Polysynchronous Clocking: Exploiting the Skew Tolerance of Stochastic Circuits

M. Hassan Najafi, *Student Member, IEEE*, David J. Lilja, *Fellow, IEEE*,
Marc D. Riedel, *Senior Member, IEEE*, and Kia Bazargan, *Senior Member, IEEE*

Abstract—In the paradigm of stochastic computing, arithmetic functions are computed on randomized bit streams. The method naturally and effectively tolerates very high clock skew. Exploiting this advantage, this paper introduces polysynchronous clocking, a design strategy in which clock domains are split at a very fine level. Each domain is synchronized by an inexpensive local clock. Alternatively, the skew requirements for a global clock distribution network can be relaxed. This allows for a higher working frequency and so lower latency. The benefits of both approaches are quantified. Polysynchronous clocking results in significant latency, area, and energy savings for wide variety of applications.

Index Terms—Polysynchronous clocking, stochastic computing, multi-clock circuits, clock distribution networks, relaxed clocking

1 Introduction

Stochastic Computing (SC), first advocated by Gaines in 1969 [12], has received renewed attention by the EDA community in recent years [5], [6], [17], [19], [22], [24], [25], [29], [30], [32], [34], [35], [36], [37], [44]. In SC designs, logical computation is performed on randomized bit streams, with numerical values encoded as probabilities: A real value x in the interval [0, 1] is represented by a stream with bits each having independent probability x of being 1.

Such a representation has an advantage over conventional binary radix in terms of error tolerance. Suppose that the environment is noisy: Bit flips occur and these afflict all the bits with equal probability. With a binary radix representation, in the worst case, the most significant bit gets flipped, resulting in a large error. In contrast, with a stochastic representation, all the bits in the stream have equal weight. A single flip results in a small error. This error tolerance scales to high error rates: Multiple bit flips produce small and uniform deviations from the nominal value.

More compelling than the error tolerance is the simplicity of the SC designs. Complex functions can be implemented with remarkably simple logic. Multiplication can be performed with a single AND gate. Functions such as polynomial approximations of trigonometric functions can be implemented with less than a dozen gates. Over a wide range of arithmetic functions, a reduction in area of 50× or 100× compared to conventional implementations is common [4], [22].

A more compelling advantage still might be the aspect of SC exploited in the paper: SC circuits naturally and effectively tolerate very high clock skew. Note that a stochastic representation is uniform: The value that is represented by a bit stream is simply the fraction of time that the signal is high. 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.

This paper introduces polysynchronous clocking, a design strategy for SC circuits in which clock domains are split at a very fine level. We explore two strategies. The first is to synchronize each domain by an inexpensive local clock, such as an inverter ring. This obviates the need for an expensive global clock distribution network (CDN). The second is to keep a global CDN but relax the clock skew requirements between domains. This allows for a higher working frequency and so lower latency.

We quantify the area, speed, and energy saving advantages of both approaches. Our experimental results show that replacing a global CDN with local clocks significantly improves the area, latency, and energy consumption for large SC designs. For smaller SC designs a "relaxed" global CDN is a more efficient choice. We show that circuits designed with either of these "polysynchronous" approaches are as tolerant of errors as conventional synchronous stochastic circuits.

The paper is structured as follows. In Section 2, we present background material, including a general discussion of CDNs, clock skew, and stochastic computing. In Section 3, we introduce polysynchronous clocking. In Section 4, we describe two approaches to polysynchronous system design: (1) replacing a global CDN with locally generated clocks, and (2) relaxing a global CDN. In Section 5, we provide a case study comparing the cost and benefits of conventional

• *The authors are with the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, MN 55455. E-mail: {najaf011, lilja, mriedel, kia}@umn.edu.*

Manuscript received 28 Aug. 2016; revised 23 Mar. 2017; accepted 17 Apr. 2017. Date of publication 24 Apr. 2017; date of current version 14 Sept. 2017. Recommended for acceptance by D. Marculescu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2017.2697881

design with CDNs to polysynchronous designs, quantifying the area, speed and energy advantages. In Section 6, we compare the error tolerance of polysynchronous stochastic circuits to conventional synchronous stochastic circuits. Finally, in Sections 7 and 8, we discuss related work and draw conclusions.

2 Background

2.1 Clock Distribution Networks

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.

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 [11], [16], [45]. 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. Temporal variations, known as jitter, affect the arrival time of the clock edges at the sequential elements across different clock cycles [10].

There are a number of factors that contribute to uncertainty: Differences in line lengths from the clock source to clocked registers; differences in delays of distributed buffers; differences in passive interconnect parameters, such as line resistivity, dielectric constants and thickness, via/contact resistance, line and fringing capacitance, and line dimensions; and differences in active device parameters, such as MOS threshold voltages and channel mobilities, which affect the delay of active buffers [11].

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.

Skew can limit circuit performance, since a circuit must be clocked at a lower frequency to tolerate it. If unaccounted for, clock skew can cause timing-related errors. 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 [45].

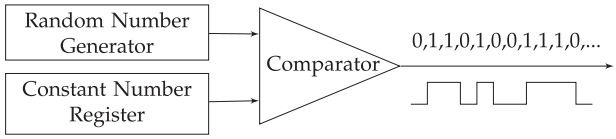

Fig. 1. Stochastic number generator.

In summary, for modern integrated circuits, the global CDN is a major bottleneck in terms of design effort, area, and performance. Stochastic computing offers skew tolerance. In Section 4, we will explain how this feature can be used to mitigate the costs: Either the global CDN can be eliminated entirely; or one can design a much less costly global CDN that tolerates skew.

2.2 Stochastic Computing

In the paradigm of stochastic computing (SC), circuits operate on random bit streams where the signal value is encoded by the probability of obtaining a one versus a zero. In the unipolar stochastic representation, each real-valued number x $(0 \leq x \leq 1)$ is represented by a sequence of random bits, each of which has probability x of being one and probability $1 - x$ of being zero. In the bipolar representation, each real-valued number y $(-1 \leq y \leq 1)$ is represented by a sequence of random bits, each of which has probability $\frac{y+1}{2}$ of being one and probability $1 - \frac{y+1}{2}$ of being zero.

This representation is much less compact than a binary radix. However, complex operations can be performed with very simple logic. In particular, arithmetic functions, consisting of operations like addition and multiplication can be implemented very efficiently. Complex functions, such as exponentials and trigonometric functions, can be computed through polynomial approximations [22], [37]. Because the bit stream representation is uniform, with all bits weighted equally, circuits designed this way are highly tolerant of soft errors (i.e., bit flips).

Given an input value, say in binary radix, the conventional approach for generating a stochastic bit stream with probability x is as follows. Obtain an unbiased random value $0 \leq r \leq 1$ from a random [9] [43] or pseudorandom source [13], [18]; compare it to the target value x; output a one if $r \leq x$ and a zero otherwise. Fig. 1 illustrates the approach. The "random number generator" is usually a linear-feedback shift register (LFSR), which produces high quality pseudo-randomness [13]. In this approach, the period of the clock feeding the generator corresponds to the duration of a single bit in the output stream. Assuming that the pseudo-random numbers are uniformly distributed between $0 \ldots 2^M - 1$, the value stored in the constant number register should be $2^M \cdot x$. In the output, each bit is one with pseudo-probability $2^M \cdot x / 2^M = x$ [7], [12].

2.3 Stochastic Operations

2.3.1 Multiplication

In SC multiplication can be implemented using a standard AND gate for the unipolar coding format and an XNOR gate for the bipolar coding format [36]. Fig. 2 shows the multiplication of two 10-bit unipolar stochastic streams using an AND gate.

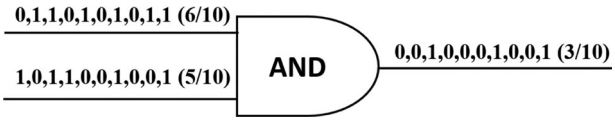

Fig. 2. Example of stochastic multiplication using an AND gate.

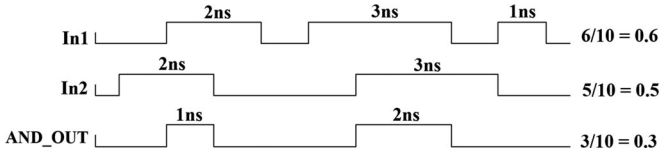

Fig. 3. Stochastic multiplication using an AND with unsynchronized bit stream.

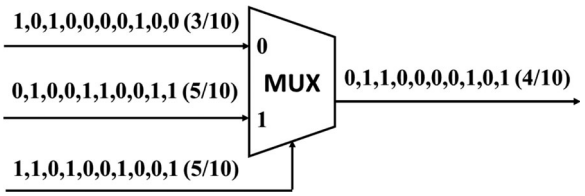

Fig. 4. Example of stochastic scaled addition using a MUX unit.

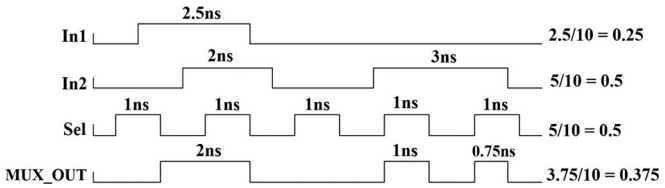

Fig. 5. Stochastic scaled addition using a MUX with unsynchronized bit streams.

The value represented by a bit stream is the time that the signal is high divided by the total length of the stream. Fig. 3 illustrates an example of multiplying two unsynchronized bit streams representing 0.6 and 0.5. As shown, the value represented by the bit stream at the output of the AND gate is 0.3, the value one expects when multiplying 0.6 by 0.5.

2.3.2 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). Fig. 4 illustrates the operation $\frac{1}{2}A + \frac{1}{2}B$.

Fig. 5 illustrates another example of scaled addition, this time on two unsynchronized bit streams representing 0.25 and 0.5. As expected, the output is a bit stream representing 0.375, the result of the scaled addition.

2.3.3 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 [7]. Li and Lilja [21] also developed an FSM-based stochastic absolute value function. The state transition diagrams of the FSMs implementing these functions are shown in Fig. 6. 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:

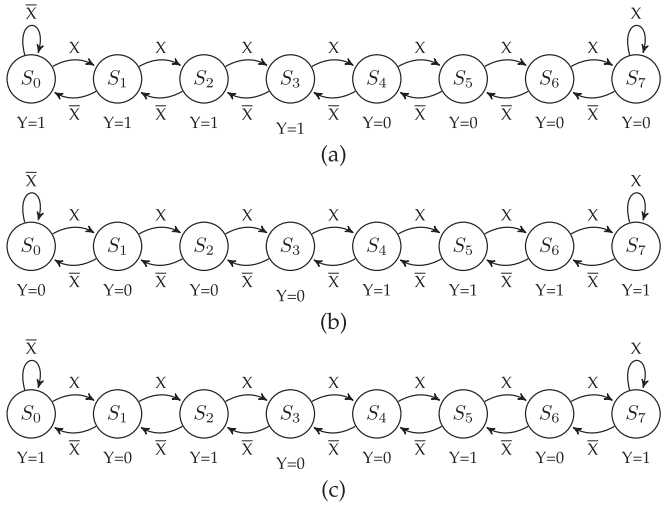

Fig. 6. State transition diagram of the FSM implementing a) the stochastic exponentiation function b) the stochastic tanh function c) stochastic absolute value function. For details of the implementation, the readers are referred to [23].

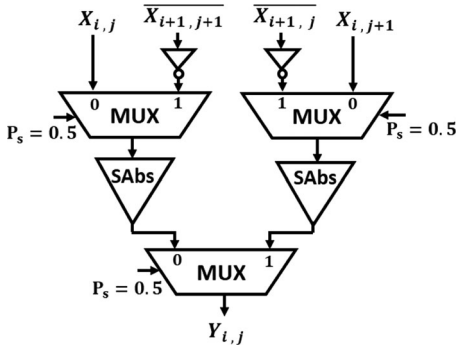

Fig. 7. Stochastic implementation of the Robert's cross edge detection algorithm [20].

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.

2.4 Stochastic Circuits

Stochastic computing has been applied to a wide variety of applications, including image and signal processing applications. In this paper, 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.

2.4.1 Robert's Cross Edge Detection

Robert's cross edge detection algorithm is a well-known and widely studied algorithm. A stochastic implementation of this algorithm, proposed in [22], is shown in Fig. 7. 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

$$y_{i,j} = \frac{1}{2} \times \left(\frac{1}{2} \left| x_{i,j} - x_{i+1,j+1} \right| + \frac{1}{2} \left| x_{i,j+1} - x_{i+1,j} \right| \right), \quad (1)$$

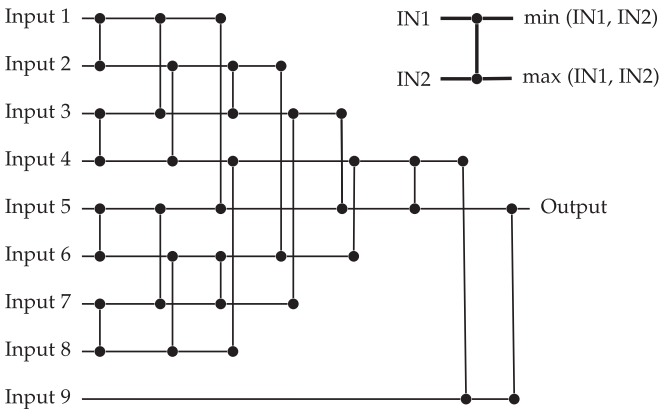

Fig. 8. Hardware implementation of the 3x3 median filter based on a sorting network [22].

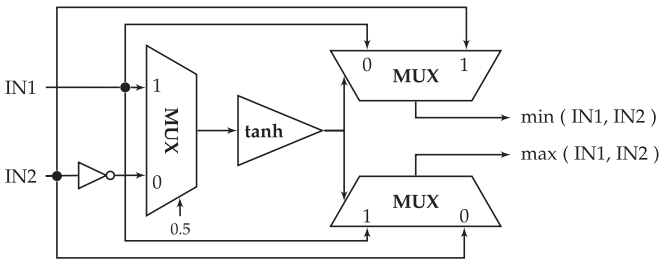

Fig. 9. Stochastic implementation of basic sorting unit.

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. 7, three multiplexers perform addition and subtraction, while two finite-state-machine based stochastic circuit perform the required absolute value operations. Since this circuit operates on signed values, all streams must be in the bipolar format.

2.4.2 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 [14]. A hardware implementation of the 3 x 3 median filter based on a sorting network is shown in Fig 8. Each basic sorting unit used in this circuit is implemented with the circuit presented in Fig. 9. In total, the median filter circuit requires 19 basic sorting units (57 MUX units and 19 FSM-based stochastic tanh circuits.)

2.4.3 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 [20], is shown in Fig. 10. Let $X_t, X_{t-1}, X_{t-2}, \ldots, Xt - n$ be recent samples of intensity values of a pixel X. The stochastic circuit proposed in [20] uses the following formula as the probability estimator

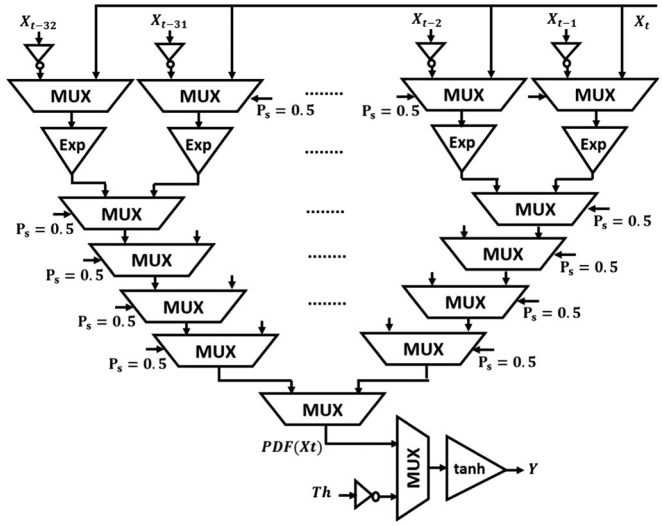

Fig. 10. Stochastic implementation of the KDE-based image segmentation algorithm [20].

$$PDF(X_t) = \frac{1}{n}\sum_{i=1}^{n} e^{-4|X_t - X_{t-i}|}. \quad (2)$$

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.

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 paper: Polysynchronous clocking.

Consider an AND gate, responsible for multiplying two unipolar input bit streams, 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. 12). 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. 11 illustrates the input signals as well as the output signal in the case where T1=1.8ns and T2=3.2 ns for 20 ns of operation. Continuing the operation for about 1,000 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 1 presents results for the three selected cases of

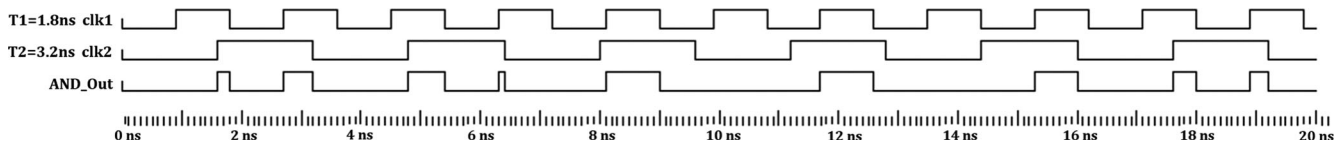

Fig. 11. Input clock signals and the corresponding output from connecting polysynchronous inputs to an AND gate.

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 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 1,000 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.

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 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 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 3 are based on bit streams of length 1,024, 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 stochastic computation [36].

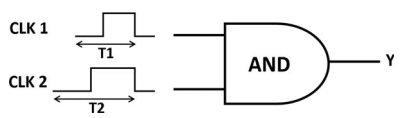

Fig. 12. An AND gate connected to polysynchronous clock sources.

TABLE 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 1,000 ns

T1=2 ns		T1=2 ns		T1=1.8 ns	
T2=3.5 ns		T2=3.2 ns		T2=3.2 ns	
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	

Proof. Polysynchronous stochastic signals can be discretized into digital stochastic bit streams 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 bit streams, $X(t)$ 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 bit streams are stochastically independent, by connecting them to the inputs of an AND gate, the output is a bit stream $Z(t)$, where

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

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

TABLE 2
The Measured Output of the MUX When Three Polysynchronous Clocks with Distinct Periods Are Connected to Its Inputs for 1,000 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 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

polysynchronous signals generated by identical SNGs but driven by different clocks, are expected to be independent, since they 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 1,000 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 circuit's clock period was fixed at 1 ns. For the polysynchronous clocking conditions, clock periods were selected randomly in the range from 1 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 1,000 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.

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. 7, to process each of the pixels concurrently. Fig. 13 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.

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. Th 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.

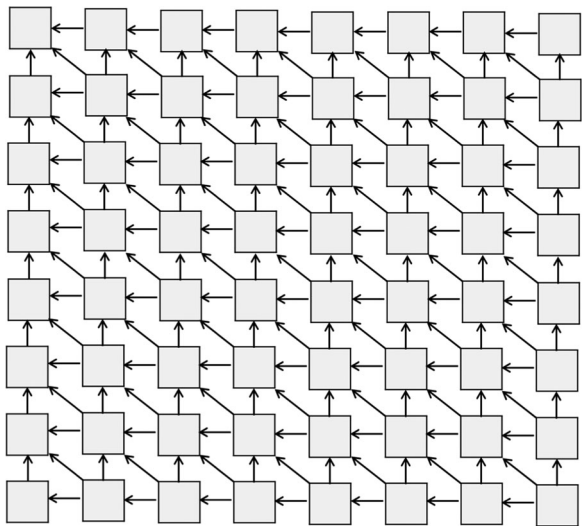

Fig. 13. 64 Robert's cross cells processing a 8×8 input image concurrently.

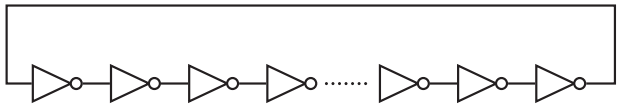

Fig. 14. Ring oscillator circuit with odd number of stages.

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 Fig. 14. 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 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 [2], [41]. 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 1 V. Thus, although relatively inexpensive, the area and power costs of inverter rings are not insignificant.

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 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 Fig. 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 bit streams 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 [42] with a 45 nm gate library to synthesize the designs.

TABLE 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 @2 Ghz (mW)
Without local clock generator	268.0	0.83
With local clock generator	291.9	1.09

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 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 1 V. Such inverter rings were added to each Robert's cross cell. Table 4 shows the area-power cost 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 and 24 percent, respectively. We will show that, for large designs, this overhead is small compared to the savings obtained by removing the CDN.

6 Experimental Results

6.1 Synthesis Results

The synthesis results, including the delay, area, total dynamic and static power consumption, energy dissipation

TABLE 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

of one clock cycle, and area-delay product, are shown in Table 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 Fig. 15.

In terms of area, both approaches decrease the cost in the three cases with 16, 64, and 256 cells, as shown in Fig. 15. 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 and Fig. 15, 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 $3.5\times$. 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

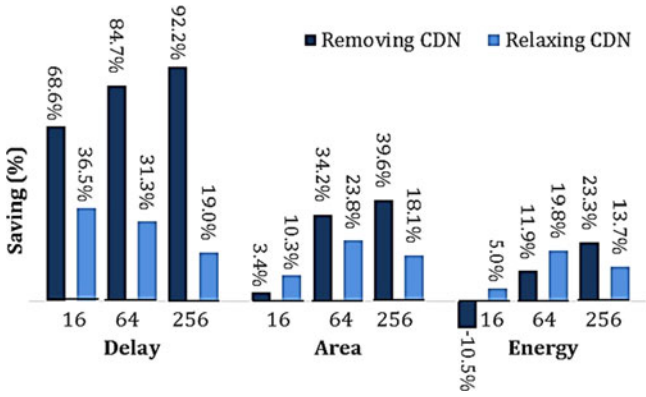

Fig. 15. Comparing the savings due to using different approaches of polysynchronous clocking on various sizes of the Robert's cross circuit.

Section 2.3: 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 0.2ns clock uncertainty when the circuits were synthesized with Design Compiler. Table. 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.

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 [1] using the KDE image segmentation circuits. For simulations with the Robert's cross circuits, image pixels were divided into groups of 16,

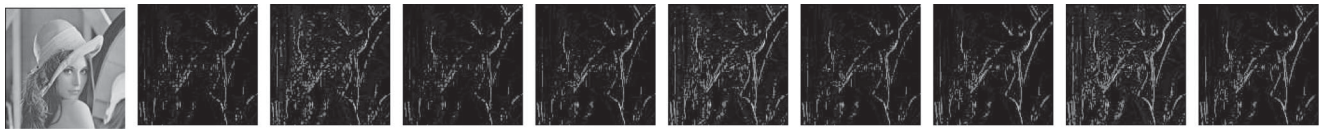

Fig. 16. The original sample input and the output images produced by post-synthesis simulations of the synthesized Robert's cross circuits.

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 1,024 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:

$$E = \frac{\sum_{i=1}^{128}\sum_{j=1}^{128}|T_{i,j} - S_{i,j}|}{255.(128 \times 128)} \times 100, \quad (3)$$

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 Fig. 16. The mean of the output error rates measured for each circuit is also shown in Table 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 Fig. 17. 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.

a)Median filter noise reduction

b)KDE image segmentation

Fig. 17. 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.

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.

7.1 Sources of Computational Errors

There are three main sources of computational errors in the conventional synchronous stochastic circuits [36]:

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. [23] 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 bit streams, the SNG rounds the input value to the closest number in the set of discrete probabilities it can generate. Increasing the length of the bit streams will reduce this quantization error [36].
3. E_R = errors due to random fluctuations. Errors due to random fluctuations are inherent in stochastic computing since the input values are intentionally randomized. The bit streams 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 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 7.3

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

$$E_{Total} = E_A + E_Q + E_R + E_C + E_S. \quad (4)$$

Based on the error rate results presented in Table 5 and Figs. 16 and 17, 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.

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 bistable 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 [39], [40]. 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 bit-stream. 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 7.1.

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 versus 256x2 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.

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 2.4, 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 [36]. 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 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 [28]. Maximal period 32-bit LFSRs were used for converting input pixel values into stochastic bit streams. Bit streams of length 1,024 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

TABLE 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

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 6.2 were used as the inputs to the circuits. Table 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 percent, the circuits will gracefully tolerate errors for frequency variations beyond 100 percent if the inputs are processed for a long enough time (e.g., 1,024 times the largest period).

9 Related Work and Discussion

Asynchronous design methodologies have been studied for decades [27], [33]. 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 [8]. GALS architectures consume less dynamic power and can achieve better performance than architectures with a single clock domain [26], [38]. However, the circuitry for domain crossings is complex and problematic. Techniques such as stretching [8], [46] and pausing the clocks [38] 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 paper, 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 stochastic computing allows independent clock domains to be connected together seamlessly without influencing the accuracy. Alternatively, it allows for a much less costly global clock distribution network, 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 [15]. 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. [3] proposed energy reduction techniques for stochastic computing. 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 percent supply voltage reduction (1 V to 0.6 V), leading to 66 percent energy saving. Similarly, a 100 percent 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 paper. 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 paper and the work of Alaghi et al. [3] 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. In future work, we will consider the impact of both energy and clock distribution optimizations for SC.

10 Conclusions

This paper 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.

Acknowledgments

This work was supported in part by National Science Foundation grant no. CCF-1408123. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. A perliminary version of this paper appeared as [31].

References

[1] Yuv video sequences, (2010). [Online]. Available: http://trace.eas.asu.edu/yuv/

[2] A. Abidi, "Phase noise and jitter in CMOS ring oscillators," *IEEE J. Solid-State Circuits*, vol. 41, no. 8, pp. 1803–1816, Aug. 2006.

[3] A. Alaghi, W.-T. J. Chan, J. P. Hayes, A. B. Kahng, and J. Li, "Optimizing stochastic circuits for accuracy-energy tradeoffs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2015, pp. 178–185.

[4] A. Alaghi and J. Hayes, "Fast and accurate computation using stochastic circuits," in *Proc. Des. Autom. Test Europe Conf. Exhibition*, Mar. 2014, pp. 1–4.

[5] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 92:1–92:19, May 2013.

[6] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI implementation of deep neural network using integral stochastic computing," *2016 9th Int. Symp. Turbo Codes Iterative Inf. Proc. (ISTC)*, pp. 216–220, Sept. 2016.

[7] B. Brown and H. Card, "Stochastic neural computation. i. computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, Sep. 2001.

[8] D. Chapiro, "Globally-asynchronous locally-synchronous systems," Stanford, CA, USA: Stanford Univ., 1984.

[9] W. H. Choi, et al., "A magnetic tunnel junction based true random number generator with conditional perturb and real-time output probability tracking," in *Proc. IEEE Int. Electron Devices Meeting*, Dec. 2014, pp. 12.5.1–12.5.4.

[10] S. Elassaad, "Clock driven design planning," PhD thesis, EECS Dept., Univ. California, Berkeley, CA, Aug. 2008.

[11] E. Friedman, "Clock distribution networks in synchronous digital integrated circuits," *Proc. IEEE*, vol. 89, no. 5, pp. 665–692, May 2001.

[12] B. Gaines, "Stochastic computing systems," in *Proc. Advances Inf. Syst. Sci.*, 1969, pp. 37–172.

[13] S. W. Golomb and G. Gong, "Signal design for good correlation: For wireless communication, cryptography, and radar," Cambridge University Press, New York, NY, USA, 2004.

[14] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2006.

[15] J. Hayes, "Introduction to stochastic computing and its challenges," in *Proc. 52nd ACM/EDAC/IEEE Des. Autom. Conf.*, Jun. 2015, pp. 1–3.

[16] Y. Jiang, et al., "Design of mixed synchronous/asynchronous systems with multiple clocks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 8, pp. 2220–2232, Aug. 2015.

[17] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proc. 53rd Annu. Des. Autom. Conf.*, 2016, pp. 124:1–124:6.

[18] K. Kim, J. Lee, and K. Choi, "An energy-efficient random number generator for stochastic circuits," in *Proc. 21st Asia South Pacific Des. Autom. Conf.*, Jan. 2016, pp. 256–261.

[19] B. Li, M. H. Najafi, and D. J. Lilja, "Using stochastic computing to reduce the hardware requirements for a restricted Boltzmann machine classifier," in *Proc. ACM/SIGDA Int. Symp. Field-Programm. Gate Arrays*, 2016, pp. 36–41.

[20] P. Li and D. Lilja, "A low power fault-tolerance architecture for the kernel density estimation based image segmentation algorithm," in *Proc. IEEE Int. Conf. Application-Specific Syst. Archit. Process.*, Sep. 2011, pp. 161–168.

[21] P. Li and D. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *Proc. IEEE 29th Int. Conf. Comput. Des.*, Oct. 2011, pp. 154–161.

[22] P. Li, D. Lilja, W. Qian, K. Bazargan, and M. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 22, no. 3, pp. 449–462, Mar. 2014.

[23] P. Li, D. Lilja, W. Qian, M. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finite-state machines," *IEEE Trans. Comput.*, vol. 63, no. 6, pp. 1474–1486, Jun. 2014.

[24] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2012, pp. 480–487.

[25] Y. Liu, H. Venkataraman, Z. Zhang, and K. K. Parhi, "Machine learning classifiers using stochastic logic," in *Proc. IEEE 34th Int. Conf. Comput. Des.*, Oct. 2016, pp. 408–411.

[26] A. Martin and M. Nystrom, "Asynchronous techniques for system-on-chip design," *Proc. IEEE*, vol. 94, no. 6, pp. 1089–1120, Jun. 2006.

[27] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Proc. 6th MIT Conf. Advanced Res. Very Large Scale Integr.*, 1990, pp. 263–278.

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

[29] M. H. Najafi, S. Jamali-Zavareh , D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani, "Time-encoded values for highly efficient stochastic circuits," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 25, no. 5, pp. 1644–1657, May 2017.

[30] M. H. Najafi and D. J. Lilja, "High-speed stochastic circuits using synchronous analog pulses," in *Proc. 22nd Asia South Pacific Des. Autom. Conf.*, Jan. 2017, pp. 481–487.

[31] M. H. Najafi, D. J. Lilja, M. Riedel, and K. Bazargan, "Polysynchronous stochastic circuits," in *Proc. 21st Asia South Pacific Des. Autom. Conf.*, Jan. 2016, pp. 492–498.

[32] M. H. Najafi and M. E. Salehi, "A fast fault-tolerant architecture for sauvola local image thresholding algorithm using stochastic computing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 24, no. 2, pp. 808–812, Feb. 2016.

[33] S. Nowick and M. Singh, "Asynchronous design (part 1): Overview and recent advances," *IEEE Des. Test*, vol. 32, no. 3, pp. 5–18, Jun. 2015.

[34] N. Onizawa, W. J. Gross, T. Hanyu, and V. C. Gaudet, "Asynchronous stochastic decoding of LDPC codes: Algorithm and simulation model," *IEICE Trans. Inf. Syst.*, vol. 97, no. 9, pp. 2286–2295, 2014.

[35] N. Onizawa, D. Katagiri, K. Matsumiya, W. J. Gross, and T. Hanyu, "Gabor filter based on stochastic computation," *IEEE Signal Process. Lett.*, vol. 22, no. 9, pp. 1224–1228, Sep. 2015.

[36] W. Qian, X. Li, M. Riedel, K. Bazargan, and D. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 93–105, Jan. 2011.

[37] W. Qian and M. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Proc. 45th ACM/IEEE Des. Autom. Conf.*, 2008, pp. 648–653.

[38] A. Rajakumari, N. Sharma, K. Kishore, and V. Petta, "A power gating gals interface implementation," in *Proc. IEEE Asia Pacific Conf. Postgraduate Res. Microelectronics Electron.*, Dec. 2013, pp. 34–39.

[39] D. Rennie, et al., "Performance, metastability, and soft-error robustness trade-offs for flip-flops in 40 nm CMOS," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 59, no. 8, pp. 1626–1634, Aug. 2012.
[40] G. Sannena and B. P. Das, "A metastability immune timing error masking flip-flop for dynamic variation tolerance," in *Proc. Int. Great Lakes Symp.*, May 2016, pp. 151–156.
[41] V. Sikarwar, N. Yadav, and S. Akashe, "Design and analysis of CMOS ring oscillator using 45 nm technology," in *Proc. IEEE 3rd Int. Advance Comput. Conf.*, Feb. 2013, pp. 1491–1495.
[42] "Synopsys," *Design Compiler User's Manual*. (2013). [Online]. Available: http://www.synopsys.com/
[43] Q. Tang, B. Kim, Y. Lao, K. Parhi, and C. Kim, "True random number generator circuits based on single- and multi-phase beat frequency detection," in *Proc. IEEE Proc. Custom Integr. Circuits Conf.*, Sep. 2014, pp. 1–4.
[44] S. Tehrani, W. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 10, pp. 716–718, Oct. 2006.
[45] G. Tosik, L. Gallego, and Z. Lisik, "Different approaches for clock skew analysis in present and future synchronous ic's," in *Proc. Int. Conf. Comput. Tool*, Sep. 2007, pp. 1227–1232.
[46] K. Yun and R. Donohue, "Pausible clocking: A first step toward heterogeneous systems," in *Proc. IEEE Int. Conf. Comput. Des.: VLSI Comput. Process.*, Oct. 1996, pp. 118–123.

M. Hassan Najafi (S'15) received the BSc degree in computer engineering from the University of Isfahan, Iran and the MSc degree in computer architecture from the University of Tehran, Iran, in 2011 and 2014, respectively. He is currently working toward the PhD degree as a research assistant at ARCTiC Labs in the Department of Electrical and Computer Engineering, University of Minnesota, Twin cities. His research interests include stochastic and approximate computing, computer-aided design of integrated circuits, low power design, and designing fault tolerant systems. He is a student member of the IEEE.

David J. Lilja (F'06) received the BS degree in computer engineering from Iowa State University, Ames, Iowa, and the MS and PhD degrees in electrical engineering from the University of Illinois at Urbana-Champaign in Urbana, Illinois. He is currently the Schnell professor of electrical and computer engineering with the University of Minnesota, in Minneapolis, Minnesota, where he also serves as a member of the graduate faculties in Computer Science, Scientific Computation, and Data Science. Previously, he served 10 years as the head of the ECE Department, University of Minnesota, and worked as a research assistant at the Center for Supercomputing Research and Development, University of Illinois, and as a development engineer at Tandem Computers Incorporated in Cupertino, California. He was elected a fellow of the Institute of Electrical and Electronics Engineers (IEEE) and the American Association for the Advancement of Science (AAAS).

Marc D. Riedel (SM'12) received the BEng degree in electrical engineering from McGill University, Montreal, QC, Canada, and the MSc and PhD degrees in electrical engineering from the California Institute of Technology (Caltech), Pasadena, California. He is currently an associate professor of electrical and computer engineering with the University of Minnesota, Minneapolis, Minnesota, where he is a member of the Graduate Faculty of biomedical informatics and computational biology. From 2004 to 2005, he was a lecturer of computation and neural systems with Caltech. He was with Marconi Canada, CAE Electronics, Toshiba, and Fujitsu Research Labs. He was a recipient of the Charl H. Wilts Prize for the Best Doctoral Research in Electrical Engineering at Caltech, the Best Paper Award at the Design Automation Conference, and the U.S. National Science Foundation CAREER Award. He is a senior member of the IEEE.

Kia Bazargan (SM'07) received the BSc degree in computer science from Sharif University, Tehran, Iran, and the MS and PhD degrees in electrical and computer engineering from Northwestern University, Evanston, Illinois, in 1998 and 2000, respectively. He is currently an associate professor in the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, Minnesota. He was a recipient of the US National Science Foundation Career Award, in 2004. He was a guest co-editor of the *ACM Transactions on Embedded Computing Systems* Special Issue on Dynamically Adaptable Embedded Systems in 2003. He was on the technical program committee of a number of the IEEE/ACM-sponsored conferences, including Field Programmable Gate Array, Field Programmable Logic, Design Automation Conference (DAC), International Conference on Computer-Aided Design, and Asia and South Pacific DAC. He was an associate editor of the *IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems* from 2005 to 2012. He is a senior member of the IEEE Computer Society.