5.93	Proposed FSM-based Correlator + CORDIV	0.52	2.8
			2	99.99	∼0	1.35	1.35	7.06			
			3	99.99	0	2.12	2.12	8.18			
			4	99.99	0	2.87	2.87	9.31			
			5	99.99	0	3.6	3.6	10.42			
		CORLD-C	2	93.71	0.52	0.52	1.04	3.47	ISCBDIV (SS Depth=FS)	2.46	
			3	98.30	0.13	0.13	0.26	2.84		2.28	
			4	99.56	0.03	0.03	0.06	2.77		2.23	
			5	99.90	∼0	∼0	0.01	2.77		2.2	
8	0.5	Synchronizer [20]	1	99.58	0.08	0.31	0.41	5.68	Proposed FSM-based Correlator + CORDIV	0.25	2.79
			2	99.997	∼0	0.64	0.64	6.25			
			3	100	0	1.02	1.02	6.83			
			4	100	0	1.41	1.41	7.42			
			5	100	0	1.79	1.79	8.01			
		CORLD-C	2	93.42	0.51	0.51	1.04	3.56	ISCBDIV (SS Depth=FS)	1.97	
			3	98.31	0.12	0.12	0.26	2.87		1.67	
			4	99.63	0.03	0.03	0.06	2.76		1.64	
			5	99.92	∼0	∼0	0.01	2.76		1.63	

1, b is the number of bit positions where X is 1 and Y is 0, c is the number of bit positions where X is 0 and Y is 1, and d is the number of bit positions where both bit-streams are 0. A SCC equal to 0 means the bit-streams are completely uncorrelated. SCC values close to +1 or -1 show high positive or negative correlation, respectively. A positive correlation of +1 means that there is a maximal correlation between the two bit-streams. Notice that because the SCC values are small numbers, *the numbers we report in the tables are SCCs multiplied by 100.*

The MAE of the minimum, maximum, and absolute difference functions is zero when their input bit-streams are positively correlated, i.e. SCC = 1.0 (100.0 when multiplying SCC by 100). As the results presented in Table 5.1 suggest, the SCC value could be deceptive where some number of 1's are removed from the input bit-stream by the correlation manipulation technique. This is especially the case with the maximum value function. When *depth* increases in the synchronizer technique, the MAE of the maximum function increases due to the number of 1's stuck in the FSM states in the final cycles. This similarly affects the absolute difference function since the bias (defined as the deviation from the original value [20]) of the larger bit-stream increases as *depth* increases. With our proposed correlator, although in some cases the output SCC is less than that of the synchronizer, the bias of both input bit-streams is zero, resulting in accurate minimum, maximum, and absolute difference functions.

We further evaluate the performance of the proposed correlator when inputs are pseudo-random bit-streams. Maximal period LFSR are used to generate pseudo-random bit-streams corresponding to each input bit-width. The SCC values and

Table 5.2. Accuracy Evaluation of the Proposed Correlator and Decorrelator when Inputs are 2^8-bit Outputs from SC Multiplication

				Synchronizer [20]			CORLD-C		Decorrelator [20]		CORLD-D	
Input Type	Input SCC	Multiplication. Function MAE (%)	Depth/ FS	Output SCC	Min. Func. MAE (%)	Max. Func. MAE (%)	Output SCC	Min. or Max. Funcs MAE (%)	Output SCC	Multiplication. Function MAE (%)	Output SCC	Multiplication. Function MAE (%)
Sobol based 1&2 and 1&2	80.29	18.96	1	99.82	0.04	0.38	N/A	N/A	N/A	N/A	N/A	N/A
			2	100	0.003	0.74	96	0.63	24.17	2.63	4.87	1.48
			3	100	∼0	1.11	98.74	0.23	14.28	1.7	2.3	0.84
			4	100	0	1.51	99.58	0.07	9.29	2.15	1.23	0.61
			5	100	0	1.88	99.86	0.02	10.55	1.47	0.78	0.35
Sobol based 1&2 and 3&4	1.27	20.08	1	98.2	0.37	0.54	N/A	N/A	N/A	N/A	N/A	N/A
			2	99.92	0.025	0.64	90.7	1.4	0.51	0.87	1.46	1.05
			3	99.99	0.005	1.03	97.48	0.44	3.95	1	0.78	0.69
			4	100	∼0	1.4	99.27	0.13	2.72	0.87	0.51	0.53
			5	100	0	1.78	99.67	0.05	5.21	1.02	0.38	0.43

the MAE rates for the minimum and maximum functions are reported in Table 5.3. As it can be seen, similar to the case with LD bit-streams, the synchronizer faces an increase in the MAE rates when the maximum function is applied on pseudo-random bit-streams and *depth* increases. The performance of CORLD-C, on the other hand, improves (MAE decreases) as *FS* increases.

Multiplication (using `AND`) is another common SC operation. As a different type of input, we evaluate the performance of CORLD-C in a common case that the inputs to the correlator are the outputs from some SC multiplication operations. The SoA work uses LD Sobol-based input bit-streams for accurate multiplication [4, 28]. The output bit-streams however are random and do not follow LD distribution. Table 5.2 reports the input and output SCC, and the MAE rates of the minimum and maximum functions for the case that the inputs to the correlator are the outputs of bit-wise

Table 5.3. Accuracy Evaluation of the Proposed Correlator (CORLD-C) with Pseudo-random Input Bit-streams.

Input Width	Input SCC	Method	Depth /FS	Output SCC	Minimum Function MAE (%)	Maximum Function MAE (%)
6	2.35	Synchronizer [20]	1	88.35	1.73	1.77
			2	98.74	0.28	2.54
			3	99.65	0.07	3.91
			4	99.92	0.02	5.34
			5	99.97	0.005	6.77
		CORLD-C	2	43.81	4.67	4.67
			3	79.37	2.39	2.39
			4	96.68	0.55	0.55
			5	99.68	0.05	0.05
7	1.81	Synchronizer [20]	1	77.04	2.70	2.47
			2	92.27	1.10	1.53
			3	97.26	0.48	2.03
			4	98.88	0.25	2.72
			5	99.56	0.14	3.48
		CORLD-C	2	46.65	4.90	4.90
			3	58.41	3.97	3.97
			4	81.02	2.05	2.05
			5	91.35	1.09	1.09
8	1.63	Synchronizer [20]	1	79.18	2.52	2.32
			2	94.27	0.95	1.08
			3	98.11	0.4	1.02
			4	99.33	0.2	1.28
			5	99.73	0.11	1.63
		CORLD-C	2	46.39	4.74	4.74
			3	69.77	3.14	3.14
			4	85.20	1.64	1.64
			5	95.57	0.5	0.5

ANDing two pairs of 256-bit LD bit-streams generated by using MATLAB built-in Sobol 1 and Sobol 2 sequences. We also evaluate a case that one pair of the inputs is generated using Sobol 3 and Sobol 4 sequences. As reported in Table 5.2, CORLD-C provides accurate outputs for both minimum and maximum functions and its MAE rates decrease by increasing *FS*.

5.4.1.2 Decorrelator

We evaluate the accuracy of the proposed decorrelator (CORLD-D) compared to the decorrelator design of [20] when connecting correlated pseudo-random and LD bit-streams of 2^N-bit length (N=6, 7, 8) to the inputs of the decorrelator circuit. The accuracy evaluation results are shown in Table 5.4. An *Input SCC* of 100 means that the input bit-streams are completely correlated. A lower *Output SCC* (closer to 0) means a better independence between output bit-streams. We evaluate the quality of the output bit-streams by measuring the MAE rate of performing SC multiplication (bit-wise AND) on the produced bit-streams. The reported MAE is the mean of the measured error rates when multiplying all possible pairs of input values (e.g., 256 × 256 combinations for 8-bit input width).

Note that using different random number sequences (e.g., different Sobol sequences) in generating the input bit-streams can result in different MAEs. Hence, in our evaluation, we selected different pairs of Sobol sequences from a large set of more than 1,100 Sobol sequences to generate LD bit-streams. The reported MAEs in Table 5.4 are the averages of the measured values. The last column of Table 5.4 reports

the MAE rate of multiplying two numbers represented by two independent Sobol-based bit-streams as a standard for our comparison. Notice that for 0 percent error, bit-streams of 2^{2N}-bit length must be processed [28][6] but here we process 2^N-bit bit-streams to have the same precision for both input and output.

As can be seen in Table 5.4, CORLD-D achieves comparable accuracy compared to the independent LD Sobol-based bit-streams for different input widths. In most cases, our decorrelator performs better than the decorrelator design of [20] for both LD and pseudo-random bit-streams.

We further evaluated CORLD-D for the case that the inputs are outputs bit-streams from multiplication operation. The output SCC and the MAE rates of the multiplication function are reported in Table 5.2. We can see that in most cases CORLD-D provides a lower output SCC and MAE compared to the SoA decorrelator [20].

5.4.2 Cost Comparison

The hardware cost of the proposed correlator and decorrelator is compared with the SoA designs in Table 5.5 and Table 5.6. We synthesized the designs using the Synopsys Design Compiler v2018.06 with the 45nm FreePDK gate library [1]. The proposed designs consist of two parts: a static part which is independent of the number of input bit-streams, and so its area is fixed, and a part that its area depends on and increases with the number of inputs. The synchronizer and decorrelator design of [20] have different structures. The minimum *depth* for the decorrelator is 2 while it is 1 for the

Table 5.4. Accuracy Evaluation of the Proposed Decorrelator (CORLD-D) for Pseudo-random and LD Bit-streams of 2^Nbit.

				Pseudo-random Bit-streams		LD Bit-streams		
Input Width	Input SCC	Method	Depth/ Fixer Size	Output SCC	MAE (%)	Output SCC	MAE (%)	MAE (%) Independent Sobol-based Bit-streams
6	100	Decorrelator [20]	2	64.54	3.83	42.43	2.59	0.65
			3	59.46	4.19	30.75	2.31	
			4	63.98	4.30	25.08	2.68	
			5	57.19	4.86	33.04	2.67	
		CORLD-D	2	52.28	3.90	9.92	1.24	
			3	22.39	2.77	1.79	0.90	
			4	8.84	2.33	-2.19	0.79	
			5	25.96	3.09	-11.94	1.84	
7	100	Decorrelator [20]	2	67.13	4.35	39.25	2.36	0.36
			3	56.12	3.48	29.33	1.85	
			4	56.81	3.32	24.87	2.64	
			5	46.23	2.52	23.39	1.66	
		CORLD-D	2	59.55	4.07	12.66	0.87	
			3	49.85	3.12	2.89	0.64	
			4	29.8	2	-0.25	0.47	
			5	32.63	1.7	-2.21	0.57	
8	100	Decorrelator [20]	2	67.67	4.54	39.32	2.49	0.19
			3	54.70	3.46	22.76	1.48	
			4	52.18	3.17	24.36	2.52	
			5	42.49	2.51	21.21	1.25	
		CORLD-D	2	49.14	3.53	13.98	0.69	
			3	35.1	2.25	3.86	0.4	
			4	32.62	1.81	0.45	0.3	
			5	5.44	0.98	-0.49	0.25	

synchronizer. The hardware area of the decorrelator varies by the input width as the size of the RNG depends on the number of bits in the input bit-streams. Although the hardware cost of the synchronizer is less than that of CORLD-C, the total cost of using the synchronizer technique depends on the number of input bit-streams. At each stage, new synchronizers are needed as the synchronizer circuit only makes two bit-streams correlated with respect to each other.

Therefore, correlating a large number of bit-streams in an SC system with multiple stages of computations needs many synchronizers. We will show this in Section 5.5 with two applications of correlated bit-streams, SC design of sorting, and median filtering. Our correlator, on the other hand, is able to correlate all bit-streams by manipulating each one only once. This leads to a significant hardware cost saving compared to the synchronizer technique. The proposed decorrelator is also advantageous from the hardware cost point of view. The hardware cost of the decorrelator design of [20] depends on the area of the SSG which is more than that of other types of RNG. However, selecting SSG as the RNG significantly improves the accuracy.

5.5 Case Studies

In this section, we evaluate the performance and the hardware cost of the proposed correlator in the SC design of the sorting and median filtering circuits which include both minimum and maximum operations. These circuits consist of multiple stages of computations requiring correlated bit-streams for correct functionality.

Table 5.5. Hardware area (μm^2) of the proposed CORLD-C and CORLD-D for different FSs

	Static Area		# of Input Dependent Area		Total Area For 1 Input		Total Area For 2 Inputs	
FS	-C	-D	-C	-D	-C	-D	-C	-D
2	45	83	170		215	253	386	424
3	67	129	238		306	367	544	605
4	90	175	298		389	474	687	772
5	112	220	354		467	574	822	929

Table 5.6. Hardware area (μm^2) of the synchronizer and decorrelator circuit of [20] for different depths

Depth	Decorrelator [20]			Synchronizer [20]
1	N/A	N/A	N/A	107
2	1225	1589	1969	166
3	1288	1652	2033	168
4	1357	1721	2101	237
5	1393	1757	2137	241
SSG Area	569	751	941	
Input Width	6-bit	7-bit	8-bit	

Figure 5.10. Schematic representation of a CAS block (a) Ascending order, (b) Descending order

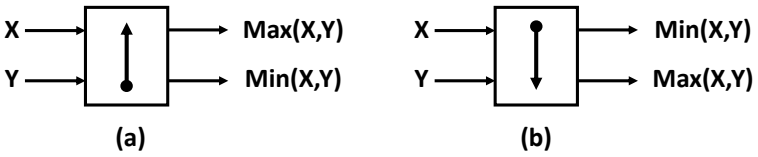
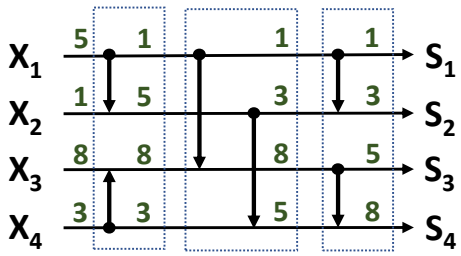


Figure 5.11. 4-input Sorting Network



5.5.1 Sorting

A sorting network is a combination of some compare-and-swap (CAS) blocks that sorts a set of input data [32]. A CAS block comprises a minimum and a maximum operation in ascending or descending order, as shown in Figure 5.10. A 4-input sorting network made of 6 CAS blocks is shown in Figure 5.11. A low-cost SC design for hardware implementation of sorting networks is proposed in [32]. In this design, each CAS block is implemented using one `AND` (for minimum operation) and one `OR` (for maximum operation) gate. For correct and accurate functionality, the input bit-streams to the CAS blocks must be correlated. Connecting uncorrelated bit-streams to the SC sorting circuit and so CAS blocks, results in inaccurate and wrong output data. Correlator circuits are therefore needed to manipulate and guarantee correlation between bit-streams.

The overhead cost of using the synchronizer technique [20] depends on, and significantly increases with, the number of CAS blocks. This is because the output bit-streams from different CAS blocks are not necessarily correlated when using this technique. Therefore, each pair of inputs to a CAS block in each stage of computations needs a separate synchronizer. On the other hand, our proposed CORLD-C needs to correlate the input bit-streams only once, in the first stage of computations, where the input bit-streams arrive.

Table 5.7 shows the MAE rates of an 8-input and a 16-input SC sorting system [32] processing data received in the form of independent LD bit-streams. The MAEs are reported for two different approaches for manipulating correlation: 1) using

Table 5.7. Accuracy Evaluation of the Stochastic Sorting System with the Proposed and SoA Correlator Technique when Sorting 256-bit independent LD input bit-streams

	Synchronizer [20]		CORLD-C	
Depth/ FS	8 inputs MAE (%)	16 Inputs MAE (%)	8 Inputs MAE (%)	16 Inputs MAE (%)
1	1.05	2.19	N/A	N/A
2	2.06	3.74	2.25	3.11
3	3.18	5.41	0.69	1.10
4	4.29	7.09	0.16	0.32
5	5.38	8.75	0.03	0.07

the SoA synchronizer with different *depth*s 2) using the proposed CORLD-C with different *FS*. As can be seen in the reported numbers, the proposed technique outperforms the SoA technique, particularly when *FS* increases. Increasing *depth* in the synchronizer technique increases the MAE rates mainly due to the weak performance of the synchronizer technique in performing the maximum operation. Table 5.8 compares the number of needed correlator units and the hardware cost of the implemented designs. As it can be seen, the hardware area saving provided by CORLD-C increases significantly when the number of inputs increases. For example, for the 256-input SC sorting system, CORLD-C with FS=2 reduces the hardware area cost by more than 17× compared to the synchronizer technique with $Depth$=2.

5.5.2 Median Filtering

A median filter is a nonlinear filter which removes impulse noises from images and videos by replacing each pixel value by the median of all pixel values in the local neighborhood. Figure 5.12 shows the structure of a 3×3 median filter circuit made of

Table 5.8. Hardware Cost (μm^2) Comparison of the SC Sorting and Median Filtering System with different number of inputs utilizing proposed CORLD-C and SoA synchronizer (# of Cor.: # of correlator needed for the design.)

		Synchronizer [20]			CORLD-C			
# of Inputs	# of CAS	# of Cor.	Area Depth=1	Area Depth=2	# of Cor.	Area FS=2	Area FS=3	
8	24	24	2,580	3,998	8	1,408	1,974	Sorting
16	80	80	8,600	13,328	16	2,771	4,850	
32	240	240	25,800	39,984	32	5,498	7,696	
64	672	672	72,240	111,955	64	10,951	15,325	
128	1,792	1,792	192,640	298,547	128	21,856	30,582	
256	4,608	4,608	495,360	767,693	256	43,667	61,098	
9 (3×3)	19	19	2,043	3,165	9	1,579	2,213	Median
25 (5×5)	246	246	26,445	40,984	25	4,305	6,027	Filter

Figure 5.12. 3×3 median filter using 19 CAS blocks [32]

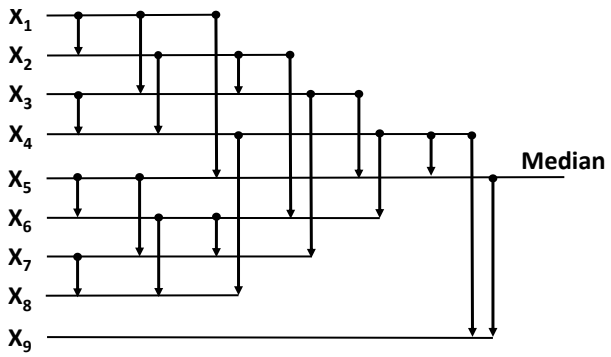


Table 5.9. Accuracy Evaluation of SC Median Filtering System with 256-bit independent LD input bit-streams

	Synchronizer [20]		CORLD-C	
Depth/ FS	9 (3×3) inputs MAE (%)	25 (5×5) Inputs MAE (%)	9 inputs MAE (%)	25 Inputs MAE (%)
1	1.09	4.07	N/A	N/A
2	2.05	8.92	2.52	3.61
3	3.25	13.68	0.8	1.36
4	4.43	18.38	0.2	0.44
5	5.59	22.9	0.04	0.11

19 CAS blocks. The design of a 5×5 median filtering circuit can be found in [32]. For correct functionality, similar to the SC sorting system, the CAS blocks need correlated inputs. Correlation manipulation units are therefore needed to guarantee correlation between the inputs of the CAS blocks. The hardware cost and the accuracy of the SC median filtering circuit implemented with different correlation manipulation approaches are reported in Table 5.8 and Table 5.9. As shown, CORLD-C achieves significantly lower MAE rates with considerable savings in the hardware area cost.

5.6 Conclusion

In this dissertation, we proposed two in-stream correlator (CORLD-C) and decorrelator (CORLD-D) techniques to increase and decrease the correlation between SC bit-streams. The proposed techniques produce high-quality LD bit-streams with no impact on the data values. The accuracy, performance, and hardware cost are all improved compared to the SoA correlation manipulation techniques. We further proposed an FSM-based correlator that improves the accuracy of the SoA SC division circuit. Evaluating the proposed techniques in the SC design of the sorting and median filtering circuits showed a significant reduction in the MAE rates and hardware area

cost. The synthesizable Verilog HDL codes of this work, including the proposed correlator and decorrelator circuits, are made publicly available at *https://github.com/asadisina/CORLD*.

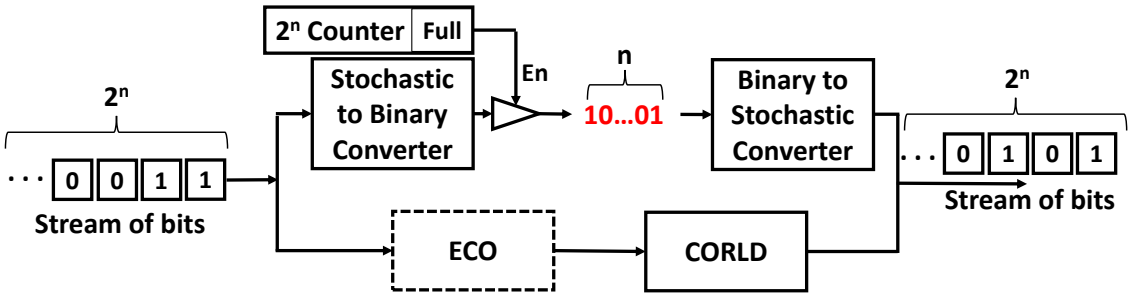
Chapter 6: ECO: Enhanced In-Stream Correlation Manipulation for Low-Discrepancy Stochastic Computing

An important part of a stochastic system is the data conversion unit that converts the input data from positional binary to bit-stream representation. The common approach for implementing this unit is by using a binary comparator and a RNG. In each clock cycle, the input data in binary format is compared with a random number from the RNG. If the random number is smaller than the input data, a 1 is yielded at the output of the comparator. Correlation between input bit-streams can be controlled at this stage when the bit-streams are generated. Sharing the same RNG between different data conversion units results in generating correlated bit-streams. Using different RNG, on the other hand, leads to generating uncorrelated bit-streams. Fig. 5.1 shows this using an example circuit.

While this method is common for controlling correlation between the inputs of the stochastic system at the first stage of computations, it cannot be used in the next stages of the system as the data are already in the bit-stream form. A naive method for manipulating correlation in the intermediate stages is to convert the bit-streams back to the positional binary format and then re-generate them with the desired correlation. This is shown in the top branch of Fig. 6.1. But this approach incurs significant area and power overheads. Meanwhile, developing low-cost correlation manipulation circuits for in-stream (i.e., without re-generating bit-streams) managing of correlation between bit-streams is in high demand.

The previous chapter covers the background studies of the proposed methods

Figure 6.1. Correlation manipulation of a stochastic bit-stream by 1) bit-stream regeneration (in top branch) and 2) in-stream manipulation with proposed method consisting of *ECO* and *CORLD* techniques (in bottom branch).



and also the new approach, COLRD. The CORLD technique successfully manipulates the correlation of different types of input bit-streams. Particularly, CORLD works very well when stochastic system inputs are LD bit-streams. However, pseudo-random-based bit-streams are still being used in SC. We observed that their correlation level negatively affects the performance of the system when using CORLD. Due to the fact that either CORLD design or correlation manipulation method [20] changes the bits pattern, the prior CORLD approach needs to be improved. By assessing the behavior of such bit-streams, we realized that the distribution of bits is not consistent in different fixed-size segments. Fig. 6.2 shows parts of an arbitrary pseudo random-based bit-stream with two 4-bit segment size. The number of 1s in the first segment is 3, and in the next segment is 1. This deviation results in higher error rates either for positively correlated input-based functions such as Minimum and Maximum functions or decorrelated input-based operations such as multiplication.

The proposed ECORLD is a combination of the prior CORLD and the

enhancing technique to balance the probability of visiting 1 in each segment. The proposed enhancer module stores the first segment and compares and swaps it with the next one consecutively. In the case of the example in Fig. 6.2, the enhancer masks the deviation in the probability of visiting 1 in two consecutive input segments and produces two segments with exactly two 1s. Fig. 6.3 shows the proposed FSM for the enhancer technique. The initial state is S_1, and the state remains there when the saved bit (S-X) equals to input bit (I-X) and the saved bit transfers to the output bit (O-X). Otherwise, if the saved bit and input bits are different, either the input bit is greater than the saved bit or vice versa, the state changes, but still, saved bits go to output and will be succeeded by the input bit. The enhancer would stay in the new state until the input bit and saved bit are different; hence, go back to the initial state. If the input bit is greater than the saved bit, it will be directly transferred to the output bit, and the saved bit will be untouched. The opposite scenario happens when the saved bit is greater than the input bit. The latent idea behind this is to keep each segment as close as possible to each other based on the definition of progressive precision [3], which is a key property of low discrepancy bit-streams that makes them suitable for SC.

6.1 Evaluation

In this section, we evaluate the accuracy and the hardware cost of the proposed ECORLD circuit compared to the SoA correlation manipulation techniques.

Figure 6.2. Proposed ECO Example with $FS = 2$.

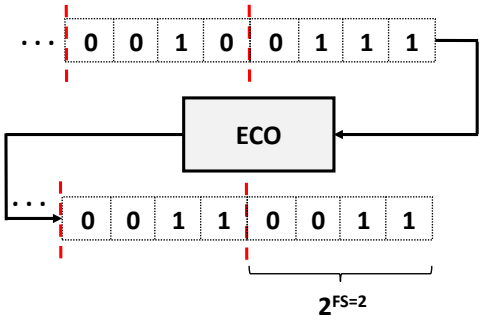
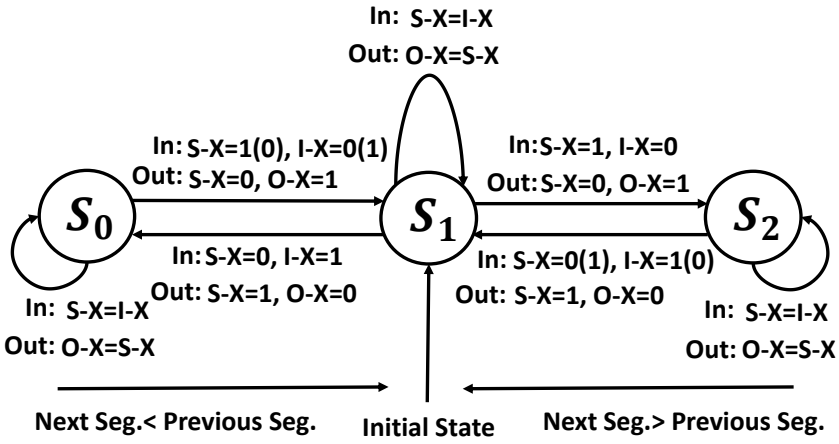


Figure 6.3. Proposed ECO FSM. (S: Saved, I: Input, O: Output).



6.1.1 Accuracy Comparison

6.1.1.1 Correlator

ECO is an improved version of COLRD specifically designed for non-LD input bit-streams. Table 6.1 shows the performance evaluation of the proposed correlator when inputs are pseudo-random bit-streams. Maximal period LFSR are used to generate pseudo-random bit-streams corresponding to each input bit-width. A similar table is provided in the previous chapter, however, the proposed ECO method is added

to compare its efficiency with respect to Synchronizer and CORLD.

The SCC values and the MAE rates for the minimum and maximum functions are reported in the table. The proposed ECO method distinctively improves the correlation hence the accuracy of Pseudo random-based bit-streams by manipulating bits of input bit-streams to maintain a steady probability pace at each segment of the stream.

6.1.1.2 Decorrelator

Table 6.2 is the extension of table 5.4 while the ECO module is utilized to further enhances the performance of the COLRD approach. As discussed, the proposed ECO method is inefficient, while the input bit-streams are LD-based due to its progressive precision property. In other words, the distribution of 1's in each segment of the LD-based bit-streams, is stable, while ECO is invented to manipulate the dispensation of 1's in continuous chunks if they are not in good order. By a quick glance at the LD-based input, it can be observed that the ECO and CORLD output the same results. On the other hand, when inputs are pseudo-random based, ECO design significantly reduces the MAE rate of performing SC multiplication (bit-wise `AND`) on the produced bit-streams. The reported MAE is the mean of the measured error rates when multiplying all possible pairs of input values (e.g., 256×256 combinations for 8-bit input width).

Table 6.1. Accuracy Evaluation of the Proposed Correlator (CORLD-C) and proposed ECO with Pseudo-random Input Bit-streams.

Input Width	Input SCC	Method	Sync. [20] Deep Size/ Fixer Size (FS)	Output SCC	Minimum Function MAE (%)	Maximum Function MAE (%)
6	2.35	Sync. [20]	1	88.35	1.73	1.77
			2	98.74	0.28	2.54
			3	99.65	0.07	3.91
			4	99.92	0.02	5.34
			5	99.97	0.005	6.77
		CORLD	2	43.81	4.67	4.67
			3	79.37	2.39	2.39
			4	96.68	0.55	0.55
			5	99.68	0.05	0.05
		ECO	2	88.98	1.41	1.41
			3	95.20	0.62	0.62
			4	99.18	0.06	0.06
			5	100	0	0
7	1.08	Sync. [20]	1	79.88	2.86	2.77
			2	91.71	1.43	1.88
			3	95.72	0.86	1.96
			4	97.51	0.61	2.45
			5	98.58	0.45	3.11
		CORLD	2	44.78	5.57	5.57
			3	67.2	3.62	3.62
			4	85.6	1.64	1.64
			5	90.31	1.08	1.08
		ECO	2	77.6	2.32	2.32
			3	89.28	1.24	1.24
			4	93.68	0.84	0.84
			5	97.83	0.3	0.3
8	1.63	Sync. [20]	1	79.18	2.52	2.32
			2	94.27	0.95	1.08
			3	98.11	0.4	1.02
			4	99.33	0.2	1.28
			5	99.73	0.11	1.63
		CORLD	2	46.39	4.74	4.74
			3	69.77	3.14	3.14
			4	85.2	1.64	1.64
			5	95.57	0.5	0.5
		ECO	2	83.97	2.02	2.02
			3	92.77	0.97	0.97
			4	95.99	0.59	0.59
			5	98.33	0.25	0.25

Table 6.2. Accuracy Evaluation of the Proposed Decorrelator (CORLD-D) and proposed ECO for Pseudo-random and LD Bit-streams of 2^Nbit.

				LFSR-based Bit-streams		LD Bit-streams		
Input Width	Input SCC	Method	Deep Size/ Fixer Size	Output SCC	MAE (%)	Output SCC	MAE (%)	MAE (%) Uncorrelated Sobol based bit-streams
6	100	Decorrelator [20]	2	54.38	2.56	19.78	0.84	0.65
			3	33.15	1.95	10.67	1.31	
			4	35.07	1.92	13.21	1.28	
			5	33.36	2.02	12.93	1.68	
		CORLD	2	58.69	4.64	13.2	0.9	
			3	25.29	2.52	0.63	0.64	
			4	14.33	1.9	-1.18	0.85	
			5	10.54	2.84	-11.94	1.84	
		ECO	2	17.64	1.75	13.2	0.9	
			3	16.02	1.28	0.63	0.64	
			4	2.56	0.96	-1.18	0.85	
			5	18.19	2.03	-11.94	1.84	
7	100	Decorrelator [20]	2	59.58	3.54	17.58	0.57	0.36
			3	47.91	2.84	23.02	1	
			4	43.01	1.62	17.54	1.07	
			5	34.22	1.56	19.02	1.11	
		CORLD	2	59.55	4.07	14.79	0.84	
			3	49.85	3.12	4.81	0.52	
			4	29.8	2	1.58	0.63	
			5	32.63	1.7	6.31	0.56	
		ECO	2	27.6	2.02	14.79	0.84	
			3	18.56	1.12	4.81	0.52	
			4	8.23	0.83	1.58	0.63	
			5	3.32	0.76	6.31	0.56	
8	100	Decorrelator [20]	2	55.98	3.4	27.55	0.98	0.19
			3	43.56	2.53	14.97	0.68	
			4	40.67	1.73	17.7	0.76	
			5	30.5	1.47	8.18	0.69	
		CORLD	2	49.14	3.53	17.04	0.55	
			3	35.1	2.25	5.33	0.36	
			4	32.62	1.81	3.84	0.19	
			5	5.44	0.98	3.22	0.23	
		ECO	2	27.55	1.86	17.04	0.55	
			3	18.36	1.02	5.33	0.36	
			4	5.06	0.53	3.84	0.19	
			5	3.5	0.45	3.22	0.23	

Table 6.3. Hardware cost of the proposed ECO for different FSs

Fixer Size	Total Area (μm^2)	Critical Path Latency (nS)	Power@Max Freq. (mW)	Energy per Cycle (pJ)
2	226.7	0.38	0.8094	0.30
3	341.7	0.45	0.9357	0.42
4	538	0.46	1.2271	0.56
5	858.8	0.53	1.6216	0.85

6.1.2 Cost Comparison

Table 6.3 represents the hardware cost and energy factors of the proposed ECO architecture. We synthesized the designs using the Synopsys Design Compiler v2018.06 with the 45nm FreePDK gate library [1]. As the table suggests, by increasing the fixer size, the area, power, and hence energy consumption increase. The selection of ECO size totally depends on the accuracy requirement and energy constraint of the application.

6.2 Conclusion

In this chapter, a new method and the corresponding architecture are designed for enhancing the COLRD-C and CORLD-D approach when non-LD-based input bit-streams are being fed to the system. We proposed an enhancer module called ECO, which masks the deviation of 1's in two consecutive input segments. The goal is to maintain the progressive precision property throughout an input bit-stream as much as possible to become closer to LD bit-streams. ECO method increased the accuracy of COLRD up to 3.6 times at the cost of some hardware unit that implements a specific finite state machine to execute the method in an in-stream manner.

Chapter 7: Summary

SC is re-emerging as an unconventional model of computing representing numbers in the form of bit-streams offering lightweight, power-efficient, and noise-tolerant properties for different arithmetic operations. For example, a single `AND` gate can perform accurate multiplication in stochastic domain while this is considered a complex operation in the binary domain. Although SC enjoys some appealing advantages, it suffers from disadvantages such as the need for costly bit-stream generators and long processing time, which often translates to high energy consumption. In this dissertation, we proposed some novel approaches to address the current challenges of SC designs. In what follows, we summarize the main contributions of this dissertation:

1. Accelerating Deterministic Methods of SC with Context-Aware Bit-stream Generation: In the very first steps of our work, we evaluated the representation of an input probability in stochastic format. We realized that by doing a simplification in probabilities, we could simplify many high-resolution numbers with lower resolution values. Representing the numbers in these simplified forms needs significantly shorter bit-streams and so lower number of clock cycles for processing without affecting the represented value. We proposed a new method and a corresponding architecture to manage this simplification. We further proposed an approach for applications that can tolerate slight inaccuracy by ignoring some LSBs. Our approach not only offers significant energy and time-saving, it comes with area saving due to removing some unnecessary components.

2. A Low-Cost FSM-based Bit-Stream Generator for Low-Discrepancy SC: the conventional approach of generating stochastic bit-streams requires some components such as random number generators and binary comparators. Although a binary comparator is relatively light, the state-of-the-art random number generators used in SC such as SOBOL sequence generators are costly to implement. This became our motivation to propose a low-cost FSM-based method for generating similar bit-streams but without the need for costly number generators. The new method produces a bit of a bit-stream by extracting and organizing the bits of the binary input at each clock cycle. The proposed approach significantly decreases the hardware cost with no impact on accuracy.

3. In-Stream Correlation Manipulation for Low-Discrepancy SC: The correct and accurate execution of SC operations depend on the correlation level between input bit-streams. A naive method to control correlation between bit-streams is to regenerate them by converting them to binary format and then convert back to bit-streams with desired correlation. However, this approach is very expensive in terms of time, area, and power. On the other hand, we are interested in developing SC systems that eradicates the dependency on the binary format and performs all operations in stochastic domain. These motivated us to propose an in-stream method to manipulate the level of correlation between bit-streams on the way without any delay just by splitting the bit-streams into multiple chunks and re-organizing each chunk. This approach provides different correlation levels based on the need of the next stage function while guaranteeing LD distribution

of the bit-streams. We further proposed a new method for producing positively correlated bit-streams. We evaluated the proposed method for SC division with significant accuracy improvement.

4. Enhanced In-Stream Correlation Manipulation: We noticed that although the proposed correlation manipulation technique (CORLD) increased the accuracy of different SC functions where input bit-streams are LD based, we enhanced our proposed technique by adding a certain unit specifically designed for non-LD based input bit-streams. Our evaluations showed higher accuracy for those non-LD bit-streams

Table7.1 shows a brief overview of the techniques proposed in this dissertation for a better understanding of their capabilities. Accuracy, time, power/energy consumption, and area are the considered criteria to have a quick comparison of how they impact the SC system.

Table 7.1. Quick Overview of the Proposed methods

Method \Parameter		Accuracy	Time efficiency	Power/Energy	Area
Context Aware (approach 1)	Accurate	Same	+++	+	-
	Error tolerant	-	+++++	++	+
FSM-based generator (approach 2)		+	+	+++	+++
FSM-based correlator generator(approach 3)		+++++	+	+	+
COLRD+ECO (approach 3 and 4)		++	Same	-	-

Bibliography

[1] NCSU FreePDK 45nm Library. `https://research.ece.ncsu.edu/eda/freepdk/freepdk45/`.

[2] A. Alaghi and J.P. Hayes. Exploiting correlation in stochastic circuit design. In *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pages 39–46, Oct 2013.

[3] A. Alaghi and J.P. Hayes. Fast and accurate computation using stochastic circuits. In *DATE'14*, pages 1–4, March 2014.

[4] A. Alaghi, W. Qian, and J. P. Hayes. The Promise and Challenge of Stochastic Computing. *IEEE Trans. on Computer-Aided Design of Integ. Circ. and Sys.*, 37(8):1515–1531, Aug 2018.

[5] Sina Asadi and M Hassan Najafi. Context-aware number generator for deterministic bit-stream computing. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, volume 2160, pages 140–140. IEEE, 2019.

[6] Sina Asadi and M. Hassan Najafi. Accelerating deterministic stochastic computing with context-aware bit-stream generator. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, page 157–162, New York, NY, USA, 2020.

[7] Sina Asadi and M. Hassan Najafi. LDFSM: A Low-Cost Bit-Stream Generator for Low-Discrepancy Stochastic Computing: Late Breaking Results. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, DAC '20. IEEE Press, 2020.

[8] Sina Asadi, M. Hassan Najafi, and Mohsen Imani. A Low-Cost FSM-based Bit-Stream Generator for Low-Discrepancy Stochastic Computing. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 908–913, 2021.

[9] Sina Asadi, M Hassan Najafi, and Mohsen Imani. Corld: In-stream correlation manipulation for low-discrepancy stochastic computing. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[10] T. Chen and J. P. Hayes. Design of division circuits for stochastic computing. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 116–121, 2016.

[11] Shao-I Chu. New divider design for stochastic computing. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(1):147–151, 2020.

[12] Robert H Dennard, Fritz H Gaensslen, Hwa-Nien Yu, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[13] S. Rasoul Faraji, M. Hassan Najafi, Bingzhe Li, Kia Bazargan, and David J. Lilja. Energy-Efficient Convolutional Neural Networks with Deterministic Bit-Stream Processing. In *DATE'19*, March 2019.

[14] B.R. Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, pages 37–172. Springer US, 1969.

[15] Brian R Gaines. Stochastic computing. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 149–156. ACM, 1967.

[16] S. Gupta, M. Imani, J. Sim, A. Huang, F. Wu, M. H. Najafi, and T. Rosing. SCRIMP: A General Stochastic Computing Architecture using ReRAM in-Memory Processing. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1598–1601, 2020.

[17] Devon Jenson and Marc Riedel. A Deterministic Approach to Stochastic Computation. In *Proceedings of the 35th Intern. Conf. on Computer-Aided Design*, ICCAD '16, 2016.

[18] Israel Koren and C. Mani Krishna. *Fault-Tolerant Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2007.

[19] S. Lee, H. Sim, J. Choi, and J. Lee. Successive log quantization for cost-efficient neural networks using stochastic computing. In *the 56th Annual Design Automation Conference 2019*, DAC '19, 2019.

[20] V. T. Lee, A. Alaghi, and L. Ceze. Correlation manipulating circuits for stochastic computing. In *DATE'18*, pages 1417–1422, 2018.

[21] V. T. Lee, A. Alaghi, J. Hayes, V. Sathe, and L. Ceze. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *DATE'17*, March 2017.

[22] V. T. Lee, A. Alaghi, R. Pamula, V. S. Sathe, L. Ceze, and M. Oskin. Architecture considerations for stochastic computing accelerators. *IEEE Trans. on Computer-Aided Design of Integ. Circ. and Sys.*, 37(11):2277–2289, 2018.

[23] Peng Li, D.J. Lilja, Weikang Qian, K. Bazargan, and M.D. Riedel. Computation on stochastic bit streams digital image processing case studies. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 22(3):449–462, 2014.

[24] S. Liu and J. Han. Energy Efficient Stochastic Computing with Sobol Sequences. In *DATE'17*, pages 650–653, March 2017.

[25] S. Liu and J. Han. Toward Energy-Efficient Stochastic Circuits Using Parallel Sobol Sequences. *IEEE Tran. on VLSI Systems*, 26(7):1326–1339, July 2018.

[26] M. H. Najafi *et al.* Time-Encoded Values for Highly Efficient Stochastic Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(5):1644–1657, May 2017.

[27] Shyamali Mitra, Debojyoti Banerjee, and Mrinal K. Naskar. A low latency stochastic square root circuit. In *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*, pages 7–12, 2021.

[28] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel. Performing Stochastic Computation Deterministically. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2925–2938, Dec 2019.

[29] M. H. Najafi and D. Lilja. High Quality Down-Sampling for Deterministic Approaches to Stochastic Computing. *IEEE Transactions on Emerging Topics in Computing*, 2018.

[30] M. H. Najafi, D. J. Lilja, and M. Riedel. Deterministic Methods for Stochastic Computing Using Low-discrepancy Sequences. In *ICCAD'18*, pages 1–8, 2018.

[31] M. Hassan Najafi, Peng Li, David J. Lilja, Weikang Qian, Kia Bazargan, and Marc Riedel. A Reconfigurable Architecture with Sequential Logic-Based Stochastic Computing. *J. Emerg. Technol. Comput. Syst.*, 13(4):57:1–57:28, June 2017.

[32] M. Hassan Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan. Low-Cost Sorting Network Circuits Using Unary Processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(8):1471–1480, Aug 2018.

[33] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja. An Architecture for Fault-Tolerant Computation with Stochastic Logic. *IEEE Trans. on Comp.*, 60(1):93–105, Jan 2011.

[34] M. Riahi Alam, M. H. Najafi, and N. TaheriNejad. Exact In-Memory Multiplication Based on Deterministic Stochastic Computing. In *2020 IEEE Intern. Symp. on Circuits and Systems (ISCAS)*, pages 1–5, 2020.

[35] Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[36] Hyeonuk Sim, Saken Kenzhegulov, and Jongeun Lee. DPS: Dynamic Precision Scaling for Stochastic Computing-based Deep Neural Networks. In *DAC'18*, pages 13:1–13:6, 2018.

[37] Hyeonuk Sim and Jongeun Lee. A new stochastic computing multiplier with application to deep convolutional neural networks. In *the 54th DAC, 2017*, pages 1–6, 2017.

[38] Bruno Tuffin. On the use of low discrepancy sequences in monte carlo methods. 1996.

[39] Di Wu and Joshua San Miguel. In-stream stochastic division and square root via correlation. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.