New Views for Stochastic Computing: From Time-Encoding to Deterministic Processing

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

M. Hassan Najafi

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Advisor: David J. Lilja

July, 2018

بسم الله الرحمن الرحیم

ای بس که نباشیم و جهان خواهد بود　　نی نام ز ما و نی نشان خواهد بود

زین پیش نبودیم و نبد هیچ خلل　　زین پس چو نباشیم همان خواهد بود

The world will long be, but of you and me
No sign, no trace for anyone to see;
The world lacked not a thing before we came,
Nor will it miss us when we cease to be.
–KHAYYAAM

Acknowledgements

First and foremost, I like to extend my sincere thanks to my advisor, Prof. David J. Lilja, for being such a wonderful mentor during my PhD. It has been a privilege to be mentored by him and my great honor to be his PhD student. He was full of encouragement and support and gave me great latitude in determining my own direction and pursuing my ideas. I learned many essential research skills from him and I appreciate his financial support which allowed me to devote all of my attention to my research.

I would like to thank the other members of my graduate committee, Prof. Marc Riedel, Prof. Kia Bazargan, and Prof. Pen-Chung Yew for not only agreeing to be on my committee but also for their valuable suggestions and feedback along the way. I like to give a big thanks to Marc and Kia for their help and input to my research. Many of the ideas of this research are derived from our weekly stochastic research group meeting with David, Marc, and Kia. I would also like to thank Shiva Jamali-Zavareh and Prof. Ramesh Harjani for their contribution to the time-based computing project, and Bingzhe Li and Rasoul Faraji for their contribution to the stochastic computing-based neural networks project.

Finally, I would like to thanks all my friends in Minneapolis. I have been fortunate to have made some wonderful friends during these four years. To Ali, Mohammad, Saeed, Arash, Amirreza, and Rasoul; I am very appreciative of the fun experiences shared with you guys as well as your support. You were my family here and I am blessed and thankful to have you as my friends.

This work was supported in part by National Science Foundation grant no. CCF-1408123 and CCF-1438286. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF. The author also acknowledges the Graduate School at University of Minnesota for the funding support through the Doctoral Dissertation Fellowship.

To My Parents and Brothers

For their endless love, support, and encouragement

Abstract

Stochastic computing (SC), a paradigm first introduced in the 1960s, has received considerable attention in recent years as a potential paradigm for emerging technologies and “post-CMOS” computing. Logical computation is performed on random bitstreams where the signal value is encoded by the probability of obtaining a one versus a zero. This unconventional representation of data offers some intriguing advantages over conventional weighted binary. Implementing complex functions with simple hardware (e.g., multiplication using a single AND gate), tolerating soft errors (i.e., bit flips), and progressive precision are the primary advantages of SC. The obvious disadvantage, however, is latency. A stochastic representation is exponentially longer than conventional binary radix. Long latencies translate into high energy consumption, often higher than that of their binary counterpart. Generating bit streams is also costly. Factoring in the cost of the bit-stream generators, the overall hardware cost of an SC implementation is often comparable to a conventional binary implementation.

This dissertation begins by proposing a highly unorthodox idea: performing computation with digital constructs on time-encoded analog signals. We introduce a new, energy-efficient, high-performance, and much less costly approach for SC using time-encoded pulse signals. We explore the design and implementation of arithmetic operations on time-encoded data and discuss the advantages, challenges, and potential applications. Experimental results on image processing applications show up to 99% performance speedup, 98% saving in energy dissipation, and 40% area reduction compared to prior stochastic implementations. We further introduce a low-cost approach for synthesizing sorting network circuits based on deterministic unary bit-streams. Synthesis results show more than 90% area and power savings compared to the costs of the conventional binary implementation. Time-based encoding of data is then exploited for fast and energy-efficient processing of data with the developed sorting circuits.

Poor progressive precision is the main challenge with the recently developed deterministic methods of SC. We propose a high-quality down-sampling method which significantly improves the processing time and the energy consumption of these deterministic methods by pseudo-randomizing bitstreams. We also propose two novel deterministic methods of processing bitstreams by using low-discrepancy sequences. We further introduce a new advantage to SC paradigm-the skew tolerance of SC circuits. We exploit this advantage in developing polysynchronous clocking, a design strategy for optimizing the clock distribution network of SC systems. Finally, as the first study of its kind to the best of our knowledge, we rethink the memory system design for SC. We propose a seamless stochastic system, StochMem, which features analog memory to trade the energy and area overhead of data conversion for computation accuracy.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Stochastic Computing (SC), first advocated by Gaines [11, 12] and Poppelbaum [13] in 1967, has received renewed attention in recent years [14, 15, 2, 16, 4, 17, 18, 19]. This is due to the growing uncertainty in design parameters, and therefore, in design functionality, as induced by imbalances in modern technology scaling. Image and video processing [1, 4, 20, 21, 22], digital filters [23, 24, 25], low-density parity check decoding and error correction [26, 27, 28, 29, 30] and neural networks [31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41] have been the main target applications for SC.

In SC, circuits operate on randomized bitstreams. Independent of the length (and interleaving of 0s and 1s), the ratio of the number of 1s to the length of the stream determines the value of the bitstream. Computation accuracy increases with the length of the bitstream. In contrast to conventional binary radix, all digits of a bitstream have the same weight. In the "unipolar" representation, a real-valued number x ($0 \leq x \leq 1$) is represented by a stream in which each bit has probability x of being one and probability $1 - x$ of being zero. In the "bipolar" representation, a real-valued number y ($-1 \leq y \leq 1$) is represented by a stream in which each bit has probability $\frac{y+1}{2}$ of being one and probability $1 - \frac{y+1}{2}$ of being zero. For example, 1101010000 is a representation of 0.4 in the unipolar and -0.2 in the bipolar format. While the unipolar format can only be used for representing positive data, the bipolar format can deal with both positive and negative values. With the same length bit-stream, however, the precision of unipolar format is twice that of the bipolar format. To represent a real number with a resolution of 2^{-M} in the unipolar format, a stream of 2^M bits is required.

Weighted binary radix has been the dominant format for representing numbers in the field of computer engineering since its inception. The representation is compact; however, computing on this representation is relatively complex, since each bit must be weighted according to its position. A stochastic representation is much less compact than conventional weighted binary radix. However, complex operations can be performed with remarkably simple logic. For example, a single standard AND gate performs multiplication with the unipolar representation; a single XNOR gate performs multiplication with the bipolar representation. A multiplexer implements scaled addition and subtraction. Complex functions, such as exponentials and trigonometric functions, can be computed through polynomial approximations with less than a dozen gates [42, 4]. Over a wide range of arithmetic functions, a reduction in area of $50\times$ or $100\times$ compared to conventional implementations is common [6], [4].

In addition to producing simple and compact logic, a stochastic representation offers the advantage of error tolerance [2, 21, 4, 1]. In a noisy environment, bit flips will affect all the bits with equal probability. With a conventional binary radix representation, the high-order bits represent a large magnitude; accordingly, faults in these bits can produce large errors. In contrast, with a stochastic representation, all the bits are equally weighted. Hence, a single flip results in a small error. This error tolerance scales to high error rates so that multiple bit flips produce only small and uniform deviations from the nominal value.

Progressive precision [1] is another interesting advantage of computation on stochastic bit-streams. The quality of the results improves as the computation proceeds. This is because short sub-sequences of long random bit-streams provide low-precision estimates of the streamsâĂŹ values. This property can be exploited in making quick decisions on the input data and so increasing the processing speed.

Given an input value, say in binary radix, the conventional approach for generating a stochastic bitstream with probability x is as follows. Obtain an unbiased random value $0 \leq r \leq 1$ from a random [43][44] or pseudorandom source [45, 46]; compare it to the target value x; output a one if $r \leq x$ and a zero otherwise. Figure 1.1 illustrates the approach. The "random number generator" is usually a linear-feedback shift register (LFSR), which produces high quality pseudo-randomness [45]. Assuming that the pseudo-random numbers are uniformly distributed between $0 \ldots 2^M - 1$, the value stored in

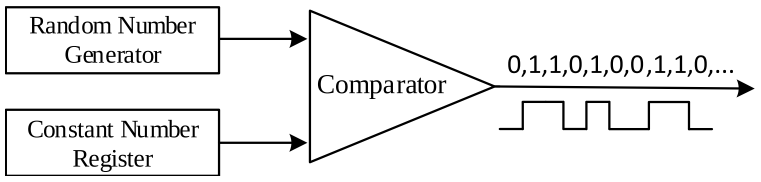


Figure 1.1: Stochastic Number Generator.

the constant number register should be $2^M \cdot x$. In the output, each bit is one with pseudo-probability $2^M \cdot x/2^M = x$ [12, 31].

The obvious disadvantage of SC, however, is the latency. A stochastic representation is exponentially longer than conventional binary radix. This translates to long operation times, particularly if high accuracy is required. Long bitstreams can be compensated for, to some extent, by shortened clock cycles. Nevertheless, long latencies translate into high energy consumption which is often higher than that of its binary counterpart. Another disadvantage is the cost overhead of generating bitstreams. While the hardware to perform the computation is simple, generating random bitstreams is costly. Indeed, in prior work, stochastic bitstream generators accounted for as much as 80% of the area and power of stochastic circuit designs [2]. Factoring in the cost of the bit-stream generators, the overall hardware cost of an SC implementation is often comparable to that of a conventional binary implementation.

Recent work has shown that the same constructs used for computation on stochastic bit-streams can be used for computation on deterministic bitstreams, if these bitstreams are generated in some specific ways [47, 48]. The results are completely accurate with no inaccuracy caused by random fluctuation or correlation. While these deterministic methods are able to provide completely accurate results, they do not offer progressive precision. The output converges to the expected correct value slowly. This slow convergence makes the deterministic approaches inefficient for applications that can tolerate some inaccuracy (e.g., image processing and neural network applications).

Furthermore, the common focus of SC proposals from 1960s onwards has been stochastic logic (arithmetic), neglecting memory, which represents a crucial system component. Due to the difference in data representation, integrating conventional memory (designed and optimized for non-SC) in SC systems inevitably incurs a significant data conversion overhead.

This dissertation provides some new views to SC paradigm to address the above-mentioned limitations and challenges. We introduce novel synthesis methodologies and research directions to SC with the goal of mitigating the hardware cost overhead, reducing the processing time and energy consumption, and improving the accuracy. The remainder of this dissertation is organized as follows.

- In Chapter 2, we introduce a new, energy-efficient, high-performance, and much less costly approach for performing SC using time-encoded pulse signals. We explore the performance of different stochastic operations for data processing of time-encoded inputs. We discuss the advantages, challenges, and potential applications for computation on such time-encoded signals.

- In Chapter 3, we propose a novel area- and power-efficient approach for synthesizing sorting network circuits based on deterministic unary-style bit-streams. To mitigate the long latency of processing input digital bitstreams, we exploit the idea of time-encoding data. We validate the method with two implementations of an important application of sorting, median filtering.

- Chapter 4 addresses an important challenge with the recently developed deterministic methods of SC, the poor progressive precision of processing unary bitstreams. We improve the progressive precision property of these deterministic methods by generating pseudo-random and low-discrepancy deterministic bitstreams. Experimental results show a significant improvement in the processing time and energy consumption compared to prior work when the application can tolerate slight inaccuracy.

- Chapter 5 introduces a new advantage to SC paradigm, the skew tolerance of SC circuits. We develop Polysynchronous Clocking, a design strategy for optimizing the clock distribution network (CDN) of stochastic systems. By removing and relaxing the clock network, we achieve a significant improvement in the latency, area, and energy consumption of stochastic systems while keeping the quality of the results. We show that circuits designed with either of these polysynchronous approaches are as tolerant of errors as conventional synchronous stochastic circuits.

- In Chapter 6, we rethink the memory system design for SC by integrating analog memory with conventional stochastic systems. A seamless stochastic system, StochMem, is introduced which features analog memory to trade the energy and area overhead of data conversion for computation accuracy. StochMem can reduce the energy wasted in the conversion units significantly at the cost of a slight loss in computation accuracy.

- Chapter 7 summarizes the contributions of this dissertation and present important future directions.

Chapter 2

Time-Based Computing with Stochastic Constructs

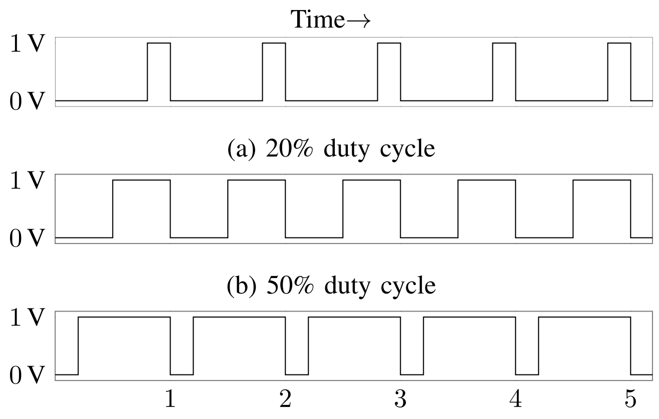
This chapter explores an evolution of the concept of stochastic computing (SC). Instead of encoding data in space, as random bitstreams, we encode values in time. Computation is performed on analog periodic pulse signals. We review the performance of different stochastic operations including operations with independent inputs and operations with correlated input for data processing of time-encoded data. We show how input data from a sensing circuit can be converted to time-encoded data and processed with digital stochastic logic. We discuss the advantages, challenges, and potential applications for computation on time-encoded signals. This chapter's material has been published in [48], [49],[18], and [50].

2.1 Motivation

A premise for SC is the availability of stochastic bitstreams with the requisite probabilities. Sensing circuits, such as image sensors, convert the sensed data (e.g., light intensity) to an analog voltage or current. The voltages or currents are then converted to digital form, as binary radix, with costly analog-to-digital convertors (ADCs). Finally, stochastic bitstream generators, consisting of random number generators (i.e., LFSRs) and comparators, are used to convert the data from binary radix format to stochastic bitstreams. Generating streams with a resolution of 2^M requires a generator that can produce 2^M unique values. Even ignoring the cost of ADCs which are similarly required in the conventional binary radix-based systems, the high cost of the pseudo-random number generation diminishes one of the main advantages of SC: low hardware cost. A high hardware cost also means a significant amount of power. Prior work has shown that bitstreams 256 to 1024 bit long are often required to satisfy output quality with SC circuits [4, 2]. Noting that $energy = power \times time$, the long run-time of stochastic circuits, together with the high power consumption of the SNGs, could lead to significantly higher energy use than their conventional binary counterparts [51].

In this chapter, we introduce a new, energy-efficient, high-performance, and much less costly approach for generating stochastic bitstreams using analog periodic pulse signals. As technology has scaled and device sizes have gotten smaller, the supply voltages have dropped while the device speeds have improved [52]. Control of the dynamic range in the voltage domain is limited; however, control of the length of pulses in the time domain can be precise [52, 53]. Encoding data in the time domain may be more accurate and efficient than converting signals into binary radix. This time-based representation is an excellent fit for low-power applications that include time-based sensors, such as image processing circuits in vision chips. Converting a variety of signals from an external voltage to a time-based representation can be done much more efficiently than a full conversion to binary radix.

The time encoding consists of periodic signals, with the value encoded as the fraction of the time that the signal is in the high (on) state compared to the low (off) state in each cycle. We call these pulse-width modulated (PWM) signals. By exploiting pulse width modulation, signals with specific probabilities can be generated by adjusting the frequency and duty cycles of the PWM signals. These signals can be treated as inputs to the same logical structures used in SC, with the value defined by the duty cycle. The duty cycle ($0 \leq D \leq 1$) describes the amount of time the signal is in the high (on) state as a percentage of the total time it takes to complete one cycle. As a result, the signal is encoded in *time*. The frequency $f = \frac{1}{T}$ of the PWM signal determines how long it takes to complete a cycle T and, therefore, how fast it switches between the high and the low



(a) 20% duty cycle

(b) 50% duty cycle

(c) 80% duty cycle

Figure 2.1: PWM signals with different duty cycles. (a) 20% duty cycle. (b) 50% duty cycle. (c) 80% duty cycle.

states. Thus, a PWM signal $f(t)$ is defined as:

$$f(t) = \begin{cases} y_{\text{high}} & N.T < t \leq N.T + (1 - D).T \\ y_{\text{low}} & N.T + (1 - D).T < t \leq (N + 1).T \end{cases}$$

where y_{high} and y_{low} are the high and low values of the signal, $N = 0, 1, 2, \cdots$ are the consecutive PWM cycles, and D is the duty cycle. Figure 2.1 shows three PWM signals with different duty cycles D when $T = 1$, $y_{\text{high}} = 1V$, and $y_{\text{low}} = 0V$.

Our approach is motivated by the following observation: a stochastic representation is a uniform, fractional representation. All that matters in terms of the value that is computed is the fraction of the time the signal is high. For example, if a signal is high 25% of the time, it is evaluated as 0.25 in the unipolar format. Similarly, PWM signals can be treated as time-encoded inputs with values defined by their duty cycle. For example, the PWM signals shown in Figure 2.1 represent 0.2, 0.5, and 0.8 in the unipolar and -0.6, 0.0, and 0.6 in the bipolar representation.

Alaghi et al. [1] proposed a specific design of an SNG unit for vision chips. Vision chips have image sensors that convert the perceived light intensity to an analog electrical voltage. The sensed voltage is converted to a stochastic number by comparing it to a random voltage generated by an LFSR-based counter and a digital-to-analog converter (DAC). Figure 2.2 illustrates their approach. We will show that, by working with PWM

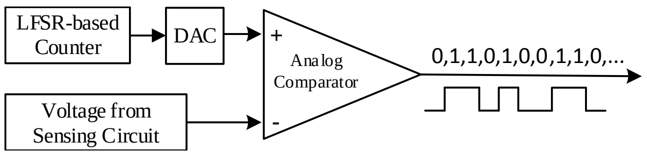


Figure 2.2: SNG proposed in [1] for vision chips.

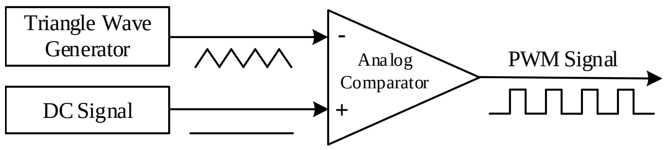


Figure 2.3: A common analog PWM generator.

signals, we can eliminate both the DAC as well as the LFSR. The result is a much less costly SNGs for applications that have analog electrical voltages as inputs.

2.2 PWM as the Stochastic Number Generator

In many electronic systems, existing analog inputs or onboard microcontrollers can be employed to generate PWM signals [54]. The simplest way to generate a PWM signal is to feed a sawtooth wave into the first input of an analog comparator and a control voltage into the second. The frequency of the sawtooth waveform determines the sampling rate of the signal. Thus, by changing the frequency of this wave, one can adjust the frequency of the generated PWM signal.

Figure 2.3 shows a common design for an analog PWM generator. The duty cycle of the PWM signal is set by changing the DC level of the input signal. The higher the DC level is, the wider the PWM pulses. The range of the DC signal varies between the minimum and maximum voltages of the triangle wave. For example, if we adjust the DC signal to have a level exactly half-way between the minimum and maximum, the circuit will generate a PWM signal with a duty cycle of 50%. This will correspond to an input value of 0.5 in the unipolar and 0.0 in the bipolar representation.

Figure 2.4 shows the design of a low-cost PWM generator, consisting of a ramp generator, a clock signal generator, and an analog comparator. The input is a current coming from a sensing circuit that controls the duty cycle of the PWM signal. The clock

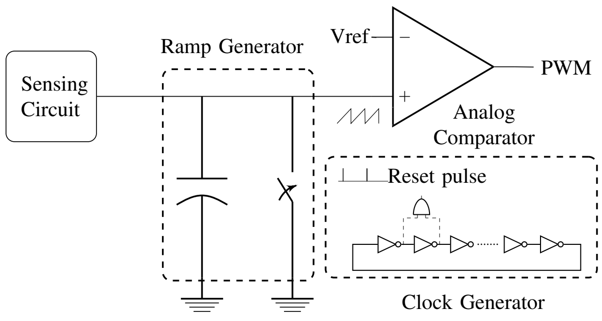


Figure 2.4: The design of our PWM generator. The duty cycle is determined by the current coming from the sensing circuit (a photodiode, or a voltage controlled current source, etc) and the Reset pulse defines the frequency of the PWM signal. Vref is a fixed reference voltage.

generator provides the required *Reset* signal which determines the frequency of the PWM signal. Ring oscillators consisting of an odd number of inverter gates can be used as the clock generator. The frequency of the *Reset* clock can be adjusted by either changing the supply voltage or changing the number of inverters in the oscillator. In the 45nm technology, a ring of approximately 89 inverter gates can generate a local clock with a period of 1ns with a supply voltage of 1.0V.

Table 2.1 shows an area-power comparison of the proposed PWM generator shown in Figure 2.4 with prior methods for SNGs: 1) the LFSR-based method in [2], and 2) the method proposed for vision chips in [1]. The results are for 45nm technology. We assume that the inputs are analog voltages or currents coming from a sensing circuit.

Table 2.1: Area-Power comparison of different SNGs

SNGs	Unit	Area (μm^2)	Power @1-3GHz (μW)
Conventional SNG [2] (with 8-bit LFSR)	LFSR+Comparator	248	335-1013
	ADC [55]	>400,000	>10,000
Special SNG for vision chips [1]	LFSR	167	298-892
	DAC + Comparator	equivalent to an ADC	
PWM Generator (1-3 GHz freq.)	Comparator	20-58	65-192
	Ramp Generator	10-32	11-29
	Clock Generator	124-37	∼175

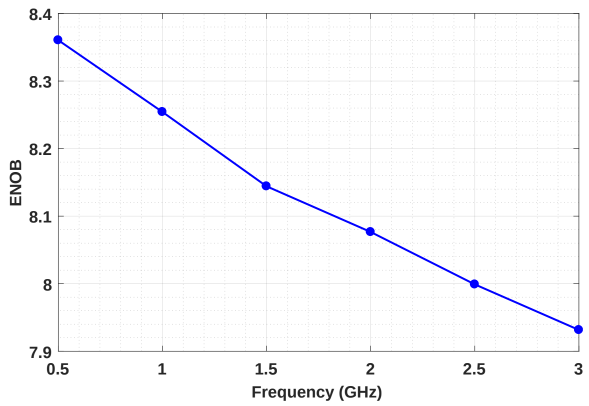


Figure 2.5: The ENOB of the proposed PWM generator shown in Figure 2.4 when generating PWM signals with frequencies from 0.5 to 3 GHz. More detail on the noise modeling of the implemented PWM generator will be discussed in Section 2.5.2.

The effective number of bits (ENOB) corresponding to different frequencies of the PWM generator is shown in Figure 2.5. Analog-to-digital convertors (ADCs) are used to obtain a digital representation for the LFSR-based method. The cost of a 45nm SAR ADC is taken from [55]. The special SNG proposed for vision chips resembles an ADC; we assume that it is roughly as expensive as a SAR ADC. The Synopsys Design Compiler was used to synthesize the SNGs. The results in Table 2.1 demonstrate that our mixed-signal method, based on PWM generators, has much lower area and power costs than the prior methods in cases where the inputs are in analog voltage or current form. Accordingly, the approach is a good fit for real-time image processing circuits, such as those in vision chips. These have image sensors that convert the perceived light intensity to an analog voltage or current.

Note that in prior methods a counter was used to convert stochastic streams back into real values in the digital domain. To convert the stochastic signals directly to a value in the analog domain, prior work used a simple RC integrator circuit to average the signal [56, 57]. For a faster response time, we use a Gm-C active integrator to average the output from processing PWM signals and measure the fraction of the time that the signal is high. For example, for a PWM signal with a period of T, duty cycle of D, $y_{\text{high}} = 1V$, and $y_{\text{low}} = 0V$, the integrator gives the average value of the first period of

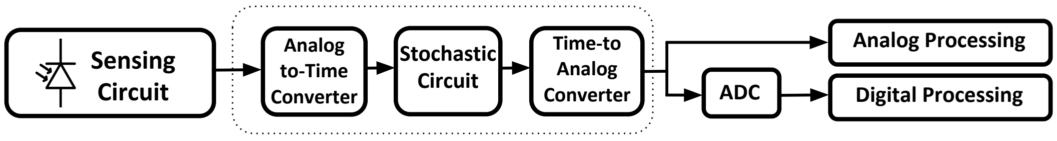


Figure 2.6: Time-based computing with stochastic constructs. An ATC converts the sensed data to a time-encoded pulse signal. The converted signal is processed using the stochastic circuit, and the output is converted back to a desired analog format using a TAC.

the signal as follow:

$$\bar{y} = \frac{1}{T}\int_0^T f(t)dt = \frac{1}{T}(\int_0^{(1-D)T} y_{\text{low}}dt + \int_{(1-D)T}^{T} y_{\text{high}}dt)$$
$$= \frac{1}{T}(T.(1-D).y_{\text{low}} + T.D.y_{\text{high}}) = D$$

2.3 Stochastic Systems with Time-Encoding Signals

Figure 2.6 shows the flow of computing on time-encoding signals. Assuming that the sensing circuitâĂŹs output is in voltage or current form, an analog-to-time converter (ATC) circuit (i.e., a PWM signal generator) is used to convert the sensed data to a time-encoded pulse signal. The converted signal is processed using the same circuit constructs as are used in SC. The output is converted back to a desired analog format using a time-to-analog converter (TAC) (i.e.,a voltage integrator). In what follows, we discuss the implementation of basic stochastic operations operating with PWM signals. Then we extend the discussion to more complex examples consisting of a multi-level combination of stochastic operations.

2.3.1 Stochastic Operations with PWM signals

Stochastic operations can be divided into two main categories with respect to correlation between their inputs: operations that require independent (i.e., uncorrelated) inputs such as multiplication and scaled addition, and operations that require highly correlated inputs such as absolute-valued subtraction, minimum and maximum value function, and comparison.

Multiplication

In the SC representation, a single AND (XNOR) gate performs multiplication if the unipolar (bipolar) format is used. The multiciplication operation presumes that the inputs are independent, uncorrelated streams [31]. Connecting two PWM signals with the same duty cycle and the same frequency to the inputs of an AND gate will evidently not work. It produces an output signal equal to the two inputs, not the square of the value as required. However, as we will show, one can use PWM signals provided that they have different frequencies (recall that we represent values by the duty cycle of PWM signals, not their frequency).

Instead of continuous-valued time signals, assume for the sake of argument that PWM signals are represented as bitstreams. For instance, assume an input value $X = 3/5$ (a signal with duty cycle of 60%) is represented by the bitstream 11100, and an input $Y = 1/2$ (a duty cycle of 50%) is represented by the bitstream 1100. Note that the stream for X has length 5 while that for Y has length 4. Suppose we multiply X and Y with an AND gate. Let the bitstreams run for 20 clock cycles, corresponding to 4 repetitions of X and 5 repetitions of Y. Taking the bit-wise AND of the streams

$$\begin{array}{r@{\,=\,}l} \text{X} & 11100111001110011100 \\ \text{Y} & 11001100110011001100 \\ \hline \text{X.Y} & 11000100000010001100 \end{array}$$

we observe $6/20$ ones in the output, the expected value, since $3/5 \times 1/2 = 6/20$. The results of this sort of multiplication operation is always correct if one chooses stream lengths that are *relatively prime* and let them run up to *the common multiple*. This is because when the length of the inputs are relatively prime, the difference between the lengths results in a new phase between the signals in each repetition until they get to the common multiple. A new initial phase in each repetition causes each bit of the first bitstream to see every bit of the second stream. This is, intuitively, equivalent to sliding one bitstream past the other. The bitstreams are therefore multiplied by convolving through sliding and ANDing repeatedly [58, 59, 47].

Proof. Let a/m be represented by a stream of m bits consisting of a bits of 1's with the rest of the bits being 0. Similarly, let b/n be represented by a stream of n bits with b bits of 1's and the remaining bits being 0. Assume that we repeat both streams to reach

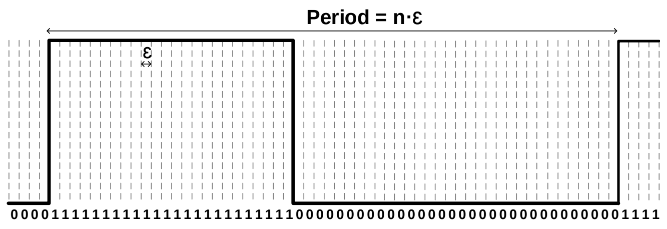


Figure 2.7: Discretizing a continuous PWM signal.

a total of the least common multiple (LCM) number of bits, or for simplicity mn bits in each stream. Applying an AND gate to these streams, we will have $a \times b$ bits of 1's if and only if the set $\{mk + i \,(\text{mod } n) : k = 0, 1, \ldots, n-1\}$ is a complete set of residues mod n. Here, i is the position of any 1 bit in the first stream. The first observation is that, whether the above holds or not does not depend on i. The second observation is that, when $i = 0$, this statement is true if and only if m and n are relatively prime. Therefore, ANDing the above streams produces $\frac{a}{m} \times \frac{b}{n}$ if and only if m and n are relatively prime. Q.E.D.

This argument can be easily expanded to analog PWM signals if the continuous signals are discretized into bitstreams, as illustrated in Figure 2.7. A PWM signal can be discretized into a bitstream by dividing the signal into pulses of size *epsilon* and assigning 0/1 bits to these pulses. The relatively prime length rule is then applicable to this discrete representation of the PWM signals and continues to hold as $\epsilon \to 0$. Note that in signal processing terminology, PWM signals with relatively prime periods are inharmonic.

To illustrate this argument, we simulated multiplication on a thousand sets of random input values represented by PWM signals in MATLAB [60]. We fixed the period of the first PWM signal at 20 ns while varying the period of the second from 1 ns to 20 ns in increments of 0.1 ns. For each pair of periods, we converted the randomly generated sets into corresponding PWM signals and then performed multiplication for 1000 ns. The accuracy of the results was verified by calculating the difference between the expected value and the measured output value for all sets. To convert the output signals into deterministic real values, we measured the fraction of the time that the output is high and divided this by the total time. The average error rates for multiplication for different

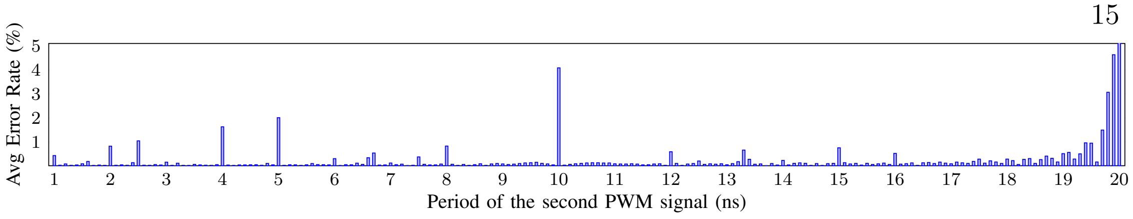


Figure 2.8: Average error rates when performing a multiplication operation using an AND gate for 1000 ns on 1000 sets of random input values when the inputs are represented using PWM signals. The period of the first input is set at 20 ns while the period of the second changes from 1 to 20 ns.

pairs of periods are shown in Figure 2.8.

As can be seen in Figure 2.8, with the period of the first PWM input signal fixed at 20 ns, choosing 1 ns, 2 ns, 2.5 ns, 4 ns, 5 ns, 8 ns, 10 ns, 12 ns, 15 ns, 16 ns, or values very close to 20 ns as the period of the second PWM input signal produces poor results. This can be attributed to an aliasing effect that occurs with periodic signals that are harmonically related. Eliminating these choices, the measured average error rate for other values was always less than 0.5%. Note that these results could ideally be extended to any other range of periods.[1] For example, knowing that 20 ns and 13 ns is a good pair, periods of 2 ns and 1.3 ns, or 10 ns and 6.5 ns would work equally well. From this observation we make our first conclusion:

Conclusion 1. Stochastic multiplication of numbers represented by PWM signals produces highly accurate results if the signals are not harmonically related.

With inharmonic PWM signals as inputs of multiplication, the fraction of time that the output signal is high will converge to the expected value eventually. However, stochastic circuits are not energy-efficient if the operations run more than what they actually need to. The question is: How many cycles of PWM signals are required to reach to a reasonable accuracy? Figure 2.9 shows an example of multiplying two stochastic numbers, 0.5 and 0.6, represented using two PWM signals. The period of the first PWM signal is 20 ns and that of the second is 13 ns. The figure shows that, after performing the operation for 260 ns, the fraction of the total time the output signal is high equals the value expected, when multiplying the two input values, namely 0.3.

[1] In practice, the resolution or effective number of bits (ENOB) of the PWM signals can affect the accuracy and so limits the extension range.

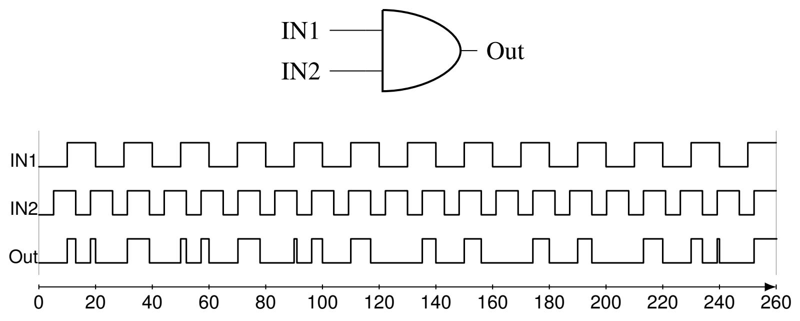


Figure 2.9: Example of multiplying two PWM signals using an AND gate. IN1 represents 0.5 (50% duty cycle) with a period of 20 ns, and IN2 represents 0.6 (60% duty cycle) with a period of 13 ns. The output signal from t=0 to 260 ns represents 0.30 (78 ns/260 ns=3/10), the expected value from multiplication of the inputs.

Expanding the example above to different operation times, Figure 2.10 shows the average error rates of multiplying 1000 pairs of random numbers represented by PWM signals when a fixed period of 20 ns is selected for the first and a fixed period of 13 ns is chosen for the second. We vary the operation time. As the figure shows, the output of multiplications converges to the expected value if the operations continue at least up to the least common multiple (LCM) of the periods of the input signals (here, $20 \times 13 = 260\ ns$). The best possible accuracy is obtained when the operation is run for exactly the LCM (260 ns) or multiples of the LCM (520 ns and 780 ns). Running the operation longer than the LCM does not help the accuracy. This is in contrast to prior SC approaches where increasing the length of bitstreams improves the quality of the results [2, 4, 1].

Let us consider the X.Y stream produced before. The LCM of the input streams was $4 \times 5 = 20$ and after exactly 20 cycles the expected output was produced. Continuing the operation for another 20 cycles produces exactly the same output with the same ratio of ones to the length of stream:

$$\begin{array}{rl} \text{X} & = 11100111001110011100\ 11100111001110011100 \\ \text{Y} & = 11001100110011001100\ 11001100110011001100 \\ \hline \text{X.Y} & = 11000100000010001100\ 11000100000010001100 \end{array}$$

Thus, we can say that the output has a period of 20 cycles. A similar result is observed when ANDing continuous PWM signals. The output has a period of the LCM. The

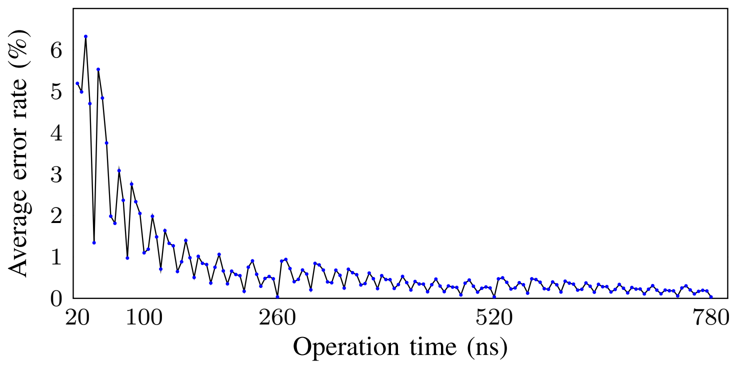


Figure 2.10: Average error rate of multiplying 1000 pairs of random numbers represented by PWM signals when varying the operation time. The period of the PWM signals corresponding to the first and to the second number in each trial is 20 and 13 ns, respectively.

signal produced from the first LCM to the second LCM is exactly the same as the signal produced from time=0 to the first LCM. This motivates our second conclusion:

Conclusion 2. The best accuracy when multiplying numbers represented by PWM signals is obtained when running the operation for the LCM or multiples of the LCM of the period of the inputs.

Knowing that relatively prime periods must be selected for the input signals and the multiplication operation should be run for the LCM of the periods, a new question arises:

Considering available sets of relatively prime periods, each with a different LCM, what is the best set of periods to reach to a desired accuracy? For example, (17 ns, 3 ns) and (17 ns, 7 ns) are two possible sets of periods to generate the PWM input signals for a multiplication operation. The first set has an LCM of 51 ns while the second's is 119 ns. Which one of these two sets is a better choice?

Figure 2.11 shows the average error rates of multiplying 1000 pairs of random numbers represented by PWM signals when different sets of relatively prime periods are selected as the periods of the input signals and the operations are run for the LCM of the periods. Each set of periods has a different LCM. As can be seen in the figure, the larger the LCM, the lower the average error rate. The reason is that larger LCMs are produced

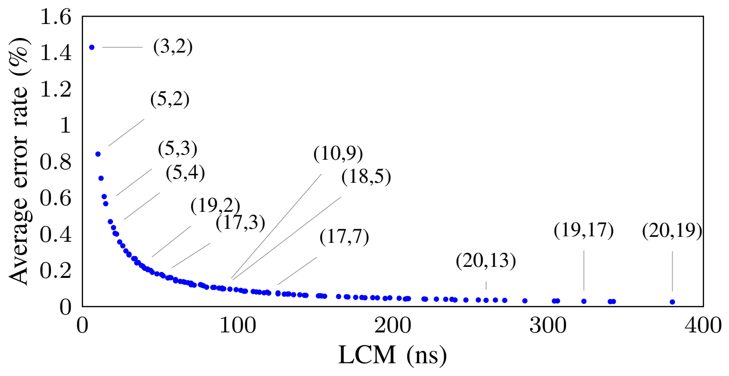


Figure 2.11: The average error rate for multiplying 1000 pairs of random numbers represented by PWM signals when the period of the first and the second PWM signal are relatively prime integers in the interval [2, 20]. A lower average error rate in the figure means a higher ENOB in the computations.

by longer periods and a longer period means a higher ENOB in representing the input values and so a higher ENOB in the computations. Note that while generating PWM signals with longer periods and so larger LCMs gives more accurate results, this requires a longer operation time. Thus, if a set of periods with a smaller LCM can satisfy the accuracy requirements, this might be the better choice. Thus, we conclude the following:

Conclusion 3. The larger the LCM of the periods of the PWM input signals, the higher the accuracy when performing multiplication.

Scaled Addition and Subtraction

Stochastic values are restricted to the interval [0, 1] (in the unipolar case) or the interval [-1, 1] (in the bipolar case). So one cannot perform addition or subtraction directly, since the result might lie outside these intervals. However, one can perform scaled addition and subtraction. These operations can be performed with a multiplexer (MUX) [4]. The performance of a MUX as a stochastic scaled adder/subtracter is insensitive to the correlation between its inputs. This is because only one input is connected to the output at a time. Thus, highly overlapped inputs like PWM signals with the same frequency can be connected to the inputs of a MUX. The important point when performing scaled

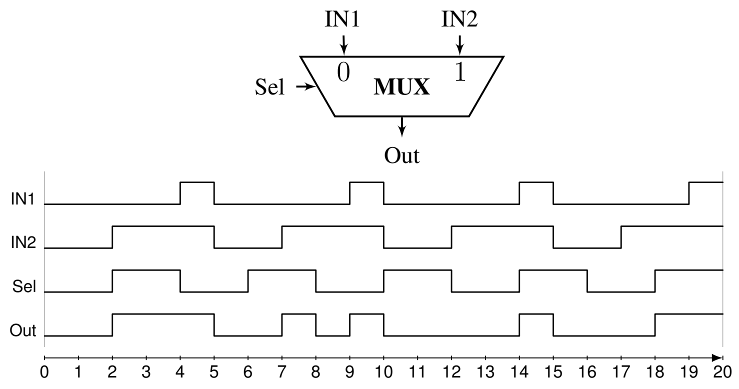


Figure 2.12: Example of the scaled addition of two PWM signals using a MUX. IN1 and IN2 represent 0.2 and 0.6 with a period of 5 ns, and Sel represents 0.5 with a period of 4 ns. The output signal from t = 0 to 20 ns represents 0.40 (8 ns/20 ns = 4/10), the expected value from the scaled addition of the inputs.

addition and subtraction with a MUX on PWM signals is that the period of the select signal should not be harmonically related to the period of the input signals. For example, 5, 5, and 4 ns is a good set of numbers for the period of the first, the second, and the select input signals, respectively.

Figure 2.12 shows an example of scaled addition on two stochastic numbers, 0.2 and 0.6, represented by two PWM signals (both have periods of 5 ns). A PWM signal with duty cycle of 50% and period of 4 ns is connected to the select input of the MUX. As shown, after performing the operation for 20 ns, the fraction of the total time the output signal is high equals the expected value, 0.40. The same argument we had for the multiplication operation also exists here – the scaled addition/subtraction operation should be run for the LCM or multiples of the LCM of the period of the input signals and that of the select signal to produce the correct output. Note that choosing different periods for the main inputs of the MUX results in a larger LCM and so results in a longer operation time. Furthermore, generating inputs with different periods requires extra clock generator circuitry. We conclude that it is most efficient to generate signals for the main inputs of the MUX with the same period.

A unique property of MUX-based operations is that large LCMs are not necessarily required to produce accurate results. Similar to what we saw for the multiplication

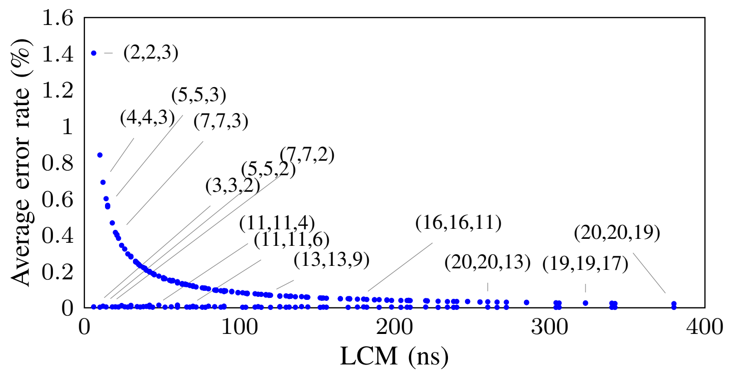
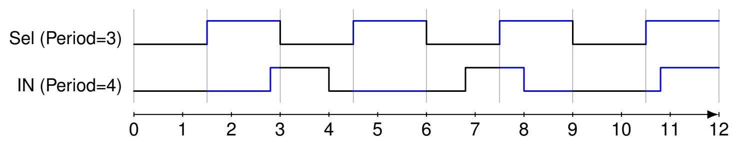


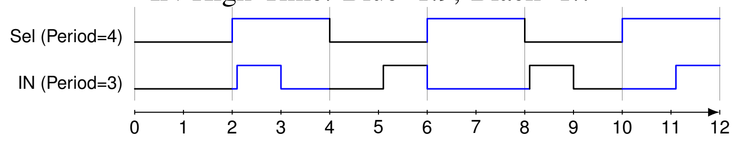
Figure 2.13: Average error rate of performing scaled addition on 1000 pairs of random numbers represented by PWM signals when the period of the first and the second PWM signal is the same but different and relatively prime with the period of the PWM select signal. The periods are selected from integers in [2, 20] interval.

operation, selecting inharmonic periods with a large LCM guarantees the accuracy of the results for the scaled addition/subtraction. However, it is possible for the stochastic MUX-based operations to produce accurate results even with inputs with very small periods. Figure 2.13 shows the average error rate of performing scaled addition when inharmonic PWM signals are connected to the main and select inputs of the MUX.

Each point in Figure 2.13 represents the accuracy and the LCM corresponding to one set of periods. The first and the second numbers in each set are the period of the main PWM inputs and the third number is the period of the select input. As the results show, when the period of the PWM select signal is an "even" value (2 ns, 4 ns,. . .) choosing "odd" periods as the period of the main PWM inputs result in highly accurate outputs. When choosing an "even" period for the inputs and an "odd" period for the select signal, a large LCM is needed to produce accurate results. The reason is shown in Figure 2.14. A select signal with an "even" period perfectly splits an input with an "odd" period in two periodic parts with the same duration at the high level. Thus, it does not matter to which input of the MUX the input signal is connected. However, in the case of an "odd" period for the select signal, connecting the input signal to different inputs of the MUX selects different parts of the input signal with different high durations. This motivates



IN High Time: Blue=1.9, Black=1.7



IN High Time: Blue=1.8, Black=1.8.

Figure 2.14: Examples of choosing an âĂIJoddâĂİ or an âĂIJevenâĂİ number as the period of the MUXâĂŹs select signal. The input is a PWM signal with D = 30%. Black (blue) lines are parts of the input signal that will be connected to the output of the MUX when the input is connected to the first (second) input.

our fourth conclusion:

Conclusion 4. Optimal choices for MUX-based operations are those with an "even" value for the period of the select input and an "odd" value for the period of the main inputs. The operation should run for the LCM of the periods.

Absolute Value subtraction

Correlation between the inputs of a stochastic circuit can sometimes change the functionality of a circuit, which might result in a more desirable operation. An XOR gate with independent inputs performs the function $z = x_1(1 - x_2) + x_2(1 - x_1)$. However, when fed with correlated inputs where the two input streams have maximum overlap in their 1s, the circuit computes $|x_1 - x_2|$. Consider $x_1 = 11101$ and $x_2 = 10001$, two 5-bit long correlated stochastic streams representing $4/5$ and $2/5$. Connecting these streams to the inputs of an XOR gate produces $Y = 01100$, the expected value from performing absolute valued subtraction. In this case, the output stream has the same number of bits as the input streams. This operation is particularly useful in stochastic implementation of image-processing algorithms, such as Robert's cross edge detection algorithm [1].

When working with PWM signals, high correlation or maximum overlap is provided

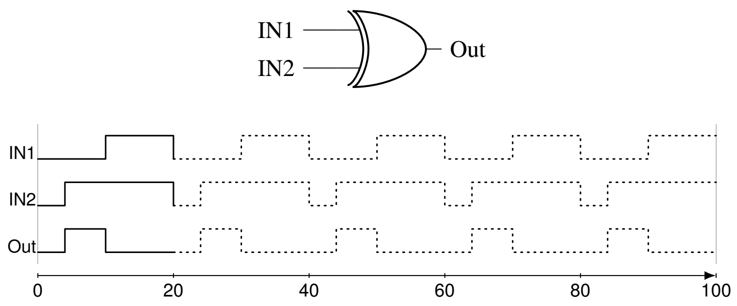


Figure 2.15: Example of performing stochastic absolute value subtraction using an XOR gate when two synchronized PWM signals are used as the inputs of the gate, one representing 0.5 (D = 50%) and the other 0.8 (D = 80%). Both PWM signals have a period of 20 ns. The output signal from t = 0 to 20 ns represents 0.3, the expected value for $|IN1 - IN2|$.

by satisfying two requirements: 1) choosing the same frequency for the input signals, and 2) having maximum overlap between the high parts of the signals. Thus, two PWM signals that have the same period, with the high part in each one located at the start or end of each period, are called correlated (or synchronized) signals. Figure 2.15 shows an example of performing absolute value subtraction on two synchronized PWM signals. As the figure shows, the correct output with the highest possible accuracy is ready right after performing the operation for only one period of the PWM input signals. Thus, the following holds.

Conclusion 5. For operations, such as absolute value subtraction, which work only with correlated inputs (synchronized PWM signals), the period of the output signal, and, thus, the operation time, equals the period of the input signals.

This conclusion introduces an important advantage of working on the synchronized PWM signals which is that they eliminate the requirement of running the operation for several repetitions of the input signals to obtain an accurate output signal. An important point, however, is that there is a limitation in using such operations that require highly correlated inputs. Providing the required synchronization (maximum high part overlap between the input signals) is difficult for the second (or higher) level of the circuit where the signals are the outputs of a previous level. Nonetheless, performing these operations

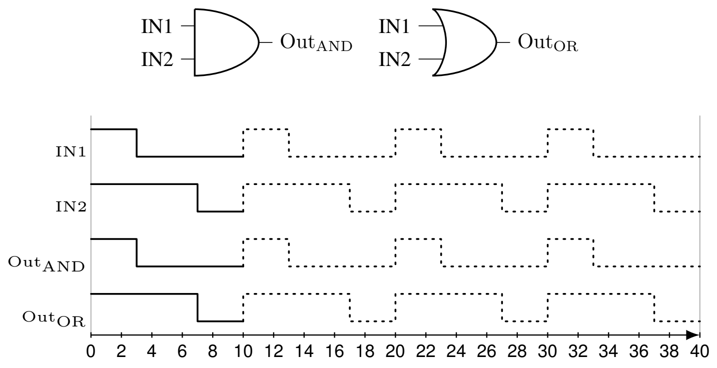


Figure 2.16: Performing minimum and maximum operations on two synchronized PWM signals: IN1 represents 0.3 and IN2 represents 0.7. Both PWM signals have a period of 10 ns.

can still be advantageous at the first level of circuits.

Minimum and Maximum

An AND gate with independent inputs works as a multiplier. However, with highly correlated inputs, it gives the minimum of the two stochastic streams. An OR gate supplied with highly correlated streams gives the maximum of the two stochastic streams. Thus, a basic sorting unit can be constructed with only an AND and an OR gate: supplied with two correlated inputs, it produces the smaller of the two values on one output line, and the greater of the two on the other. Such a low-cost implementation of sorting can save orders of magnitude in hardware resources and power when compared to the costs of a conventional binary implementation. As we will discuss in Chapter 3, such circuits are important for low-cost implementation of applications such as median filtering.

Figure 2.16 shows two synchronized PWM signals and the outputs of performing the minimum and maximum operations on these. As can be seen, the expected output is produced after a single cycle of the PWM input signals. Continuing the operations for additional cycles (the dotted lines) does not improve the accuracy of the results.

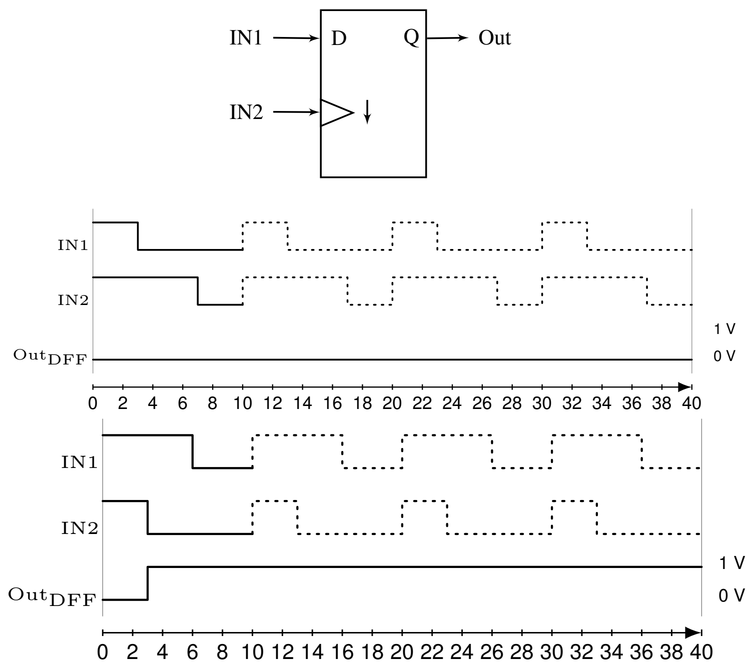


Figure 2.17: Comparing stochastic numbers (SNs), represented by synchronized PWM signals, using a D-type flip-flop: (up) IN1 $<$ IN2, and thus Out=0; (down) IN1 $>$ IN2, and thus Out=1.

Comparison

Comparison of stochastic numbers (SNs) is another common operation in stochastic circuits. Li and Lilja [61] proposed a stochastic comparator using an FSM-based stochastic *tanh* circuit developed by Brown and Card [31]. However, FSM-based circuits are often very expensive to implement. Figure 2.17 shows how to use a simple D-type flip-flop as a stochastic comparator. For correct functionality, the inputs of the flip-flop must be correlated. For a digital representation, all 1s in each stream must be placed together at the beginning of the stream. The first SN should be connected to the D input, and the second one should be connected to the falling edge triggered clock input. The output of comparing two SNs, N1 and N2, will be 0 if IN1 $<$ IN2, and 1 otherwise.

Figure 2.17 also shows two possible cases of comparing SNs, represented by PWM signals using a D-type flip-flop. When IN1 is smaller than IN2, the falling edge of the PWM signal representing N2 causes the flip-flop to sample a low-level signal, and thus logical-0 is produced at the output. When N1 is greater than N2, the PWM signal representing N1 is still at a high level when the falling edge of IN2 occurs. So, logical-1 will be produced at the output of the flip-flop.

2.3.2 Multilevel Circuit PWM signals

In the following, we briefly discuss the functionality of multilevel stochastic logic when PWM signals are used as the inputs of the circuit. An interesting point in performing stochastic operations on PWM signals is that the output of each level can be used as the input of the next level even though the output is not a PWM signal. When connecting two PWM signals to a stochastic operator, the output is a conventional stochastic number whose value cannot be found from the duty cycle but rather by probability of being in the âĂIJhighâĂİ state. However, the main difference between such an output with a conventional random stochastic signal is that, since the primary inputs were PWM signals, the generated output is a periodic signal. This property allows us to use the output of each level as the input to the next level. By knowing the period of the output signal, the obtained signal and some new signals that are not harmonically related can be used in the subsequent levels.

Consider the example presented in Figure 2.18.a, a three-level circuit multiplying four PWM signals with periods of $P1, P2, P3$ and $P4$. We want to choose the periods of the inputs and the required operation time which can lead to accurate outputs. Based on the conclusions in Section 2.3.1, $P1$ and $P2$ should not be harmonically related. The output of the AND1 gate is a signal with a period of $P1 \times P2$. The accuracy of the output produced by AND2 depends on the output of AND1 and also on $P3$, the period of the third PWM signal. $P3$ should not be harmonically related to the period of the signal generated at the output of AND1, and so to $P1$ and $P2$. Finally, P4 should not be harmonically related to $P1$, $P2$, and $P3$. The final output has a period of $P1 \times P2 \times P3 \times P4$, so the circuit must run for this amount of time to produce an accurate result.

Expanding the example mentioned earlier to circuits multiplying N PWM signals with N periods that are not harmonically related, the operation time must be the LCM of *all* these periods. The important trade-off here is to select small or large periods for these signals. Small periods results in a small LCM, and so need a shorter operation

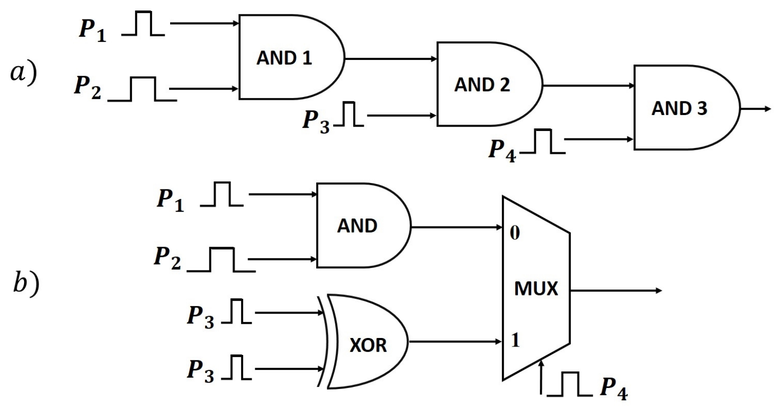


Figure 2.18: Examples of multilevel stochastic circuits.

time. Larger periods have larger LCMs and so require a longer running time. As shown in Figure 2.11, the larger the LCM, the higher the accuracy of multiplication. Thus, selecting the period of the PWM signals for such circuits depends on the accuracy and timing expectations.

The circuit presented in Figure 2.18.b incorporates all three sorts of basic operations. The AND gate's output has a period of $P1 \times P2$ while the output of the XOR gate has a period equal to the period of its inputs, or $P3$. The minimum operation time for this circuit is obtained when the MUX's inputs have the same periods ($P1 \times P2 = P3$). $P3$ must be an "odd" number while a small even value must be selected for P4. For this circuit the total operation time will be $P3 \times P4$. In cases where $P3 \neq P1 \times P2$, the total operation time will be the LCM of the period of all inputs, or $P1 \times P2 \times P3 \times P4$.

2.4 Experimental Results

To validate our ideas, we used stochastic implementations of two well-known digital image processing algorithms, the Robert's cross edge detection algorithm and the Gamma correction function. The core stochastic computation circuit for the Robert's cross algorithm was taken from [1], and the core logic for the gamma correction algorithm was taken from [2] (both shown in Figure 2.19). In the rest of this section, when we refer to the "prior" approach, we pair the core stochastic logic with input SNGs (LFSR+comparator as shown in Figure 1.1), and output counters to convert stochastic bitstreams to binary

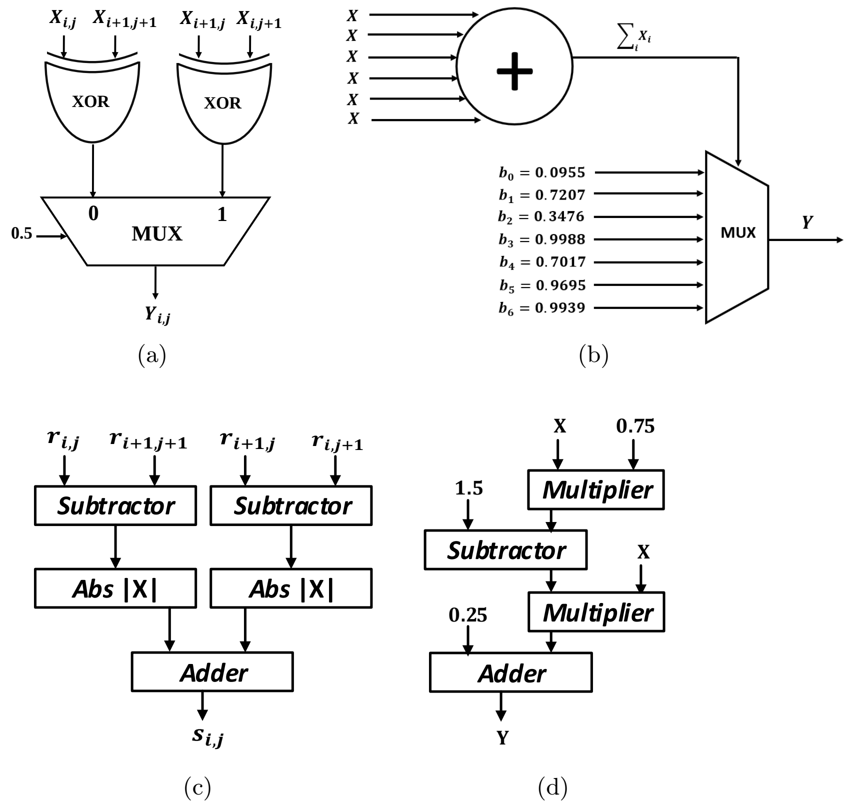


Figure 2.19: Robert's cross edge detection algorithm a) core stochastic logic [1], c) conventional binary implementation. Gamma correction function b) core stochastic logic based on ReSC architecture [2], d) a conventional binary implementation [1].

numbers. When we refer to the "PWM" approach, we pair the core stochastic logic with PWM generators (Figure 2.3) and a voltage integrator to generate the analog output. The conventional binary implementations of the selected algorithms are also shown in Figure 2.19.

We implemented SPICE netlists for the stochastic circuits described earlier. Two 128×128 sample images (16384 pixels each) were selected for the simulations. Simulations were carried out using a 45-nm gate library in HSPICE. We implemented the PWM generator proposed in Figure 2.4 for converting input pixel values into the corresponding PWM signals. Figure 2.20 shows the input sample images as well as the output of processing these images using a deterministic, software-based implementation of each

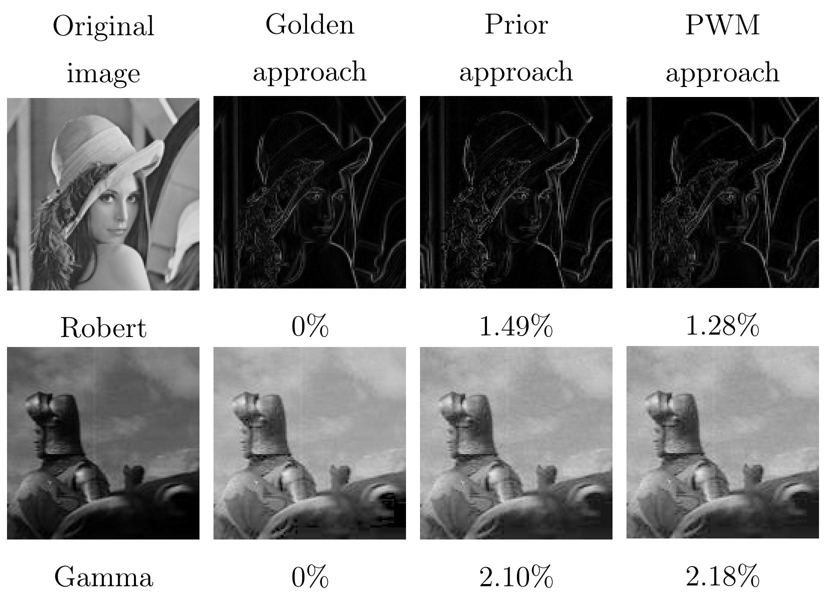


Figure 2.20: Original 128×128 sample images and the outputs of processing the input images using the "golden approach", the "prior approach", and the proposed PWM approach with the Robert's cross stochastic circuit (first row) and the Gamma correction stochastic circuit (second row).

algorithm in MATLAB. We call this the "golden" approach, with a 0% average error rate. Also, we simulated the circuit operation on randomized stochastic streams in the "prior" approach. The conventional SNG described in Figure 1.1 was used for converting input pixel intensities into stochastic bitstreams. An 8-bit maximal period LFSR was used as the pseudo-random number generator. Bitstreams 256 bit long were generated for each input value. We calculate the average output error rate for the output image produced by the "prior" approach as follows:

$$E = \frac{\sum_{i=1}^{128}\sum_{j=1}^{128}|T_{i,j} - S_{i,j}|}{255.(128 \times 128)} \times 100$$

where $S_{i,j}$ is the expected pixel value in the output image and $T_{i,j}$ is the pixel value produced using the circuit.

To compare the operation time of the PWM approach with the delay of the prior approach, and also that of the conventional binary approach, we synthesized the

Robert's cross and the gamma correction circuits using the Synopsys Design Compiler vH2013.12 [62] with a 45nm gate library. The stochastic circuits had a critical path of 0.34 and 0.60 ns, respectively. In Sections 2.4.1 and 2.4.2, we first describe the process of synthesizing the selected circuits with the proposed PWM approach and then compare performance, area, and energy dissipation of the implemented circuits.

2.4.1 Case Study 1: Robert's cross edge detector

Each Robert's cross operator consists of a pair of 2×2 convolution kernels that process an image pixel based on its three neighbors as follows:

$$S_{i,j} = \frac{1}{2} \times (|r_{i,j} - r_{i+1,j+1}| + |r_{i,j+1} - r_{i+1,j}|)$$

where $r_{i,j}$ is the value of the pixel at location (i, j) of the original input image and $S_{i,j}$ is the output value computed for the same location in the output image. Figure 2.19.a shows the stochastic implementation of the Robert's cross algorithm proposed by Alaghi and Hayes [1], consisting of a MUX for the scaled addition and two XOR gates to perform the absolute value subtractions. This circuit is the core computation logic and is shared between the "prior" stochastic approach and our PWM approach.

Prior Method [1]

To generate the circuit for the prior approach, we pair the core stochastic logic of Figure 2.19.a with one LFSR and four comparators to generate the input streams feeding the XOR gates. Only one LFSR is used for the XOR input lines because Alaghi's approach relies on *correlated* bitstreams. Another LFSR and comparator is also necessary to generate the select stream. Note that when the input is given in analog voltage, coming from a sensing circuit, an ADC must also be used to convert the analog input signal into digital form. We ignore the ADC unit in our comparisons. If the cost of the ADC were to be added, our approach would have shown even larger gains compared to prior work. The output of the prior approach circuit is fed to a counter to convert the bitstream to a binary number.

The PWM Method

Next, we describe how we implemented the Robert's cross algorithm using the PWM approach. The core stochastic logic of Figure 2.19.a is paired with PWM generators that provide the input signals feeding the XOR gates, and the output of the MUX is fed to a voltage integrator circuit. The following steps are used to synthesize the circuit in the PWM approach:

Step 1. Frequency Selection. When using PWM signals as inputs of a stochastic circuit, one has to select appropriate frequencies. As discussed in Section 2.3.1, the inputs to an XOR gate must be two synchronized PWM signals to compute the absolute value subtraction. Since the MUX unit is also insensitive to the correlation between input signals, four synchronized PWM signals corresponding to four pixels of the image can be connected to the main inputs of the Robert's cross circuit. The important point here is to appropriately select the frequency of the PWM signal connected to the select line of the MUX. This select signal can be a clock signal which is a PWM signal with 50% duty cyle. The period of this signal must not be harmonically related to the period of the main inputs of the MUX. Since the period of the signal produced at the output of the XOR gates is the same as the period of their inputs, the period of the clock signal must not be harmonically related to the period of the circuit's main inputs. Considering the critical path (0.34 ns) as the minimum allowed period of the PWM signals, we chose 0.51 ns as the period of the main PWM input signals and 0.34 ns as the period of the select signal. These numbers are obtained by scaling (3 and 2 ns) down which is one of the best set of periods extracted in Section 2.3.1.

Step 2. Operation Time Determination. We showed that the results of performing stochastic absolute value subtraction is ready after running the operation for only one period of the input PWM signals. For scaled addition/subtraction operations, the best operation time is the LCM of the periods of the MUX select and input signals. Since we scaled (3 and 2 ns) down to (0.51 and 0.34 ns), the best operation time is also obtained by scaling their LCM down by the same scaling factor. Thus, the best operation time for the synthesized Robert's cross circuit in the PWM approach is 1.02ns.

Step 3. Clock Generation. Since the frequency of all four PWM inputs is the same, a clock generator with an oscillation period of 0.51 ns is enough to drive the main PWM generators. A second clock signal with a period of 0.34 ns is also necessary for the select line of the MUX. Thus, a total of two clock generators would be sufficient for generating the inputs of the Robert's cross circuit. We used rings of 43 and 29 inverters to generate the required clock signals.

Comparison

We processed each image pixel separately and computed the corresponding output value. Comparing the produced output image in the PWM approach with the golden image, the mean of the output error rates was 1.28%. Thus, the proposed approach could decrease the average error rate of processing the sample image when it is compared with that of the prior stochastic approach with 256-bit streams (1.49%). Considering the delay of the prior stochastic approach ($256 \times 0.34ns = 87.04ns$), the PWM approach decreases the processing time of each pixel by more than 98%, to only 1.02ns. Even if one argues that the quality of the 32-bit streams (1.98%) is enough for the prior approach, still the PWM approach has improved the operation time by 90%. The area, power, and energy consumption of the circuit when working with PWM signals are also presented and compared with the prior approach in Table 2.2. From the area, area $\times$ delay and energy numbers, we see that the proposed PWM approach has a significant cost advantage when compared with the prior stochastic approach.

Compared to the conventional binary implementation, although the PWM approach is slightly slower, it costs 63% less area, dissipates 12% less energy, and reduces the area-delay product by more than 50%. The main barrier to practical use of the prior stochastic implementation was its long latency and correspondingly high energy use. However, as the results presented in Table 2.2 show, the proposed PWM approach is able to implement the Robert's cross edge detection algorithm with the advantages of the stochastic design but as fast and energy-efficiently as the conventional binary design.

2.4.2 Case Study 2: Gamma Correction

A flexible and straight-forward method to utilize SC in different applications is to synthesize the SC circuits with a MUX-based architecture, called ReSC [2]. This design approach is simple and area-efficient, and is able to realize polynomial functions that can be translated to Bernstein Polynomials. The gamma correction function ($f(x) = x^{\gamma}$) is a popular pixel value transformation that can change luminance and tri-stimulus values in

Table 2.2: Area, delay, power and energy comparison of the implemented circuits for the conventional binary, prior stochastic and the proposed PWM approach. For the prior stochastic approach, we ignore the cost of the ADC. Delay and power numbers are reported for the maximum working frequency.

Circuit	Approach	Area (μm^2)				Delay	Power (μW)	Energy	Area×Delay
		Core	SNG	Output Circt.	Total	(ns)	(@max freq.)	(pJ)	($\mu m^2 \times \mu s$)
Robert	Conventional binary	1626	-	-	1626	0.78	1415	1.10	1.26
	Stochastic-Prior	22	739	199	960	87.04	2813	244.8	83.55
	Stochastic-PWM	22	464	110	596	1.02	943	0.96	0.60
Gamma	Conventional binary	1980	-	-	1980	1.03	973	1.00	2.03
	Stochastic-Prior	76	982	199	1257	153.6	1672	256.8	181.4
	Stochastic-PWM	76	678	110	864	1.8	1690	3.04	1.42

video and image processing systems. This function can be approximated using a Bernstein polynomial. A stochastic implementation of the gamma correction function for $\gamma = 0.45$ based on the ReSC architecture is shown in Fig. 2.19.b. The inputs to this system consist of six independent bitstreams, each with a probability corresponding to the value x of the input pixel (denoted as x in the figure), as well as seven random bitstreams set to constant values, corresponding to the Bernstein coefficients, $b_0 = 0.0955$, $b_1 = 0.7207$, $b_2 = 0.3476$, $b_3 = 0.9988$, $b_4 = 0.7017$, $b_5 = 0.9695$ and $b_6 = 0.9939$. Additional details of the circuit can be found in [2].

In the following, just as we did in Case Study 1, we use the same core stochastic logic for the prior and the PWM methods, but use different input SNG and output accumulation circuits.

Prior Method [2]

Based on the analysis done in [63], we can use delayed outputs of the same bitstream to generate multiple bitstreams with small correlations. That results in significant area savings to the original implementation in [2]. A second LFSR was used for generating the Bernstein coefficients, making a total of two LFSRs and eight comparators to generate all the necessary bitstreams in the "prior" approach.

The PWM Method

Here, we discuss the process of synthesizing the gamma correction circuit using the PWM approach. The same process can be easily adapted to implement any other function that can be realized with the ReSC architecture.

Step 1. Frequency selection. At any time, only one input of the MUX is selected to be connected to the output. As a result, the PWM signals corresponding to the Bernstein coefficients can be generated with the same frequency. However, the circuit needs some level of independence between the six PWM signals corresponding to the input value of x. Fortunately, providing the required independence does not necessarily require generating signals with different frequencies, as was the case with multiplication. In the prior stochastic approach, such independence could be provided by shifting the x streams for one or a few bits and so have a huge savings in the cost of SNG [63][64]. Similarly, we can use a phase shift technique for the PWM approach to make independent copies of x. An additional step will select the best set of shift phases for the x signals that can lead to high quality outputs. Synthesis results showed a critical path of 0.60 ns for the gamma correction circuit. Thus, accordingly, we chose 0.60 ns as the period of the x signals and 0.9 as the period of the Bernstein coefficient signals. These periods are the scaled versions of (2 and 3 ns).

Step 2. Operation Time Determination. Since the gamma correction circuit is built on a MUX-based architecture, accurate outputs can be produced if the circuit runs for the LCM of the period of the inputs and the period of the PWM signals corresponding to the input x. Thus, the best operation time for the selected periods is their first common multiple or 1.8 ns. Note that using the phase shifting technique does not increase the operation time and highly accurate output can still be produced in LCM time by choosing the phases of the x signals appropriately.

Step 3. Clock Generation. Two clock generators are necessary for the Gamma correction circuit. One for generating a clock signal with a period of 0.9 ns for the Bernstein PWM signals and another one for generating a clock signal with a period of 0.6 ns. The latter drives the PWM generators responsible for generating the x signals. We used rings of 79 and 53 inverters to generate the required clock signals with periods of 0.9 ns and 0.6 ns, respectively.

Step 4. Phase Shift Calibration. A supplementary step is required to synthesize the ReSC architecture in the PWM approach. In the ReSC circuits, the results of adding independent copies of signal x determine which input of the MUX at any time must be connected to the output. Having six similar PWM signals, each signal can be shifted for a phase between 0 to the period of the signal. When using a ring of inverters as the clock generator, clock signals with the same frequency but different phases can be extracted from different stages of the ring. For the gamma correction circuit, we needed six clock signals all with a fixed period of 0.6 ns but each with a different phase. In several trials, we measured the average error rates of processing 1000 random pixels when clock signals with different phases were extracted from different stages of the ring. For the final implementation, we chose the set of ring stages that led to the minimum average error rate.

Comparison

The pixels of the sample image were converted to their corresponding PWM signals and then processed by the implemented circuit. The mean of the error rates in processing all pixels of the sample image in the PWM approach was 2.18%, which is very close to the number reported for processing the sample image by the prior stochastic approach. The operation time for processing each image pixel has decreased from 153.6 ns for the prior approach to only 1.8 ns in the PWM approach. Also, the area×delay cost and energy consumptions are all significantly improved by the PWM approach when compared to the prior stochastic implementation. Note that we did not consider the cost of the required clock generator in the prior approach. If this cost were to be added, the improvement from the PWM approach would have been even greater.

Comparing the conventional binary implementation of the gamma correction function with the prior stochastic approach, we see that the latency of processing each image pixel, the energy dissipation, and the area-delay product of the stochastic approach are all significantly increased. The benefits of the prior stochastic approach are limited to around a 36% area saving and adding the ability to tolerate noise, which is an inherent property in SC. The PWM approach, on the other hand, not only inherits the noise tolerance advantage of the stochastic design, it also increases the area saving to 56% and brings the latency very close to the latency of the conventional binary design. Although the energy dissipation of the PWM approach is still more than that of the conventional design, it is much less than the energy dissipation of the prior stochastic approach.

2.5 Error Analysis

In this section, we first define different sources of error in performing stochastic operations on PWM signals and then discuss the noise model and noise performance of the implemented PWM generator.

2.5.1 Sources of Computational Error

There are five primary sources of error in performing stochastic operations on PWM signals.

1. E_G = Error in generating the PWM signals.

A PWM generator has some inherent inaccuracies in converting real values to corresponding PWM duty cycles. This inaccuracy can be defined as the difference between the expected and the measured duty cycle in the generated signal.

$$E_G = |D - \frac{1}{T} \times T_{high}|$$

In addition, achieving the desired frequency for the PWM signals is not always feasible, particularly when using ring oscillators as the clock generator. Changing the number of inverters is the simplest way to adjust the frequency of the oscillator. The oscillation period is twice the sum of the delay of all inverter gates, where the delay of one inverter gate in the selected 45-nm library is 5.69 ps. Considering that an odd number of inverter gates is required, the period can be increased (decreased) by adding (removing) an even number of inverters. Thus, the minimum change in period for this generator is 0.022 ns. This limitation in controlling the period of the PWM generators can affect the accuracy of operations. Note that in our simulations, the error introduced in generating PWM signals was always less than 0.4%.

2. E_S = Error due to skew noise.

For some stochastic operations, such as absolute value subtraction using XOR gates, perfectly synchronized PWM signals are necessary to produce accurate results. On-chip variations or other noise sources affecting ring oscillators can result in deviations from the expected period, phase shift, or the slew rate of the signals.

3. E_M = Error in measuring output signals.

An analog integrator can be used to measure the fraction of the time the output signal is high. Longer rise and fall times and imperfect measurement of the high and low voltages (corresponding to digital ”1” and ”0” values) result in inaccuracies in measuring the correct output value. We compared the output values measured by our SPICE-level implementation of the integrator with the expected values from measuring the outputs produced by the Robert’s cross and Gamma circuit under ideal signal levels (HSPICE .ideal) when processing sample images. The average error rate of the measurements was 0.16% for the Robert’s cross and 0.12% for the Gamma correction circuit.

4. E_T = Error due to truncation.

Truncation is another source of error in the PWM-based approach if the operation runs for any time other than the required operation time. For example, the multiplication operation must run the LCM or multiples of the LCM of the period of the PWM inputs to generate an accurate output. Running the operation for any time less or more than the LCMs introduces truncation error.

5. E_A = Error due to function approximation.

Functions implemented with SC typically must be approximated since a given function usually cannot be mapped directly to a stochastic operation. Our gamma correction operation, for example, used a Bernstein approximation of the exponential function. Prior work [2] has shown that a Bernstein approximation of degree of six is usually sufficient to reduce the average approximation error to below 0.1%.

The overall error, E_{Total}, for the stochastic operations performed on PWM signals is bounded by the sum of the above error components

$$E_{Total} = E_G + E_S + E_M + E_T + E_A$$

.

Considering the error rates we measured when processing the sample images using the synthesized Robert’s cross and Gamma correction circuits with the PWM approach, some of these sources of errors can offset or compensate for each other, resulting in an acceptable total error. Note that, in an actual chip fabrication, the effect of thermal noise and the influence of process and temperature variations might introduce more inaccuracy in the generated signals which could produce higher error rates. Still, as Figure 2.21

shows, we expect that even if these fabrication sources of error introduce up to 20% relative error in the duty cycle and period of the PWM input signals, the stochastic circuits can still produce outputs with acceptably small errors.

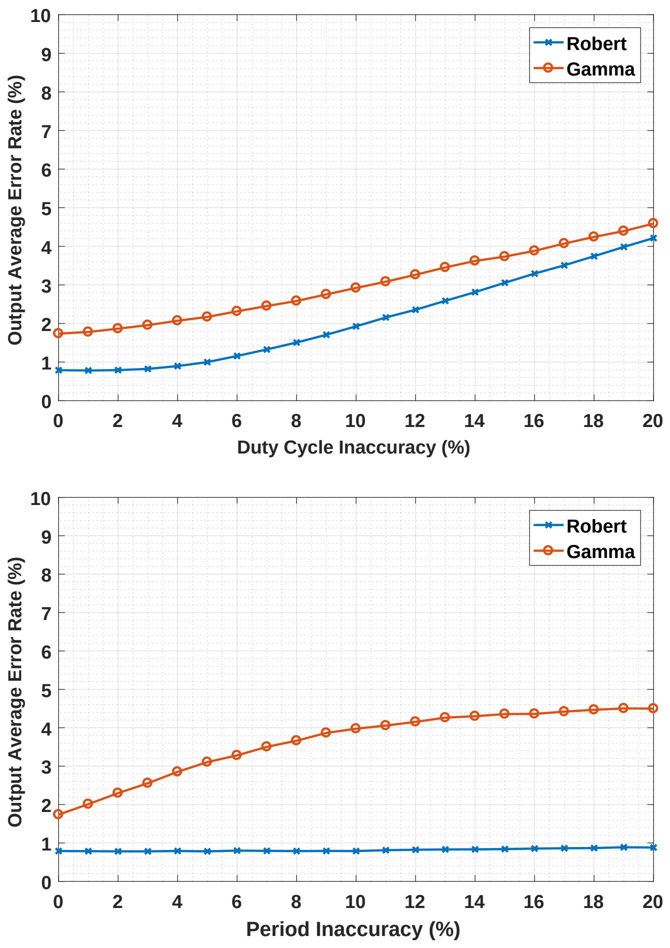


Figure 2.21: Average error rate of the output images when processing the sample images using the proposed PWM-based approach for different rates of inaccuracy in the duty cycle (top) and in the period (bottom) of the PWM input signals. PWM signals are generated using an ideal PWM generator in HSPICE and the output signals are converted back to real values using an ideal integrator. Twenty trials were performed for each inaccuracy rate to ensure statistically significant results.

2.5.2 Noise Modeling

Noise and linearity are definitely the most important concerns in analog circuits. In the following discussion, we analyze the noise contribution of each component in the PWM generator, and show that the proposed technique can satisfy the accuracy requirements even in the presence of thermal noise or process variations.

The ramp required for pulse width modulation is generated by charging a capacitor with a slope proportional to the input signal. If the input is coming from an image sensor, for instance, the output of the sensor is a current and can be directly integrated on a capacitor. On the other hand, there are cases such as the coefficient inputs of the ReSC architecture where the input signal is a constant voltage and an active integrator, such as Gm-C or R-OTA-C integrator, must be used. We analyze these two cases separately.

Input source: Image sensor. In order to achieve 8-bit accuracy in PWM generation, the pulse width error must be less than $(1/2^9) \times T \approx 0.002\ T$, where T is the period of the PWM signal. There are two sources of error in the PWM generator:

1) Thermal noise

a) Switched-capacitor noise: capacitors are inherently noiseless, but when they get switched, the thermal noise of the switch resistance accumulates on the capacitor, resulting in an equivalent rms noise voltage of KT/C [65]. This noise depends only on the capacitor size. Therefore, the maximum tolerable noise defines the minimum capacitance that can be used:

$$10 \log_{10} \frac{\frac{0.5^2}{2}}{\frac{KT}{C}} \geq 8 \times 6.02 + 1.78 = 50\ dB$$

$$\frac{KT}{C} < \frac{0.5^2}{2} 10^{-5} = 1.2510^{-6} \rightarrow C > 3.3\ fF$$

Since $C = 3.3\ fF$ was derived for room temperature, we choose $C = 5\ fF$ to allow some margin for temperature and process variations. This analysis shows a trade-off between capacitor area and circuit noise.

b) Comparator: the comparator is the key element in PWM generation. The comparator's resolution, i.e. the minimum voltage that causes a change in the output, determines the minimum detectable input current:

$$\text{The integration slope: } m_{LSB} = \frac{V_{res}}{t_{LSB}} = \frac{CMP_{res}}{0.002\ T} = \frac{i_{LSB}}{C}$$
$$\rightarrow i_{LSB} = \frac{CMP_{res}}{0.002\ T}$$

The comparator's resolution depends on the architecture. A typical comparator consists of a differential pair followed by a latch. The resolution of the comparator is given by $(V_{dd}/Comp_{gain})$ where $Comp_{gain} = \textit{pre-amplifier}_{gain} \times exp\ (t/\tau)$. We show the $\textit{pre-amplifier}_{gain}$ with Av. τ is the latch time constant measured by

$$\tau = \frac{C_L \text{ (load capacitance at the output of the comparator)}}{G_m \text{ (transconductance of the cross-coupled latch)}}$$

The above-mentioned equation shows that the comparator's resolution improves with time, i.e. one can achieve better resolution at the expense of longer delay [66, 67].

For 8-bit resolution with $1\ V\ V_{dd}$ for $1\,GHz$, the frequency$\rightarrow$ $Comp_{gain} > 512$, $t << 1\ ns$, $C_L = 1\ fF$, $Av = 16$, and t_d, or the maximum time that the comparator has to make a decision, is 0.001=1 ps. Thus,

$$Av * exp(t \times 10^{15} \times G_m) = 512$$

$$\rightarrow exp(10^3 \times G_m) > 32 \rightarrow 10^3 * G_m > \ln(32) = 3.45$$
$$\rightarrow G_m > 3.45\,mA/v$$

Since we have high gain in the input stage, the noise of the latch does not matter (because the latch noise is divided by the input gain). The noise of the input transistors can result in pulse width variations, also known as jitter. A common formula for calculating jitter noise is [65]

$$Jitter_{RMS} = \frac{Vnoise_{RMS}}{Slew\ rate}$$

Based on [68], we have

$$Jitter_{RMS} = \frac{\sqrt{4KT\gamma/G_m} \times \sqrt{f}}{I/Cm}$$

It is worth noting that the effect of comparator noise on PWM generator is the same as the ADC circuit presented in [1]. Also, note that process and temperature variation only affect the gain of the comparator, which can be considered during the design process.

2) Resetting speed

In each pulse generation cycle, the integrating capacitor must be discharged (reset) within the minimum time step, i.e. $\frac{T}{2^{N+1}}$. Therefore, the reset pulse width shrinks as the PWM frequency increases, imposing a limit on the maximum achievable speed. As calculated before, for 1-ns period and 8-bit accuracy, t_{min}=2 ps.

In summary, we have three sources of noise in the PWM generator: switched capacitor noise (KT/C), integrator noise, and comparator noise, where the following holds.

- KT/C is constant, because we change the current but the capacitor is fixed. For C=5 fF and room temperature, $(KT/C) = -57.81\,dB$
- The current has to scale linearly with speed, so the integrator noise decreases.
- Comparator noise results in jitter, so the impact increases with frequency. For $f = 1GHz$ it is $60\,dB$.
- Total distortion = integrator distortion (i.e. nonlinearity) = $-60\,dB$
- $SNDR = 10 * \log_{10}(\frac{0.5 \times V_{sig}^2}{Total_{noise}} + Total_{distortion}) = 6.02 * N + 1.76\ (dB)$.
- For $V_{dd} = 1v \rightarrow 0.5 \times V_{sig}^2 = 0.5$.
- For $f = 1GHz$, $Total_{noise} = 3 \times 10^{-6}$, so $SNDR = 51.5\,dB$ and $ENOB = 8.25$.

Input source: constant voltage. In case of voltage inputs, the transconductor (Gm cell) or the amplifier in the integrator also introduces noise, but the total noise is small and does not degrade the performance substantially.

2.6 Applications

Growth in digital and video imaging cameras, mobile imaging, biomedical imaging, robotics, and optical sensors has spurred demand for low-cost, energy-efficient circuits for image processing. Prior work on SC has shown this computing paradigmâĂŹs potential in low-cost implementation of image and video-processing algorithms. Image processing based on time-encoded signals could have significant impact in this application area, particularly when power constraints dominate. Time-encoded, mixed-signal processing can be performed on the same chip, with analog-to-time conversion followed by logical computation on the time-encoded signals, using stochastic constructs.

Mixed-signal design is attractive for VLSI implementations of neural networks (NNs) for reasons of speed and energy efficiency. Also, mixed-signal solutions do not suffer from the quantization effects that arise with analog-to-digital conversion. NNs are computationally complex, which makes them a good candidate for processing with low-cost stochastic logic. Digital bitstream-based processing of data in stochastic NN often requires running for more than 1,000 clock cycles to achieve an accuracy close to that of conventional deterministic fixed-point binary designs, which then leads to high energy consumption. Time-based SC has the potential to mitigate these costs, offering energy-efficient designs. Unlike conventional SC, the computations can be completely accurate with no random fluctuation. The approach could have a significant impact in the design of near-sensor NN accelerators.

2.7 Challenges

In this section, we briefly discuss different challenges in the development and application of the proposed time-based computing.

2.7.1 Analog Noise

Recent work has shown that by properly structuring digital bitstreams, completely deterministic computation can be performed with stochastic logic [47, 48]. The results are completely accurate with no random fluctuations. Due to the mixed-signal nature of time-based processing, computations on time-encoded signals are susceptible to noise; one cannot promise 100 percent accuracy. Analog noise cannot be completely eliminated from signals and therefore from computation. By careful design of ATC and TAC, and by choosing appropriate frequencies, however, the error can be made very low (less than 0.001 percent mean absolute error).

2.7.2 Resolution

The resolution in time-based processing is limited by noise, rather than by the length of bitstreams, as it is with SC. While there is no limit in the resolution of SNs represented by digital bitstreams, the resolution in our time-encoded approach is limited by the maximum ENOB of the ATC (that is, the PWM generator). For a minimum frequency of 10 MHz, current ATCs can achieve a maximum ENOB of 11 to 12 bits.

2.7.3 Truncation

With time-encoded signals, operations should run for a specific amount of time to produce correct results. For operations with independent inputs, this time equals the product of the period of the input signals; for operations with correlated inputs, it equals the period of the input signals. As we discussed in Section 2.5, running the operation for longer or shorter than the required time results in truncation error. In contrast, stochastic bitstreams have the property of progressive precision, meaning that short subsequences of an SN can provide low-precision estimates of its value [1]. The longer the stream runs, the more precise the value. Given enough time, the output converges to the expected correct value, and consequently, the truncation error is generally low.

2.7.4 Synchronization

Operations using synchronized PWM signals are limited to only the first level of logic in a circuit. Providing the required synchronization- that is, having maximal overlap between the high part of the input signals-is difficult to achieve for the second and higher logic levels.

A naive solution is to convert the output of each level back to an analog format, then perform an analog-to-time conversion and feed this to a higher level. However, this naive method decreases the accuracy and is costly in terms of latency, area, and energy.

2.7.5 Skew

The synchronization must be perfect in operations that require synchronized inputs. On-chip variations or noise sources affecting clock generators can result in deviations from the expected period, phase shift, or slew rate of the signals. Different delays for AND and OR gates, for example, can be a source of significant skew in implementing sorting-based circuits. The skew in each stage is propagated to the next, resulting in a considerable skew error for large circuits. Mitigating the skew by delaying some signals is complex and costly, and may offset gains in area and power.

2.7.6 Rotation

Relatively prime stream length method of Section 2.3.1, and the clock division and rotation methods explored in [47] are three methods for processing bitstreams deterministically. Choosing inharmonic frequencies for the time-encoded signals corresponds to the "relatively prime" method. A high-frequency time-encoded PWM signal was connected to the select input of the MUX in Section 2.3.1 for an accurate scaled addition operation. This approach corresponds to the "clock division" method of [47]. In the "rotation" method of [47], digital bitstreams are stalled for one cycle at powers of the stream length, causing each bit of one bitstream to see each bit of the other stream exactly once. Considering the high working frequency of time-based SC, stalling PWM signals for a very short and precise amount of time might not be possible.

2.7.7 Sequential Circuits

Sequential finite-state machine (FSM)-based approaches exist for implementing complex functions with SC [31, 9, 16]. These methods depend on randomness in different ways than combinational methods do. It is not clear how to translate these sequential constructs to deterministic computation on time-based PWM signals.

2.8 Conclusion

With a stochastic representation, computation has a pseudo *analog* character, operating on real-valued signals. This is certainly counterintuitive: why impose an analog view on digital values? Prior work has demonstrated that it is often advantageous to do so, both from the standpoint of the hardware resources required as well as the error tolerance of the computation. Many of the functions that we seek to implement for computational systems such as signal processing are *arithmetic* functions, consisting of operations like addition and multiplication. Complex functions, such as exponentials and trigonometric functions, are generally computed through polynomial approximations, so through multiplications and additions. Operations such as these can be implemented with remarkably simple hardware in the stochastic paradigm.

The cost incurred is to provide randomness. While randomness is never free, pseudo-randomness often suffices. The strategy proposed in this chapter was to provide a form of pseudorandomness through time encoding of signals using pulse width modulation. Such signals can be constructed with very common and inexpensive analog circuit structures. We have demonstrated that all the basic operations discussed in the literature on SC can be implemented on PWM signals.

Prior approaches to stochastic circuit design suffered from high run-time latency and correspondingly high energy use. Although the hardware cost of the core stochastic logic was negligible compared to the hardware cost of the conventional binary design, expensive stochastic number generators made them area and energy inefficient. With the proposed PWM approach, however, the latency, area and energy dissipation are all greatly reduced compared to the prior stochastic approaches. This new time-encoded approach inherits the fault tolerance advantage of stochastic design while working as fast and energy-efficiently as the conventional binary design. Fault tolerance capability, a lower hardware cost and a smaller area-delay product make the proposed PWM approach a better choice than the conventional binary design.

Chapter 3

Low-Cost Sorting Network Circuits

This chapter presents an application of the proposed time-based computing in low-cost and energy-efficient implementation of Sorting Network circuits. We first discuss our motivation and present a brief background on sorting networks. We then use unary-style bitstreams in low-cost deterministic implementation of sorting networks. To mitigate the long latency and so high energy consumption of processing digital bitstreams, we use the time-based encoding method of chapter 2. We validate the idea with two implementations of an important application of sorting: median filtering. This chapter's material has been published in [69] and [70].

3.1 Motivation

Sorting is an important task in applications ranging from data mining to databases [71, 72, 73], to ATM and communication switching [74], [75], to scientific computing [76], to scheduling [77], to artificial intelligence and robotics [78], to image [4], video [79], [80], and signal processing [81]. For applications that require high performance, sorting is often performed in hardware with application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs) [82]. Based on the target applications, hardware sorting units vary greatly in the way that they are configured. The number of inputs can be as low as nine for some image processing applications (e.g., median filtering) or as high as tens of thousands. The data inputs are sometimes binary values, integers, or floating-point numbers ranging from 4- to 256-bit precision.

Hardware cost and power consumption are the dominant concerns with hardware implementations. The total chip area is limited in many applications. As fabrication technologies continue to scale, keeping chip temperatures low is an important goal since leakage current increases exponentially with temperature. Power consumption must be kept as low as possible. Developing low-cost, power-efficient hardware-based solutions to sorting is an important goal.

The usual approach is to wire up a network of compare-and-swap (CAS) units in a configuration called a batcher (or bitonic) network. Such networks can readily be pipelined. The parallel nature of hardware-based solutions allows them to outperform sequential software-based solutions. The hardware cost and the power consumption depend on the number of CAS blocks and the cost of each CAS block.

In this chapter, we propose a novel area- and power efficient approach to sorting networks based on "unary processing." Data are encoded as serial bit streams, with values represented by the fraction of 1's in a stream of 0's and 1's. This is an evolution of prior work on stochastic processing. Our designs inherit the fault tolerance and low-cost design advantages of stochastic processing while producing completely accurate and deterministic results. As with stochastic processing, however, the approach is handicapped in term of latency. A serial representation is exponentially longer than a conventional binary positional representation.

To mitigate the long latency issue of unary processing, we adopt the mixed-signal time-encoding approach of Chapter 2. The approach is different to the work on continuous time mixed-signal designs of [83] and [84] in the sense that instead of converting data to (from) binary format by using costly analog to digital (digital to analog) converters and processing in binary domain, the data is encoded in time using low-cost analog-to-time converters and processed in unary domain. We represent the data with time-encoded pulse signals. Time-encoding the data provides a significant improvement in the latency and energy consumption with only a slight loss in accuracy.

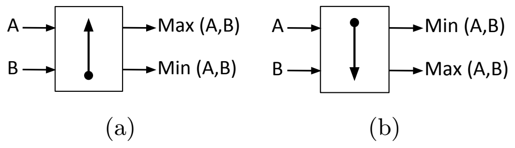


Figure 3.1: The schematic symbol of a CAS block a) ascending b) descending

3.2 Background

3.2.1 Sorting Networks

A sorting network is a combination of CAS blocks that sorts a set of input data. Each CAS block compares two input values and swaps the values at the output, if required. There are two variants: an "ascending" type and a "descending" type. Figure 3.1 shows their schematic symbols. In a conventional design, each CAS block consists of an M-bit comparator and two M-bit multiplexers, where M is the data-width of the inputs.

Sorting networks are fundamentally different from software algorithms for sorting such as QuickSort, MergeSort, BubbleSort, etc., since the order of comparisons is fixed in advance; the order is not data dependent as is the case with software algorithms. The bitonic and odd-even merge sorting networks proposed by Batcher [85] are two popular configurations of sorting networks [86][87]. They have the lowest known latency for hardware-based sorting [3][88].

Bitonic sort uses a key procedure called bitonic merge (BM). Given two equal size sets of input data, sorted in opposing directions, the BM procedure will create a combined set of sorted data. It recursively merges an ascending and a descending set of size N/2 to make a sorted set of size N [89]. Figure 3.2 shows the CAS network for an 8-input bitonic sorting network made up of ascending and descending BM units. The total number of CAS blocks in an N-input bitonic sorting is $N \times log_2(N) \times (log_2(N) + 1)/4$. Thus, 8-input, 16-input, 32-input, and 256-input bitonic sorting networks require 24, 80, 240, and 4,608 CAS blocks, respectively [3].

An odd-even merge sorting network recursively merges two ascending sequences of length N/2 to make a sorted sequence of length N. Odd-even merge sorting units requires fewer CAS blocks than bitonic sorting units, but often have more complex wiring [3]. Due to their simpler structure, we will present designs based on bitonic sort networks.

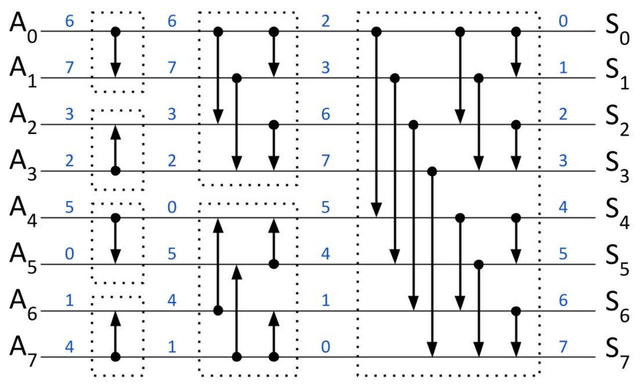


Figure 3.2: The CAS network for an 8-input bitonic sorting [3].

The proposed design approach, however, is applicable to any sorting network topology, including odd-even sorting networks; it will accrue the same advantages.

3.2.2 Unary processing

A recent evolution of the idea of SC has been to perform the processing completely deterministically [47][48][49]. If properly structured, computation on deterministic bitstreams can be performed with same circuits as are used in SC. The results are completely accurate with no random variations; furthermore, the latency is greatly reduced. The idea of unary (or burst) processing was first introduced in 1980s [58] [90] as a hybrid information processing technique that has characteristics common to both conventional binary and to stochastic processing. It is deterministic, but borrows the concept of averaging from stochastic methods. In this chapter, we apply unary processing to problem of desiging low-cost, power-efficient sorting networks.

Unary streams. In unary processing, numbers are encoded uniformly by a sequence of one value (say, 1) followed by a sequence of the other value (say, 0) (See Figure 3.3). This uniform sequence of bits is called a unary stream. To convert binary input data to unary streams, an increasing/decreasing value from an up/down counter is compared to the target value. As with stochastic streams, all the bits have equal weight. This property provides the immunity to noise. Multiple bit flips in a long unary stream produce small and uniform deviations from the nominal value. In stochastic processing, only real-valued

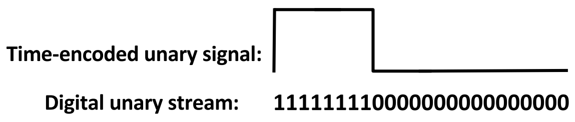


Figure 3.3: Time-based vs. digital-stream unary representation.

numbers can be represented: numbers in the [0, 1] interval with the unipolar format and numbers in the [-1, 1] interval with the bipolar format. In contrast, with unary streams both real-valued and integer numbers can be represented. In representing real-value numbers, the number of ones divided by the length of stream determines the value. In representing integer values, the number of ones directly determines the value. For example, when using unary streams in the real domain, the streams 1000 and 11000000 are both representations of the value 0.25. In the integer domain, on the other hand, these streams represent 1 and 2, respectively. Similar to the bipolar format for stochastic streams, negative numbers can also be represented with unary streams using a simple linear transformation [14].

Unary Operations. The maximum (max) and minimum (min) value functions are two useful functions with simple and low cost unary implementation. In a weighted binary design, data-width-dependent comparator and multiplexer units must be used to implement these functions. In unary processing, individual gates can synthesize these functions: an AND gate gives the minimum of two unary streams when two equal-length unary streams are connected to its inputs; an OR gate gives the maximum value when its inputs are fed with two equal-length unary streams. These gates showed a similar functionality when fed with correlated stochastic bit-streams (See Section 2.3.1).

Figure 3.4 shows an example of finding the minimum and maximum values in unary processing. An important advantage of unary processing is that synthesizing a function is independent of the resolution of data (length of streams). The same core logic is used for processing 128-bit unary streams that is used for processing 256-bit unary streams. While developing a general method for synthesizing all operations with unary processing is still a work in progress, we showed absolute-value subtraction (using an XOR gate), comparison (using a D-type flip-flop), and multiplication (using an AND gate) of unary streams in Section 2.3.1.

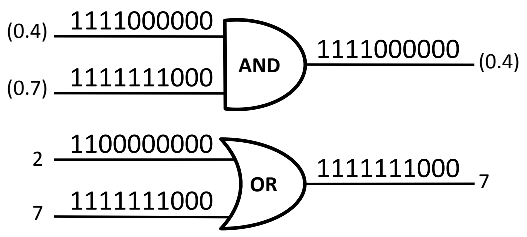


Figure 3.4: Example of performing maximum and minimum operations on unary streams.

Time-based unary streams. The representation of numbers in unary processing is not limited to purely digital bitstreams. A time-based interpretation of numbers is also possible using pulse modulation of data [48]. Figure 3.3 shows both approaches. While both approaches can operate on the same unary logic, the time-based representation offers a seamless solution to the increasing number of time-based sensors and, as we will show, can be exploited in addressing the long latency problem of unary circuits.

3.3 Complete Sort System

In this section, we discuss hardware implementation of complete sort networks. We first discuss the conventional binary design and then present the synthesis approach based on unary processing.

3.3.1 Conventional Design

As discussed in Section 3.2, sorting networks are made of CAS blocks. The hardware cost of a sorting network is therefore a direct function of the number of CAS blocks and the cost of each block. As shown in Figure 3.5a, in a weighted binary design with a data-width of M bits, each CAS block consists of one M-bit comparator and two M-bit multiplexers. Thus, by increasing the resolution of data, the complexity of the design will also be increased. Increasing the complexity of the design directly affects the cost of the hardware implementation, latency, power, and as a result, energy consumption. Another issue with the conventional binary design is noise immunity and fault tolerance. In a noisy environment, faults due to bit flips on high-order bits can produce large errors. Thus, additional fault-tolerance techniques must be used if the goal is to design a noise

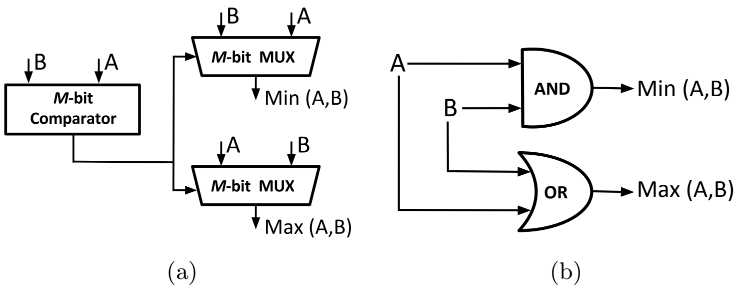


Figure 3.5: Hardware implementation of a CAS block a) Conventional binary design b) Unary design.

tolerant system.

3.3.2 Unary Design

The essential operations in CAS blocks are maximum and minimum functions. This makes unary processing a good fit for hardware implementation of CAS blocks and sorting networks. As shown in Figure 3.5, instead of data-width dependent complex logic, one AND and one OR gate is sufficient to synthesize the CAS block in unary domain. The sorting networks can therefore be synthesized regardless of the resolution of the input data. While the synthesized circuit will be much less costly than the circuit synthesized in the binary approach, additional overhead must be incurred for conversion units which are required to convert the data between the binary and the unary fomart and a longer operation time due to performing the operation on 2^M-bit long streams.

Assuming that the input data is given in binary format and the result must again be in binary, a unary stream generator is required to convert the data from binary to unary and a counter is required to count the number of ones in the final unary stream and convert the result back into binary. Figure 3.6 shows the design of a unary stream generator responsible for converting the data from binary to unary. For each input data, one unary stream generator and, for each output, one counter is required. A significant cost saving in implementing the CAS blocks, particularly for large-scale sorting circuits, will compensate for the overhead of converters in unary designs. Note that while the converters are data-width dependent, the CAS blocks synthesized with the unary approach are independent of data resolution.

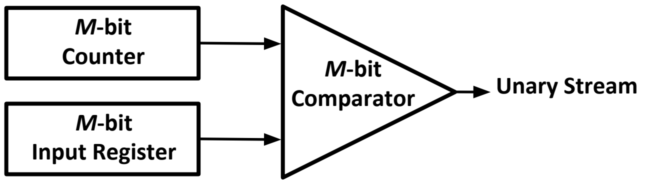


Figure 3.6: Unary stream generator.

3.3.3 Design Evaluation

In order to evaluate the costs and benefits of the proposed design approach, we developed Verilog hardware descriptions of complete bitonic sorting networks for 8, 16, 32, 64, 128, and 256 data inputs, for both the conventional binary and for the proposed unary approach. For the unary approach, the architectures include the required conversion units from/to binary. The developed designs are synthesized using the Synopsys Design Compiler vH2013.12 and a 45-nm standard-cell library. We report synthesis results for three different data widths of 8, 16, and 32 bits. In order to find the minimum hardware cost and also the maximum speed of the developed architectures we synthesized a non-pipelined and also a pipelined version of each architecture.

Non-Pipelined Design

Table 3.1 shows the synthesis results for the non-pipelined implementations. As can be seen, the unary approach could save the hardware cost of the implemented sort networks up to 91%. For small networks like the 8-input sort networks, the cost overhead of unary stream generators and output converters was comparable to the saving due to using a low-cost CAS implementation and so lower savings are achieved. By increasing the number of inputs and so the number of CAS blocks, the savings dominate the overheads and a hardware area saving of around 91% is achieved when implementing the 256-input sorting network with the unary approach.

The total (dynamic plus static) power consumption of the synthesized designs at the maximum feasible working frequency of each architecture, and also at a constant working frequency of 50 MHz, are presented in Table 3.1. The static power or leakage is the dominant power when the system operates at low frequencies. It is directly proportional to the hardware cost and so a sort network with a lower hardware cost will have a lower

Table 3.1: Synthesis results of complete bitonic sort networks (Non-Pipelined).

# of inputs and outputs	# of CAS units	Data width	Area (μm^2) Conven.	Area (μm^2) Unary	Critical Path (ns) Conven.	Critical Path (ns) Unary	Power (@max f) (mW) Conven.	Power (@max f) (mW) Unary	Power (@50MHz) (mW) Conven.	Power (@50MHz) (mW) Unary
8	24	8-bit	3,086	2,194	1.85	0.74	1.30	3.26	0.12	0.13
		16-bit	6,865	4,531	2.05	0.75	2.63	5.59	0.27	0.23
		32-bit	14,868	9,456	2.41	0.77	4.90	10.1	0.62	0.44
16	80	8-bit	10,534	4,511	2.73	0.87	3.66	5.30	0.49	0.25
		16-bit	22,920	8,901	3.42	0.89	6.61	8.94	1.17	0.44
		32-bit	49,812	17,274	3.80	0.93	13.4	15.9	2.63	0.83
32	240	8-bit	32,508	9,235	4.06	1.07	8.86	8.40	1.75	0.49
		16-bit	68,621	17,643	5.05	1.13	16.6	13.8	4.18	0.86
		32-bit	149,669	27,811	5.90	1.12	31.8	25.4	11.3	1.52
64	672	8-bit	90,691	19,028	5.71	1.33	19.8	13.4	5.48	0.96
		16-bit	191,174	29,259	7.03	1.35	39.2	22.5	13.6	1.60
		32-bit	431,182	56,598	8.00	1.37	78.5	41.2	33.1	3.03
128	1,792	8-bit	242,049	33,916	7.49	1.62	44.4	21.4	15.7	1.80
		16-bit	523,565	60,686	9.27	1.63	89.8	37.1	41.1	3.19
		32-bit	1,047,646	115,835	10.14	1.63	165.7	69.1	85.4	6.05
256	4,608	8-bit	586,456	74,719	9.71	1.91	88.7	36.5	42.2	3.64
		16-bit	1,239,154	126,804	11.79	1.94	181.3	62.1	102	6.40
		32-bit	2,560,803	234,957	12.89	1.97	367.7	113	221	12.0

leakage power. When a system works at its maximum frequency, dynamic power, which is an increasing function of the working frequency, is the dominant one. Thus, although the unary designs would have a much lower power consumption at low speeds, due to a lower critical path (CP) latency and so a higher maximum working frequency, the power numbers reported for unary implementation of the 8- and 16-input sorting networks are greater than the power numbers reported for their corresponding binary implementations. As given in Table 3.1, for larger sorting networks (32-input and above), the simplicity of the unary design has led to even a lower power consumption at the maximum working frequency than the power consumption of the binary implementation.

Due to a simpler architecture, the CP latency of the designs synthesized with the unary approach is lower than that of the conventional binary designs. However, the total latency of the unary approach which is the product of the CP latency and 2^M (the number of clock cycles the system must operates to generate and process the unary stream), is much more than the latency of the conventional design (one clock cycle $\times$ CP

Table 3.2: Synthesis results of complete bitonic sort networks (Pipelined).

# of inputs	CAS	Pipeline	Data	Area (μm^2)		Critical Path (ns)		Power (@max freq) (@50MHz) (mW)			
and outputs	units	Stages	width	Conven.	Unary	Conven.	Unary	Conven.	Unary	Conven.	Unary
			8-bit	6,926	2,659	0.42	0.39	19.5	8.1	0.46	0.16
8	24	6	16-bit	14,383	5,024	0.49	0.42	35.6	11.6	0.97	0.27
			32-bit	25,066	9,916	0.53	0.49	66.5	17.2	1.88	0.48
			8-bit	19,338	5,834	0.42	0.40	67.9	17.0	1.50	0.37
16	80	10	16-bit	39,554	10,323	0.48	0.44	126	23.3	3.17	0.56
			32-bit	83,102	18,065	0.52	0.50	241	33.9	6.64	0.94
			8-bit	57,900	13,095	0.42	0.41	213	38.5	4.68	0.86
32	240	15	16-bit	118,202	17,029	0.50	0.46	381	48.2	9.95	1.16
			32-bit	248,129	29,682	0.53	0.50	748	70.0	21.0	1.88
			8-bit	161,934	25,248	0.42	0.44	602	83.0	13.3	1.92
64	672	21	16-bit	329,787	37,726	0.50	0.47	1105	104	28.7	2.61
			32-bit	718,216	63,144	0.52	0.50	2201	149	61.9	4.02
			8-bit	431,062	59,579	0.42	0.47	1625	182	36.0	4.53
128	1,792	28	16-bit	901,206	84,646	0.49	0.50	3070	221	78.8	5.90
			32-bit	1,834,850	134,746	0.52	0.52	5990	310	167	8.70
			8-bit	1,107,998	140,006	0.42	0.49	4228	407	93.0	10.6
256	4,608	36	16-bit	2,294,989	189,903	0.49	0.51	7859	489	204	13.3
			32-bit	4,714,805	289,723	0.54	0.54	15024	648	437	18.9

latency). Although the longer latency of the unary approach is still acceptable for many applications, a more important issue is the energy consumption. Energy consumption is evaluated by the product of the processing time and the total power consumption. Although the unary implementations of the sorting networks have often shown a lower power consumption for a fixed frequency, a very long processing time would lead to a higher energy consumption than their conventional binary counterparts. We will address the long latency and high energy consumption problem of unary designs in Section 3.4.

Pipelined Design

Table 3.2 shows the synthesis results for a fully pipelined structure (only one CAS block between pipeline registers) of the developed designs. Although due to using a large number of pipeline registers, the fully pipelined structure is significantly more costly than the non-pipelined structure, a higher working frequency is achieved with the pipelined one. Designing the sorting network with only one CAS block between pipeline registers leads to a higher latency and total area than the case with more number of CAS blocks

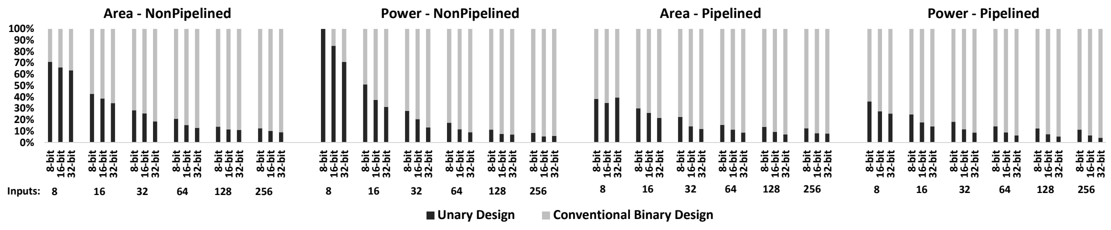


Figure 3.7: Normalized area and power (@50MHz) cost numbers reported for the non-pipelined and pipelined structures of the implemented complete sort networks.

between pipeline registers. However, the one CAS block approach (fully pipelined) results in a higher sorting throughput [3]. Thus, choosing the number of CAS blocks between pipeline registers is a trade-off between the total area and latency, and the throughput, and is a design decision.

As can be seen in Table 3.2, the hardware area cost of the pipelined unary designs are 61%-92% lower than the hardware cost of the pipelined binary designs. Observing a high saving in the area of the small-scale sorting circuits, such as the 8-input sorting network (61% for 8-bit data), is due to using simpler pipeline registers (1-bit instead of M-bit) in the pipelined unary design compared to the pipelined binary design. Figure 3.7 shows normalized diagrams for area and power cost numbers of the synthesized architectures. In each configuration, the results are normalized to the value of the conventional design with that configuration.

Critical path latency of the unary design in the pipelined structure of small sorting networks was slightly lower than that of the binary designs. The reason was a simpler CAS block between the pipeline registers in the unary approach. For large networks (e.g. 128-input, 256-input), however, the CP latency of binary design was lower than the unary implementation. Although in these designs still the CAS blocks of the unary approach are simpler, a more complex unary stream generator and a larger output counter limit the performance of the circuit and increase the CP. The total processing time of the pipelined binary design is the product of the CP latency and the number of pipeline stages. The throughput, however, is higher than the non-pipelined binary design because at each cycle a new set of inputs can enter the system and a set of sorted numbers is leaving the system. For pipelined unary designs, the total latency is the CP latency $\times$ number of pipeline stages $\times 2^M$, where M is the data-width. Thus, similar to the non-pipelined structure, the total latency of the pipelined unary implementations is much higher than the total latency of their conventional binary counterparts. This long latency, further, makes the total energy consumption higher than the energy consumption of the binary designs. We will address this issue in the next section by time-encoding of data using a mixed-signal design of sorting network-based median filtering.

3.4 Highly Efficient Median Filters

A median filter is a popular non-linear filter widely used in image, speech, and signal processing applications. It replaces each input data with the median of all the data in a local neighborhood. This results in filtering out impulse noise and smoothing of the image while preserving important properties such as the edge information [91]. In real-time image and video applications, the digital image data are affected by noise resulting from image sensors or transmission of images. A hardware implementation of the median filter is, therefore, required for denoising. The high computational complexity of median filters, however, makes their hardware implementation expensive and inefficient for many applications. In this section, we first propose a low-cost implementation of median filters similar to the unary sorting networks introduced in Section 3.3. We then exploit the time-based representation of input data using pulse-width modulation to address the long latency problem of the implemented circuits.

3.4.1 Circuit Design

There are a variety of methods for hardware implementation of median filters [92, 93]. Sorting network-based architectures [94] consisting of a network of CAS blocks are one of the most common approaches. The incoming data is sorted as it passes the network. The middle element of the sorted data is the median. As the sorting network can be easily pipelined, the approach provides the best performance [91]. The local neighborhood in median filtering is often a 3×3 or 5×5 window with the target input data at the center. Figures 3.8 and 3.9 show the sorting networks for a 3×3 and a 5×5 median filter, respectively. We developed a non-pipelined and a pipelined structure of these median filters with both the conventional binary and the proposed unary design approach with

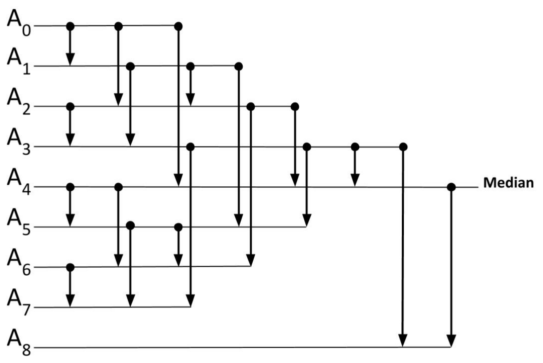


Figure 3.8: The CAS network for a 3x3 Median Filter made of 19 CAS blocks [4].

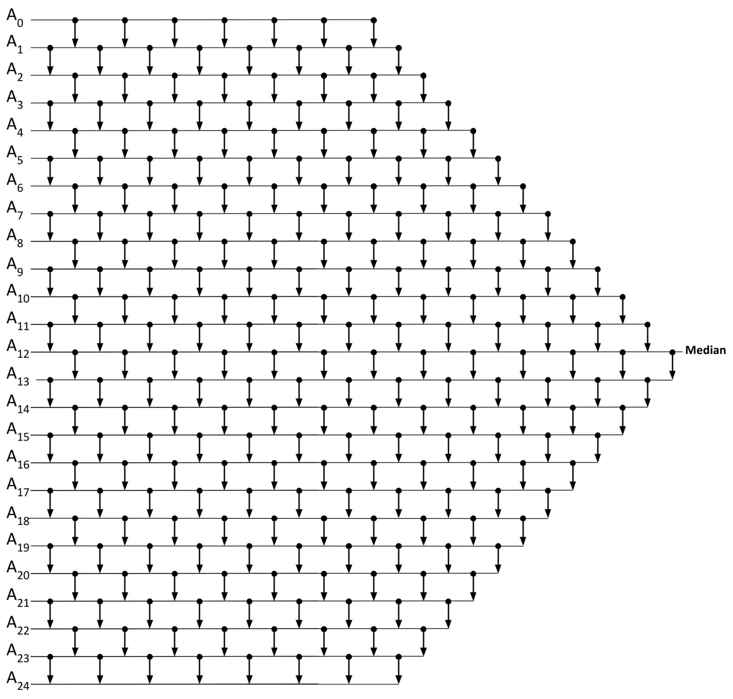


Figure 3.9: The CAS network for a 5x5 Median Filter made of 246 CAS blocks [5].

8-bit input data resolution. The CAS blocks presented in Figure 3.5 were used in the developed architectures. A separate unary stream generator was used for converting each input data and a counter was used for converting the output median stream back to binary form in the unary designs.

Table 3.3 shows the synthesis results for the developed architectures. For now, let

Table 3.3: Synthesis results of the sorting network-based median filters for data-width=8.

Median Filter	Design Approach	Area (μm^2)			Latency (ns)		Power (mW)	Energy (pJ)
		CAS Logic	Overhead	Total	CP	Total	(@max freq)	
3x3	Binary-NonPipelined	2,167	-	2,167	2.10	2.10	1.03	2.1
	Binary-Pipelined (8-stage)	2,167	3,384	5,551	**0.43**	3.44	15.56	6.6
	Unary-NonPipelined	79	917	**996**	0.70	179.2	0.95	170.2
	Unary-Pipelined (8-stage)	79	1,292	1,371	0.40	102.4	3.08	315.3
	Unary-Time-based	79	776	**855**	0.39	**0.39**	1.78	**0.69**
5x5	Binary-NonPipelined	32,772	-	32,772	6.77	6.77	5.76	38.9
	Binary-Pipelined (26-stage)	32,772	28,208	60,980	**0.43**	11.18	219	94.1
	Unary-NonPipelined	1,051	1,988	**3,039**	1.07	273.9	0.93	254.7
	Unary-Pipelined (26-stage)	1,051	6,377	7,428	0.40	102.4	19.68	2015.2
	Unary-Time-based	1,051	1,960	**3,011**	0.78	**0.78**	2.71	**2.11**

us ignore the rows representing Unary-Time-based designs, they will be discussed in Section 3.4.2. The overhead in pipelined designs includes pipeline registers and for unary designs include the required converters from/to binary. Similar to the results reported for the complete sort networks, the unary implementation of the median filters significantly improves the hardware cost, up to 90% for the 5×5 median filter architecture. The pipelined implementations have a higher working frequency and a higher throughput. Comparing the power consumption of the pipelined implementations show that, for the same working frequency, the unary designs have a significantly lower power consumption. For applications in which hardware cost and power consumption are the main priorities, the proposed unary designs outperform the conventional weighted binary designs. However, for high-performance low-energy applications the binary design can be a better choice. In the following section, we exploit the concept of near sensor processing and time-based representation of data to improve the latency and energy consumption of the unary-based median filtering designs at the cost of a slight accuracy loss.

3.4.2 Time-based unary design

1) Overview

Image sensors convert the light intensity to an analog voltage/current. The conventional approach for processing these sensed data is to first convert the analog data to digital

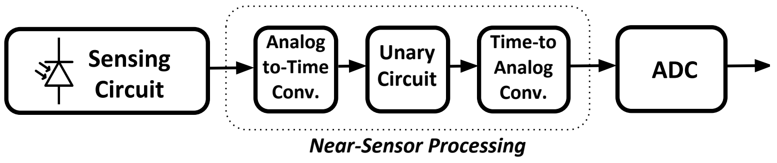


Figure 3.10: Near Sensor Processing with unary circuits.

binary form using a conventional analog-to-digital (ADC) and then process the binary data using digital logic. In unary processing, this binary data is first converted to a unary bitstream and then processed using unary circuits. Processing of image pixels with 8-bit resolution requires running the unary circuit for 256 cycles. Even with a higher working frequency, due to a large number of clock cycles running the circuit, the total latency of the processing using unary circuit is more than that of processing with the binary design.

Near sensor image processing (NSIP) [95] is an interesting concept that suggests integrating some of the processing circuits (i.e., median filter circuit) with the sensing circuit. This can potentially improve the power consumption, size, and costs of vision chips. With more and more sensors providing time-encoded outputs and ways to convert signals from voltage or current to time signals [96], the sensed data in the form of time-encoded signals can directly be fed to unary circuits. Inspired from the NSIP concept and based on the idea of time-encoding data introduced in Chapter 2, we time-encode the sensed input data to address the long latency of processing using unary circuits. Figure 3.10 depicts a simple flow of the method. Assuming that the output of the sensing circuit is in voltage or current form, an analog-to-time converter (ATC) (i.e., low-cost circuit shown in Figure 2.4) is used to convert the sensed data to a time-encoded pulse signal. The converted signal is processed using the unary circuit and the output is converted back to a desired analog format using a time-to-analog converter (TAC) (i.e., a voltage integrator).

2) Evaluation

Table 3.3 shows the area, latency, power, and energy consumption of the implemented median filtering circuits synthesized with the conventional binary, digital bit-stream based unary, and the proposed time-based unary approach. The low-cost pulse-width modulator shown in Figure 2.4 was used as the ATC and a Gm-C active integrator [97] was used as the TAC to convert the output signal back to analog form in the time-based unary designs. While a pulse-width modulator generates a periodic signal with a specific duty cycle and frequency, as we discussed in Section 2.3.1, only one period of the generated signal will be sufficient for processing the data using the implemented unary designs. The duty cycle of the generated signal is determined by the DC level of the sensed data. The hardware cost and the energy consumption of the implemented ATC and TAC are a function of the target working frequency and reported as the overhead of the time-based unary design in Table 3.3.

A separate ATC is used for time-encoding each input data (9 ATCs for 3×3 median filter circuit). For each time-based unary design, the reported overhead numbers are for a working frequency equal to the inverse of the critical CP of the circuit. Assuming that the clock signal that drives the ATC is available in the system, a lower working frequency translate to a lower area and energy overhead. As can be seen in Table 3.3, the total area of the time-based designs including the overhead of ATCs and TAC is lower than the area cost of the digital bit stream-based non-pipelined version of the unary design. The total latency and the energy consumption of the time-based unary designs are better than those of the pipelined and non-pipelined structure of the unary design and also lower than those of the binary designs. A lower CP latency in the time-based unary designs in comparison to the non-pipelined unary design is due to not using unary stream generator and counter in the time-based approach.

The down-side of the time-based unary design, however, is a slight accuracy loss. The working frequency of the ATC affects the effective number of bits in representing and processing data, hence the accuracy of computation. To evaluate the performance of the median filtering unary designs when working with time-encoded input signals, we developed SPICE netlists of both 3×3 and 5×5 median filtering circuits and simulated their operation on a 128×128 noisy soldier image. The sample input image is shown in Figure 3.11. Simulations were carried out using a 45-nm standard cell library in HSPICE. Table 3.4 shows the average output error rates for the images produced using the time-based unary designs. Image pixel intensities were converted to pulse signals using the ATC shown in Figure 2.4 and also using the HSPICE built-in pulse generator (an ideal ATC). In Table 3.4, these two methods correspond to the rows “ATC of this work"and “Ideal ATC", respectively. Comparing the output images with the expected

Table 3.4: Average error rate of processing the sample image using the time-based unary circuits.

Median Filter Time-based Unary		Length of input signals (1/freq.)			
		CP	1ns	2ns	5ns
3x3	Ideal ATC	2.09%	0.84%	0.45%	0.19%
	ATC of this work	2.65%	1.05%	0.56%	0.21%
5x5	Ideal ATC	4.70%	3.33%	1.83%	0.94%
	ATC of this work	4.86%	3.66%	1.90%	1.01%

output image (produced using a software-based implementation of the algorithm in MATLAB), the mean of the output error rates was calculated as follows:

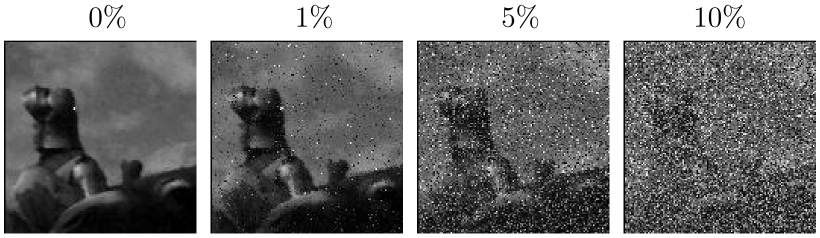
$$AverageErrorRate = \frac{\sum_{i=1}^{W}\sum_{j=1}^{H}|P_{i,j} - E_{i,j}|}{255.(W \times H)} \times 100$$

where $E_{i,j}$ is the expected value for location (i, j) in the output image, $P_{i,j}$ is the pixel value for the same location produced using the circuit, and W and H are the dimensions of the image. As can be seen in Table 3.4, increasing the length of the input signal (a lower working frequency) leads to a higher accuracy in the time-based approach. An average error rate of less than 1% is achieved in the 3×3 median filtering circuit with 1 ns and in the 5×5 circuit with 5 ns processing time. The inherent inaccuracy in converting the values with the ATC of Figure 2.4 resulted in a slightly higher error rates when comparing to the error rates where using ideal ATC.

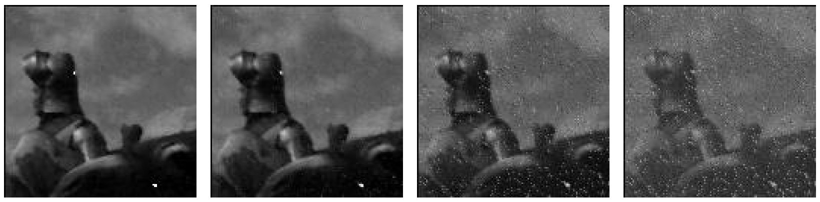
3) Sources of inaccuracy

Error in generating pulse signals (analog value to time conversion), error in measuring the output signal (time to analog conversion), and error due to skew noise are the main sources of errors in the time-based unary processing. A different gate delay for AND and OR gates, particularly, can be a main source of skew in the unary sorting networks. Such a skew is negligible for small sorting networks (e.g., 3×3 median filtering). However, for large sorting networks (e.g., 5×5 median filtering) the skew in each stage is propagated to the next stage, resulting a considerable skew error. With careful gate sizing and adjusting gate delays, or simply increasing the length of the input signals we can mitigate this source of inaccuracy in the time-based unary design.

(a) Sample Input Image



(b) Conventional Binary Implementation



(c) Proposed Unary Implementation

Figure 3.11: (a) Sample input image, and comparison of the noise-tolerance capability of (b) the conventional binary vs. (c) the proposed unary implementation for the 3×3 median filtering circuit for different noise injection rates.

3.5 Noise-Tolerant Behavior

To evaluate the noise-tolerance of the proposed unary designs in comparison to that of the corresponding conventional binary implementations, we randomly injected soft errors, i.e., bit flips, for 0%, 1%, 5%, and 10% noise injection rates on the inputs of CAS blocks of the 3×3 median filtering circuits and measured the corresponding average output error rates. A noise injection rate of 10% means that 10% of the total bits in the inputs of CAS blocks are randomly chosen and flipped. The sample image shown in Figure 3.11 was used as the input to the circuits. For the conventional binary implementation the data-width was fixed at 8 bits and bitstreams of length 256 were used to represent values in the unary designs. Figure 3.11 shows the performance of the implemented circuits at various noise injection rates. As can be seen, the proposed unary implementation has shown a higher noise-tolerance compared to the conventional binary implementation. For injection rates higher than 1%, the quality of the output image produced by the binary design degrades drastically leading to a useless image for injection rates higher than 5%. This noise-immunity observed in the unary design is mainly due to its data encoding approach, a common property between the unary and the stochastic processing. Bits are equally weighted in unary streams and so bit flips produce small and uniform deviation from the nominal value.

3.6 Summary

A conventional weighted binary-based implementation of a large sorting networks is costly considering the large number of compare-and-swap (CAS) units that such a network entails. The VLSI cost increases significantly with increasing resolution of the input data. The high hardware cost and the high power consumption of such networks restrict their application. In this chapter, we proposed an area and power efficient implementation of sorting networks based on unary processing. The core processing logic consists of simple gates and is independent of the resolution of data. The only overhead in the approach, the cost of converting data from/to binary, is small. More than 90% area and power savings are observed when compared to the costs of a conventional weighted binary implementation.

The penalty is latency. Processing digital unary streams, requires a relatively long running time, e.g., more than 100 ns to process each set of input data. Although this is a 100$\times$ increase in latency over conventional weighted binary, this increase may be tolerable for many applications. For example, ten gray-scale HD (1280×720) images or four gray-scale Full HD (1920×1080) images can be processed per second with the proposed scheme for a task such as median filtering, when operating on 256-bit long unary streams. In spite of the latency, a 90% decrease in power consumption might often make this a winning proposition.

To mitigate the latency of the approach, we further developed a time-based unary design approach in which the input data is encoded in time and represented with pulse signals. The result is a significant improvement in the latency and energy consumption, at the cost of a slight loss in accuracy. For example, more than 1000 gray-scale HD images or 400 gray-scale Full HD images can be processed per second with the proposed time-based unary implementation of the 3×3 median filtering at the cost of only 1% loss in accuracy.

Chapter 4

Fast-Converging Deterministic Bitstream Computing

In Chapter 2, we showed that by choosing *relatively prime lengths* for unary bitstreams, and repeating the streams up to the least common multiple of the stream lengths, a deterministic and completely accurate output can be produced by stochastic logic. *Clock division* and *rotation* methods have been also proposed in the literature to process unary bitstreams deterministically. In this chapter, we first propose a high-quality down-sampling method to improve the progressive precision and so the processing time and energy consumption of the three current deterministic methods of SC by pseudo-randomizing bitstreams. We then propose two novel deterministic methods of processing bitstreams by using low-discrepancy sequences. The material of this chapter has been taken from [98, 99, 100, 101], and [102].

4.1 Motivation

SC has been around for many years as a noise-tolerant approximate computing approach. Random Fluctuation has always made SC somewhat inaccurate. Due to random fluctuation, stochastic operations often need to run for a very long time to produce highly accurate results. Some operations, such as multiplication, also suffer from correlation between bitstreams. For these operations, the input bitstreams must be independent to produce accurate results. To produce an output with n-bit precision, the input bitstreams length, and so the number of cycles performing the operation, must be greater than 2^{2ni-2}, where i is the number of independent inputs in the circuit [47].

Recent progress in the idea of SC [48][47], however, has revolutionized the paradigm. If properly structured, random fluctuation can be removed and SC circuits can produce deterministic and completely accurate results. In Chapter 2, we showed that by choosing relatively prime lengths for a specific class of stochastic streams–called unary streams, and repeating the streams up to the least common multiple of the stream lengths, a deterministic and completely accurate output can be produced. Jenson and Riedel [47] further proposed two deterministic approaches of processing unary streams, clock division and rotation of bitstreams. The proposed approaches not only are able to produce completely accurate results (i.e., zero percent error rate), but they also improve the hardware cost and the processing time of stochastic operations significantly when compared to those of the computations performed on the conventional randomized stochastic bitstreams.

While the unary stream-based deterministic approaches are able to produce completely accurate results (i.e., results that are the same as the results of binary-radix computation), they suffer from a poor *progressive precision* property; The output converges to the expected correct value very slowly. This drawback can be a major limitation to wide use of these approaches in different applications. While ideally we are interested in producing completely accurate results, decision making on some inputs, particularly in image processing and neural network applications, do not require high precision operation and a low-precision estimate of the output value is sufficient. In such cases, due to the poor progressive precision property of unary streams, stochastic operations must run for a much longer time than the cases with conventional random bitstreams to produce acceptable results with small levels of inaccuracy. When small rates of inaccuracy are acceptable, using the unary stream-based deterministic approaches will lead to a very long operation time and consequently a very high energy consumption.

Fig. 4.1 compares the progressive precision of different stochastic approaches when multiplying 8-bit precision input values. As can be seen in the figure, the conventional random stream-based stochastic approach shows a much better progressive precision than the unary stream-based deterministic approaches and so is the preferable choice for any application that can tolerate some errors, such as image processing and neural network applications.

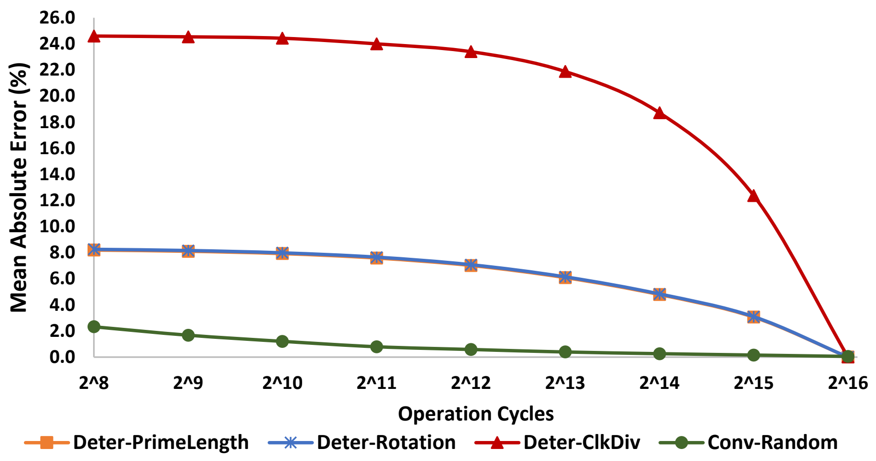
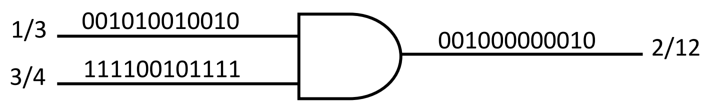


Figure 4.1: Progressive Precision comparison of the conventional random stream-based SC with the unary stream-based deterministic approaches of SC when multiplying two 8-bit precision input values.

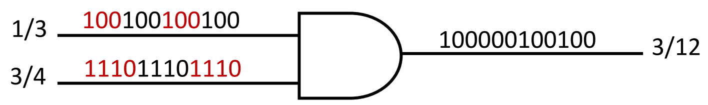
In this chapter, we show that by modifying the structure of the stream generators, the deterministic methods not only are able to produce completely accurate results, they are also able to produce acceptable results in a much shorter time and with a much lower energy consumption compared to the current architectures that generate and process unary streams.

4.2 Deterministic Approaches to Stochastic Computing

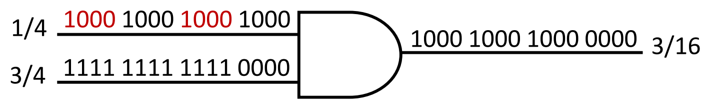
Recent work on SC [48][47][69] has shown that SC does not necessarily have to be an approximate approach and the result of computation can actually be completely accurate and deterministic. Instead of random stochastic bitstreams, logical computation is performed on a specific class of bitstreams, called unary streams. A unary stream consists of a sequence of 1s followed by a sequence 0s. For example, 1111000000 is a unary stream representing 0.4 in the unipolar format. To represent a value with resolution of $1/2^n$ (n-bit precision), the unary stream must be 2^n bit long. For operations that require independent inputs, the independence between the input unary streams is provided by using relatively prime stream lengths (See Chapter 2), rotation, or clock division [47].



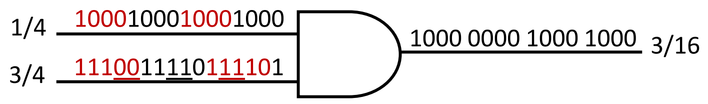
(a) Approximate: Random Streams



(b) Deterministic: Relatively Prime Stream Lengths



(c) Deterministic: Clock Division



(d) Deterministic: Rotation

Figure 4.2: Examples of performing stochastic multiplication: a) conventional approximate SC with random bitstreams (b)-(d) recently proposed deterministic approaches to SC with unary bitstreams.

Fig. 4.2(b)-(d) exemplifies these three deterministic approaches to SC.

To produce accurate results with these deterministic approaches, the operation must run for an exact number of clock cycles which is equal to the product of the length of the input bitstreams. For example, when multiplying two n-bit precision input values represented using two 2^n-bit streams, the operation must run for exactly 2^{2n} cycles [47]. Running the operation for fewer cycles (e.g., 2^{2n-1} cycles) will lead to a poor result with an error out of the acceptable error bound. As we discussed in Section 2.5, this important source of inaccuracy in performing computations on unary streams is called "truncation error".

As an example, assume we want to multiply two 8-bit precision numbers, represented using unary streams, with the rotation or clock division deterministic approaches. The operation must run for exactly $2^{16} = 65536$ cycles to produce a completely accurate result. Exhaustively testing the multiplication operation for every possible pair of input values when running the operation for 2^{15} and 2^{10} cycles shows a mean absolute error (MAE) of 3.10% and 7.99%, respectively, for the rotation approach, and 12.3% and 24.4% for the clock division approach. With the conventional approach of processing random bitstreams when exhaustively testing the operation on a large set of random pairs of input values, although we could not produce completely accurate multiplication results in 2^{16} cycles, a good progressive precision property could lead to acceptable results when running the operation for the same number of operation cycles (MAE of 0.15% after 2^{15} and 1.20% after 2^{10} cycles).

4.3 Pseudo-Random Bitstreams for Deterministic SC

In this section, we propose a high-quality down-sampling method to improve the progressive precision of current deterministic methods of SC. We first describe the proposed method and compare its performance with prior methods for the multiplication operation. We then evaluate the proposed method using the Robert's cross edge detection stochastic circuit as a case study.

4.3.1 Proposed Method

While the randomness inherent in stochastic bitstreams was one of the main sources of inaccuracy in SC, distributing the ones across the stream instead of grouping them (i.e., first all ones and then all zeros) may be able to provide a good progressive precision property for representing stochastic numbers and, therefore, for the computation. With randomized bitstreams the result quality improves as the computation proceeds. This is because short sub-sequences of long random stochastic bitstreams provide low-precision estimates of the streams' values. This property can be exploited in many applications of SC for making quick decisions on the input data and so increasing the processing speed.

Deterministic approaches proposed in Chapter 2 and in [47] perform computation on unary streams. Due to the nature of unary representation, truncating the bitstream leads to a high truncation error and so a significant change in the represented value. Here, we propose a high quality down-sampling method for these approaches by bringing randomization back into the representation of bitstreams. Similar to processing unary streams, the computations are completely accurate when the operations are executed

0 1 0 1 0 1 1 1 0 0 1 0 0 0 0 1 : 7/16

(a) Stochastic Randomized Bit-Stream

1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 : 8/16

(b) Deterministic Unary Bit-Stream

1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 : 8/16

(c) Deterministic Pseudo-Randomized Bit-Stream

Figure 4.3: Different types of stochastic bitstreams.

for the required number of cycles. However, by pseudo-randomizing the streams, the computation will have a good progressive precision property and truncating the output streams by running for fewer clock cycles still produces high quality outputs.

For a deterministic and predictable randomization of the bitstreams, we propose to use *maximal period* pseudo-random sources (i.e., a maximal period LFSR) to generate the bitstreams. The important point is that the period of the pseudo-random source should be equal to the length of the bitstream. By using such a source to generate random numbers, we are able to convert an input value into a pseudo-random but completely accurate stochastic representation. Fig.4.3 illustrates an example of representing 0.5 value with a random, a unary, and our proposed pseudo-randomized bitstream.

Table 4.1 compares the MAEs of the conventional random stream-based SC and the unary stream-based deterministic approaches of Chapter 2 and [47] with the proposed method by exhaustively testing multiplication of two 8-bit precision stochastic streams on a large set of random input values for the conventional random SC and for the proposed approach, and on every possible input value for the unary deterministic approaches. For the conventional random stochastic approach, we evaluate the accuracy with two different structures for converting the input values to randomized stochastic bitstreams: 1) using maximal period 8-bit LFSRs, and 2) using maximal period 16-bit LFSRs to emulate a true-random number generator. Two different LFSRs (i.e., different designs[1] with different seeds) are used in each case to generate independent bitstreams. While the first structure can accurately convert the input values to 256-bit[2] pseudo-random bitstreams,

[1] Two out of 16 different designs of maximal period 8-bit LFSRs and two out of 2,048 different designs of maximal period 16-bit LFSRs described in [103] are randomly selected for each run.

[2] An n-bit maximal period LFSR has a period of $2^n - 1$, as the 0-state in the LFSR is normally

Table 4.1: Mean Absolute Error (%) comparison of the prior random and deterministic approaches to stochastic computing and the proposed deterministic approaches based on pseudo-randomized streams when multiplying two 8-bit precision stochastic streams with different numbers of operation cycles.

Design Approach	SNG	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}
Conventional	Prior work [2, 14]: two LFSR-8	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83	2.54	4.25
Random Stoch.	Prior work [2, 14]: two LFSR-16	0.05	0.15	0.26	0.39	0.58	0.79	1.20	1.67	2.32	3.32	4.72
Deterministic	Method of Ch. 2: two counter-8	0.00	3.03	4.70	6.01	7.08	7.62	7.90	7.98	8.11	33.2	51.5
Prime Length	Proposed method: two LFSR-8	0.00	0.09	0.16	0.24	0.34	0.47	0.60	0.72	0.85	2.56	4.22
Deterministic	Prior work [47]: two counter-8	0.00	12.3	18.7	21.8	23.4	24.0	24.4	24.5	24.9	49.6	62.2
Clock Division	Proposed method: two LFSR-8	0.00	1.44	2.48	3.74	5.28	7.18	9.91	14.2	24.8	25.0	25.8
Deterministic	Prior work [47]: two counter-8	0.00	3.10	4.84	6.15	7.08	7.66	7.99	8.17	8.26	33.1	51.8
Rotation	Proposed method: two LFSR-8	0.00	0.09	0.16	0.24	0.35	0.47	0.60	0.71	0.82	2.56	4.26

the second structure converts the inputs to any stream with a length less than 2^{16} to give an approximate representation of the value. With the first structure, after 256 cycles, the generated bitstreams repeat and so the accuracy of the operation never improves after this time. Due to a more precise representation, the first structure shows a better MAE for low stream lengths. However, for very long bitstream lengths, the second structure can produce a better MAE. The hardware cost of the second structure is twice that of the first one because of using larger LFSRs. Note that due to random fluctuation and correlation, neither of these two structures can produce completely accurate results in 2^{16} cycles.

As shown in Table 4.1, the deterministic approaches proposed in Chapter 2 and [47] are able to produce completely accurate results when running the operation for 2^{16} cycles. Due to using unary bitstreams, however, the MAE of the computation increases significantly when running the operation for fewer cycles. Instead of unary streams, we use pseudo-randomized but accurate bitstreams. Integrating these bitstreams with the deterministic approaches results in completely accurate computation when it is run for the required number of cycles while still producing high quality results if the output stream is truncated.

As discussed, in the deterministic approaches to SC, the required independence

not used. Here, for a fair comparison with the unary stream-based deterministic approaches, we add a 0-state to the set of the states of each LFSR to generate 2^n unique numbers.

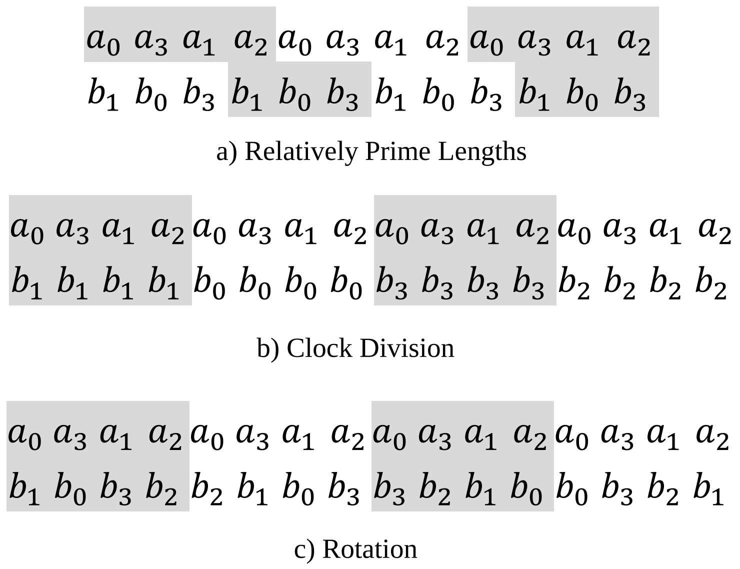

$a_0\ a_3\ a_1\ a_2\ a_0\ a_3\ a_1\ a_2\ a_0\ a_3\ a_1\ a_2$
$b_1\ b_0\ b_3\ b_1\ b_0\ b_3\ b_1\ b_0\ b_3\ b_1\ b_0\ b_3$
a) Relatively Prime Lengths
$a_0\ a_3\ a_1\ a_2\ a_0\ a_3\ a_1\ a_2\ a_0\ a_3\ a_1\ a_2\ a_0\ a_3\ a_1\ a_2$
$b_1\ b_1\ b_1\ b_1\ b_0\ b_0\ b_0\ b_0\ b_3\ b_3\ b_3\ b_3\ b_2\ b_2\ b_2\ b_2$
b) Clock Division
$a_0\ a_3\ a_1\ a_2\ a_0\ a_3\ a_1\ a_2\ a_0\ a_3\ a_1\ a_2\ a_0\ a_3\ a_1\ a_2$
$b_1\ b_0\ b_3\ b_2\ b_2\ b_1\ b_0\ b_3\ b_3\ b_2\ b_1\ b_0\ b_0\ b_3\ b_2\ b_1$
c) Rotation

Figure 4.4: Deterministic approaches to SC by two pseudo-randomized bitstreams.

between input streams is provided by using relatively prime lengths, rotation, or clock division. When running the operations for the product of the length of the streams, these three methods cause *every bit of the first stream to interact with every bit of the second stream* [47, 48]. The computation is therefore performed deterministicly and accurately irrespective of the location of the ones in each stream. Thus, as demonstrated in Fig. 4.4, with the interaction of two pseudo-randomized bitstreams, there is actually no requirement to use unary-style streams and, instead, we use pseudo-randomized bitstreams for the deterministic approaches.

We use different LFSRs (different LFSR designs and different seeds) for generating pseudo-randomized bitstreams. The period of the LFSR should be maximal and equal to the length of the bitstream to accurately represent each value. Thus, for 8-bit precision inputs, an 8-bit size maximal period LFSR is required. Table 4.1 also compares the MAE of the deterministic approaches when multiplying the inputs streams generated using the proposed approach. Similar to the unary stream-based deterministic approaches, the proposed method results in completely accurate results when running the operation for 2^{16} cycles, but produce a much lower MAE when running for fewer cycles. Compared to the conventional random SC, the relatively prime length and the rotation approaches produce results with a lower MAE.

Note that, similar to the unary-stream based deterministic approaches that require n separate counters for generating n independent input bitstreams [47], sharing LFSRs in the proposed method is not possible. In the clock division deterministic approach, each LFSR must be driven with a different clock source which as a result prevents using optimization techniques such as sharing LFSRs+shifting [63] to save hardware cost. Similarly, the limitation of using number sources with different periods in the relatively prime approach and stalling number generators in the rotation approach prevent us from sharing pseudo-random number generators in the proposed method.

4.3.2 Evaluation

To evaluate the proposed idea, we used the stochastic implementation of the Robert's cross edge detection algorithm. Fig. 2.19a shows the stochastic implementation of this algorithm proposed in [1]. The two XOR gates compute absolute value subtraction when they are fed with correlated input streams (streams with maximum overlap between 1s). Sharing the same source of numbers (i.e., same LFSR) for generating the input streams can provide correlated streams. The MUX unit, on the other hand, performs scaled addition irrespective of correlation between its main input streams. The important point, however, is that the select input stream (here, a stream with the value 0.5) should be independent to the main input streams to the MUX. Thus, for the Robert's cross stochastic circuit, the four main input streams (the inputs to the XOR gates) should be correlated to each other, but should be independent of the select input of the MUX. Two number generators are, therefore, required for this circuit– one for converting the main inputs and one for generating the select input stream.

We evaluate the performance, the hardware area, the power, and the energy consumption of the Robert's cross stochastic circuit in three different cases: 1) the conventional approach of processing random streams, 2) the prior deterministic approaches of processing unary streams, and 3) the proposed deterministic approaches of processing pseduo-randomized streams. The circuit shown in Fig. 2.19a is the core stochastic logic and will be shared between all cases.

Fig. 4.5 shows our proposed structures of the sources for generating pseudo-random numbers for the three deterministic approaches. For the relatively prime length approach,

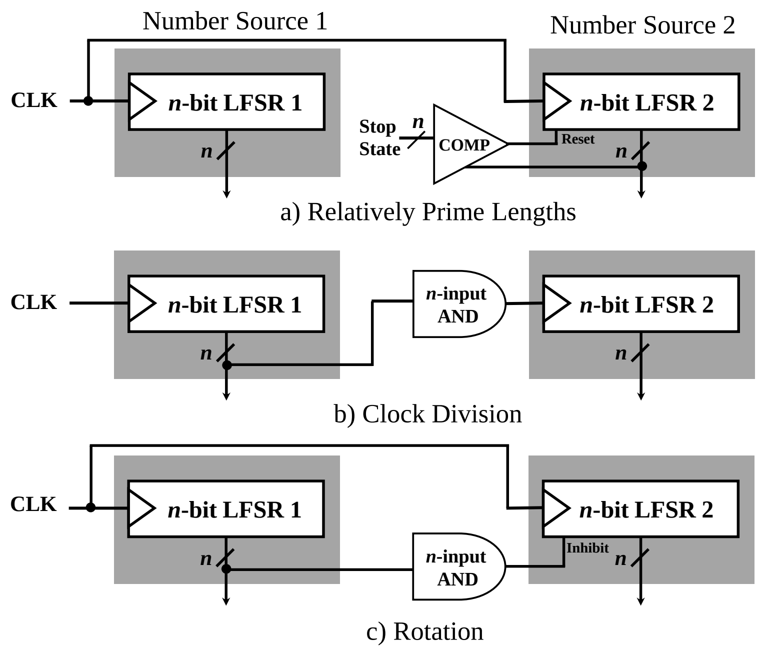


Figure 4.5: Proposed sources of generating pseudo-random numbers for the three deterministic approaches to SC.

we assume the first number source has a period of $2^n - 1$ and we control the period of the second source by setting a stop state. Here, for the Robert's cross circuit, pseudo-random number sources with periods of $2^8 - 1$ and $2^8 - 2$, are implemented. When the state (the output number) of LFSR 2 equals the stop state, LFSR 2 is restarted to its initial state. For the clock division structure, LFSR 2 is clock divided by the period of LFSR 1 through detecting the all one state using an AND gate. Similarly, the rotation structure uses an AND gate to inhibit or stall every $2^n - 1$ cycles when the all one state is detected.

These units are used as the number source in the stochastic stream generator shown in Fig. 1.1. For the unary stream-based prior deterministic approaches, we optimized and implemented the counter-based architectures of [47]. For the conventional random stream-based implementations, we used two different 8-bit or two different 16-bit LFSRs as the required sources of random numbers. We used the Synopsys Design Compiler vH2013.12 with a 45nm gate library to synthesize the designs.

As shown in Table 4.2, the hardware area cost of the proposed deterministic designs

Table 4.2: Area (um^2), Power (mW) (@ 1GHz), and Energy consumption (pJ) of the Robert's cross stochastic circuit synthesized with the 8- and 16-bit conventional random approach, and also the prior structures and the proposed structures of the three deterministic approaches (Relatively Prime Lengths, Clock Division, and Rotation).

			Target Error (MAE)													
Design	Area	Power	0%		0.1%		0.3%		0.5%		1.0%		2.0%		3.0%	
Approach	(μm^2)	(mW)	Cycle	Energy	Cycle	Energy	Cycle	Energy	Cycle	Energy	Cycle	Energy	Cycle	Energy	Cycle	Energy
Conv-Random-8	671	0.818	-	-	-	-	-	-	-	-	-	-	170	**139**	100	**82**
Conv-Random-16	1415	1.633	2^16>	10^5>	54410	88,852	11370	18,567	4380	7,153	1220	1,992	300	490	130	212
PrimeL-Prior	689	0.560	64770	**36,271**	64210	35,958	63150	35,364	62070	34,759	59510	33,326	54420	30,475	49080	27,485
PrimeL-Proposed	746	0.848	64770	54,925	25100	**21,285**	4000	**3,392**	1500	**1,272**	470	**399**	170	**144**	100	**85**
CLKDIV-Prior	665	0.304	65025	**19,768**	64830	**19,708**	63850	19,410	62900	19,122	60740	18,465	56740	17,249	53250	16,188
CLKDIV-Proposed	688	0.527	65025	34,268	63730	33,586	54310	28,621	42640	22,471	20780	10,951	6500	3,426	2980	1,570
Rotation-Prior	667	0.493	65025	**32,057**	64665	31,880	63445	31,278	62405	30,766	59855	29,509	54875	27,053	49575	24,440
Rotation-Proposed	725	0.732	65025	47,598	29305	**21,451**	4045	**2,961**	1495	**1,094**	465	**340**	170	**124**	100	**73**

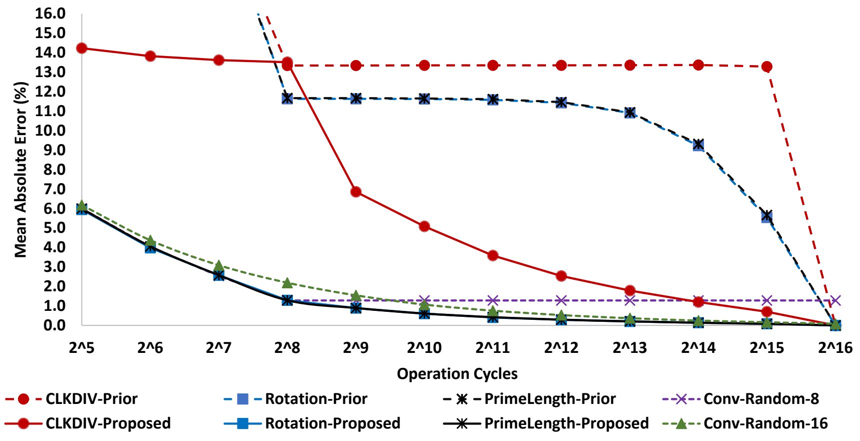


Figure 4.6: Mean Absolute Error (%) when processing random input values with the Robert's cross stochastic circuit using different stochastic approaches.

is slightly (<10%) more than that of their corresponding prior deterministic implementations. Due to replacing counters with LFSRs in the proposed architectures, the power consumption has also increased in all cases. The important metric, however, in evaluating the efficiency of the implemented designs is energy consumption, defined as the product of the power consumption and processing time.

We evaluate the energy-efficiency of the different designs by measuring the energy consumption of each one in achieving a specific accuracy in processing the inputs. MAE is used as the accuracy metric (a lower MAE means a higher accuracy). To comprehensively test the designs, we simulate the operation of the Robert's cross circuit in each design approach by processing 10,000 sets of 8-bit precision random input values. For accurate representation of input values in each design approach, we randomly choose an integer value between zero and the period of the (pseudo-random) number generator and divide it by the period. Fig. 4.6 and Fig. 4.7 present the MAE and the standard deviation of processing random input values in different design approaches. Table 4.2 further shows the number of processing cycles and the energy consumption of each design to achieve different accuracies.

When completely accurate results are expected, the proposed designs must run for the same number of cycles as required by the prior deterministic designs (product of the periods of the number generators). Considering the higher power consumption of the

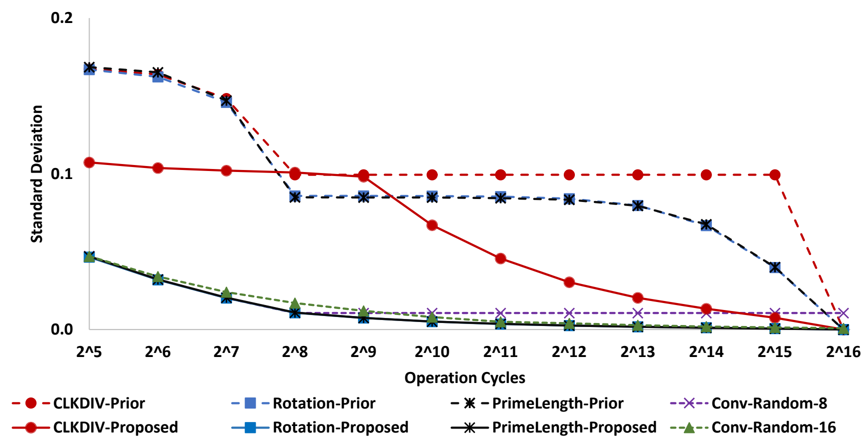


Figure 4.7: Standard deviation of the absolute error of processing random input values with the Robert's cross stochastic circuit using different stochastic approaches.

proposed designs, the prior deterministic implementations consume less energy to achieve completely accurate results. The great advantage of the proposed architectures starts when slight inaccuracy in the computation is acceptable. In such cases, the proposed designs start showing a much lower energy consumption by converging to the expected accuracy in a much shorter time.

For the relatively prime and the rotation approaches, the proposed designs improve the processing time by 61% and 55%, respectively, resulting in an energy consumption savings of 41% and 33% when accepting an MAE as low as 0.1%. For an MAE of 3.0%, these architectures consume 324 and 334 times lower energy by improving the processing time up to 500X compared to prior unary-based architectures. For the clock division approach, the proposed design is more energy efficient if at least an MAE of 1.0% is acceptable. The energy consumption is reduced 10 times for this method for an MAE of 3%.

Compared to the conventional random stream-based architectures (Conv-Random-8 with 8-bit LFSRs and Conv-Random-16 with 16-bit LFSRs) the proposed structures are more energy-efficient than the 16-bit conventional architecture but are at the same level with the 8-bit implementation. The important point, however, is that the 8-bit conventional architecture cannot achieve an MAE of 1.0% or lower and the 16-bit architecture requires a very long processing time and consumes significant energy to get close to completely accurate results.

4.4 Low-Discrepancy Bistreams for Deterministic SC

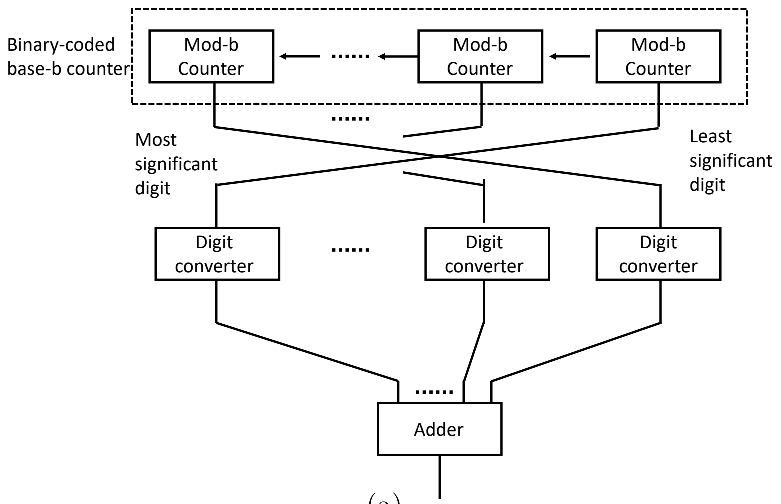
In Section 4.3, we used pseudo-random sequences to improve the poor progressive precision of the current deterministic methods of SC. In this section, we propose two new deterministic methods for computation with stochastic bitstreams using low-discrepancy sequences. We first describe the proposed methods and their hardware structures, and then evaluate the accuracy and hardware costs compared to prior state-of-the-art work.

4.4.1 Low-Discrepancy Sequences in SC

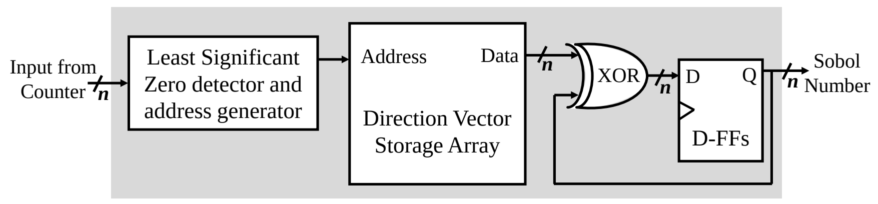
Low discrepancy (LD) sequences were traditionally used to accelerate the convergence in Monte-Carlo simulations [8]. Recent work on SC [6][7] utilized these sequences in improving the speed of computation on stochastic bitstreams. With LD sequences, 1âĂŹs and 0âĂŹs in the stochastic streams are uniformly spaced, the streams do not suffer from random fluctuations. The bitstreams can quickly and monotonically converge to the target value, producing acceptable results in a much shorter time [6].

Alaghi and Hayes proposed the use of LD Halton sequences for SC [6]. A Halton sequence generator consists of a binary-coded base-b counter, where b is a prime number. For d independent input streams in a SC system, d counters with different prime bases must be used. For instance, in the simplest case of multiplying two stochastic bitstreams using an AND gate, one base-2 and one base-3 counter is required. The order of the counter's output digits are reversed and the reordered digits are converted to equivalent binary numbers. The structure of the Halton sequence generator proposed in [6] is shown in Figure 4.8.a. Stochastic bitstreams generated using Halton-based sequences can significantly improve the processing time of SC for achieving the same accuracy compared to the conventional LFSR-based pseduo-random bitstreams. However, the base conversion required in the structure of this sequence generator results in additional hardware overhead [7].

Liu and Han [7] recently proposed another LD-based stochastic stream generator based on Sobol sequences. Compared to Halton sequence generator, generation of Sobol sequences does not need the additional base-conversion hardware. The Sobol sequence



(a)


Input from Counter
n
Least Significant Zero detector and address generator
Address
Data
Direction Vector Storage Array
n
XOR
n
D
Q
D-FFs
n
Sobol Number

(b)

Figure 4.8: (a) Halton sequence generator [6] (b) Sobol sequence generator [7, 8].

generator, instead, consists of an address generator that detects the position of the least significant zero, a storage array storing the values of the direction vectors as intermediate variables for sequence generation, and a pair of XOR gate and D-type flip-flop for recursively generating random numbers. The structure of the Sobol sequence generator, shown in Figure 4.8.b, is proposed in [8] and used in [7] for generating LD stochastic bitstreams. Different Sobol sequences can be generated by changing the values of the direction vectors.

The authors in [7] showed that the Halton sequence based stochastic multiplier takes about twice the sequence length to achieve a similar accuracy as the Sobol sequence-based design. Thus, both approaches consume almost the same energy for the same accuracy requirement. Due to the limitation of the Halton sequences to prime bases, in this work, we focus on the Sobol sequences which can cover different precisions of the base-2

Table 4.3: Comparing different sources of generating numbers (3-bit precision) for stochastic stream generator

Counter	0	1	2	3	4	5	6	7
	0	1/8	1/4	3/8	1/2	5/8	3/4	7/8
LFSR	0	3	7	1	2	6	4	5
	0	3/8	7/8	1/8	1/4	3/4	1/2	5/8
Sobol Generator	0	4	2	6	1	5	3	7
	0	1/2	1/4	3/4	1/8	5/8	3/8	7/8

numbers.

Table 4.3 compares three different sources of generating numbers for the stochastic stream generator of Fig. 1.1. A counter is being used to generate unary streams. An LFSR is being used to generate pseudo-random bitstreams. A Sobol sequence generator, on the other hand, is used to generate a LD-based stochastic stream.

4.4.2 First Proposed Method

The first method uses the LD Sobol sequences and is independent of prior deterministic methods. The required independence between the input bitstreams is guaranteed by simply using different Sobol sequences for generating the bitstreams and processing the streams for a specific number of cycles. The important point for this method is that the precision of the LD sequence generator should be i times the precision of the input data, where i is the number of independent bitstreams. Each input data must be converted to a stream of 2^{in} bits by comparing the input value to 2^{in} different numbers from the sequence generator. For example, to multiply two n-bit precision input data, two $2n$-bit precision Sobol sequence generators are required. Each input data is converted to a 2^{2n} length bitstream by comparing to the first 2^{2n} numbers from one of the two Sobol sequence generators. The generated bitstreams are then connected to an AND gate and the deterministic accurate output bitstream is ready after 2^{2n} cycles.

In the following, we see an example of multiplying two 2-bit precision input values using the first proposed method. The first input value is converted to a bitstream representation using the simplest Sobol sequence (Sobol seq. 1 in Fig. 4.9). The second input value is converted using the second Sobol sequence from the MATLAB built-in

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**
	a_0	a_2	a_1	a_3	a_0	a_2	a_1	a_3	a_0	a_2	a_1	a_3	a_0	a_2	a_1	a_3
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**
	b_0	b_2	b_3	b_1	b_2	b_0	b_1	b_3	b_3	b_1	b_0	b_2	b_1	b_3	b_2	b_0
Sobol Seq 3	**0**	**1/2**	**1/4**	**3/4**	**7/8**	**3/8**	**5/8**	**1/8**	**11/16**	**3/16**	**15/16**	**7/16**	**5/16**	**13/16**	**1/16**	**9/16**
	c_0	c_2	c_1	c_3	c_3	c_1	c_2	c_0	c_2	c_0	c_3	c_1	c_1	c_3	c_0	c_2
Sobol Seq 4	**0**	**1/2**	**3/4**	**1/4**	**7/8**	**3/8**	**1/8**	**5/8**	**7/16**	**15/16**	**11/16**	**3/16**	**9/16**	**1/16**	**5/16**	**13/16**
	d_0	d_2	d_3	d_1	d_3	d_1	d_0	d_2	d_1	d_3	d_2	d_0	d_2	d_0	d_1	d_3

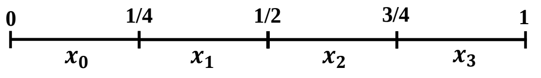


Figure 4.9: First 16 numbers of the first four Sobol sequences from MATLAB built-in Sobol sequence generator, and the category of each one based on their position in the [0, 1] interval.

Sobol sequence generator (Sobol seq. 2 in Fig. 4.9). Note that, when converting to a bitstream representation, a one is generated if the Sobol number is *less* than the input target number.

Example . Deterministic 2-bit precision multiplication using the first proposed method:

$$\begin{array}{r@{\,=\,}l} 1/4 & 1000\ 1000\ 1000\ 1000 \\ 3/4 & 1101\ 1110\ 0111\ 1011 \\ \hline 3/16 & 1000\ 1000\ 0000\ 1000 \end{array}$$

As can be seen, the accurate output of multiplying the two 2-bit precision input values is obtained by directly converting the inputs to 2^4-bit streams, by comparing them to the first 2^4 numbers of two Sobol sequences, and ANDing the generated bitstreams.

To prove why the first proposed method produces deterministic accurate results, we use two important properties of the Sobol sequences:

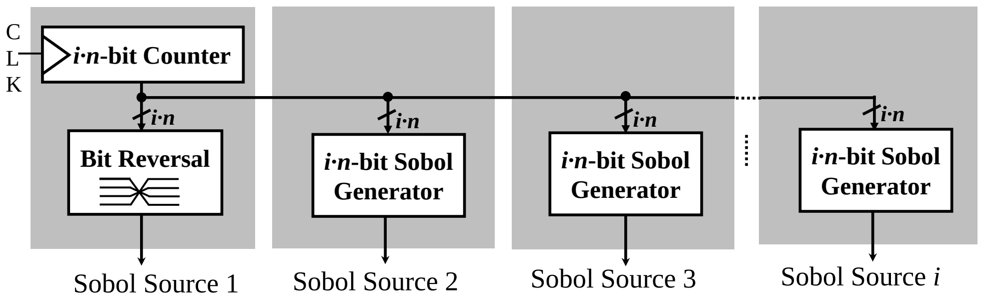
- The first 2^n numbers of any Sobol sequence include all n-bit precision values in [0, 1) interval.
- If equally split [0, 1) interval into 2^n sub-intervals, in any consecutive group of 2^n Sobol numbers starting at positions $i \times 2^n$ $(i = 0, 1, 2, \ldots)$, there is exactly one member in each sub-interval.

Fig. 4.9 categorizes consecutive groups of 2^2 numbers in the first four Sobol sequences. Each Sobol number in each group is labeled with a number from 0 to 3 depending on its sub-interval. For example 1/8 in Sobol sequence 1 is labeled with a_0 because it is a member of the first sub-interval, [0, 1/4), and 5/8 in Sobol sequence 2 is labeled with b_2 because it is a member of the third sub-interval, [1/2, 3/4). When converting a 2-bit precision input value into a 2^4-bit stream by comparing it to the first 2^4 numbers of a Sobol sequence, the result is the same for the Sobol numbers with the same label. For example, comparing 3/4 to 5/8 and 11/16 from the Sobol sequence 2 generates the same bit of '1' as both 5/8 and 11/16 are a member of [1/2, 3/4) (label b_2) and so are both less than the input value of 3/4. As can bee seen in Fig. 4.9, any selected group of 2^2 numbers includes all labels from 0 to 3, and as a result, all groups of the same Sobol sequence will produce the same number of 1s. All groups can accurately present the target input value and their difference will only be in the order of bits (order of labels).

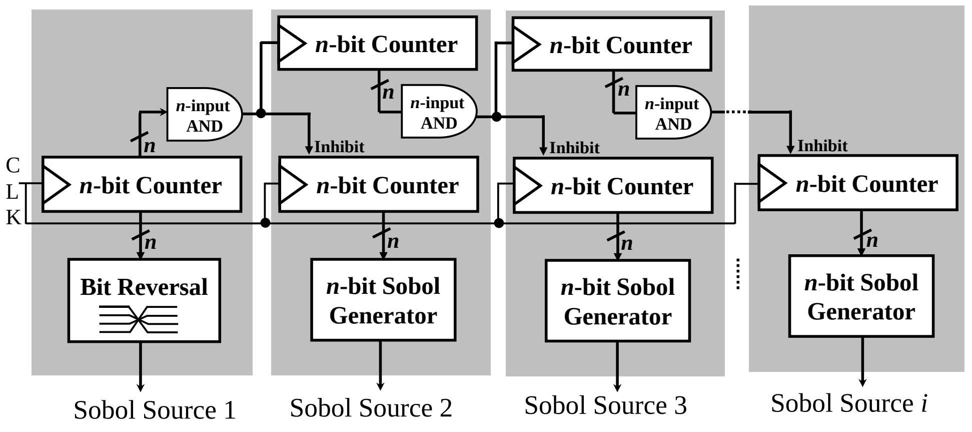
The result of multiplying two input values, represented by two bitstream, is deterministic and completely accurate if every bit of one bitstream meets every bit of the other stream exactly once [47]. As shown in Fig. 4.9 for $n = 2$, for any pair of two different Sobol sequences, every label u ($u = 0, 1, 2, 3$) in x_u ($x = a, b, c, d$) meets every label t ($t = 0, 1, 2, 3$) in y_t ($y = a, b, c, d$) exactly once if considering the first 2^4 numbers of each sequence. So, the result of multiplying two 2-precision numbers by ANDing their 2^4-bit stream representation, generated based on two different Sobol sequences, is deterministic and completely accurate.

This argument can be easily extended to multiplication of i n-bit precision numbers when converting the input numbers to bitstreams of $2^{i.n}$-bit length by comparing them to $2^{i.n}$ numbers from i different Sobol sequences. The generated bitstreams can be divided into groups of 2^n bits with different groups of a bitstream representing same n-bit precision value but with a different order (except the case of using Sobol Seq. 1, because labels in different groups of Sobol Seq. 1 have the same order). Every bit (label) from a bitstream interacts with every bit (label) of the other bitstreams exactly once, which results in a deterministic accurate output bitstream.

Fig. 4.10.a shows the structure of the sources of generating Sobol sequences for the first proposed method. These are used as the number sources in the stochastic stream generator of Fig. 1.1. Note that the simplest Sobol sequence is simply the reverse of the



(a)



(b)

Figure 4.10: Structures of the sources of generating Sobol sequences based on (a) first proposed method (b) second proposed method.

output bits of a binary counter and so we generate the first Sobol sequence by hard-wiring the output bits of a counter at no extra hardware cost.

4.4.3 Second Proposed Method

The second method is based on the prior deterministic methods introduced in [47]. Inspired from the idea of using pseudo-randomized bitstreams with the three state-of-the-art deterministic approaches (see Section 4.3), we propose to integrate the LD-sequences with the previously proposed deterministic methods. In Section 4.3, maximal period pseudo-random sources (i.e., maximal period LFSRs) are used to generate deterministic accurate bitstreams. The important point is that the period of the pseudo-random source should be equal to the length of the bitstream. By using such a source to generate random numbers, the input value could be converted into a pseudo-random but completely accurate stochastic representation. Instead of pseudo-random sources, here, we use LD sequence generators. In contrast to our first method, for the second method, the precision of the sequence generator is equal to the precision of the input data. For example, for multiplication of two n-bit precision inputs data, two n-bit LD sequence generators are required. In case of using LD Halton sequences, which are generated based on prime numbers, the relatively prime length method of [47] must be used to guarantee the required independence between the bitstreams. The Sobol sequences, on the other hand, must be integrated with the clock division or the rotation method of [47]. The operations then must continue for the product of the length of the bitstreams to produce deterministic complete accurate results.

In Section 4.3, we showed that the rotation method has a faster convergence property and is more energy efficient than the clock division deterministic method. So, for the rest of this section, for the second proposed method, we integrate the LD Sobol sequences with the rotation method. While we limit our reported results to LD Sobol sequences and the rotation approach, the proposed idea can similarly be applied to LD Halton sequences and the relatively prime length method.

The rotation method of [47] guarantees a deterministic accurate output by rotating the bitstreams through inhibiting or stalling on powers of the stream lengths. Fig. 4.10.b shows the structure of the sources of generating Sobol sequences for the second proposed method based on the rotation method. The first Sobol source repeats every 2^n cycles and does not rotate. Other Sobol sources (source k=2, 3, ..., i) have a period of 2^n but rotate every $2^{(k-1)\cdot n}$ cycles by inhibiting. Additional counters in the structure of the second method control these inhibits. We will show that, due to using n-bit Sobol generators, instead of expensive $i \cdot n$-bit generators, the structure of the second proposed method has a lower hardware cost than that of the first porposed method.

In the following, we see an example of multiplying two 2-bit precision input values using the second proposed method based on the first two Sobol sequences.

Example 8. Deterministic 2-bit precision multiplication using the second proposed method:

Sobol source 1 with a period of 2^2 and no rotation:

0,1/2,1/4,3/4, 0,1/2,1/4,3/4, 0,1/2,1/4,3/4, 0,1/2,1/4,3/4

Sobol source 2 with a period of 2^2 and inhibiting after every 2^2 cycles:

0,1/2,3/4,1/4, 1/4,0,1/2,3/4, 3/4,1/4,0,1/2 1/2,3/4,1/4,0

$$\begin{array}{r@{\,=\,}l} 2/4 & 1010\ 1010\ 1010\ 1010 \\ 3/4 & 1101\ 1110\ 0111\ 1011 \\ \hline 6/16 & 1000\ 1010\ 0010\ 1010 \end{array}$$

As can be seen, by exploiting the rotation approach, every number in the first four numbers of the Sobol source 1 pairs with every number in the first four numbers of the Sobol source 2 exactly once. This has led to a deterministic accurate multiplication when these rotated sequence of numbers are used in converting the input values, 2/4 and 3/4, into bitstream representation.

4.4.4 Accuracy Evaluation

For accuracy comparison of the proposed methods with prior works, we exhaustively tested multiplication of two 8-bit precision input data in the [0, 1] interval from a large set of random input values for the conventional approximate SC [12][2] and for the pseudo-random rotation approach of Section 4.3, and on every possible input value for the unary-stream based deterministic approach [47] and on the two proposed LD bitstream-based methods. Table 4.4 compares the mean absolute errors (MAEs) of the conventional approximate SC (using two different 16-bit LFSRs as the number generators), the prior deterministic unary-stream based rotation approach (using two 8-bit counters as the number generators), the deterministic pseudo-random based rotation approach (using two different 8-bit LFSRs as the number generators), the first proposed method based on LD Sobol sequences (using two 16-bit Sobol Sequence generators), and the second proposed LD Sobol sequence-based rotation approach (using two 8-bit Sobol generators). Note that for the two required Sobol sequences in the proposed methods we use the simplest Sobol sequence and the second Sobol sequence from the MATLAB Sobol sequence generator.

As can be seen in Table 4.4, similar to the deterministic approaches proposed in [47] and in Section 4.3, the two proposed methods could produce completely accurate results

Table 4.4: Mean Absolute Error (%) comparison of the prior approximate and deterministic approaches to SC and the proposed deterministic approach when multiplying two 8-bit precision stochastic streams with different numbers of operation cycles.

Design Approach	Area(μm^2)	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6
Conv. Approx. SC [12, 2]	781	0.05	0.15	0.26	0.39	0.58	0.79	1.20	1.67	2.32	3.32	4.72
Deter. Rotation Unary [47]	492	0.00	3.10	4.84	6.15	7.08	7.66	7.99	8.17	8.26	33.1	51.8
Deter. Rotation Pseudo-Random [98]	536	0.00	0.09	0.16	0.24	0.35	0.47	0.60	0.71	0.82	2.56	4.26
Proposed LD Method 1- Deter. Sobol	3361	0.00	0.0003	0.0013	0.0035	0.009	0.019	0.041	0.092	0.190	0.451	0.921
Proposed LD Method 2- Deter. Rotation Sobol	1277	0.00	0.0013	0.0033	0.0075	0.014	0.031	0.059	0.112	0.190	0.451	0.921

Table 4.5: Hardware area cost (μm^2) of the bitstream generator for different data precision and number of inputs (N=Input Data Precision - i=Number of Inputs)

Design Approach	N=4 i=2	N=4 i=3	N=4 i=4	N=8 i=2	N=8 i=3	N=8 i=4	Stream Generator Structure
Conv. Approx. SC [12, 2]	397	821	1394	781	1622	2799	i i*N-bit LFSRs + i N-bit Comparator
Deter. Rotation Unary [47]	224	342	459	492	754	1016	1 N-bit Counter + i-1 N-bit CounterEN + i N-bit Comp
Deter. Rotation Pseudo [98]	262	411	560	536	832	1127	1 N-bit LFSR + i-1 N-bit LFSREN + i N-bit Comp
Proposed LD Method 1	1005	3740	9127	3361	13193	32406	1 i*N-bit Counter + i-1 i*N-bit Sobol Gen+ i N-bit Comp
Proposed LD Method 2	456	806	1156	1277	2324	3371	i-1 N-bit Count+i-1 N-bit (CountEN+Sobol Gen)+i N-bit Comp

in 2^{16} cycles. Due to using LD bitstreams, however, the MAE of the computation is significantly lower than that of the prior approximate and deterministic approaches when truncating the bitstreams (running the operation for fewer cycles). For example, when running the multiplication operation for 2^{15} cycles (processing 2^{15}-bit streams), the proposed methods have shown a MAE of around 10^{-3}, which is 100X lower than the MAE of the deterministic pseudo-random rotation method of Section 4.3 and 3000X lower than that of the deterministic unary stream-based rotation approach of [47]. Thus, our methods have a much better progressive precision property and converge to the correct result much faster than prior methods.

4.4.5 Cost Comparison

The hardware area costs of the proposed methods for the case of implementing a 2-input 8-bit precision multiplier are also compared with the costs of the prior methods in Table 4.4. Each design includes two (random) sequence generators and two comparators to generate two independent stochastic bitstreams. We synthesized the designs using the Synopsys Design Compiler vH2013.12 with a 45-nm gate library. As can be seen in the table, the proposed methods have a higher cost than prior methods due to using costly Sobol sequence generators. The first proposed method is even 2.6X more costly than the second proposed method because of implementing two expensive 16-bit Sobol sequence generators. The important metric, however, to evaluate the efficiency of different methods is the area-delay product as an estimation of the energy consumption. As we will show in the next section, due to a very fast converging property, our proposed methods could satisfy a fixed accuracy expectation in a much shorter time, which will lead to a much lower area-delay product than prior methods. This, in particular, makes the proposed methods interesting for applications that can tolerate some degree of inaccuracy such as image processing and neural network applications.

4.5 Scalability Evaluation

Limited scalability has been an important challenge of prior deterministic methods of SC. As the authors in [15] discuss, when many mutually independent stochastic bitstreams are needed, the hardware cost significantly increases with the number of inputs. Stochastic bitstreams generated via LD sequences has a faster convergence than the pseudo-random sequences and of course than the unary counter-based sequences. However, the benefits of using LD sequences diminish as the number of inputs increase, because the cost of generating them is much higher than pseudo-random number generation. In this section, we evaluate the scalability of the proposed methods compared to prior methods and show that the second proposed method which integrates the LD sequences with the rotation approach has the best scalability compared to prior deterministic and also conventional approximate SC.

We implemented and synthesized 2-input, 3-input, and 4-input stochastic multipliers, with different design approaches, for multiplication of input data with 4-bit and 8-bit data precisions. The hardware area costs are reported in Table 4.5. As can be seen in the reported numbers, the deterministic rotation approach based on unary streams has the lowest hardware cost with the lowest cost increase rate (2X) from the 2-input to the 4-input multiplier. The first proposed LD method which has the fastest converging property (see Table 4.4) has the highest hardware cost with highest cost increase rate (9X) from the 2-input to the 4-input multiplier. The second LD method, on the other hand, not only has a very fast converging property, it also has a cost increase rate (X2.5) very close to the cost increase rate of the rotation unary stream-based method.

The MAE of the implemented multipliers for different stream lengths (different operation cycles) are presented in Figs. 4.11 and 4.12. As can be seen in Fig. 4.11, for the 4-bit precision multipliers, the proposed methods (DETER. SOBOL and DETER. ROTATION SOBOL) show a better progressive precision property than prior deterministic methods and their computation accuracy scales with increasing the number of inputs. Only the conventional approximate SC approach shows a better scalability than the second proposed LD method but it lacks the ability of generating completely accurate results. For the 2-input and 3-input multipliers with 8-bit precision (Fig. 4.12) we achieved the best accuracy performance by using the two proposed LD methods. Both methods converge to the expected correct value very fast and scale well with increasing the number of inputs.

Finally, we show the area-delay product of the implemented 8-bit precision multipliers for different MAEs in Fig. 4.13. We first exhaustively tested each design approach with a large set of input values and found the average processing time of each one to achieve

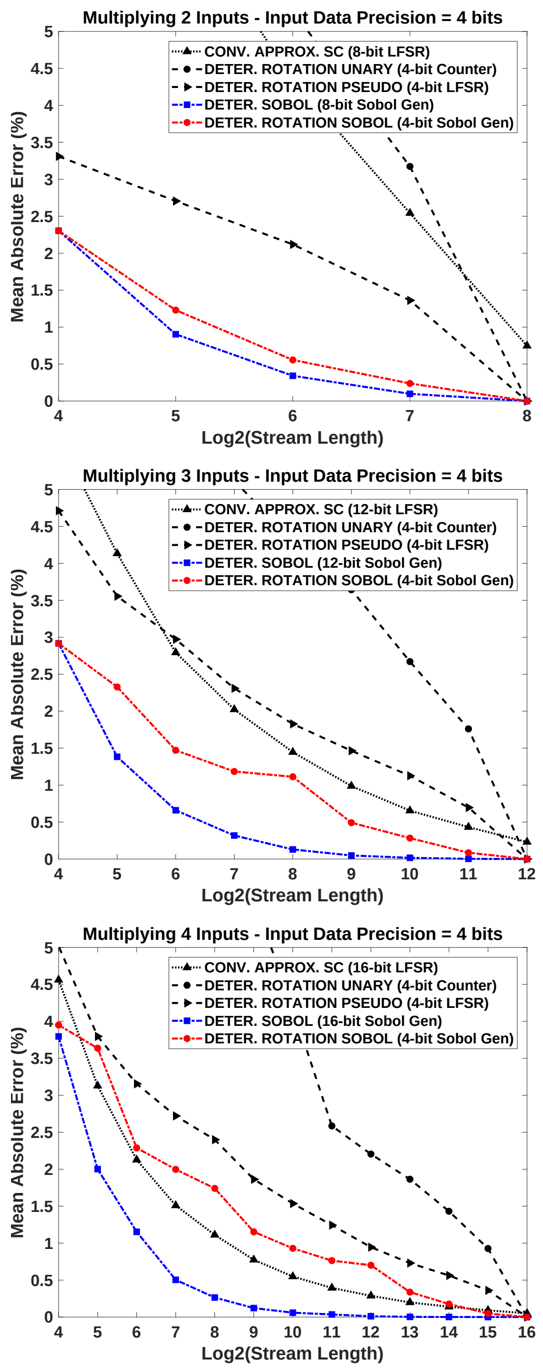


Figure 4.11: MAEs of 4-bit precision multipliers for different stream lengths.

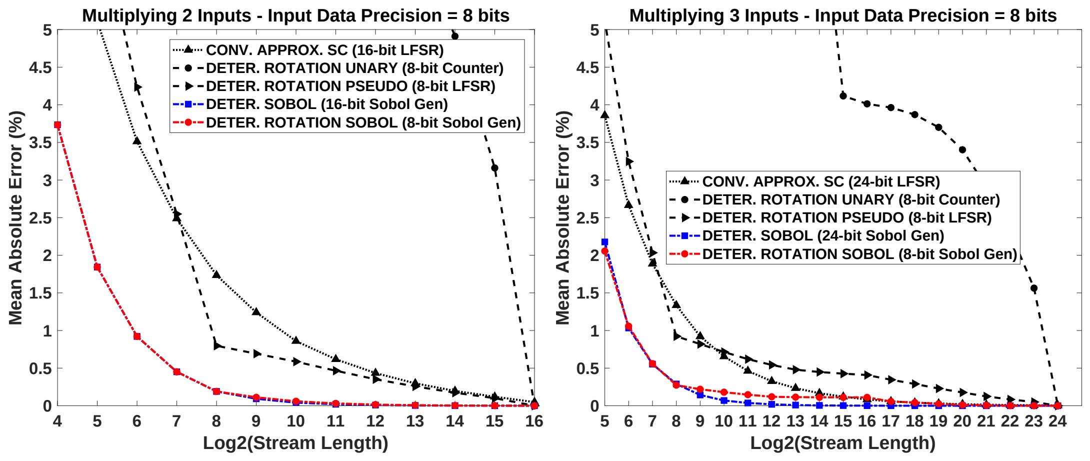


Figure 4.12: MAEs of 8-bit precision multipliers for different stream lengths.

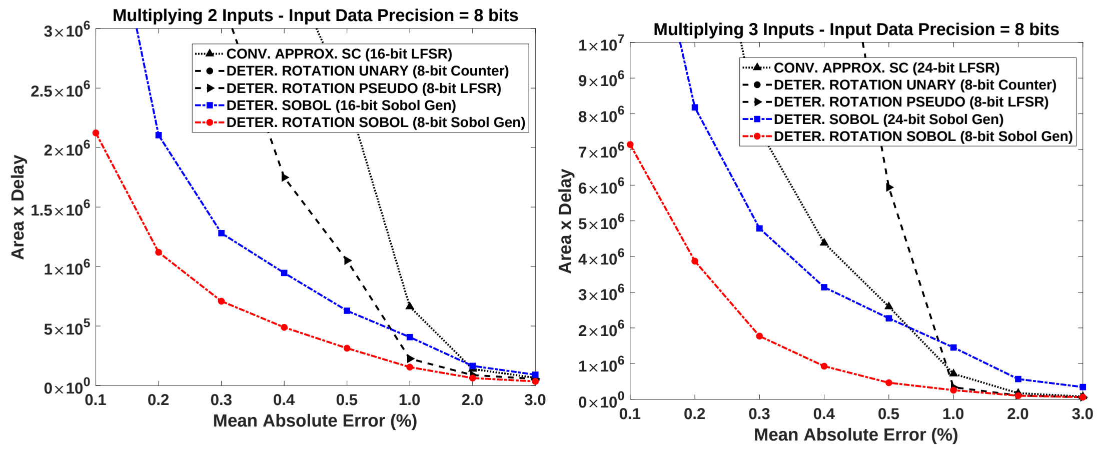


Figure 4.13: Area x Delay of 8-bit precision multipliers for different MAEs. Note that the Area x Delay numbers for the deterministic rotation unary method were much larger than other method and out of the range shown in the figure)

a specific MAE. We then multiplied the processing time with the corresponding design hardware area cost to produce the area-delay product. As can be seen in Fig. 4.13, the second proposed LD method (red lines) has the lowest area-delay product between different approximate and deterministic state-of-the-art methods which shows its superiority to prior methods and its scalability when the number of inputs increases.

4.6 Summary

Recent work on SC has shown that computation using stochastic logic can be performed deterministically and accurately by properly structuring unary-style bitstreams. While these approaches are appealing by generating completely accurate results, the cost of precise results makes them energy-inefficient for the cases that slight inaccuracy is acceptable. In this chapter, we first exploited pseudo-randomness in improving the progressive precision property of the previously proposed deterministic approaches of SC. We then proposed two new fast-converging scalable deterministic approaches of processing bitstreams based on low-discrepancy (LD) sequences. The first LD-based method provides the best accuracy for a fixed processing time while the second LD-based method has the lowest area-delay product. Completely accurate results are produced when running the operation for the required number of cycles. When slight inaccuracy is acceptable, significant improvement in the processing time and energy consumption is observed compared to the prior unary stream-based deterministic approaches and also the conventional random stream-based approaches.

Chapter 5

Polysynchronous Stochastic Circuits

This chapter introduces a new advantage to SC paradigm. We show that SC circuits naturally and effectively tolerate very high clock skew. Exploiting the skew tolerance of SC circuits, we introduce polysynchronous clocking, a design strategy for optimizing the clock distribution network (CDN) of SC-based systems. We describe two approaches to polysynchronous system design: (1) replacing the global CDN with locally generated clocks, and (2) relaxing the global CDN. We provide a case study comparing the cost and benefits of conventional design with CDNs to polysynchronous designs, quantifying the area, speed and energy advantages. We finally compare the error tolerance of polysynchronous stochastic circuits to conventional synchronous stochastic circuits. This chapter's material has been published in [104], [17], and [105].

5.1 Motivation

All electronic systems are inherently asynchronous in nature. By carefully choreographing transitions with clock signals, asynchronous circuitry can be adapted to appear to behave synchronously. Such synchronism brings significant advantages: it greatly simplifies the design effort; also, with predictable timing, one can make performance guarantees. However, synchronism comes at a significant cost: one must create a clock distribution network (CDN).

The CDN distributes the clock signal from a single oscillator to stateholding components, such as flip-flops. The primary design goal for CDNs is to maintain signal integrity while distributing the clock widely. In the ideal case, transitions in the clock signal should arrive at all state-holding elements at precisely the same moment (so there is zero clock uncertainty). Achieving this is difficult and costly in terms of design effort and resources. In modern large-scale integrated circuits, the CDN accounts for significant area, consumes significant power, and often limits the overall circuit performance [106, 107, 108]. With increasing variation in circuit parameters, designing CDNs with tolerable clock uncertainty is becoming a major design bottleneck.

There are two kinds of variations that lead to uncertainty in the arrival time of the clock edge at sequential circuit elements: spatial and temporal. Spatial variations, known as skew, affect the arrival of the various clock edges at the sequential elements within a single clock cycle. Skew can limit circuit performance, since a circuit must be clocked at a lower frequency to tolerate it. There is a designer's rule of thumb that clock skew should be less than 10 percent of the clock period. As clock frequency goes up, more complex CDNs are required to keep skew at a constant fraction of the clock period. Increasing die size, clock loads, and process variability magnify the challenge [108].

Temporal variations, known as jitter, also affect the arrival time of the clock edges at the sequential elements across different clock cycles [109]. Even when designed to be zero, environmental and processing variations can nonetheless lead to significant amounts of clock uncertainty. Various strategies are used to minimize the uncertainty in the delivery of clock signals. For instance, buffers and inverters can be inserted to balance the delays between the clock source and the clock sinks. However, this costs– both in area and design effort.

Stochastic computing offers skew and clock uncertainty tolerance. SC circuits can naturally and effectively tolerate very high clock skew. Suppose that the bits in different input streams are temporally misaligned, that is to say, the bit transitions do not line up correctly in time. The SC circuit will compute an output value based on the input values it sees at any moment in time (ignoring subtleties such as setup and hold times). Since it is only the fraction of time that the signal is high that matters, averaged over time, the result of the SC operation will be correct. In this chapter, we will explain how this feature can be used to mitigate the costs of implementing SC-based systems: either the global CDN can be eliminated entirely; or one can design a much less costly global CDN that tolerates skew.

5.2 Background

5.2.1 Stochastic Operations

Multiplication

As we discussed in Section 2, multiplication is implemented using a standard AND gate for the unipolar coding format. Fig. 5.1 shows the multiplication of two 10-bit unipolar stochastic streams using an AND gate. The value represented by a bitstream is the time that the signal is high divided by the total length of the stream. Fig. 5.2 illustrates an example of multiplying two unsynchronized bitstreams representing 0.6 and 0.5. As shown, the value represented by the bitstream at the output of the AND gate is 0.3, the value one expects when multiplying 0.6 by 0.5.

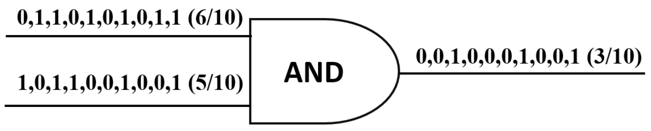


Figure 5.1: Example of stochastic multiplication using an AND gate.

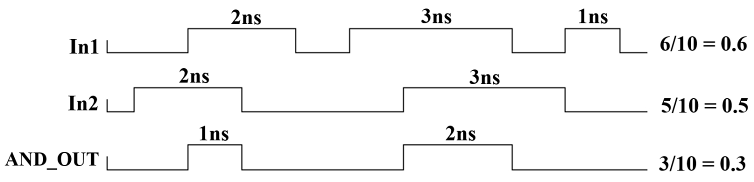


Figure 5.2: Stochastic multiplication using an AND with unsynchronized bitstream.

Scaled Addition and Subtraction

Stochastic values are restricted to the interval [0, 1] (in the unipolar case) or the interval [-1, 1] (in the bipolar case). So one cannot perform addition or subtraction directly, since the result might lie outside these intervals. However, one can perform scaled addition and subtraction. These operations can be performed with a MUX. Fig. 5.3 illustrates the operation $\frac{1}{2}A + \frac{1}{2}B$ with digital bitstreams. Fig. 5.4 illustrates another example of scaled addition, this time on two unsynchronized bitstreams representing 0.25 and

0.5. As expected, the output is a bitstream representing 0.375, the result of the scaled addition.

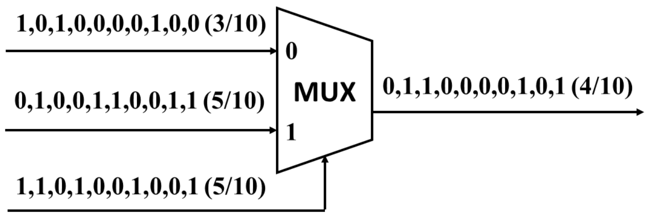


Figure 5.3: Example of stochastic scaled addition using a MUX unit.

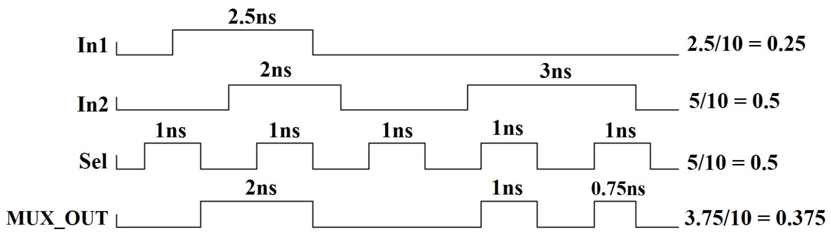


Figure 5.4: Stochastic scaled addition using a MUX with unsynchronized bitstreams.

FSM-based operations

More complex functions can be implemented in SC using finite state machines (FSMs). The stochastic implementation of the exponentiation function and the tanh function were developed by Brown and Card [31]. Li and Lilja [61] also developed an FSM-based stochastic absolute value function. The state transition diagrams of the FSMs implementing these functions are shown in Fig. 5.5. Assuming that the input to these FSMs is a random signal that is high a fraction X of the time, the output signal Y converges to expected value: a fraction of time at high equal to $\exp(X), \tanh(X)$ and $\text{abs}(X)$. Note that these FSMs only differ in how the outputs are computed from the current state. Transition diagrams with 8 states are shown here; these can readily be generalized to FSMs with more states [16].

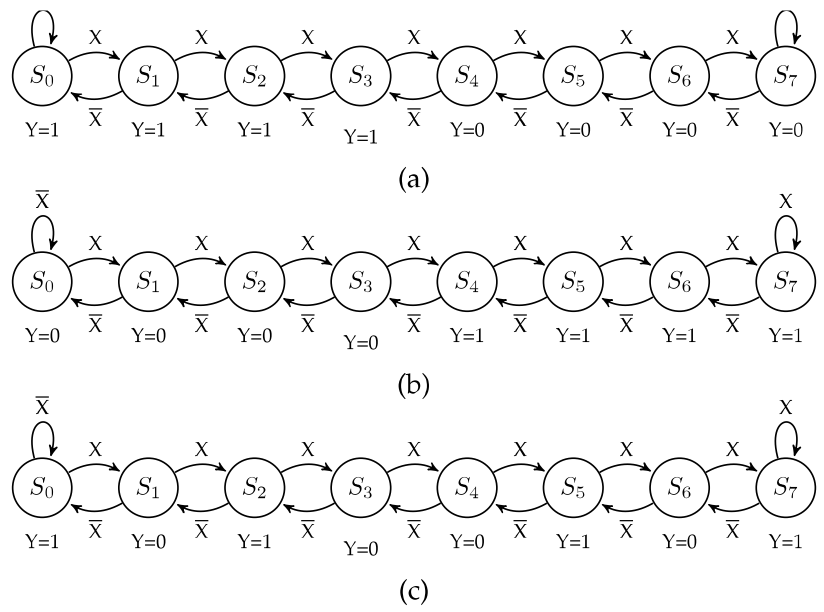

S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7
X
$\overline{X}$
(a) Y=1 Y=1 Y=1 Y=1 Y=0 Y=0 Y=0 Y=0
(b) Y=0 Y=0 Y=0 Y=0 Y=1 Y=1 Y=1 Y=1
(c) Y=1 Y=0 Y=1 Y=0 Y=0 Y=1 Y=0 Y=1

Figure 5.5: State transition diagram of the FSM implementing a) the stochastic exponentiation function b) the stochastic tanh function c) stochastic absolute value function. For details of the implementation, the readers are referred to [9].

5.2.2 Stochastic Circuits

SC has been applied to a wide variety of applications, including image and signal processing applications. In this chapter, we use circuit implementations of three fairly complex image processing algorithms as case studies: Robert's cross edge detection, Median filter based noise reduction circuit, and image segmentation based on stochastic kernel density estimation.

Robert's cross edge detection

Robert's cross edge detection algorithm is a well-known and widely studied algorithm. A low-cost implementation of the Robert's cross method which works on correlated input bitstreams was shown in Figure 2.19a. A more complex FSM-based stochastic implementation of this algorithm which works with independent random bitstreams is proposed in [4] and shown in Fig. 5.6. In this circuit, each Robert's cross operator consists of a pair of 2×2 convolution kernels that process an image pixel based on its

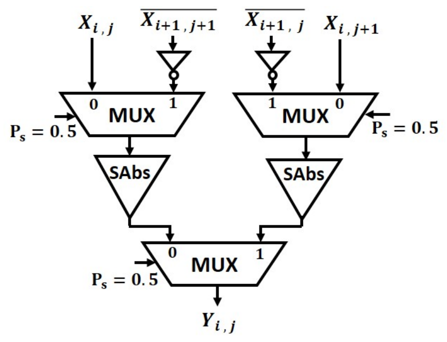


Figure 5.6: Stochastic implementation of the Robert's cross edge detection algorithm [4].

three neighbors as follows

$$y_{i,j} = \frac{1}{2} \times (\frac{1}{2}|x_{i,j} - x_{i+1,j+1}| + \frac{1}{2}|x_{i,j+1} - x_{i+1,j}|) \tag{5.1}$$

where $x_{i,j}$ is the value of the pixel at location (i, j) of the original input image and $y_{i,j}$ is the output value computed for the same location in the output image. In the circuit of Fig. 5.6, three multiplexers perform addition and subtraction, while two FSM based stochastic circuits perform the required absolute value operations. Since the Robert's cross circuit operates on signed values, all streams must be in the bipolar format.

Median Filter Noise Reduction

The median filter replaces each pixel of an input image with the median of neighboring pixels. It is quite popular because, for certain types of random noise, it provides excellent noise-reduction capabilities [110]. A hardware implementation of the 3x3 median filter based on a sorting network was shown in Fig 3.8. Instead of the AND-OR based implementation of the basic sorting unit which requires highly correlated input bitstreams (e.g., unary-style bitstreams), here, we use the FSM-based sorting circuit shown in Fig. 5.7 in implementing the median filter circuit. This implementation works with independent random bitstreams. In total, the median filter circuit requires 19 basic sorting units (57 MUX units and 19 FSM-based stochastic tanh circuits.)

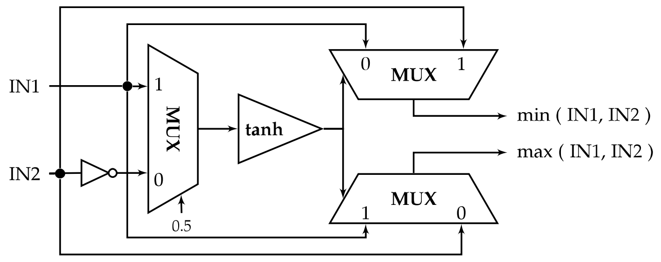


Figure 5.7: Stochastic implementation of basic sorting unit.

Kernel Density Estimation-based Image Segmentation

Image Segmentation based on Kernel density estimation is an image processing algorithm which is used in object recognition and tracking applications to extract changes in a video stream in real time. Using a probability density function (PDF), the distribution of intensity values a pixel will have at time t can be estimated. A stochastic implementation of this algorithm based on 32 recent frames of the input video, proposed in [10], is shown in Fig. 5.8. Let $X_t, X_{t-1}, X_{t-2}, ..., Xt - n$ be recent samples of intensity values of a pixel X. The stochastic circuit proposed in [10] uses the following formula as the probability estimator:

$$PDF(X_t) = \frac{1}{n}\sum_{i=1}^{n} e^{-4|X_t - X_{t-i}|} \tag{5.2}$$

Using this probability estimator, a pixel is considered a background pixel if $PDF(X_t)$ is less than a predefined threshold value. In total, the circuit includes 64 MUXs, 32 FSM-based stochastic exponentiation circuits, and one FSM-based stochastic tanh circuit.

5.3 Polysynchronous Clocking

With a stochastic representation, computational units can tolerate skew in the arrival time of their inputs. This stems from the fact that the stochastic representation is uniform: all that matters in terms of the value that is computed is the fraction of time that the signal is high. The correct value is computed even when the inputs to a computational unit are misaligned temporally. Consequently, precise synchronization between the arrival time of input values to logic gates does not matter. This observation motivates the topic of this section: polysynchronous clocking.

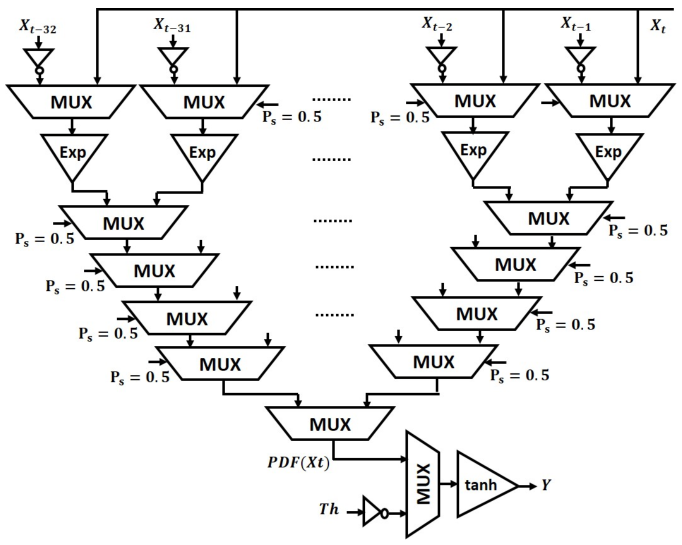


Figure 5.8: Stochastic implementation of the KDE-based image segmentation algorithm [10].

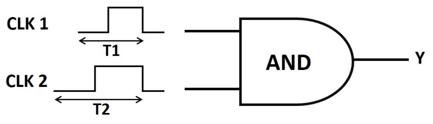


Figure 5.9: An AND gate connected to polysynchronous clock sources.

Consider an AND gate, responsible for multiplying two unipolar input bitstreams, P1 and P2, generated by stochastic number generators driven by two clocks with different periods, T1 and T2. To simplify the problem, we first connect two clocks with 50 percent duty cycles directly to the inputs of an AND gate (Fig. 5.9). This is equivalent to connecting two stochastic streams both representing P=0.5. Therefore, the expected output value is Y=0.25. We want to verify the functionality of performing multiplication using an AND gate according to three different scenarios: 1) T1=2 ns, T2=3.5 ns, 2) T1=2 ns, T2=3.2 ns, and 3) T1=1.8 ns, T2=3.2 ns.

Fig. 5.10 illustrates the input signals as well as the output signal in the case where T1=1.8 ns and T2=3.2 ns for 20 ns of operation. Continuing the operation for about 1000 ns will produce a good view of the different lengths of high pulses that are observed at the output of the AND gate. Dividing the total fraction of the time that the output signal is high by the total time gives the result of the multiplication operation. Table 5.1

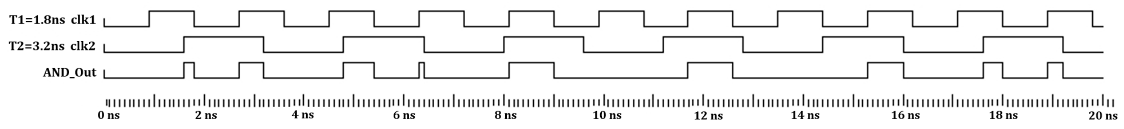


Figure 5.10: Input clock signals and the corresponding output from connecting polysynchronous inputs to an AND gate.

presents results for the three selected cases of clock periods. It lists the number of occurrences of high pulses of each length that is observed, as well as the total time of the high pulses.

As can be seen in Table 5.1, when we vary the periods of the two clock sources, the total time that the output is high does not change much. The length of the observed high pulses and the number of occurrences of each changes, but the total fraction of the time that the output is high is very close to 250 ns. Dividing 250 ns by 1000 ns produces 0.25, the expected output of multiplying the two input streams. This example provides an intuitive explanation of why polysynchronous stochastic operations work: temporal misalignment of input values does not affect the accuracy of the computation.

Table 5.1: Different observed lengths of high pulses at the output of the AND gate and the number of occurrences of each one for three pairs of clock periods when executing the multiplication operation for 1000 ns.

T1=2ns		T1=2ns		T1=1.8ns	
T2=3.5ns		T2=3.2ns		T2=3.2ns	
Length	#	Length	#	Length	#
0.25	72	0.2	63	0.1	35
0.50	72	0.4	63	0.2	35
0.75	71	0.6	62	0.3	35
1.00	142	0.8	62	0.4	35
-	-	1.0	125	0.5	35
-	-	-	-	0.6	35
-	-	-	-	0.7	35
-	-	-	-	0.8	34
-	-	-	-	0.9	138
Total High	249.25		249.60		249.40

Table 5.2: The measured output of the MUX when three polysynchronous clocks with distinct periods are connected to its inputs for 1000 ns.

T1	T2	T3	Total High Time	Measured Output	Expected Output
2.00	1.80	3.75	499.43	0.499	0.500
1.90	2.63	2.12	500.21	0.500	0.500
3.20	1.60	2.00	498.80	0.499	0.500
2.87	2.43	2.10	499.23	0.499	0.500

Table 5.3: Stochastic multiplication and scaled addition, using an AND gate and a MUX, respectively, with inputs generated by unsynchronized SNGs.

					AND Output		MUX Output	
In1	T1(ns)	In2	T2(ns)	T3(ns)	Measured	Expected	Measured	Expected
0.50	2.10	0.50	2.30	2.00	0.247	0.250	0.502	0.500
0.35	2.82	0.66	3.11	3.68	0.237	0.231	0.498	0.505
0.27	2.81	0.48	2.36	3.61	0.128	0.129	0.372	0.375
0.18	1.60	0.53	3.70	2.20	0.096	0.095	0.350	0.355

Next, we analyze the functionality of a MUX unit performing scaled addition with temporally misaligned inputs. The analysis is similar to that of an AND gate performing multiplication. Note, however that the MUX unit has an extra select stream performing the scaling. To study the functionality of the MUX unit we connect three polysynchronous clocks with distinct periods, T1, T2, and T3, to the first, second, and select inputs of the MUX. We compare the fraction of time that the output is high divided by the total time to the expected value, $(1/2+1/2)/2$. The results are shown in Table 5.2. These results are similar to what we saw for the multiplication operation. The measured output values are essentially equal to the expected output value of 0.5.

Now we discuss the general case of operations on stochastic streams generated by SNGs that are driven by separate clocks, and so are not synchronized. Table 5.3 presents the results of trials for stochastic multiplication and scaled addition. In this table, T1 and T2 are the periods of the clocks of the SNGs responsible for generating the first and the second streams, respectively. For the scaled addition operations, T3 is the period of the clock of the SNG responsible for generating the select stream, which is set to

0.5. Note that the results presented in Table 5.3 are based on bitstreams of length 1024, generated with 32-bit LFSRs. This configuration produces a good Bernoulli distribution of probabilities for the individual bits in the stream. As can be seen in this table, all of the measured values are very close to the expected values. Indeed, in spite of the polysynchronous clocking, the results are accurate to within the error bound expected for SC [2].

Proof. Polysynchronous stochastic signals can be discretized into digital stochastic bitstreams by dividing the signals into pulses of size ϵ and assigning 0/1 values to these pulses. Suppose that we discretize two polysynchronous signals, X and Y, into digital bitstreams, $X(t)$ and $Y(t)$. Assuming that the fraction of time the polysynchronous signals are high are x and y, respectively, the probability that each bit in the discretized streams is one is also $P(X = 1) = x$ and $P(Y = 1) = y$, respectively. If the discretized bitstreams are stochastically independent, by connecting them to the inputs of an AND gate, the output is a bitstream $Z(t)$, where:

$$\begin{aligned} Z = P(Z = 1) &= P(X = 1 \ \ \mathit{and} \ \ Y = 1) \\ &= P(X = 1)P(Y = 1) = x \cdot y \end{aligned}$$

Thus, correspondingly, for any two independent polysynchronous signals, an AND gate computes the product of the values:

$$\int_0^T Z \ dt = \int_0^T XY \ dt = x \cdot y$$

as ϵ approaches zero. Similarly, we can show that connecting independent polysynchronous signals to the main and to the select inputs of a MUX produces the result of scaled addition/subtraction. Note that polysynchronous signals generated by identical SNGs but driven by different clocks, are expected to be independent, since they are not synchronized in any way.

For a circuit-level verification of the polysynchronous idea, we implemented the SPICE netlist of the Robert's cross stochastic circuit. Simulations were carried out using a 45-nm gate library in HSPICE on 1000 sets of random input values, for both synchronous and polysynchronous clocking conditions. Each set of inputs consisted of four different random values. For the conventional synchronous clocking condition, the circuitâĂŹs clock period was fixed at 1 ns. For the polysynchronous clocking conditions, clock periods were selected randomly in the range from 1 ns to 2 ns (so 100 percent variation). Note that the period corresponds to a single bit in the random stream.

The accuracy of the results was computed by calculating the difference between the expected value and the measured value. On 1000 trials, we found that the mean of the output error rates was 4.85 percent for the synchronous and 4.45 percent for the polysynchronous approach. Hence, the polysynchronous stochastic circuits are essentially as accurate as conventional synchronous circuits.

With polysynchronous clocking, the global clock signal of a circuit and its associated CDN can be replaced by multiple inexpensive clocks for different local domains. The division into domains can be performed down to a very fine level, even up to a handful of gates. The local clocks can be generated with simple inverter rings. In subsequent sections, we evaluate the idea of polysynchronous clocking with case studies, presenting detailed experimental results.

5.4 Polysynchronous System Design: A Case Study

In the polysynchronous stochastic design paradigm, the system is divided into three main units: 1) stochastic number generators (SNGs) that convert input values, perhaps from analog sources, into the corresponding stochastic signals; 2) computational units that accept stochastic input signals, and perform operations, producing stochastic output signals; and 3) stochastic output converters that produce output signals, perhaps for analog outputs such as voltage accumulators. The output converters measure the fraction of time the output signals are high divided by the total operation time to produce the final values.

Suppose that we are given an input $n \times n$ gray-scale image to process with a Robert's cross circuit. We can use n^2 instances of the Robert's cross circuit, presented in Fig. 5.6, to process each of the pixels concurrently. Fig. 5.11 shows a diagram of such a parallel circuit for $n = 8$. Call each instance a Robert's cross *cell*. Each cell converts one input pixel value, represented as a stochastic signal, into an output pixel value, represented as stochastic signal. An SNG in each cell is responsible for the input conversion. The cell communicates with its neighbor cells to receive their pixel values, all represented as stochastic signals.

We consider three different cases to validate the concept of polysynchronous clocking. First, we implement our case study using a conventional synchronous approach: a global CDN that synchronizes all cells. Next, we remove the global CDN and instead use locally generated clocks for each cell; now the cells will not operate synchronously. Finally, we synthesize the circuit with a“relaxed CDN.” In each case, we quantify the costs for the Robert’s cross circuits with 16, 64, and 256 cells.

5.4.1 Synchronous Design: Global CDN

In the conventional approach, a global CDN is synthesized to synchronize all components of the system with a common clock signal. The arrival time of the clock signal needs to be synchronized throughout. With variations, this requirement for zero clock skew is challenging, requiring considerable design effort. The larger the circuit, the more complex the CDN. Often, a large number of buffers must be inserted throughout the CDN to balance the clock tree and satisfy the arrival time requirements. In addition to the high amount of design effort expended, the CDN consumes considerable area and power.

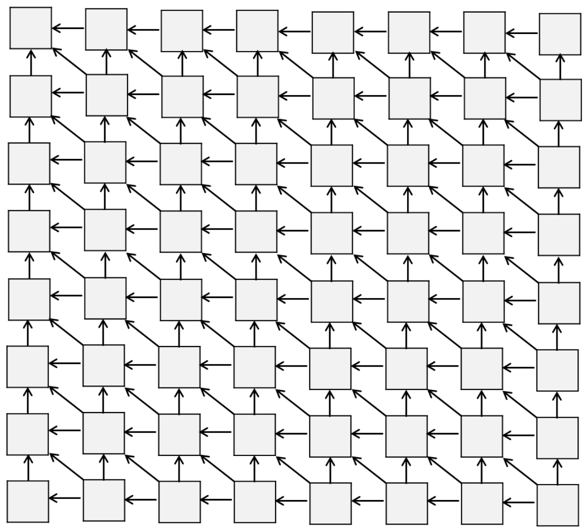


Figure 5.11: 64 Robert’s Cross Cells processing a 8×8 input image concurrently.

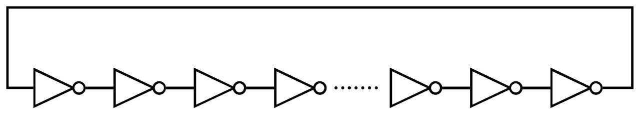


Figure 5.12: Ring oscillator circuit with odd number of stages

5.4.2 Polysynchronous Design: Removing the CDN

In the first polysynchronous approach, we replace the global CDN with unsynchronized local clocks. Two different approaches can be used to supply local domains with clock signals: 1) Using clock signals from external sources, and 2) self-timed local clock generators. Because of the limitation and extra costs of I/O ports, the first approach is more practical when there are a small number of clock domains. With a large number of domains, self-timed local clock generators are generally advantageous. In what follows, we evaluate the second approach. We present quantitative comparisons of the performance-cost gain when the global CDN is replaced with multiple local clock generators.

Ring oscillators can be used as fast and inexpensive local clock generators. A ring oscillator consists of an odd number of inverter gates connected in a ring, as shown in Figure 5.12. NAND and NOR gates can also be used to build ring oscillators. Due to their longer delay, a smaller number of NAND or NOR gates are required to achieve the same oscillation period as an inverter ring. As a result, the area cost of the NAND- and NOR-based oscillators is lower than that of an inverter-based oscillator. However, due to its lower power consumption, an inverter-based oscillator is generally more energy-efficient. The oscillation period of a ring oscillator is twice the sum of the gate delays. The frequency can be increased by either increasing the supply voltage or by decreasing the number of inverters [111, 112]. Note that a ring of approximately 110 inverter gates is necessary to generate a local clock with a period of 1 ns in 45-nm technology when the supply voltage is 1V. Thus, although relatively inexpensive, the area and power costs of inverter rings are not insignificant.

5.4.3 Polysynchronous Design: Relaxed CDN

Instead of eliminating the CDN, an alternative approach is to relax the requirements on it, permitting significant clock skew throughout the system. This can significantly simplify the synthesis process, saving area, lowering power, and increasing performance by permitting the system to be clocked at a higher speed. Obviously, this approach does not entail the use of local clock generators.

A significant advantage that such a "relaxed CDN" provides is ease in controlling the working frequency. With local clocks, generated by inverter rings, the frequency will generally be fixed (some implementations of ring oscillators do allow for slight adjustments to the period; however, the possible range of values is more or less fixed by the number of inverters used). In contrast, the frequency of an external clock provided to a "relaxed CDN" can be changed freely, in some cases permitting significant over-clocking.

5.5 Experimental Setup

In order to quantify the performance and cost benefits of both approaches to polysynchronous design, that is, by removing the CDN or relaxing it, we implemented the Robert's cross circuit for values of $n = 4, 8$, and 16 in Verilog. The SNG unit presented in Figure 1.1 was used in each cell to convert the input pixel value into a corresponding stochastic signal. A 10-bit maximal period LFSR was used in each cell to supply the SNG with pseudo-random numbers. We used different random number generators (different LFSR designs, with different seeds) in the different cells to ensure that the stochastic bitstreams are uncorrelated. Applying polysynchronous clocking can further help de-correlate stochastic streams and can introduce additional randomness. FSM-based SAbs circuits with 16 states were used to implement the required absolute value function. We used the Synopsys Design Compiler vH2013.12 [62] with a 45-nm gate library to synthesize the designs.

For synthesizing the circuits with conventional global CDNs, we considered a "clock uncertainty" value of at most 10 percent (0.1 ns for the smaller 16-cell circuit, and of 0.2 ns for the larger 64 and 256-cell circuits). This uncertainty parameter in the Synopsys Design Compiler represents process variations and other sources of variability that cause variations in the clock delay. In the synthesis flow, the tool uses extra elements, mainly

Table 5.4: Synthesis results for a single Robert's cross cell with and without a local clock generator.

One Robert's cross cell	Area (μm^2)	Power @2Ghz (mW)
Without local clock generator	268.0	0.83
With local clock generator	291.9	1.09

delay buffers, to ensure near zero clock skew in the signal arrival time at all components. It produces a circuit with cells that are nearly perfectly synchronized.

For the "relaxed CDN" approach, we allow for significant skew and jitter by defining a clock source uncertainty of zero and accepting some timing violations. As a result, the tool ignores the delays due to the clock network latency and the propagation delay in different paths. It does not add any buffers to compensate for clock uncertainty. With this approach, different cells are at differing distances from the clock input source. As a result, the clock signals arriving at different cells are not synchronized. We use this configuration to test the ability of the polysynchronous approach to tolerate the clock skew and jitter.

For the approach where we eliminate the global CDN entirely by replacing it with local unsynchronized clocks, we synthesized the system with 16, 64, and 256 cells, with each cell containing an inverter ring. In order to design the inverter rings, we first synthesized a single Robert's cross cell and found its critical path to be 0.49 ns. SPICE-level simulations showed that 45 inverter gates are required to generate a clock signal with this period in the 45-nm technology when using a supply voltage of 1V. Such inverter rings were added to each Robert's cross cell. Table 5.4 shows the area and power costs of a single Robert's cross cell before and after adding the inverter rings. Adding the inverter ring incurs area and power overhead of 8 percent and 24 percent, respectively. We will show that, for large designs, this overhead is small compared to the savings obtained by removing the CDN.

Table 5.5: Delay, area, power, and average error rate comparison of the implemented circuits for different approaches of synthesizing the CDN.

Circuit	CDN	Delay (ns)	Area (μm^2)	Power (mW)	Energy (pJ)	Area×Delay ($\mu m^2 \times \mu s$)	Error Rate (percent)
Robert 16-cell	Synchronous	1.56	4485	5.41	8.44	7.00	2.20
	Poly Local	0.49	4332	19.04	9.33	2.12	1.77
	Poly Relaxed	0.99	4025	8.1	8.02	3.98	2.12
Robert 64-cell	Synchronous	3.20	25438	13.25	42.40	81.40	2.56
	Poly Local	0.49	16750	76.26	37.37	8.21	1.67
	PolyRelaxed	2.20	19391	15.45	33.99	42.66	2.57
Robert 256-cell	Synchronous	6.30	111319	31.06	195.68	701.31	2.68
	Poly Local	0.49	67242	306.18	150.03	32.95	1.87
	Poly Relaxed	5.1	91121	33.12	168.91	464.72	2.37
Median Filter	Synchronous	2.91	3169	1.39	4.04	9.22	2.64
	Poly Relaxed	2.45	2694	1.45	3.55	6.60	2.62
KDE	Synchronous	2.14	4921	3.08	6.60	10.53	1.70
	Poly Relaxed	1.75	4443	3.42	5.99	7.78	1.69

5.6 Experimental Results

5.6.1 Synthesis Results

The synthesis results, including the delay, area, total dynamic and static power consumption, energy dissipation of one clock cycle, and area-delay product, are shown in Table 5.5. The reduction in delay, seen as equivalent to increasing the working frequency, is the most significant benefit of polysynchronous clocking. The results show that increasing the number of cells limits the performance of the system when a global CDN with zero clock uncertainty is implemented. Providing all the cells with synchronized clock signals is costly. For the system with 256 cells, removing the CDN and instead using locally generated clocks improves the maximum working frequency by around 12x. As a result, the output converges to an accurate value much faster. With a relaxed CDN, the benefit is also significant, although not as great as with locally generated clocks. The savings gained by these approaches are presented in Figure 5.13.

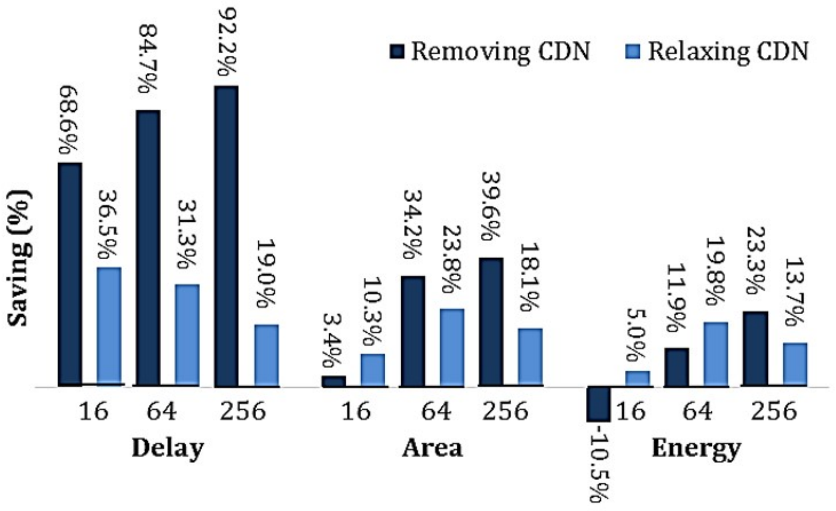


Figure 5.13: Comparing the savings due to using different approaches of polysynchronous clocking on various sizes of the Robert's cross circuit.

In terms of area, both approaches decrease the cost in the three cases with 16, 64, and 256 cells, as shown in Figure 5.13. As expected, for large-scale systems (64 and 256 cells), removing the CDN provides more area saving than simply relaxing the CDN. It provides up to a 39 percent area reduction in the system with 256-cells. However, for smaller systems, the area overhead incurred by the local clock generators diminishes the benefits. We conclude that relaxing the CDN instead of completely eliminating it is the better approach for small circuits.

As shown in Table 5.5 and Figure 5.13, removing the CDN results in an overall energy dissipation reduction, except for the 16-cell circuit. For the 16-cell circuit, removing the CDN improves the latency and area by 68 percent and 3 percent, respectively. However, the power consumption of the circuit with the highest frequency increases around 3.5×. This increase in power consumption occurs because the local clock's power consumption outgrows the power savings obtained by eliminating the CDN, which is small for this circuit. A higher working frequency also increases the power. Consequently, a 10 percent increase in the energy dissipation is observed. Thus, unless improving the working frequency is the main goal, relaxing the CDN or using a zero-clock-skew CDN might be better choices for smaller circuits. However, for larger circuits, eliminating the global CDN and using locally generated clocks is a winning proposition.

To further evaluate idea of relaxing the CDN in stochastic circuits, we implemented two complex circuits, discussed in Section 5.2.1: a median filter based noise reduction circuit and a kernel density estimation based image segmentation circuit. These were implemented: 1) using a conventional synchronous approach with zero clock uncertainty tolerance; and (2) in the proposed polysynchronous approach with a relaxed CDN. FSM-based stochastic circuits with 32 states were used to implement the required *tanh* and *exp* functions. We used a 0.2 ns clock uncertainty when the circuits were synthesized with the Synopsys Design Compiler. Table. 5.5 compares the delay, area, power, and energy results extracted for these circuits. As can be seen, relaxing the CDN improves the performance and saves area for both circuits. The power consumption when using the maximum working frequency is higher with a relaxed CDN due to the increase in the frequency. However, more importantly, the total energy dissipation (power $\times$ delay) of the circuits is improved.

5.6.2 Performance Comparisons

In order to evaluate the performance of the synthesized circuits, we performed post-synthesis simulations and processed the 128×128 Lena image using the Robert's cross circuits, a 128×128 noisy image using the median filter circuits, and 32 144×144 subsequent frames of the "Hall Monitor" test video sequence [113] using the KDE image segmentation circuits. For simulations with the Robert's cross circuits, image pixels were divided into groups of 16, 64, and 256 pixels, depending on the number of circuit inputs. Input pixels in each group were converted to stochastic signals and processed by the Robert's cross cells concurrently. To produce the output image, we measured the fraction of the time the circuits' output signals were high for 1024 cycles. The output image produced by each circuit was compared with a "Golden" output image produced by MATLAB and an average error rate was calculated as follows:

$$E = \frac{\sum_{i=1}^{128} \sum_{j=1}^{128} |T_{i,j} - S_{i,j}|}{255.(128 \times 128)} \times 100 \tag{5.3}$$

where $S_{i,j}$ is the expected pixel value in the perfect output image and $T_{i,j}$ is the pixel value produced using post-synthesis simulations including timing violations (setup and hold). The output images produced by post-synthesis simulation of the Robert's cross circuits are shown in Figure 5.14. The mean of the output error rates measured for

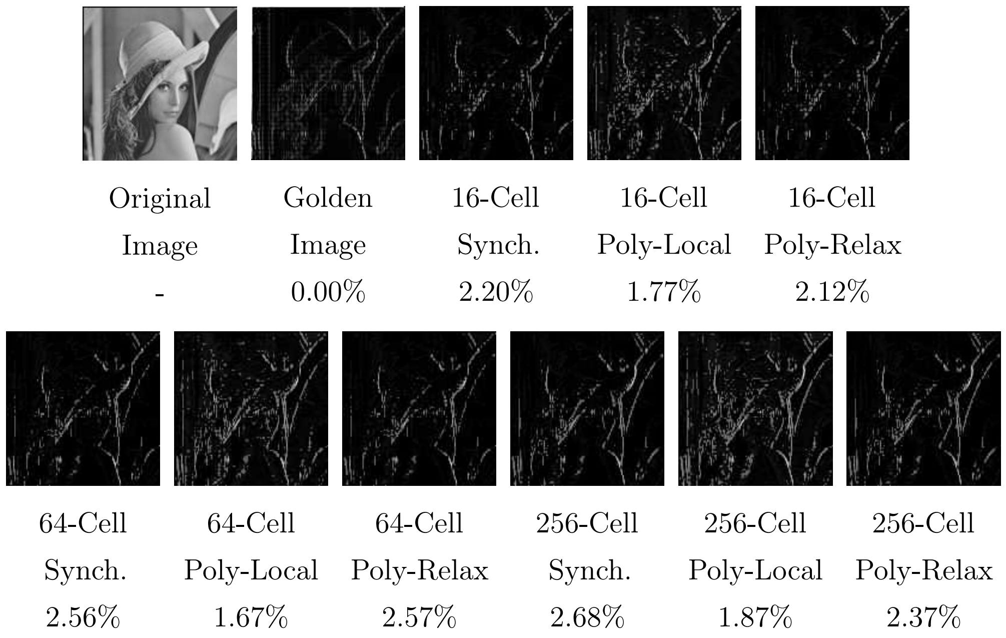
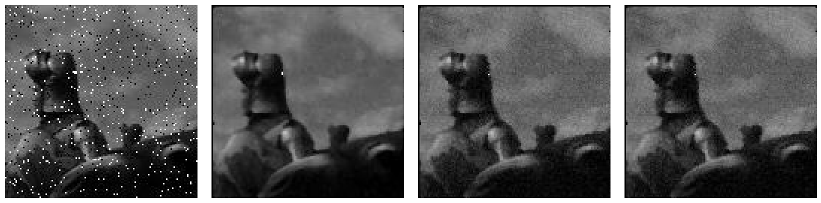


Figure 5.14: The original sample input and the output images produced by post-synthesis simulations of the synthesized Robert's cross circuits.

each circuit is also shown in Table 5.5. The outputs from processing the sample images using the median filter noise reduction and the KDE image segmentation circuits in the synchronous and polysynchronous versions of the circuits with a relaxed CDN are shown in Figure 5.15. As can be seen in these results, removing and relaxing the CDN not only has not decreased the quality of the results, but also in most cases has actually improved the average error rate of processing image pixels. This improvement in the quality of the results is mainly due to the additional randomness introduced by the polysynchronous clocking.

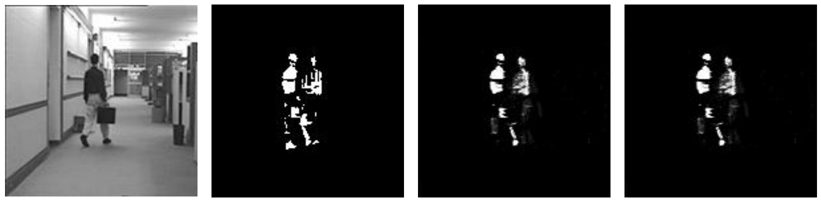
5.7 Error Analysis

There are several error sources in polysynchronous circuits. We analyze the effects of these error sources by first examining the computational errors inherent in stochastic circuits, and then examining errors that are unique to polysynchronous circuits.



Original Image	Golden 0.00 percent	Synch. 2.64 percent	Poly-Relax 2.62 percent

a) Median filter noise reduction



Original Image	Golden 0.00 percent	Synch. 1.70 percent	Poly-relax 1.69 percent

b) KDE image segmentation

Figure 5.15: The original sample inputs and the outputs of processing the sample images by post-synthesis simulations of the synthesized circuits with a relaxed CDN: a) Median filter noise reduction circuit, b) KDE image segmentation circuit.

5.7.1 Sources of Computational Errors

There are three main sources of computational errors in the conventional synchronous stochastic circuits [2]:

1. E_A = function approximation error. This error stems from the fact that we are computing a mathematical approximation of the desired function. For instance, the FSM-based stochastic absolute value function used in the Robert's cross circuit is an approximation of the desired absolute value function. The approximation error for such FSM-based functions depends on the number of states. The more states we use to implement the FSM, the smaller approximation error. Peng et al. [9] have reported 0.03 percent function approximation error for a 32-state FSM-based implementation of the

stochastic exponentiation function. The function approximation errors in the 16-state implementation of stochastic Abs function and the 32-state version of the stochastic tanh function are very close to zero.

2. E_Q = quantization error. In converting the input values in the interval [0, 1] or [-1, 1] into stochastic bitstreams, the SNG rounds the input value to the closest number in the set of discrete probabilities it can generate. Increasing the length of the bitstreams will reduce this quantization error [2].

3. E_R = errors due to random fluctuations. Errors due to random fluctuations are inherent in SC since the input values are intentionally randomized. The bitstreams can be described as a Bernoulli distribution and can be quantified using the variance of the distribution. Thus, these errors are inversely proportional to the square root of the length of the stream.

In addition to these errors, the polysynchronous clocking approach introduces two extra sources of error:

4. E_C = errors due to temporally misaligned bits in the streams. As the average error rate results presented in Table 5.5 show, temporal misalignment of inputs is an unbiased source of error that can either increase or decrease the mean of the total error in the polysynchronous circuits. We conclude from these results that, for polysynchronous clocking, the effect of temporally misaligned inputs on accuracy is, in fact, minimal.

5. E_S = errors due to stall time. When inputs to a component arrive at different times, the output will be invalid for a short time, called the "stall time." Reading the output during this short interval can reduce the accuracy of the computation. The error due to stall time will be discussed further in Section 5.7.3.

Summing all of these error sources, the total error for a polysynchronous circuit is no worse than:

$$E_{Total} = E_A + E_Q + E_R + E_C + E_S \tag{5.4}$$

Based on the error rate results presented in Table 5.5 and Figures 5.14 and 5.15, we conclude that removing or relaxing the CDN allows the maximum frequency of the circuit to be increased without affecting the accuracy of the computation compared to a conventional synchronous stochastic implementation of the circuits.

5.7.2 Metastability

In modern CMOS processes, the effects of metastability have become increasingly significant, especially in high-speed applications. Metastability is a phenomenon where a bi-stable element, such as a flip-flop, enters an undesirable third state in which the output is at an intermediate level between logic 0 and 1. A system's reliability is compromised when this occurs [114, 115]. An incorrect value might be sampled which would introduce an error in the computation. The effect of metastability can propagate to multiple registers and thereby get amplified. In conventional deterministic systems with multiple clock domains, each domain crossing represents a location where metastability could occur.

In SC circuits, however, metastability is not a major issue. The effect of metastability on the registers can be considered as a source of error that sometimes causes a change from 0 to 1 and sometimes 1 to 0. The important point is that these changes in the value of the signals have minimal effect on the numerical value represented by a long bitstream. On average they tend to cancel each other out, and will ultimately produce an acceptable total error. The experimental results that we showed for the polysynchronous implementation of complex stochastic circuits (i.e., the median filter noise reduction and the KDE image segmentation circuits) demonstrate that SC circuits are robust to the effects of metastability and propagated metastability, since these circuits average the signal value which then masks timing errors. We can consider the inaccuracy introduced by metastability as an error caused by temporally misaligned bits in the streams, or E_C, as discussed in Section 5.7.1.

5.7.3 Input to Output Synchronization

Assume we have a polysynchronous system processing a large set of inputs with a limited number of cells that work concurrently. The input source and so the input data for each cell changes periodically. For each new set of data, the input values must be converted to the corresponding stochastic signals and then transferred to the cells that require the new information. When neighboring cells work with polysynchronous clocks, there might be a very short time, called the "stall time", between the first and the last input signals arriving at the cells. For this short period of time, the output is believed to be invalid.

In a conventional binary system a synchronizer is required to deal with the stall time.

In a stochastic system, however, the designer can simply consider the output produced during this short time interval as a valid output. Comparing the stall time with the total processing time of each set of input data (e.g. 2 ns vs. 256×2 ns) allows the designer to start sampling (or measuring the fraction of high time) of the output signals immediately after first input arrives, or immediately after the input changes. Sampling the output during this small interval does not significantly influence the accuracy of the computation, given the nature of the stochastic representation. Eliminating the synchronizer circuitry further reduces the area overhead and design complexity.

5.8 Fault Tolerance of Polysynchronous Circuits

We compare the error tolerance of our polysynchronous stochastic circuit designs to conventional synchronous designs. To do so, we preformed trials on the circuits discussed in Section 5.2.2, randomly injecting soft errors, i.e., bit flips, on the internal signal lines and measuring the corresponding average output error rates.

For the synchronous circuits, the inputs were generated with SNGs driven by synchronized clocks each with a period of 2 ns. For the polysynchronous circuits, the inputs were generated by SNGs driven by clocks with periods varying randomly between 2 and 4 ns. Note that this range of values provides a variation of up to 100 percent in the clock periods. To approximate hardware conditions in which short pulses ("spikes") cannot satisfy the setup and hold time requirements of logic gates, high output pulses that were less than 10 percent of the 2 ns clock period (0.2 ns) were filtered out by setting them to zero.

Soft errors were simulated by independently flipping a given fraction of the input and output signals of each computing element. For example, a soft error rate of 20 percent means that 20 percent of the total bits in an input value are randomly chosen and flipped. To inject soft errors into a computational element such as a MUX, we insert XOR gates into all of its inputs and outputs. For each XOR gate, one of its inputs is connected to the original signal of the MUX and the other is connected to a global random soft error source, implemented using an LFSR and a comparator [2]. Note that we do not simultaneously inject soft errors on the input and output signals of any given component. Also, we do not inject soft errors more than once on the intermediate line between two

Table 5.6: The average error rate of the stochastic circuits for different soft error injection rates.

Circuit	Clocking Approach	Injection Rate			
		0 percent	5 percent	10 percent	20 percent
Robert's Cross	Synchronous	2.6	2.6	2.7	2.94
	polysynchronous	2.59	2.6	2.7	2.94
Median Filter	Synchronous	3.03	3.08	3.28	4.08
	polysynchronous	3.13	3.08	3.22	4.04
KDE	Synchronous	1.21	1.26	1.62	2.84
	polysynchronous	1.24	1.40	1.67	2.93

components (thereby potentially undoing a bit flip).

We apply this approach to all of the basic computational elements of the stochastic circuits. Hardware simulations were performed using the ModelSim hardware simulator [116]. Maximal period 32-bit LFSRs were used for converting input pixel values into stochastic bitstreams. Bitstreams of length 1024 were used to represent the values. The processing time, however, is determined by the longest clock period among the SNGs that generate inputs to the circuit. Thus, for inputs with shorter clock periods, longer streams are required compared to those with longer periods. Ten trials were performed for each case to ensure statistically significant results. For each trial we used a different initial condition with ten different LFSR seed values for each SNG. Simultaneously, ten different sets of values for the periods of the polysynchronous clocks were used. We present the average results of these trials.

The sample images shown in Section 5.6.2 were used as the inputs to the circuits. Table 5.6 shows the average output error rates of the two design approaches under different soft error injection rates. As can be seen, the polysynchronous stochastic circuits are as error tolerant as the synchronous versions. For both polysynchronous and synchronous circuits, the error tolerance scales gracefully to very large numbers of errors. Note that, while we presented the error-tolerance results for a frequency variation of 100%, the circuits will gracefully tolerate errors for frequency variations beyond 100% if the inputs are processed for a long enough time (e.g., 1,024 times the largest period).

5.9 Related Work and Discussion

Asynchronous design methodologies have been studied for decades [117],[118]. Instead of synchronizing transitions with a global clock, asynchronous systems are organized as a set of components which communicate using handshaking mechanisms. The drawback of asynchronous methodologies is the overhead required for the handshaking mechanisms.

Circuits with multiple independent clock domains, dubbed “globally asynchronous locally synchronous” (GALS), have been widely studied [119]. GALS architectures consume less dynamic power and can achieve better performance than architectures with a single clock domain [120, 121]. However, the circuitry for domain crossings is complex and problematic. Techniques such as stretching [119][122] and pausing the clocks [120] have been proposed. Nevertheless, the circuitry for the handshaking needed at domain crossings is costly. Consequently, the splitting typically is only performed at a coarse level.

Asynchronous and GALS design methodologies are applicable to both SC and conventional designs. The paradigm advocated in this chapter, however, is only applicable to SC systems and differs from the asynchronous and GALS approaches in that no complex handshaking mechanisms are needed. The skew tolerance provided by SC allows independent clock domains to be connected together seamlessly without influencing the accuracy. Alternatively, it allows for a much less costly global CDN, with relaxed constraints. This, in turn, provides very significant benefits in terms of area, performance and energy. The increase in performance, in particular, can be quite significant. For applications that require modest accuracy, this increase in performance could more than offset the latency incurred by adopting a stochastic representation.

High energy dissipation is one of the main challenges in the practical use of SC [123]. Stochastic circuits are compact and so consume little power. However, given the high latency, the energy consumption (which is power multiplied by time) is high. In recent work, Alaghi et al [124] proposed energy reduction techniques for SC. Theses techniques exploit the tolerance that SC offers to timing errors. This permits very aggressive voltage scaling without significant quality degradation. Their simulation results show that SC circuits can tolerate aggressive voltage scaling with no significant SNR degradation after 40% supply voltage reduction (1V to 0.6V), leading to 66% energy saving. Similarly, a

100% frequency boosting of the optimized circuits leads to no significant SNR degradation for several representative circuits.

The approach of Alaghi et al. is conceptually similar and complementary to the one that we propose in this chapter. The impact of timing errors due to voltage scaling is similar to the impact of clock skew errors. In both cases, SC naturally and effectively provides error tolerance. To our knowledge, the work in this chapter and the work of Alaghi et al. [124] are the first to introduce and exploit the skew tolerance advantage of SC circuits. This work focuses on optimizing CDNs while the work of Alaghi et al. studies the effects of voltage and frequency scaling.

5.10 Summary

In this chapter, we proposed polysynchronous clocking, a design strategy for exploiting the skew tolerance of SC circuits. We showed that, from basic stochastic operations, such as multiplication and scaled addition, to complex stochastic circuits, the correct output is computed even when the inputs are not synchronized. We explored two approaches of polysynchronous system design to mitigate the costs of the CDNs. In the first approach, we removed the global CDN and instead used locally generated clocks to design the Robert's cross stochastic system. Quantifying the costs and benefits, the maximum working frequency, the area, and the energy consumption improved by up to 12x, 39 percent, and 23 percent, respectively, for the Robert's cross system with 256 cells. For smaller systems, the area and energy overhead incurred by the local clock generators diminished the benefits of removing the CDN.

Experimental results showed that, for small scale stochastic circuits such as the Robert's cross circuits with 16 cells, the median filter noise reduction circuit, and the kernel density estimation based image segmentation circuit, relaxing the CDN is a more efficient choice. The area, speed, and energy are all improved by a relaxed CDN. Post-synthesis simulations on sample images showed that removing and relaxing the CDN not only did not degrade the quality of the output, but in some cases it actually improved the accuracy of results by introducing additional randomness. We showed that circuits designed with either of these polysynchronous approaches are as tolerant of errors as conventional synchronous stochastic circuits.

Chapter 6

Seamless Memory Design for SC

Due to the difference in data representation, integrating conventional memory (designed and optimized for non-stochastic computing) in SC systems inevitably incurs a significant data conversion overhead. This chapter presents a seamless stochastic sytem, StochMem, which features analog memory to trade the energy and area overhead of data conversion for computation accuracy. We compare the proposed system with a beseline near-sensor stochastic image processor featuring conventional digital memory. We evaluate the performance of the proposed design on five representative image processing applications. This chapter's material is taken from [125].

6.1 Motivation

The common focus of SC proposals from 1960s onwards has been stochastic logic (arithmetic), neglecting memory, which represents a crucial system component. Memory mainly serves as a repository for data collected from external resources (e.g., sensors) or data generated by previous steps of computation, to be used at later stages of computation. Algorithmic characteristics dictate both, the memory capacity requirement and the memory access pattern (particularly for data re-use). Most SC proposals deploy conventional digital memories (designed and optimized for non-stochastic computing) to address such algorithmic needs. Unfortunately, this practice increases hardware design complexity due to the discrepancy in conventional digital (i.e., non-stochastic) and stochastic data representations. Digital to/from stochastic data conversion can reach 80% or more of the overall energy consumption and hardware cost, which can easily diminish any benefit from SC [1, 2]. *In this chapter, we rethink the memory system design for stochastic computing.*

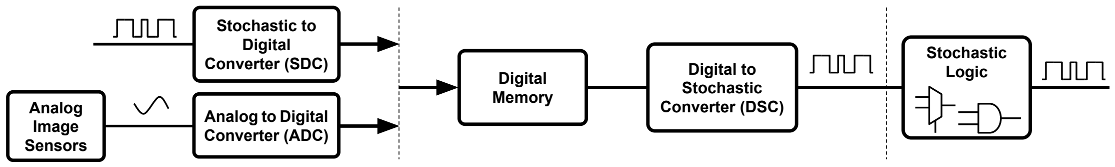
Practically *seamless* conversion options between analog and stochastic data representations [126, 127] makes analog memory stand out as a particularly promising point in the memory design space for SC. The downside is potential loss in data accuracy, where a divergence between the written/stored and the read values (at the same memory address) often becomes inevitable, however, which stochastic logic can mask due to its implicit tolerance to inaccuracy in input data operands.

This chapter quantitatively characterizes the potential of analog memory for seamless SC, using a representative near-sensor stochastic image processing system as a case study. Non-stochastic, analog near-sensor image processing accelerators such as [128] exist. The focus of this chapter is not design and exploration of image processing accelerators. The scope rather is memory system design for SC where we use a representative stochastic system to characterize the impact of memory. We will refer to the resulting (practically) seamless stochastic system as StochMem.

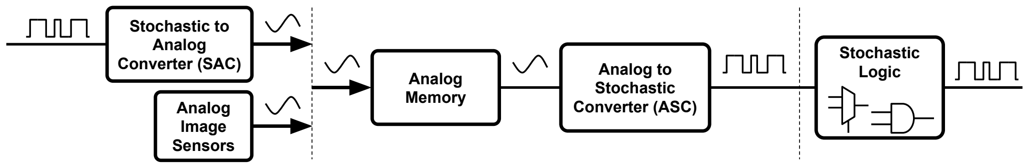
Cameras have already become ubiquitous sensors. There is a demand for near-sensor image processing both to reduce costly communication with the cloud and to enhance security and privacy. Real-time image processing algorithms often track differences between a stream of frames. It is not uncommon that the processing of the instantaneous frame requires comparison to a history of previously processed frames, which has to be stored in and retrieved from some form of memory. In this chapter, we will cover five representative image processing applications which span diverse compute and memory access characteristics.

6.2 Toward Seamless SC

We will first compare and contrast StochMem featuring analog memory with the corresponding stochastic near-sensor image processor featuring conventional digital memory as a representative baseline.


Stochastic to Digital Converter (SDC)
Analog Image Sensors
Analog to Digital Converter (ADC)
Digital Memory
Digital to Stochastic Converter (DSC)
Stochastic Logic

(a) Baseline Near-Sensor Stochastic Image Processor featuring Digital Memory


Stochastic to Analog Converter (SAC)
Analog Image Sensors
Analog Memory
Analog to Stochastic Converter (ASC)
Stochastic Logic

(b) StochMem featuring Analog Memory

Figure 6.1: Baseline Near-Sensor Stochastic Image Processor vs. StochMem.

6.2.1 Baseline: Stochastic Logic + Conventional Memory

Fig. 6.1a provides an overview for the baseline stochastic near-sensor image processor featuring conventional digital memory (designed and optimized for non-stochastic computing). The input data operands may represent the result bitstreams of previous steps of (stochastic) computation, or may directly come from analog image sensors. To be able to store such input data in conventional digital memory, a *Stochastic to Digital Converter*, SDC (for stochastic input bitstreams) or an *Analog to Digital Converter*, ADC (for analog inputs coming from sensors) become necessary. Moreover, further (stochastic) processing of the stored data necessitates a *Digital to Stochastic Converter*, DSC, upon data retrieval from digital memory. In the following we briefly describe key system components.

Stochastic Logic incorporates a circuit of basic Boolean gates to carry out the application-specific stochastic computation (Section 6.3.2). The inputs and outputs are both stochastic bitstreams.

Stochastic to Digital Converter (SDC) can generate the conventional binary representation for any stochastic bitstream. A digital counter usually serves the purpose, by keeping track of the number of 1s in the input bitstream to be converted. An SDC carries out data conversion if the inputs to the stochastic system represent result bitstreams from previous steps of (stochastic) computation.

Analog to Digital Converter (ADC) becomes necessary if the inputs to the stochastic system directly come from analog image sensors. Conventional ADCs can serve the purpose. For most applications of SC (including the case study in this chapter) an 8 to 10-bit ADC is sufficient [4].

Digital to Stochastic Converter (DSC) transforms conventional binary data retrieved from digital memory (for further stochastic processing) to stochastic bitstreams. Commonly, DSC achieves this by comparing an unbiased random number (obtained from a random number generator) to the binary value to be converted. A one is attached to the output (stochastic) bitstream if the random number is less than the binary value (to be converted); zero, otherwise. The random number generator can rely on physical random sources or pseudo-random constructs such as LFSRs.

6.2.2 StochMem: Stochastic Logic + Analog Memory

The data converters (SDC or ADC and DSC) incorporated into the baseline stochastic system from Fig. 6.1a each has a significant energy and area footprint [1], which can easily nullify potential benefits from SC. In order to reduce this overhead, StochMem replaces the conventional digital memory with its analog counterpart. Fig. 6.1b provides the overview for the resulting SC system. In the following, we briefly describe key StochMem components:

Stochastic Logic is the same as under the baseline system.

Stochastic to Analog Converter (SAC) replaces the SDC of the conventional system. SAC can generate the analog representation for any stochastic bitstream. A conventional analog integrator can serve the purpose, by measuring the fraction of time a stochastic input bitstream stays at logic 1. Such an integrator usually has a smaller energy and area footprint than the SDC of the baseline system (Section 6.3.3). A SAC carries out data conversion if the inputs to StochMem represent result bitstreams from previous steps of (stochastic) computation.

Analog to Stochastic Converter (ASC) transforms data from analog memory (for further stochastic processing) to stochastic bitstreams, similar to the DSC of the conventional system. As representative examples, [126, 127] both cover energy-efficient ways for generating stochastic bitstreams from analog inputs.

6.3 Evaluation Setup

6.3.1 System Design

We evaluate three stochastic near-sensor image-processing designs: two different implementations of the baseline from Fig. 6.1a ($Conv_{LFSR}$ and $Conv_{MTJ}$) and StochMem. The two baseline designs differ in the implementation of data converters as follows:
$Conv_{LFSR}$: The baseline SC system featuring a 10-bit LFSR and a comparator as the DSC unit.
$Conv_{MTJ}$: The baseline featuring a DAC followed by an MTJ-based ASC as a more energy-efficient DSC. The rest of the system is identical to $Conv_{LFSR}$.

All systems first store the input in the memory. Then, they convert it to stochastic, and feed it to the stochastic logic.

6.3.2 Stochastic Applications

To evaluate *Stochastic Logic* from Fig. 6.1, we use stochastic circuits of five representative image processing applications: *Robert* (Robert's cross edge detection), *Median* (median filter noise reduction), *Frame* (frame difference-based image segmentation) from [4]; *Gamma* (gamma correction) from [2]; and *KDE* (kernel density estimation-based image segmentation) from [10].

As input, we use 128×128 gray-scale images for *Robert, Median, Frame*, and *Gamma*;

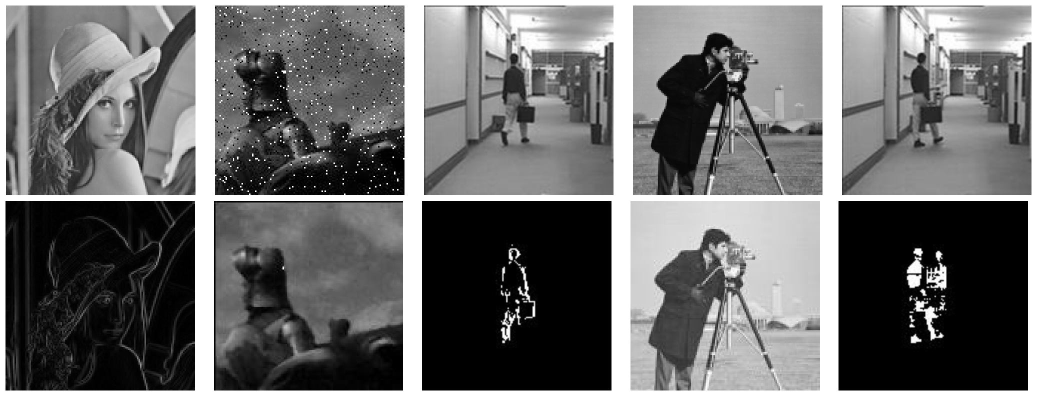

(a) *Robert* (b) *Median* (c) *Frame* (d) *Gamma* (e) *KDE*

Figure 6.2: Input (expected output) per application on top (bottom).

and 33 recent frames of a video, for *KDE*. Fig. 6.2 shows the input (expected output) images used for each application on the top (bottom) row. Expected output captures the maximum-possible accuracy. To calculate the accuracy of the end results, we calculate the average pixel-by-pixel difference between the output image of each stochastic circuit and the corresponding maximum-possible-accuracy output.

6.3.3 Hardware Parameters

Table 6.1 summarizes the area and energy consumption of different units of the evaluated stochastic systems. We synthesize logic units (including the stochastic circuit implementations of the five benchmark applications from Section 6.3.2), LFSR, digital comparator, and counter units using Synopsys Design Compiler vH2013.12 with a 45-nm gate library. The Floating-Gate (FG) analog memory implementation follows [129]. To model inaccuracy of FG memory, we add Gaussian noise (with standard deviation from measured data in [129]) to the stored data.

For a fair evaluation, we assume that the input to both the baseline designs and StochMem directly comes from analog image sensors. All designs output a stochastic bitstream. Therefore, the evaluated systems do not feature an SDC or SAC on the feedback path from memory (Fig. 6.1). However, we include these units in Table 6.1 for the sake of completeness. SAC area (energy) cost is 2.3× (17.9×) less than SDC. Accordingly, if the evaluated systems deployed these units (as explained in Section 6.2), StochMem would have shown even larger gains when compared to the baseline.

6.4 Evaluation

Since all three alternative designs operate at the same frequency, they have similar throughput. So, we start the evaluation with a quantitative characterization of the accuracy loss in the outputs due to the potential read-write discrepancy of the analog memory incorporated in StochMem. We continue with energy consumption and conclude with area cost.

Table 6.1: Area and energy breakdown.

Stochastic Logic		
Circuit	Area (um^2)	Energy (pJ)(@1GHz)
Robert	339	0.440
Median	5382	4.090
Frame	457	0.413
Gamma	76	0.042
KDE	8691	7.094
Baseline System Parameters		
Unit	Area (um^2)	Energy (pJ)(@1GHz)
ADC 10-bit [55, 130]	50,000	20
SRAM cell	0.35	10
DSC: 10-bit LFSR	194	0.355
DSC: 10-bit Comparator	96	0.041
DSC: DAC 8-bit [131]	16,000	64
SDC: 10-bit Counter	254	0.179
StochMem System Parameters		
Unit	Area (um^2)	Energy (pJ)(@1GHz)
Analog memory cell [129]	58.7	10 (RD) / 100 (WR)
ASC [127]	15	0.030
SAC (integrator)	110	0.010

6.4.1 Output Accuracy of StochMem

A known downside of analog memory technologies is the potential discrepancy between values read and written/stored. We model the impact of this discrepancy after the accuracy measurements of a representative analog memory implementation [129]. All evaluated benchmark applications produce images as output. Therefore, we capture the accuracy loss in the output by the average per-pixel deviation (and SSIM [132]) from the "expected" output for each application as shown in the bottom row of Fig. 6.2.

Fig. 6.3 demonstrates the % output inaccuracy (in terms of average per-pixel deviation) of StochMem and the baseline designs for all applications under a stochastic bitstream length of 1024. The y-axis is normalized to the expected accuracy values corresponding to the images in the bottom row of Fig. 6.2. The two baseline designs evaluated, $Conv_{LFSR}$ and $Conv_{MTJ}$ (Section 6.3.1), feature the very same output inaccuracy, as given by the

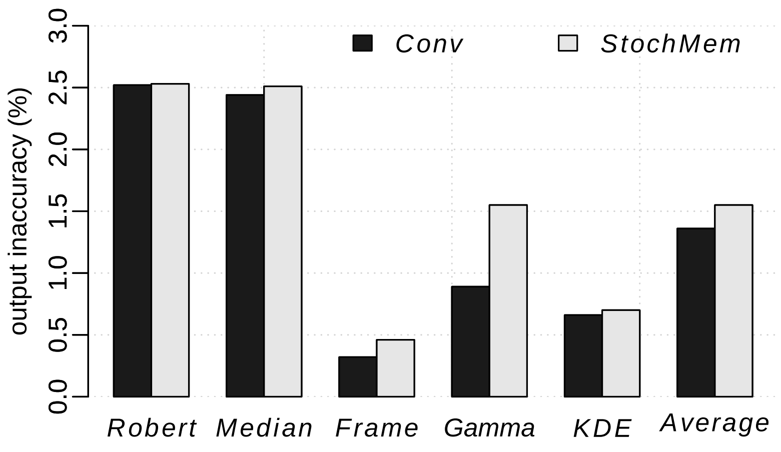


Figure 6.3: Output inaccuracy of the baseline vs. StochMem.

Conv bar in Fig. 6.3. We observe that, overall, the degradation (with respect to *Conv*) in the output accuracy of StochMem remains negligible. Only for *Gamma*, the inaccuracy becomes around 0.7% worse than *Conv*. For all other applications, the inaccuracy worsens by less than 0.15%. On average, the % output inaccuracy of StochMem is 1.55%; of *Conv*, 1.36%, with respect to the expected outputs. Besides, on average SSIM gets 3.2% worse for different applications. For the worst-case application, *Gamma*, SSIM gets 7.3% worse than *Conv*.

Fig. 6.4 tabulates the output images for all benchmark applications under StochMem and *Conv*. In accordance with the comparison results from Fig. 6.3, the difference in output accuracy is barely perceivable.

We repeat these experiments for three different bitstream lengths: 128, 256, and 512 bits. The average output inaccuracy of StochMem with respect to *Conv* increases from 4.08% to 4.21%, from 2.63% to 2.77%, and from 1.87% to 2.03%, as the bitstream length increases from 128 to 512, respectively. The relatively small degradation in the output inaccuracy is in line with the experimental outcomes summarized in Fig.s 6.3 and 6.4.

6.4.2 Reduction in Energy Consumption

We next compare and contrast the energy consumption of the evaluated stochastic designs. In the following, we report the experimental results for a bitstream length of 1,024 without loss of generality. As Fig. 6.5 depicts, due to its more energy-efficient DSC implementation,

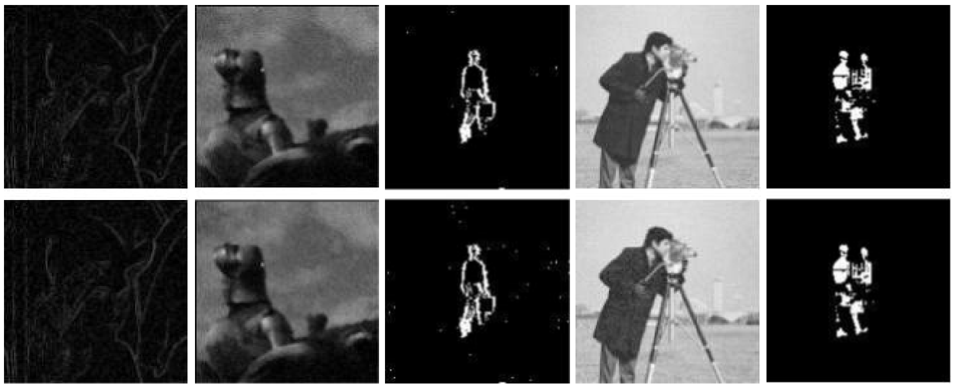

(a) *Robert* (b) *Median* (c) *Frame* (d) *Gamma* (e) *KDE*

Figure 6.4: Output images: Baseline (StochMem) on top (bottom).

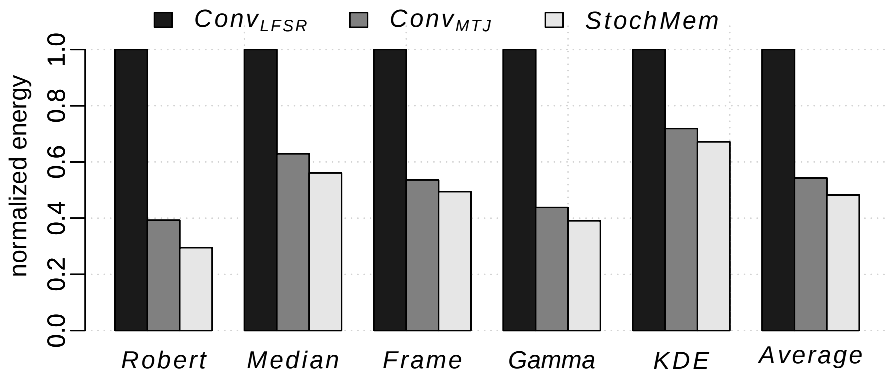
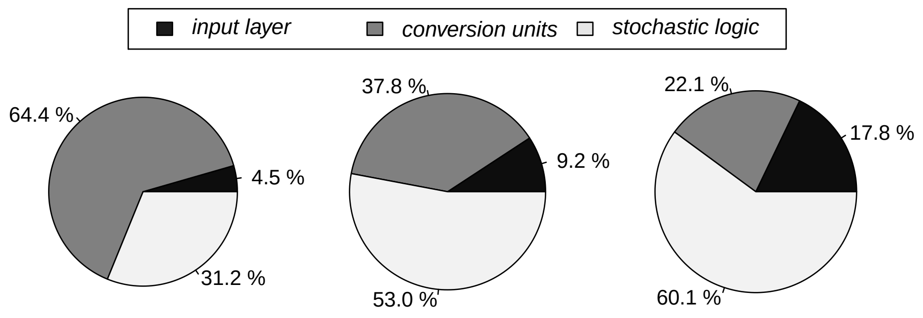


Figure 6.5: Energy consumption normalized to $Conv_{LFSR}$.

$Conv_{MTJ}$ can decrease the energy consumption with respect to $Conv_{LFSR}$ significantly, by 45.7% on average. Introducing analog memory– i.e., StochMem – can reduce the energy consumption further, by 11.1% on average over $Conv_{MTJ}$.

To demonstrate the sources of these energy gains, we quantify the share of energy spent in different units. We expect an energy-efficient stochastic system to spend most of its energy budget on computation, rather than on data conversion and input operand retrieval. Pie charts from Fig. 6.6 differentiate between the shares of energy spent in the *input layer* (which covers the input operand retrieval and hence constitutes the ADC, if applicable, and memory units); in the *conversion units* (which constitute the ASC or DSC); and in the *stochastic logic* (which captures the actual computation).



(a) $Conv_{LFSR}$ (b) $Conv_{MTJ}$ (c) StochMem

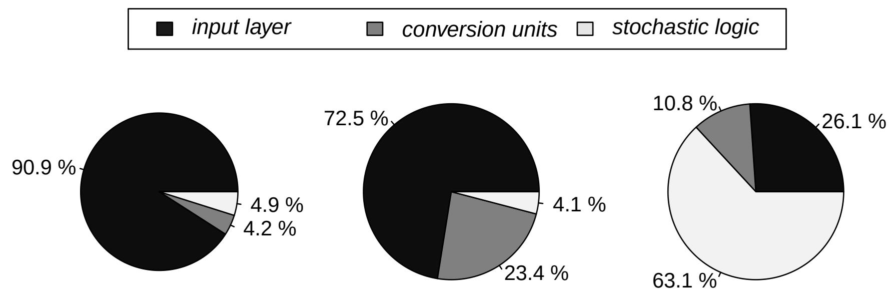
Figure 6.6: Share of energy consumed by different units.

Figures 6.6a, 6.6b, and 6.6c, show the shares for $Conv_{LFSR}$, $Conv_{MTJ}$, and StochMem separately (Section 6.3.1). As the charts reveal, share of *stochastic logic* (*conversion units*) increases (decreases) from 31.2% (64.4%) to 53.0% (37.8%) and to 60.1% (22.1%), as we move from $Conv_{LFSR}$ to $Conv_{MTJ}$ and to StochMem , respectively. StochMem represents the most energy efficient design, featuring the lowest (highest) energy share for data conversion (computation), when compared to $Conv_{LFSR}$ and $Conv_{MTJ}$.

6.4.3 Reduction in Area

In this section, we evaluate the area cost of each alternative. Since tailoring ADC and DAC units to each application was out of the scope of this study, for the baselines (i.e., $Conv_{LFSR}$ and $Conv_{MTJ}$) we deploy an ADC and a DAC unit of minimal area (which represents the hypothetical best-case in terms of area cost), even if these units fail short of providing the required precision. Accordingly, if we were to incorporate realistic ADC or DAC units (which would likely incur a much higher area overhead), StochMem (which does not employ any ADC or DAC) would have shown even larger area savings in comparison to the baseline.

Table 6.2 summarizes the area cost for the evaluated stochastic designs (columns) for the stochastic benchmark applications (rows). While $Conv_{MTJ}$ consumes notably less energy than $Conv_{LFSR}$ (Section 6.4.2), it requires an extra DAC which increases the area overhead (with respect to $Conv_{LFSR}$) by 20.0% on average. On the other hand, StochMem can cut the area cost significantly, by about 93.7% (with respect to $Conv_{LFSR}$)



(a) $Conv_{LFSR}$ (b) $Conv_{MTJ}$ (c) StochMem

Figure 6.7: Pie-charts demonstrating share of hardware cost (in terms of area) across different units.

on average, by eliminating the need for costly conversion units. Only StochMem can deliver area and energy benefits at the same time.

Fig. 6.7 depicts a detailed break-down of area consumption among different units. Similar to Fig. 6.6, pie charts from Fig. 6.7 differentiate between the shares of area in the *input layer*, *conversion units*, and *stochastic logic*, respectively. Only 4.9% of the area in $Conv_{LFSR}$ goes to the *stochastic logic*, while the *input layer* consumes 90.9%. *Stochastic logic* in $Conv_{MTJ}$ has even a smaller share of area (4.1%) when compared to $Conv_{LFSR}$. On the other hand, in StochMem, 63.1% of the area goes to *stochastic logic*; only 10.8%, to *conversion units*.

Data conversion in conventional SC systems necessitates high-overhead units such as LFSRs+comparators, ADCs, or DACs. StochMem-like SC systems, on the other

Table 6.2: Area in μm^2.

Apps	Logic	$\boldsymbol{Conv_{LFSR}}$				$\boldsymbol{Conv_{MTJ}}$					***StochMem***		
		Memory	ADC	DSC	Total	Memory	ADC	DAC	ASC	Total	Memory	ASC	Total
Robert	339	21		1450	51810	21			75	66435	183	75	597
Median	5382	38		2900	58320	38			150	71570	336	150	5868
Frame	457	17	50000	772	51246	17	50000	16000	60	66534	153	60	670
Gamma	76	35		1156	51267	35			120	66231	306	120	502
KDE	8691	122		6166	64979	122			630	75443	1071	630	10392

hand, can eliminate or replace these units with lighter-weight counterparts leading to substantial energy and area savings.

6.5 Summary

A challenging artifact of modern technology scaling is growing uncertainty in design parameters, and therefore, in design functionality. This renders SC a particularly promising paradigm, which represents and processes information as quantized probabilities. Numerous SC proposals from 1960s onwards, however, focus on stochastic logic (mainly arithmetic), neglecting memory. Unfortunately, deploying conventional (digital) memory in a stochastic system is particularly inefficient due to the difference in data representations, which can easily incur a significant data conversion overhead.

In this chapter, we proposed a seamless memory system design for SC systems to minimize the data conversion overhead, which can reach 80% of overall hardware cost, considering image processing as a case study. Analog memory is particularly promising due to seamless conversion options between analog and stochastic data representations, despite the potential loss in data accuracy which stochastic logic can easily mask due to its implicit fault-tolerance. We evaluated analog memory for seamless SC, using a representative stochastic near-sensor image processing system as a case study. We demonstrated how such a system can reduce energy consumption and area cost by up to 52.8% and 93.7%, while keeping the accuracy loss as incurred by analog memory below 0.7%.

Chapter 7

Concluding Remarks

7.1 Summary of Contributions

In this dissertation, we first provided background information on stochastic computing (SC) including different encoding schemes, advantages, and weaknesses of this unconventional computing paradigm.

Second, we explored an evolution of the concept of SC and proposed a highly unorthodox idea: performing computation with digital constructs on time-encoded analog signals. Instead of encoding data in space, as random bitstreams, we encoded values in time. We showed how analog periodic pulse signals can be used in performing essential stochastic operations. The approach is an excellent fit for low-power applications that include time-based sensors, for instance image processing circuits in vision chips. Implementation results on image processing applications showed up to a 99% performance speedup, 98% saving in energy consumption, and 40% area reduction compared to prior stochastic implementations [49, 48, 133, 18, 50].

Third, we proposed a novel area- and power-efficient synthesis approach for implementation of sorting network circuits based on unary bitstreams. The proposed method inherits the fault tolerance and low-cost design advantages of processing random stochastic bitstreams while producing completely accurate result. Synthesis results of complete sorting networks showed up to 90% area and power saving compared to the conventional binary implementations. However, the latency increased. To mitigate the increased latency, we used our developed time-encoding method. The approach was validated by implementing a low-cost, high-performance, and energy-efficient implementation of an important application of sorting, median filtering [69, 70].

Fourth, we proposed high-quality down-sampling methods to solve an important issue with the recently developed deterministic methods of SC. Relatively prime stream length, clock division, and rotation of bitstreams are the three deterministic methods of processing bitstreams that are initially proposed based on unary bitstreams. For applications that slight inaccuracy is acceptable, these unary stream-based approaches must run for a relatively long time to produce acceptable results. This long processing time makes these approaches energy-inefficient compared to the conventional random stream-based SC. We exploited pseudo-randomness and proposed new LFSR-based structures to improve the progressive precision property of these deterministic methods. We also proposed two new deterministic methods of processing bitstreams based on low-discrepancy sequences. Significant improvement in the processing time and energy consumption is observed using the proposed structures [99, 98, 102, 100, 101].

Fifth, we demonstrated that computation on stochastic bitstreams has another compelling advantage: circuits naturally and effectively tolerate very high clock skew. Exploiting this advantage, we investigated Polysynchronous Clocking, a design strategy for optimizing the clock distribution networks of SC systems. Clock domains are split at a very fine level, reducing power on an otherwise large global clock tree. Each domain is synchronized by an inexpensive local clock. Alternatively, the skew requirements for a global clock tree network can be relaxed. The proposed design approach allows for a higher working frequency and so lower latency. It also results in significant area and energy savings for a wide variety of applications [104, 17, 105].

Finally, we integrated analog memory with conventional stochastic systems to reduce the energy wasted in conversion units. We proposed a seamless stochastic system, StochMem, which features analog memory to trade the energy and area overhead of data conversion for computation accuracy. Comparing to a baseline system which features conventional digital memory, StochMem can reduce the energy and area significantly at the cost of slight loss in computation accuracy [125].

Beside the topics discussed in this dissertation, we developed a low-cost SC-based hardware implementation of a large Restricted Boltzmann Machine (RBM) Classifier completely on a single FPGA [32, 33]. Conventional binary implementation of a fully parallel design of a large neural network is expensive, involves extra design overheads, and in most cases cannot be fit on a single FPGA. We also developed a new reconfigurable architecture and methodology for synthesizing any given target function stochastically using FSMs [16]. When the target function is relatively complex, such as the exponentiation, the hyperbolic tangent, or high-order polynomial functions, our developed sequential logic-based implementation is more efficient than the prior combinational architectures. Our synthesis method also has the ability of implementing multi-input functions at a very low cost. Compared to prior combinational logic-based approaches, the proposed reconfigurable architecture can save hardware area and energy consumption by up to 30% and 40%, respectively, while achieving a higher processing speed.

7.2 Future Directions

The proposed design methodologies of this dissertation can be used in responding to the high-demand request for implementing ultra-low-power signal processing systems and energy-efficient real-time machine learning systems.

With the growth in image and video processing systems (e.g. mobile cameras, biomedical imaging, robotics), speech and voice recognition systems, and in general many sensor-based signal processing systems that can tolerate small rates of inaccuracy, developing ultra-low power systems using unconventional paradigms such as SC has become a research area of substantial interest. Optimizing and improving the speed and energy consumption of signal processing systems by converting these systems from a pure-digital or pure-analog design to efficient mixed-signal designs using novel approaches such as the proposed time-based design is an interesting direction to proceed. Studying the feasibility of using the proposed methods in designing systems that work with extreme power budget constraints, developing vertically-integrated high-performance signal processing chips based on the proposed methods, and evaluating the impact of applying both of the conventional energy reduction techniques (e.g., voltage and frequency scaling) and the developed energy-optimization methods (e.g., polysynchronous clocking) in energy-efficient design of image, video, and speech processing systems are other important future directions.

In recent years, machine learning has been used by almost all high-technology companies in developing intelligent systems. Data security, healthcare and medical diagnosis, search engines, smart cars, bioinformatics, computer vision and object recognition, speech and handwritten recognition, recommender systems, and translation systems are only a subset of applications of machine learning. Low-cost energy-efficient hardware implementation of machine learning algorithms has been an attractive and high demand research area in recent years. High computational complexity, however, makes the hardware implementation expensive, energy inefficient, and in many cases impractical with limited hardware resources. SC has been used for low-cost implementation of machine learning algorithms [31, 33, 37, 35, 34, 38, 40, 134, 135, 39]. A higher latency and energy consumption, and a lower output quality compared to the conventional fixed-point binary implementations are still the main barriers in wide application of SC-based machine learning systems. Recent progress in the SC field, from mixed-signal time-based encoding to deterministic processing of bitstreams, has raised new hope to solve the important challenges inefficient design of these systems. The proposed methodologies of this dissertation can particularly have a significant impact in the design of near-sensor neural network accelerators.

An important part of future efforts could be to lay the theoretical foundations for the developed deterministic methods of SC including the time-based SC to provide bounds on the accuracy of computations. From conventional digital stochastic bitstreams to the time-based encoding, we can perform rigorous mathematical analysis. We intend to examine issues such as the interaction between signals with different frequencies; limits on the classes of functions that can be synthesized using a given set of gates; and the effect of truncating periodic signals. Number-theoretic techniques can be employed to optimize designs, for instance when generating coefficients for polynomial approximations. Solving the current challenges of the developed time-based computing such as resolution limitation, truncation error, difficulty of synchronization, skew propagation, and translating the time-encoded signals for the FSM-based stochastic circuits is another interesting future direction.

References

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

[2] Weikang Qian, Xin Li, M.D. Riedel, K. Bazargan, and D.J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *Computers, IEEE Trans. on*, 60(1):93–105, Jan 2011.

[3] A. Farmahini-Farahani, H. J. Duwe III, M. J. Schulte, and K. Compton. Modular design of high-throughput, low-latency sorting units. *IEEE Transactions on Computers*, 62(7):1389–1402, July 2013.

[4] 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.

[5] Jesse Scott. Analysis of two-dimensional median filter hardware implementations for real-time video denoising. MS thesis, Penn State University, December 2010.

[6] A. Alaghi and J.P. Hayes. Fast and accurate computation using stochastic circuits. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–4, March 2014.

[7] S. Liu and J. Han. Energy efficient stochastic computing with sobol sequences. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 650–653, March 2017.

[8] I. L. Dalal, D. Stefan, and J. Harwayne-Gidansky. Low discrepancy sequences for monte carlo simulations on reconfigurable platforms. In *2008 International Conference on Application-Specific Systems, Architectures and Processors*, pages 108–113, July 2008.

[9] Peng Li, D.J. Lilja, W. Qian, M.D. Riedel, and K. Bazargan. Logical computation on stochastic bit streams with linear finite-state machines. *Computers, IEEE Transactions on*, 63(6):1474–1486, June 2014.

[10] Peng Li and D.J. Lilja. A low power fault-tolerance architecture for the kernel density estimation based image segmentation algorithm. In *Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on*, pages 161–168, Sept 2011.

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

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

[13] W. J. Poppelbaum, C. Afuso, and J. W. Esch. Stochastic computing elements and systems. In *Proceedings of the Joint Computer Conference*, AFIPS '67 (Fall), pages 635–644, New York, NY, USA, 1967. ACM.

[14] Armin Alaghi and John P. Hayes. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):92:1–92:19, 2013.

[15] Armin Alaghi, Weikang Qian, and John P Hayes. The promise and challenge of stochastic computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[16] 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.

[17] M. Hassan Najafi, David J. Lilja, Marc D. Riedel, and Kia Bazargan. Polysynchronous clocking: Exploiting the skew tolerance of stochastic circuits. *IEEE Transactions on Computers*, 66(10):1734–1746, Oct 2017.

[18] M. Hassan Najafi, Shiva Jamali-Zavareh, David J. Lilja, Marc D. Riedel, Kia Bazargan, and Ramesh Harjani. An Overview of Time-Based Computing with Stochastic Constructs. *IEEE Micro*, 37(6):62–71, November 2017.

[19] K. Parhi and Y. Liu. Computing arithmetic functions using stochastic logic by series expansion. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2017.

[20] D. Fick, G. Kim, A. Wang, D. Blaauw, and D. Sylvester. Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2d edge detection and noise filtering. In *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, pages 1–4, Sept 2014.

[21] M. Hassan Najafi and Mostafa E. Salehi. A Fast Fault-Tolerant Architecture for Sauvola Local Image Thresholding Algorithm Using Stochastic Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(2):808–812, Feb 2016.

[22] N. Onizawa, D. Katagiri, K. Matsumiya, W. J. Gross, and T. Hanyu. Gabor filter based on stochastic computation. *IEEE Signal Processing Letters*, 22(9):1224–1228, Sept 2015.

[23] Y. Liu and K. K. Parhi. Architectures for recursive digital filters using stochastic computing. *IEEE Transactions on Signal Processing*, 64(14):3705–3718, July 2016.

[24] H. Ichihara, T. Sugino, S. Ishii, T. Iwagaki, and T. Inoue. Compact and Accurate Digital Filters Based on Stochastic Computing. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2017.

[25] Yin Liu and Keshab K Parhi. Linear-phase lattice fir digital filter architectures using stochastic logic. *Journal of Signal Processing Systems*, 90(5):791–803, 2018.

[26] S.S. Tehrani, S. Mannor, and W.J. Gross. Fully parallel stochastic ldpc decoders. *Signal Processing, IEEE Transactions on*, 56(11):5692–5703, Nov 2008.

[27] A. Naderi, S. Mannor, M. Sawan, and W.J. Gross. Delayed stochastic decoding of ldpc codes. *Signal Processing, IEEE Transactions on*, 59(11):5617–5626, Nov 2011.

[28] Naoya Onizawa, Warren J. Gross, Takahiro Hanyu, and Vincent C. Gaudet. Asynchronous stochastic decoding of ldpc codes: Algorithm and simulation model. *IEICE Transactions on Information and Systems*, 97(9):2286–2295, 2014.

[29] X. R. Lee, C. L. Chen, H. C. Chang, and C. Y. Lee. A 7.92 gb/s 437.2 mw stochastic ldpc decoder chip for ieee 802.15.3c applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(2):507–516, Feb 2015.

[30] S.S. Tehrani, W.J. Gross, and S. Mannor. Stochastic decoding of ldpc codes. *Communications Letters, IEEE*, 10(10):716–718, Oct 2006.

[31] B.D. Brown and H.C. Card. Stochastic neural computation. i. computational elements. *Computers, IEEE Transactions on*, 50(9):891–905, Sep 2001.

[32] B. Li, M. Hassan Najafi, and David J. Lilja. An FPGA implementation of a Restricted Boltzmann Machine Classifier using Stochastic Bit Streams. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 68–69, July 2015.

[33] Bingzhe Li, M. Hassan Najafi, and David J. Lilja. Using stochastic computing to reduce the hardware requirements for a restricted boltzmann machine classifier. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 36–41, New York, NY, USA, 2016. ACM.

[34] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic Energy-accuracy Trade-off Using Stochastic Computing in Deep Neural Networks. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 124:1–124:6, New York, NY, USA, 2016. ACM.

[35] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross. VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2688–2699, Oct 2017.

[36] B. Li, Y. Qin, B. Yuan, and D. J. Lilja. Neural network classifiers using stochastic computing with a hardware-oriented approximate activation function. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 97–104, Nov 2017.

[37] Vincent T. Lee, Armin Alaghi, John P. Hayes, Visvesh Sathe, and Luis Ceze. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '17, pages 13–18, 3001 Leuven, Belgium, Belgium, 2017. European Design and Automation Association.

[38] Yuan Ji, Feng Ran, Cong Ma, and David J. Lilja. A hardware implementation of a radial basis function neural network using stochastic logic. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 880–883, San Jose, CA, USA, 2015. EDA Consortium.

[39] H. Sim, D. Nguyen, J. Lee, and K. Choi. Scalable Stochastic Computing Accelerator for Convolutional Neural Networks. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 696–701, Jan 2017.

[40] H. Sim and J. Lee. A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017.

[41] Bingzhe Li, M. Hassan Najafi, B. Yuan, and David J. Lilja. Quantized neural networks with new stochastic multipliers. In *2018 19th International Symposium on Quality Electronic Design (ISQED)*, pages 376–382, March 2018.

[42] W. Qian and M.D. Riedel. The synthesis of robust polynomial arithmetic with stochastic logic. In *45th ACM/IEEE Design Automation Conference, DAC'08*, pages 648–653, 2008.

[43] Qianying Tang, Bongjin Kim, Yingjie Lao, K.K. Parhi, and C.H. Kim. True random number generator circuits based on single- and multi-phase beat frequency detection. In *Custom Integrated Circuits Conference (CICC), 2014 IEEE Proceedings of the*, pages 1–4, Sept 2014.

[44] Won Ho Choi, L.V. Yang, Jongyeon Kim, A. Deshpande, Gyuseong Kang, Jian-Ping Wang, and C.H. Kim. A magnetic tunnel junction based true random number generator with conditional perturb and real-time output probability tracking. In *Electron Devices Meeting (IEDM), 2014 IEEE International*, pages 12.5.1–12.5.4, Dec 2014.

[45] Solomon W. Golomb and Guang Gong. Signal design for good correlation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2004.

[46] K. Kim, J. Lee, and K. Choi. An energy-efficient random number generator for stochastic circuits. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 256–261, Jan 2016.

[47] Devon Jenson and Marc Riedel. A deterministic approach to stochastic computation. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, pages 102:1–102:8, New York, NY, USA, 2016.

[48] M. Hassan Najafi, Shiva Jamali-Zavareh, David J. Lilja, Marc D. Riedel, Kia Bazargan, and Ramesh Harjani. Time-Encoded Values for Highly Efficient Stochastic Circuits. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 25(5):1–14, 2017.

[49] M. Hassan Najafi and David J. Lilja. High-Speed Stochastic Circuits using Synchronous Analog Pulses. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 481–487, Jan 2017.

[50] Mohammadhassan Najafi, Shiva Jamalizavareh, David J. Lilja, Marcus Riedel, and Kiarash Bazargan. Stochastic Computation using Pulse-Width Modulated Signals. U.S. Patent App. 15869453, pending, filed Jan 2018.

[51] B. Moons and M. Verhelst. Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 4(4):475–486, Dec 2014.

[52] International Technology Roadmap for Semiconductors (ITRS). ITRS 2.0. 2015.

[53] Gordon W. Roberts and M. Ali-Bakhshian. A brief introduction to time-to-digital and digital-to-time converters. *IEEE Transactions on Circuits and System-II*, 57(3):153–157, January 2010.

[54] Xueqin Lu, Shuguo Chen, Chenning Wu, and Mingzhu Li. The pulse width modulation and its use in induction motor speed control. In *Computational Intelligence and Design (ISCID), 2011 Fourth International Symposium on*, volume 2, pages 195–198, Oct 2011.

[55] B. Murmann. "ADC Performance Survey 1997-2016," [online]. Available: http://web.stanford.edu/ murmann/adcsurvey.html, 2016.

[56] S. L. Toral, J. M. Quero, and L. G. Franquelo. Stochastic pulse coded arithmetic. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 1, pages 599–602 vol.1, 2000.

[57] J. M. Quero, C. L. Janer, J. G. Ortega, and L. G. Franquelo. D/A Converter ASIC Uses Stochastic Logic. In *EDN*, pages 86–88, Oct. 1996.

[58] W.J. Poppelbaum, A. Dollas, J.B. Glickman, and C. O'Toole. Unary processing. In *Advances in Computers*, volume 26, pages 47 – 92. Elsevier, 1987.

[59] P. Mars and W.J. Poppelbaum. Stochastic and Deterministic Averaging Processors. IEE digital electronics and computing series. The institution of Electrical Engineers, 1981.

[60] MATLAB. *version 9.0.0 (R2016a)*. The MathWorks Inc., Natick, Massachusetts, 2016.

[61] Peng Li and D.J. Lilja. Using stochastic computing to implement digital image processing algorithms. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 154–161, Oct 2011.

[62] Synopsys. *Design Compiler UserâĂŹs Manual.* http://www.synopsys.com/, 2013.

[63] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue. Compact and accurate stochastic circuits with shared random number sources. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pages 361–366, Oct 2014.

[64] Zhiheng Wang, Naman Saraf, Kia Bazargan, and Arnd Scheel. Randomness meets feedback: Stochastic implementation of logistic map dynamical system. In *Design Automation Conference (DAC)*, 2015.

[65] D. JOHNS and K. MARTIN. *Analog Integrated Circuit Design.* 1997.

[66] B. Razavi. The cross-coupled pair - part i [a circuit for all seasons]. *IEEE Solid-State Circuits Magazine*, 6(3):7–10, Summer 2014.

[67] B. Razavi. The cross-coupled pair - part ii [a circuit for all seasons]. *IEEE Solid-State Circuits Magazine*, 6(4):9–12, Fall 2014.

[68] T. Sepke, P. Holloway, C. G. Sodini, and H. S. Lee. Noise analysis for comparator-based circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(3):541–553, March 2009.

[69] M. Hassan Najafi, David J. Lilja, Marc Riedel, and Kia Bazargan. Power and Area Efficient Sorting Networks Using Unary Processing. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 125–128, Nov 2017.

[70] M. Hassan Najafi, David J. Lilja, Marc D. Riedel, and Kia Bazargan. Low-Cost Sorting Network Circuits Using Unary Processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 1–10, 2018.

[71] Goetz Graefe. Implementing sorting in database systems. *ACM Comput. Surv.*, 38(3), September 2006.

[72] Naga Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. Gputerasort: High performance graphics co-processor sorting for large database management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 325–336, New York, NY, USA, 2006. ACM.

[73] Basel A. Mahafzah. Performance assessment of multithreaded quicksort algorithm on simultaneous multithreaded architecture. *The Journal of Supercomputing*, 66(1):339–363, Oct 2013.

[74] A. A. Colavita, A. Cicuttin, F. Fratnik, and G. Capello. Sortchip: a vlsi implementation of a hardware algorithm for continuous data sorting. *IEEE Journal of Solid-State Circuits*, 38(6):1076–1079, June 2003.

[75] J. P. Agrawal. Arbitrary size bitonic (asb) sorters and their applications in broadband atm switching. In *Conference Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications*, pages 454–458, Mar 1996.

[76] Alberto Colavita, Enzo Mumolo, and Gabriele Capello. A novel sorting algorithm and its application to a gamma-ray telescope asynchronous data acquisition system. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 394(3):374 – 380, 1997.

[77] D. C. Stephens, J. C. R. Bennett, and Hui Zhang. Implementing scheduling algorithms in high-speed networks. *IEEE Journal on Selected Areas in Communications*, 17(6):1145–1158, Jun 1999.

[78] V. Brajovic and T. Kanade. A vlsi sorting image sensor: global massively parallel intensity-to-time processing for low-latency adaptive vision. *IEEE Transactions on Robotics and Automation*, 15(1):67–75, Feb 1999.

[79] K. Ratnayake and A. Amer. An fpga architecture of stable-sorting on a large data volume : Application to video signals. In *2007 41st Annual Conference on Information Sciences and Systems*, pages 431–436, March 2007.

[80] C. Chakrabarti and Li-Yu Wang. Novel sorting network-based architectures for rank order filters. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):502–507, Dec 1994.

[81] D. S. K. Pok, C. I. H. Chen, J. J. Schamus, C. T. Montgomery, and J. B. Y. Tsui. Chip design for monobit receiver. *IEEE Transactions on Microwave Theory and Techniques*, 45(12):2283–2295, Dec 1997.

[82] R. Chen and V. K. Prasanna. Computer generation of high throughput and memory efficient sorting designs on fpga. *IEEE Transactions on Parallel and Distributed Systems*, 28(11):3100–3113, Nov 2017.

[83] N. Guo, Y. Huang, T. Mai, S. Patil, C. Cao, M. Seok, S. Sethumadhavan, and Y. Tsividis. Energy-efficient hybrid analog/digital approximate computation in continuous time. *IEEE Journal of Solid-State Circuits*, 51(7):1514–1524, July 2016.

[84] Y. Tsividis. Continuous-time digital signal processing. *Electronics Letters*, 39(21):1551–1552, Oct 2003.

[85] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM.

[86] K. Kantawala and D. L. Tao. Design, analysis, and evaluation of concurrent checking sorting networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(3):338–343, Sept 1997.

[87] S. Y. Kuo and S. C. Liang. Design and analysis of defect tolerant hierarchical sorting networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1(2):219–223, June 1993.

[88] Sherenaz W. Al-Haj Baddar and Basel A. Mahafzah. Bitonic sort on a chained-cubic tree interconnection network. *J. Parallel Distrib. Comput.*, 74(1):1744–1761, January 2014.

[89] Buğra Gedik, Rajesh R. Bordawekar, and Philip S. Yu. Cellsort: High performance sorting on the cell processor. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 1286–1297, 2007.

[90] W.J. Poppelbaum. Burst processing: A deterministic counterpart to stochastic computing. In *Proceedings of the 1st International Symposium on Stochastic Computing and its Applications*. 1978.

[91] E. Nikahd, P. Behnam, and R. Sameni. High-speed hardware implementation of fixed and runtime variable window length 1-d median filters. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 63(5):478–482, May 2016.

[92] D. S. Richards. VLSI median filters. *IEEE Trans. on Acoustics, Speech, and Signal*, 38(1):145–153, Jan 1990.

[93] M. Karaman, L. Onural, and A. Atalar. Design and implementation of a general-purpose median filter unit in cmos vlsi. *IEEE Journal of Solid-State Circuits*, 25(2):505–513, Apr 1990.

[94] C. Chakrabarti. Sorting network based architectures for median filters. *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, 40(11):723–727, 1993.

[95] J. E. Eklund, C. Svensson, and A. Astrom. Vlsi implementation of a focal plane image processor-a realization of the near-sensor image processing concept. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(3):322–335, Sept 1996.

[96] Vishnu Ravinuthula, Vaibhav Garg, John G. Harris, and JosÃľ A. B. Fortes. Time-mode circuits for analog computation. *Intern. Journal of Circuit Theory and Applications*, 37(5):631–659, 2009.

[97] WMC Sansen. Analog design essentials, ser. the international series in engineering and computer science, 2006.

[98] M. Hassan Najafi and David J. Lilja. High Quality Down-Sampling for Deterministic Approaches to Stochastic Computing. *IEEE Transactions on Emerging Topics in Computing*, PP(99):1–1, 2018.

[99] M. Hassan Najafi and David J. Lilja. High Quality Down-Sampling for Deterministic Approaches to Stochastic Computing. In *Computer Design (ICCD), 2017 IEEE 35th International Conference on*, Nov 2017.

[100] M. Hassan Najafi, David J. Lilja, and Marc Riedel. Fast-Converging, Scalable, Deterministic Bit-Stream Computing using Low-Discrepancy Sequences. In *2018 27th International Workshop on Logic and Synthesis (IWLS)*, pages 1–6, June 2018.

[101] M. Hassan Najafi, David J. Lilja, and Marc Riedel. Deterministic Methods for Stochastic Computing using Low-Discrepancy Sequences. In *Proceedings of the 37th International Conference on Computer-Aided Design*, ICCAD '18, pages 1–8, 2018.

[102] Mohammadhassan Najafi and David J Lilja. High Quality Down-Sampling for Deterministic Bit-Stream Computing. U.S. Provisional Patent App. 62643369, filed March 2018.

[103] P. Koopman. Maximal Length LFSR Feedback Terms https://users.ece.cmu.edu/ koopman/lfsr/index.html, 2017.

[104] M. Hassan Najafi, David J. Lilja, Marc Riedel, and Kia Bazargan. Polysynchronous Stochastic Circuits. In *2016 21st ASP-DAC*, pages 492–498, Jan 2016.

[105] David J Lilja, Mohammadhassan Najafi, Marcus Riedel, and Kiarash Bazargan. Polysynchronous Stochastic Circuits. U.S. Patent App. 15448997, pending, filed March 2018.

[106] E.G. Friedman. Clock distribution networks in synchronous digital integrated circuits. *Proceedings of the IEEE*, 89(5):665–692, May 2001.

[107] Y. Jiang, H. Zhang, H. Zhang, H. Liu, X. Song, M. Gu, and J. Sun. Design of mixed synchronous/asynchronous systems with multiple clocks. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1–1, 2014.

[108] G. Tosik, L.M.S. Gallego, and Z. Lisik. Different approaches for clock skew analysis in present and future synchronous ic's. In *EUROCON, 2007. The International Conference on Computer as a Tool*, pages 1227–1232, Sept 2007.

[109] Shauki Elassaad. *Clock Driven Design Planning*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2008.

[110] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[111] A.A. Abidi. Phase noise and jitter in cmos ring oscillators. *Solid-State Circuits, IEEE Journal of*, 41(8):1803–1816, Aug 2006.

[112] V. Sikarwar, N. Yadav, and S. Akashe. Design and analysis of cmos ring oscillator using 45 nm technology. In *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, pages 1491–1495, Feb 2013.

[113] Martin Reisslein, Lina J Karam, Patrick Seeling, FH Fitzek, and Tatjana K Madsen. YUV Video Sequences. *http://trace.eas.asu.edu/yuv/*, Online Access: May 2015.

[114] G. Sannena and B. P. Das. A metastability immune timing error masking flip-flop for dynamic variation tolerance. In *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, pages 151–156, May 2016.

[115] D. Rennie, D. Li, M. Sachdev, B. L. Bhuva, S. Jagannathan, S. Wen, and R. Wong. Performance, metastability, and soft-error robustness trade-offs for flip-flops in 40 nm cmos. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(8):1626–1634, Aug 2012.

[116] Mentor Graphics. *ModelSim PE Student Edition*. https://www.mentor.com/company/higher_ed/modelsim-student-edition, 2015.

[117] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In *Proceedings of the Sixth MIT Conference on Advanced Research in VLSI*, AUSCRYPT '90, pages 263–278, Cambridge, MA, USA, 1990. MIT Press.

[118] S.M. Nowick and M. Singh. Asynchronous design (part 1): Overview and recent advances. *Design Test, IEEE*, 32(3):5–18, June 2015.

[119] D. Chapiro. Globally-asynchronous locally-synchronous systems. *Stanford University*, 1984.

[120] A. Rajakumari, N.S.M. Sharma, K.L. Kishore, and V.K. Petta. A power gating gals interface implementation. In *Microelectronics and Electronics (PrimeAsia),*

2013 IEEE Asia Pacific Conference on Postgraduate Research in, pages 34–39, Dec 2013.

[121] A.J. Martin and M. Nystrom. Asynchronous techniques for system-on-chip design. *Proceedings of the IEEE*, 94(6):1089–1120, June 2006.

[122] K.Y. Yun and R.P. Donohue. Pausible clocking: a first step toward heterogeneous systems. In *Computer Design: VLSI in Computers and Processors, 1996. ICCD '96. Proceedings., 1996 IEEE International Conference on*, pages 118–123, Oct 1996.

[123] J.P. Hayes. Introduction to stochastic computing and its challenges. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–3, June 2015.

[124] Armin Alaghi, Wei-Ting J. Chan, John P. Hayes, Andrew B. Kahng, and Jiajia Li. Optimizing stochastic circuits for accuracy-energy tradeoffs. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, ICCAD '15, 2015.

[125] S. K. Khatamifard, M. Hassan Najafi, A. Ghoreyshi, U. R. Karpuzcu, and D. J. Lilja. On Memory System Design for Stochastic Computing. *IEEE Computer Architecture Letters*, 17(2):117–121, July 2018.

[126] D. Fick, G. Kim, A. Wang, D. Blaauw, and D. Sylvester. Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2d edge detection and noise filtering. In *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, pages 1–4, Sept 2014.

[127] N. Onizawa, D. Katagiri, W. J. Gross, and T. Hanyu. Analog-to-stochastic converter using magnetic tunnel junction devices for vision chips. *IEEE Transactions on Nanotechnology*, PP(99):1–1, 2015.

[128] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. Redeye: Analog convnet image sensor architecture for continuous mobile vision. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 255–266, Piscataway, NJ, USA, 2016. IEEE Press.

[129] Junjie Lu, Steven Young, Itamar Arel, and Jeremy Holleman. A 1 tops/w analog deep machine-learning engine with floating-gate storage in 0.13 μm cmos. *IEEE Journal of Solid-State Circuits*, 50(1):270–281, 2015.

[130] Lukas Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. Meyer Anderson, and Yusuf Leblebici. A 90GS/s 8b 667mW 64x Interleaved SAR ADC in 32nm Digital SOI CMOS. In *Proceedings of the 2014 ISSCC*, 2014.

[131] W. T. Lin and T. H. Kuo. A 12b 1.6gs/s 40mw dac in 40nm cmos with 70db sfdr over entire nyquist bandwidth. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 474–475, Feb 2013.

[132] I. Akturk, K. Khatamifard, and U. R. Karpuzcu. On Quantification of Accuracy Loss in Approximate Computing. In *12th Annual Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, June 2015.

[133] M. Hassan Najafi, Shiva Jamali-Zavareh, David J. Lilja, Marc D. Riedel, Kia Bazargan, and Ramesh Harjani. Time-Encoded Values for Highly Efficient Stochastic Circuits. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.

[134] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han. A stochastic computational multi-layer perceptron with backward propagation. *IEEE Transactions on Computers*, pages 1–1, 2018.

[135] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'17, pages 405–418, New York, NY, USA, 2017. ACM.