Polysynchronous Stochastic Circuits

M. Hassan Najafi, David J. Lilja, Marc Riedel, and Kia Bazargan
Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, USA
{najaf011, lilja, mriedel, kia}@umn.edu

Abstract— Clock distribution networks (CDNs) are costly in high-performance ASICs. This paper proposes a new approach: splitting clock domains at a very fine level, down to the level of a handful of gates. Each domain is synchronized with an inexpensive clock signal, generated locally. This is possible by adopting the paradigm of stochastic computation, where signal values are encoded as random bit streams. The design method is illustrated with the synthesis of circuits for applications in signal and image processing.

I. Introduction

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) that supplies a common reference signal to all synchronous components. Historically, the primary design goal for CDNs has been to ensure that a single clock signal arrives at every synchronous component at precisely the same time (so there is zero clock skew). 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 [1][2]. With increasing variation in circuit parameters, designing CDNs with tolerable clock skew is becoming a major design bottleneck.

Completely asynchronous design methodologies have been studied for decades, but these have never gained widespread acceptance (in spite of strong advocacy by proponents). Circuits with multiple independent clock domains – so circuits that are globally asynchronous, but locally synchronous (GALS) – have become common [3]. Splitting the domains reduces the cost of the distribution network, but relatively complex circuitry for handshaking is needed at domain crossings, so the splitting is only performed at a coarse level.

This paper proposes a radically new approach: splitting clock domains at a very fine level, with domains consisting of only a handful of gates each. Each domain is synchronized by an inexpensive clock signal, generated locally. This is feasible if one adopts a stochastic representation for signal values [4, 5, 6, 7, 8, 9]. Logical computation is performed on randomized bit streams, with signal values encoded in the statistics of the streams: a real value x in the interval [0, 1] is represented by a stream with bits each having independent probability x of being 1.

Compared to a binary radix representation, such a stochastic representation is not very compact. With M bits, a binary radix representation can represent 2^M distinct numbers. To represent real numbers with a resolution of 2^{-M}, i.e., numbers of the form $\frac{a}{2^M}$ for integers a between 0 and 2^M, a stochastic representation requires a stream of 2^M bits. The two representations are at opposite ends of the spectrum: conventional binary radix is a maximally compressed, positional encoding; a stochastic representation is an uncompressed, uniform encoding.

A stochastic representation, although not very compact, has an advantage over 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 designs in the stochastic paradigm. Complex functions can be implemented with remarkably simple logic. A reduction in area of 50x or 100x compared to conventional implementations is common [8][10].

A more compelling advantage still of the stochastic paradigm could be the idea advocated in this paper. 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. We will demonstrate that the correct value is computed even when the inputs are misaligned temporally. Accordingly, adopting the stochastic paradigm obviates the need for a global clock signal and the associated CDN. Instead one can simply use local clock signal generators throughout. We call this approach *polysynchronous stochastic* (to dis-

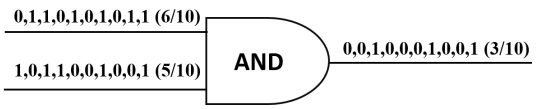


Fig. 1. Example of stochastic multiplication using an AND gate.

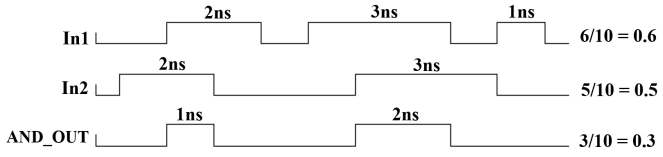


Fig. 2. Stochastic multiplication using an AND with unsynchronized bit streams.

tinguish it from asynchronous and GALS methodologies).

The paper is structured as follows. In Section II, we provide some background on stochastic computing. In Section III, we introduce polysynchronous stochastic concepts and demonstrate how to implement basic operations. In Section IV, we describe our experimental methodology and present experimental results. Finally, in Section V, we present conclusions and discuss future work.

II. BACKGROUND

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 [5, 8]. 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) [6].

Critical to the ideas in this paper is the observation that the stochastic representation is a uniform fractional representation: all that matters is the fraction of time that the signal is high. Consequently, precise synchronization between the arrival time of input values to logic gates does not matter. This is illustrated in the next section.

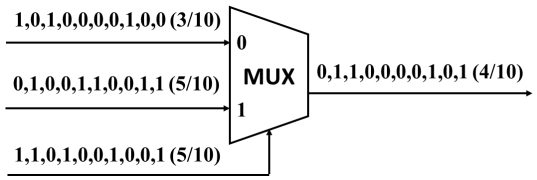


Fig. 3. Example of stochastic scaled addition using a MUX unit.

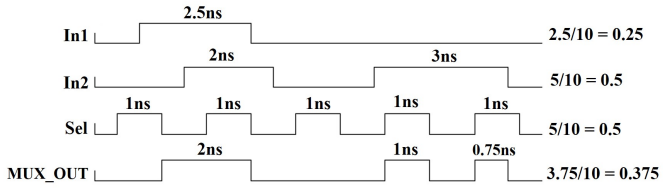


Fig. 4. Stochastic scaled addition using a MUX with unsynchronized bit streams.

A. Stochastic Operations

Multiplication Multiplication can be implemented using a standard AND gate [6]. Fig. 1 shows the multiplication of two 10-bit stochastic streams using an AND gate.

The value represented by a bit stream is the time that the signal is high divided by the total length of the stream. Fig. 2 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.

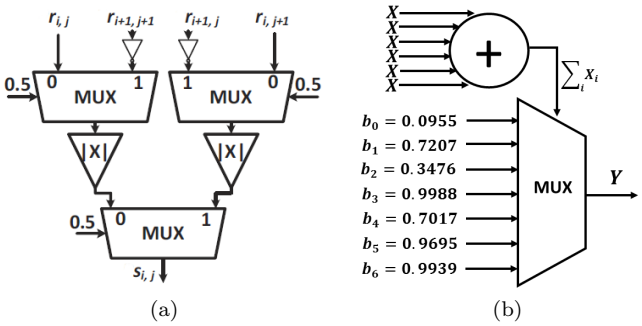
Scaled Addition and Subtraction Stochastic values are restricted to the interval [0, 1] (in the unipolar case) or the interval [-1, 1] (in the bipolar case). So one cannot perform addition or subtraction directly, since the result might lie outside these intervals. However, one can perform scaled addition and subtraction. These operations can be performed with a multiplexer (MUX). Fig. 3 illustrates the operation $\frac{1}{2}A + \frac{1}{2}B$.

Fig. 4 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.

B. Stochastic Circuits

Stochastic computing has been applied to a wide variety of applications, ranging from image processing [8, 11, 12, 13, 14, 15] to decoding of low-density parity check codes [16, 17, 18]. In this paper, we use the stochastic implementations of three digital image processing algorithms as case studies to evaluate the polysynchronous stochastic paradigm.

Robert's cross edge detection A stochastic implementation of Robert's cross edge detection algorithm, pro-



(a) (b)

Fig. 5. a) Stochastic implementation of the Robert's cross edge detection algorithm. For details of the implementation, the readers are referred to [8] b) Stochastic implementation of the gamma correction function using a MUX-based Bernstein polynomial architecture [6].

posed in [8], is shown in Fig. 5.a. Each operator consists of a pair of 2×2 convolution kernels that process an image pixel based on its three neighbors as follows:

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

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. Since this circuit works with signed values, all streams should be in the bipolar format.

Gamma correction Gamma correction is a nonlinear function used to code and decode luminance and tristimulus values in video and image processing systems. The required function in the gamma correction processes is $f(x) = x^\gamma$, where x is the value of a pixel in a given gray-scale image and γ is the gamma factor. For example, $\gamma = 0.45$ is the gamma value used in most TV cameras. A stochastic architecture for computing functions such as gamma correction was proposed in [6].

An example of stochastic gamma correction is shown Fig. 5.b. The inputs to this system consist of six independent bit streams, each with probability corresponding to the value x of the input pixel, as well as seven random bit streams 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 $b_6 = 0.9939$. For further details, readers are referred to [6].

Noise Reduction using a Median Filter An efficient technique for noise reduction in gray-scale images is to use a median filter. It replaces each pixel with the median value of its neighboring pixels. So the operation considers a local window around each pixel, computes the median value of the pixels inside that window, and replaces the pixel with the computed value. A stochastic implementation of a 3×3 median filter was proposed in [8].

C. Stochastic Number Generators (SNG)

A premise for stochastic computing is the availability of stochastic bit streams with the requisite probabilities. Such streams can either be generated from physical random sources [19, 20] or with pseudo-random constructs such as linear feedback shift registers (LFSRs). Given an input value, say in binary radix, the procedure for generating a stochastic bit stream with probability x is as follows. Obtain an unbiased random value r from the random or pseudorandom source; compare it to the target value x; output a one if $r \leq x$ and a zero otherwise.

III. POLYSYNCHRONOUS STOCHASTIC CIRCUITS

As was discussed in the introduction, stochastic computation has the advantage that it can tolerate input values that are misaligned temporally. Consequently, we can eliminate a global clock and instead use local clocks. Alternatively, we can continue to use a global clock, but relax the arrival time requirements to different components. We call this approach "polysynchronous clocking." First we will discuss the functionality of basic stochastic computational elements, such as the multiplier (an AND gate) and the scaled adder (a MUX unit) when these receive inputs driven by different clock sources. Next we discuss the polysynchronous scheme more generally. Finally, we present and evaluate three examples of image processing circuits, driven by polysynchronous clocks.

A. Basic Stochastic Operations with Polysynchronous Inputs

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 show how an AND gate works when two unsynchronized clocks are connected directly to its inputs. Then we discuss the behavior with arbitrary stochastic input streams. Connecting two clocks with 50% duty cycles directly to the inputs of an AND gate 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=2ns, T2=3.5ns, 2) T1=2ns, T2=3.2ns, and 3) T1=1.8ns, T2=3.2ns. Fig. 6 illustrates the input signals as well as the output signal in the case where T1=1.8ns and T2=3.2ns for 20ns of operation. Continuing the operation for about 1000ns 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 I presents results for the three selected cases of clock

TABLE I
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 1000NS.

T1=2ns T2=3.5ns		T1=2ns T2=3.2ns		T1=1.8ns 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 II
THE MEASURED OUTPUT OF THE MUX WHEN THREE POLYSYNCHRONOUS CLOCKS WITH DISTINCT PERIODS ARE CONNECTED TO ITS INPUTS FOR 1000NS.

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

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 I, 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 250ns. Dividing 250ns by 1000ns 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 main difference between this unit and the AND gate performing multiplication is 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 inputs. 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 II. 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 III 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 III are based on bit streams 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 stochastic computation [6].

TABLE III
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)	Meas.	Expec.	Meas.	Expec.
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

B. Stochastic Circuits with Polysynchronous Inputs

We extend our analysis of polysynchronous clocking to more complex stochastic circuits, namely the stochastic image processing circuits discussed in Section II. Suppose that we are given an input 4x4 gray-scale image to process by a stochastic Robert's cross edge detection circuit. An efficient way of processing the image is to use 16 instances of the Robert's cross stochastic circuit to process each of the pixels concurrently. Fig. 7 shows a diagram of such a parallel circuit. Call each instance a Robert's cross cell. Each cell has its own local clock; it converts its input pixel value, presented as a stochastic bit stream, into an output pixel value, presented as stochastic bit stream. The input bit stream is generated by an SNG driven by the cell's local clock. The cell communicates with its neighbor cells to receive their pixel values, presented as stochastic bit streams. These bit streams arriving from neighboring cells are generated by SNGs driven by their local clocks, so the input bit streams will not all be synchronized. The clocks will potentially all have different frequencies and phases.

Consider the first cell in Fig. 7. This cell is responsible for processing the image pixel 1 to decide whether it is on an edge or not. This cell takes a pixel intensity value and converts it to a stochastic bit stream using an SNG driven by a local clock. It does so while receiving streams corresponding to the values of neighboring pixels 2, 5, and 6. The pulses that the first cell receives from cells 2, 5, and 6 are all generated by SNGs driven by local clocks. Accordingly, the input bit streams are all potentially misaligned

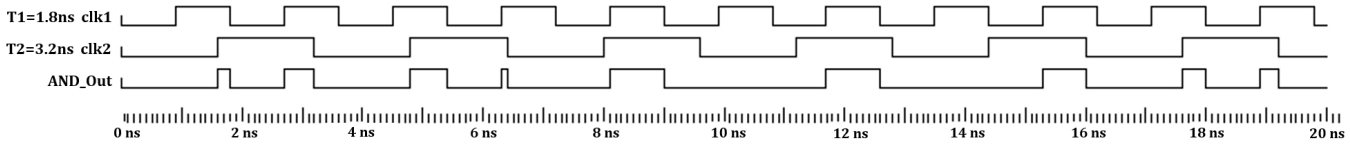


Fig. 6. Input clock signals and the corresponding output from connecting polysynchronous inputs to an AND gate.

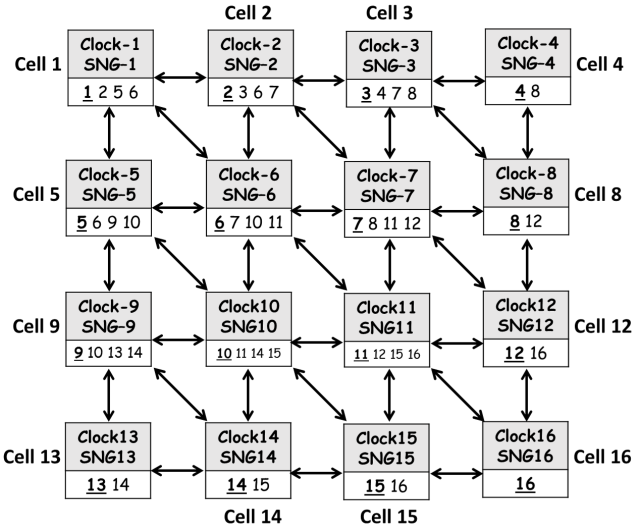


Fig. 7. 16 Robert's Cross Cells processing a 4x4 input image concurrently.

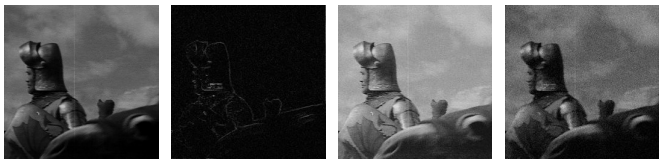
temporally. Nevertheless, as the results in the next section show, the computation is accurate.

IV. EXPERIMENTAL RESULTS

A. Methodology

We implemented the three stochastic image processing circuits discussed in Section II, namely the circuit for Robert's cross edge detection, Gamma correction, and noise reduction, in Verilog. For the Robert's cross circuit, three out of four streams are received asynchronously with respect to the local clock of each cell. Similarly, for the noise reduction circuit, eight out of nine streams are received asynchronously with respect to the local clock of each cell. For the Gamma correction circuit, we generate the bit streams for the Bernstein coefficients streams with SNGs driven by local clocks. We also generate the bit streams for the independent copies of the input value x using SNGs driven by local clocks.

We selected a 256×256 sample input image, so an image with 65536 pixels, for our simulations. The simulations were performed using the ModelSim hardware simulator. We implemented the SNG unit described in [6] for converting input pixel values into stochastic bit streams, using a 32-bit maximal period LFSR. This pseudorandom number generator was seeded with a random initial value for each trial; 10 trials were performed to ensure statistically significant results. Bit streams of length 1024 were used



(a) (b) (c) (d)

Fig. 8. a) The original 256×256 sample image, and the outputs of processing the input image using: b) Stochastic Robert's cross edge detection, c) Stochastic Gamma Correction, d) Stochastic Noise Reduction Median Filtering, with synchronized local clocks.

to represent the values. To convert the output bit streams into deterministic real values, we measured the fraction of the time the output signal is high and divide by the total time of the computation. For example, if the output signal was high 25% of the time, it is evaluated as 0.25.

To evaluate the correct functionality of our polysynchronous circuits, we first generated a "golden case" – we processed the sample image using stochastic circuits with all local clocks synchronized. Fig. 8 shows the sample image and the resulting output images. The synchronized clocks had a period of 2ns.

For the experiments we compare six different clocking schemes when evaluating the three stochastic image processing circuits:

Scheme 1. The period of the local clock in all processing cells is fixed at 2ns (the 'golden case" above).

Scheme 2. The period of the local clock in each cell is a random real value between 2-3 ns (so 50% variation between the clock periods).

Scheme 3. The period of the local clock in each cell is a random real value between 2-4ns (so 100% variation).

Scheme 4. The periods of the local clocks are random values between 2-4ns, but high output pulses that are less than 10% of the 2ns clock period (0.2ns) are filtered out (i.e., they are set to 0).

Scheme 5. Same as Scheme 4, but we filter out high output pulses that are less than 15% of the 2ns clock period.

Scheme 6. Same as Scheme 4, but we filter out high output pulses that are less than 20% of the 2ns clock period.

The motivation for schemes 2 and 3 is to study the impact of having more variation between the local clocks. For schemes 4, 5 and 6, it is to approximate hardware conditions in which short pulses – call them "runt pulses" – do not reach a valid high or low level, and so cannot change the output states of the gates.

TABLE IV
THE MEAN OF THE OUTPUT ERROR RATES FOR THE THREE IMPLEMENTED STOCHASTIC CIRCUITS, SIMULATED IN SIX DIFFERENT CLOCKING SCHEMES.

	Clocking Schemes					
Circuit	S.1	S.2	S.3	S.4	S.5	S.6
Robert.	2.88%	2.89%	2.94%	2.89%	2.92%	2.88%
Gamma.	2.56%	2.50%	2.49%	2.51%	2.59%	2.64%
Median.	3.15%	3.19%	3.31%	3.28%	3.39%	3.31%

B. Simulation Results

Starting with the stochastic Robert's cross circuit, we simulate the processing of the sample image using the six schemes described above. We simulate each one of the three stochastic circuits based on these six schemes 10 times, each time with different initial conditions: 10 different LFSR seed values for each SNG and 10 different sets of values for the periods of the local clocks. The results are the average results of these trials. For each output image we calculate the average output error rate as follows:

$$E = \frac{\sum_{i=1}^{256}\sum_{j=1}^{256}|T_{i,j} - S_{i,j}|}{255.(256 \times 256)} \times 100$$

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

Table IV shows the mean of the error rates of the results produced by processing the sample image with the six schemes described above. By comparing the measured accuracies of the first scheme, i.e., the "golden case", to the five polysynchronous schemes, we conclude that the quality of the results and the accuracy of the computations are essentially independent of how well synchronized the local clocks are. In fact, as this table shows, the clock periods can vary by up to 100% without affecting the accuracy of the results.

As can be seen in Table IV, in some cases, the mean of the error in the polysynchronous circuits is actually slightly below that of the synchronous case. This improvement can occur because polysynchronous clocks can produce more random-looking input streams. So polysynchrony might actually help instead of hurting stochastic computation! The results from schemes 4-6 show that filtering out runt pulses still produces statistically acceptable results.

C. SPICE-Level Verification

For a circuit-level verification of the proposed 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 500 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 1ns. For the polysynchronous clocking conditions, clock periods were selected randomly in the range from 1ns to 2ns (so 100% 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 500 trials, we found that the mean of the output error rates was 4.91% for the synchronous and 4.42% for the polysynchronous approach. Hence, we can conclude that polysynchronous stochastic circuits are essentially as accurate as conventional synchronous circuits.

V. CONCLUSIONS AND FUTURE DIRECTIONS

This paper presented a novel paradigm for sequential computation that is synchronous, but only loosely so. It is predicated on the paradigm of stochastic computing. 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 consist of multiplications and additions. Such functions can be implemented with remarkably simple logic in the stochastic paradigm. Simple hardware generally translates to low power consumption. Certainly, it translates to low leakage power consumption, a metric of eminent concern to modern integrated circuit designers. Admittedly, stochastic computations are slow due to the length of the bit streams. The latency scales very poorly with accuracy. However, simple logic translates to short critical paths, so stochastic circuits can potentially run at much higher clock rates.

Another important benefit of the stochastic paradigm is the flexibility that it provides with respect to the clocking mechanism. Indeed, stochastic logic computes accurately irrespective of the temporal alignment of input values, so it can tolerate arbitrary amounts of clock skew. As a result, we can replace a global clock and its associated clock distribution network with locally generated clocks. These can be simple, fast inverter rings, for instance.

This paper discusses polysynchronous stochastic computation as a proof of concept. We demonstrate the critical feature that the accuracy of the computation is not impacted if a global clock is replaced with unsynchronized local clocks. We did not perform a full implementation to quantify the savings obtained by eliminating the CDN in integrated circuits. Obviously, the area, power and design complexity will all be impacted in positive sense if we can eliminate the CDN. Also, we did not discuss the input and output mechanisms needed to support the stochastic paradigm.

Indeed, the question of how unsynchronized values can

interface with non-stochastic logic is an important one. In our vision, circuits will process stochastic values from input to output. We have been exploring efficient mechanisms for analog-to-digital (A/D) conversion at the circuit inputs, with devices that generate stochastic bit streams directly from analog sources. These devices – essentially modified sigma-delta converters – are highly efficient. They provide random bit streams at no extra cost; in fact, they are significantly less costly in terms of area and power than full sigma-delta converters. Similarly we are exploring highly efficient digital-to-analog (D/A) converters for the circuit outputs. These produce accurate analog signals directly based on the fraction of time that digital signals are high, irrespective of pulse widths. Our polysynchronous stochastic approach provides exactly this form of output: the values correspond to the fraction of time that signals are high, without any concern for pulse widths.

We conclude that the polysynchronous stochastic approach is a good fit for applications that require only modest accuracy but call for low cost, low power and high resiliency. In future work, we will make this case with a comparison of full integrated circuit designs, including clock mechanisms and I/O circuitry.

VI. Acknowledgement

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.

References

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

[2] 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.

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

[4] 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.

[5] 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.

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

[7] 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 *Computer-Aided Design, 2012. ICCAD 2012. IEEE/ACM International Conference on*. IEEE, 2012.

[8] 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, March 2014.

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

[10] 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.

[11] 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.

[12] 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.

[13] M.H. Najafi and M.E. Salehi. A fast fault-tolerant architecture for Sauvola local image thresholding algorithm using stochastic computing. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, May 2015.

[14] M. Ranjbar, M.E. Salehi, and M.H. Najafi. Using stochastic architectures for edge detection algorithms. In *Electrical Engineering (ICEE), 2015 23rd Iranian Conference on*, pages 723–728, May 2015.

[15] Bingzhe Li, M.H. Najafi, and D.J. Lilja. An FPGA implementation of a restricted boltzmann machine classifier using stochastic bit streams. In *Application-specific Systems, Architectures and Processors (ASAP), 2015 IEEE 26th International Conference on*, pages 68–69, July 2015.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.