A Variable-Latency Architecture for Accelerating Deterministic Approaches to Stochastic Computing

Alexander J. Groszewski and Earl E. Swartzlander, Jr.
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712 USA
groszewa@utexas.edu, eswartzla@aol.com

***Abstract*—Traditional Stochastic Computing methods suffer from high latency due to the required bit stream length and the area incurred from generating them. Using deterministic bit streams reduces this area, greatly improves latency, and produces fully-accurate results. However, even though latency is improved, it is still high. This paper presents multiple novel, variable-latency architectures which exhibit superior performance and energy efficiency when compared to current state-of-the-art designs.**

***Index Terms*—Stochastic Computing, variable latency**

I. Introduction

Stochastic Computing (SC) refers to a set of techniques which operate on probabilistic bit streams [1]. The main benefits to this type of representation are that complex arithmetic operations can be performed with incredibly simple logic and the underlying number representation is intrinsically resilient to Single Event Upsets (SEUs). However, these benefits also come at a cost. Due to correlation between streams and random fluctuations, stochastic computations tend to suffer from errors when bit streams are short. Protecting against this requires longer bit streams, which causes SC to exhibit high area and long runtimes. However, recent work suggests that randomness is not actually a requirement for stochastic operations, and that deterministic bit streams can replace stochastic ones with improved results [2]. Using deterministic bit streams, all approximation is removed and the results are fully accurate. Rather than increasing the length of the bit streams to increase precision, the number of cycles required to obtain a fully-accurate result is known *a priori* which leads to improved performance. Furthermore, deterministic bit streams can be generated with simple counters which eliminates the need for costly hardware pseudo-random number generators, leading to significant area reduction.

Deterministic approaches to Stochastic Computing (DSC) is clearly an improvement on traditional SC methods, exhibiting reduced area and latency and producing fully-accurate results. However, even though latency is improved, it is still relatively high and in many cases this could be the deciding factor against choosing a stochastic design for a given application. For this reason, improvements to performance and energy-efficiency are crucial to making DSC viable for adoption into real-world applications. In this work, we propose multiple novel, variable latency-architectures which run DSC operations for only as many cycles as absolutely necessary. At the very minimum, this architecture boasts an 50% average improvement over traditional DSC methods with extremely little hardware overhead cost. Through some extra design effort and area overhead in explicitly ordering inputs, this improvement is seen to be as high as 70% and increases with the number of inputs.

II. Background

A. Stochastic Computing

A stochastic bit stream $X(t)$ of length L (where t = 1, 2,... L) is defined in time and has two possible coding formats: unipolar and bipolar. In the unipolar format, $X(t)$ is mapped to a real number x in the unit interval such that x is the number of 1's that appear over the total length of the stream. For example, using a 4-bit stream the representable values are $\{0, 0.25, 0.5, 0.75, 1\}$ and the number 0.75 can be represented by any of the four streams: 0111, 1011, 1101, 1110. In the bipolar format, the range of x is extended to $[-1, 1]$ and the probability is defined as the number of 1's minus the number of 0's in a stream over the total length.

SC allows for incredibly simple implementations of arithmetic operations. For example, multiplication between two bit streams can be accomplished using a single AND gate while scaled addition is done with a 2:1 MUX (Figure 1).

There are two main causes of accuracy loss in SC: fluctuations inherent in random numbers and correlation between stochastic bit streams being processed. The astute reader might notice that certain encoding combinations in Figure 2-1(a) will produce incorrect results. For example, if the same inputs were instead encoded as 111000 and 110000 (maximally correlated), then the result would incorrectly be 1/3. For this reason, bit streams must be generated using random or pseudo-random sources in order to ensure correlation between inputs is minimized.

Large, costly Linear Feedback Shift Registers (LFSRs) are typically used as pseudo-random Stochastic Number Generators (SNGs) which convert binary numbers to stochastic bit streams that are suitably uncorrelated. Each LFSR has a length requirement to ensure suitable randomness and independence: research has shown that in a system with i inputs with n-bit resolution, each LFSR must have at least length $2ni$ [2].

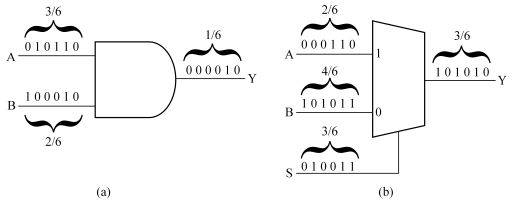

Fig. 1. Stochastic implementation of arithmetic operations: (a) multiplication and (b) scaled addition (after [2]).

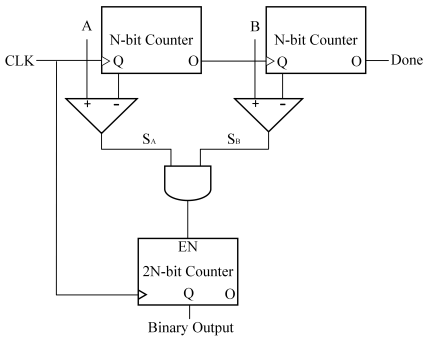

Fig. 2. N-bit clock-division DSC multiply.

Converting a stochastic bit stream to binary is relatively simple and can be achieved using a simple counter.

In addition to the aforementioned concerns regarding area due to costly SNG circuits, there is also the issue of bit stream length requirements that lead to high latency operations. In order to obtain a stochastic number within an error margin ϵ, minimum bit stream length N is given by (1).

$$N > \frac{(p)(1-p)}{\epsilon^2} \tag{1}$$

Given that the maximum value the $p(1-p)$ term can be is 2^{-2} , this means that in order to represent a stochastic number with binary resolution 2^{-n} then the error margin ϵ must equal $2^{-(n+1)}$, where n is the binary bit width [3]. Therefore, in order to represent a number with this resolution 2^{-n} stochastically, the bit stream must be greater than 2^{2n} bits long [2]. This means that stochastic bit streams grow exponentially with the desired resolution, leading to very long bit streams which limit overall performance.

B. Deterministic Approaches to Stochastic Computing

Recent work suggests randomness is actually not a requirement for computations on bit streams, and that deterministic bit streams can replace stochastic ones with improved results [2]. The authors argue that in traditional stochastic methods, computation is happening in a statistical sense on the "average" number of 1's and 0's. That is, because the probability a bit stream represents is equal to its expected value, it can alternatively be viewed as the number of 1's and 0's that one would expect to see on average. Guaranteeing that stochastic bit streams are suitably random, uncorrelated, and independent allows them to passively maintain the property that the average bits of one stream are operated by the average bits in the other. However, if one could actively enforce the property of every bit in one stream being operated on by every bit in another stream rather than leaving it up to probability, then the result is guaranteed to always be fully-accurate. This is akin to convolution between bit streams.

The key to achieving this "convolution" between bit streams lies in the structure of the SNG, of which there are three main approaches: relatively prime bit lengths, clock-division, and rotation. The relatively prime bit length method ensures the "convolutional" property by guaranteeing that all of the bit streams in a computation have lengths which are relatively prime to one another. Clock-division works by clock-dividing operands and rotation explicitly rotates bit streams, both methods allow for bit streams of arbitrary lengths.

There are two main benefits to producing deterministic bit streams in this way. Firstly, large LFSRs which are longer than $2ni$ bits wide can be replaced with more area-efficient counters of length n (where n is the binary bit width and i is the number of input variables). Secondly, the number of cycles to complete a given operation is known *a priori* and yields fully-accurate results. Replacing long, stochastic bit streams with deterministic ones can show a cycle time reduction of as much as $\frac{1}{2^n}$ [2].

An illustrative example of how a multiplication between deterministic bit streams using the clock-division method can be seen in Figure 2. The overflow output of the "A" counter (i.e. maximum representable value transitions to zero) is active for 1 cycle and acts as the clock to the "B" counter. The overflow of the "B" counter acts as a "done" signal which indicates operation is complete.

Much research has been aimed at improving Stochastic Computing circuits in terms of latency. One way of improving performance is to organize the structure of the bit stream representation such that it exhibits progressive precision, a property that guarantees that initial parts of a bit stream provide steadily improving estimates of its true value [4]. However, one trait of this method is that accuracy is traded off for performance, which in many cases is not an option. This work opts to always retain full accuracy. Recent work using deterministic bit streams has shown that binary inputs can be broken down into lower-resolution parts which can be processed in parallel and then combined at the end of the operation [5]. Another work pre-loads bit streams into shift registers and computes up to 2^n operations in parallel, leading to a cycle time reduction of up to $\frac{1}{2^{n-1}}$ [6]. However, two common trait shared by parallel DSC methods is the fact that extra area is added to achieve parallelism and binary operations are introduced into the stochastic operation which takes away from the error tolerance capability. Finally, [7] exploits input data encoding by utilizing shuffling networks to only run an operation for the minimum number of cycles. However, while this method shows great performance and area improvements, the main drawback is that it only works for single variable functions (e.g. $f(x) = x^3$).

III. Variable Latency Methods

A. Architectural Overview

The fact that convolution via clock-division produces unary or "thermometer-coded" bit streams (all 1's followed by all 0's) leads to certain DSC operations completing accumulation earlier than others. This difference in time to completion is dependent on the values of the inputs. This work presents a novel circuit architecture deemed "early shutoff" (ES) which takes advantage of the value of the clock-divided bit stream in order to determine when an operation can be terminated early. A simple 2-bit multiplication example can be seen in Table I. While the required number of cycles to complete a traditional DSC multiplication between two 2-bit binary numbers is 16 (2^{ni}, n=2, i=2), it can be seen that the output of the multiplication for the upper operation is ready earlier than the 16^{th} cycle, when the "done" signal would typically be asserted from the overflow of the "B" counter. Once the value of the clock-divided bit stream becomes 0 in cycle 13, the remaining cycles do not accomplish any useful work and the operation can be terminated early. Therefore, a new form of the "done" signal can be derived which is dependent on both the overflow of the clock divided SNG and the complement of the clock-divided bit stream. The lower operation in the table shows that this same operation can be optimized further if the input operands are flipped such that the smallest operand is the one which is clock-divided. In this example, the operation can be terminated in cycle 9 (4 cycles earlier than the previous example). Traditional DSC methods can be seen as imposing a rigid "performance envelope" of 2^{ni} cycles on any given operation. ES methods, on the other hand, exhibit performance values which vary as a function of the input operands. A visual representation of what this means can be found in Figures 3 and 4, which profile an ES operation for n=4 and i=2. Using a naïve (unordered) ES approach leads to each of the possible 2^{ni} input combinations being split into bins of size 2^n (each of which are size 2^n) (Figure 3). Smaller values of the clock-divided input lead to shorter runtimes and the value of the other input does not matter. Applying implicit ordering to guarantee the clock-divided operand is the smallest still splits the possible combinations into 2^n bins, but now the bins are of varying sizes (Figure 4). In fact, the largest bin is the one with the best performance, and the size of the bins monotonically decrease as performance gets worse. Due to the fact that now the size of both inputs is taken into consideration, the probability that an input combination will fall into a bin with desirable performance is increased.

It is useful to derive a single performance number for comparison between traditional and ES methods. If we assume that all inputs are possible with equal probability, this can be viewed as an expected value problem. For both naïve and explicitly ordered ES methods, the latency of an operation as a function of the clock-divided input x_{div} is given by (2).

$$latency(x_{div}) = \begin{cases} 1 & x_{div} = 0 \\ x_{div} \cdot 2^{(i-1)n} & else \end{cases} \tag{2}$$

For the naïve approach, the probability of falling into a particular bin is simply given by (3). The expected value is calculated as the sum of all possible values each multiplied by the probability of its occurrence (4).

$$P(x_{div} = X)_{naive} = \frac{1}{2^n} \tag{3}$$

$$\begin{aligned} \mathbb{E}[latency]_{naive} &= \sum_{x=0}^{2^n} latency(x) \cdot P(x_{div} = x) \\ &= \frac{1 + \sum_{x=1}^{2^n} x \cdot 2^{(i-1)n}}{2^n} \end{aligned} \tag{4}$$

The probability of falling into a particular bin when explicit ordering of inputs is introduces is a bit more complicated as it is a now a function of the clock-divided input (5), which is then used to find the expected value in (6).

$$P(x_{div} = X)_{ord} = \frac{(2^n - x_{div})^i - (2^n - x_{div} - 1)^i}{2^{ni}} \tag{5}$$

TABLE I
2-bit DSC multiply using clock-division

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A < B																
A=2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
B=3	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
A*B	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0
output ctr	1	2	2	2	3	4	4	4	5	6	6	6	6	6	6	6
B < A																
A=3	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0
B=2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
A*B	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0
output ctr	1	2	3	3	4	5	6	6	6	6	6	6	6	6	6	6

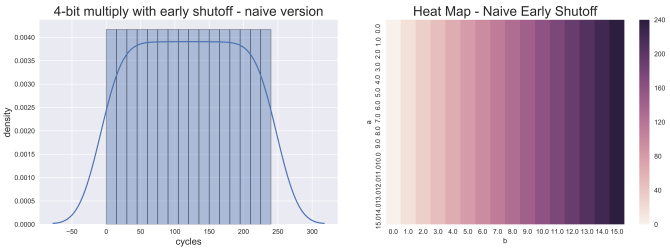

Fig. 3. Density plot and input heatmap for naïve ES.

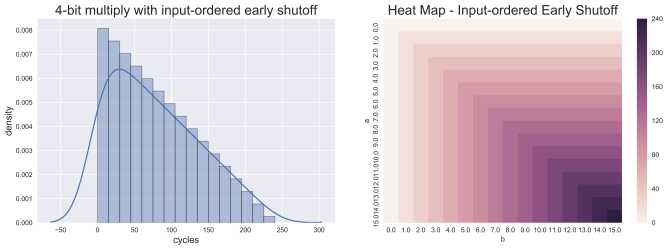

Fig. 4. Density plot and input heatmap for explicitly ordered ES.

$$\mathbb{E}[latency]_{ord} = \frac{1 + \sum_{x=1}^{2^n} (2^n - x_{div})^i - (2^n - x_{div} - 1)^i}{2^{ni}} \tag{6}$$

Dividing (4) by the equivalent number of cycles in a traditional DSC operation (2^{ni}) yields the performance improvement over the baseline and can be simplified to (7).

$$\frac{\mathbb{E}[latency]_{naive}}{latency_{DSC}} = \frac{1}{2} + 2^{-(n+1)} + 2^{-n(i+1)} \tag{7}$$

Taking the limits with respect to n and i lead to (8) and (9), respectively.

$$\lim_{n\to\infty} \frac{\mathbb{E}[latency]_{naive}}{latency_{DSC}} = \frac{1}{2} \tag{8}$$

$$\lim_{i\to\infty} \frac{\mathbb{E}[latency]_{naive}}{latency_{DSC}} = \frac{1}{2} + 2^{-(n+1)} \tag{9}$$

As the number of bits approaches infinity, the improvement percentage approaches a constant (non-zero) value of $\frac{1}{2}$. Doing the same with the number of inputs, the percentage becomes purely a function of the number of bits. This proves that the most improvement that can ever be obtained from the naïve implementation is $\frac{1}{2}$. However, the same does not hold true for the version with explicit ordering of inputs. Dividing (6) by 2^{ni} does not have as clean of a solution as (7). However, plots of the proportion across various numbers of inputs and bit widths (Figure 5) show that as n approaches infinity the proportion approaches a non-zero number (which is always less than $\frac{1}{2}$) while the limit as i approaches infinity actually converges to zero! This is because with more inputs, the probability of falling into a high latency bin gets smaller while the probability of falling into a lower latency bin gets larger. So while explicit ordering of inputs shows further improvement on the naïve implementation with respect to increasing input resolution, it can actually provide considerably larger performance improvements as the number of inputs to an operation are scaled up.

B. Methods for Explicitly Ordering Inputs

Ordering of binary values can be done using a Compare and Shift (CAS) block. A form of the CAS block is used in standard Floating-Point addition units to guarantee the input with the larger exponent is always fed into a particular data path. However, this work attempts to perform input ordering on bit streams as well. Prior work has been done to develop hardware sorting units to order bit streams for use in image processing and switching systems [8] and can be seen in Figure 6. While traditional SC operations strive to reduce correlation of inputs as much as possible, it has been shown that increasing correlation to the maximum amount between deterministic bit streams can lead to optimized circuitry for specific operations such as calculating the minimum (MIN) or maximum (MAX) of two bits streams. Since the thermometer coding of data leads to maximally correlated inputs, this allows for the MIN and MAX operations to be done with a single AND and OR gate, respectively. This work uses a similar method for guaranteeing bit stream ordering.

While Figure 6 shows how a 2-input binary and stochastic CAS block can be built, this work also explores operations with more than two inputs as well. Binary CAS blocks of more than 2 inputs can be created using 2-input CAS blocks as building blocks. 2 and 3-input binary CAS circuits utilized in this work can be seen in Figure 7. Unary CAS units are much more compact than their binary counterparts. While both 3 and 4-input unary CAS find the minimum and maximum value identically to the 2-input variant (with more inputs to the gates), extra circuitry is required to find the remaining values. For a 3-input unary CAS, the middle value is obtained through the circuit in Figure 8a. A 4-input unary CAS requires circuitry to find the second smallest and second largest inputs, which can be seen in Figure 8b and 8c, respectively.

IV. Evaluation

In this section, variable-latency designs will be compared to traditional DSC methods with respect to area, performance, worst-case timing path, power, and energy. The target circuit will be multiplication and will be designed across multiple resolutions and numbers of inputs. The reference DSC multiply circuit can be seen in Figure 2. The unordered (naïve) variable latency design is formed by creating a new "done" signal by feeding the original signal in Figure 2 and the complement of the most clock-divided bit stream into an OR gate. This work achieves explicit ordering in two ways: one does so on the binary inputs using a binary CAS block while the other does the ordering at the bit stream level and uses unary CAS blocks. In the former case, the CAS block uses the same circuit as the naïve case, but the inputs are first fed through the CAS block. The latter case requires i comparators per input which are then

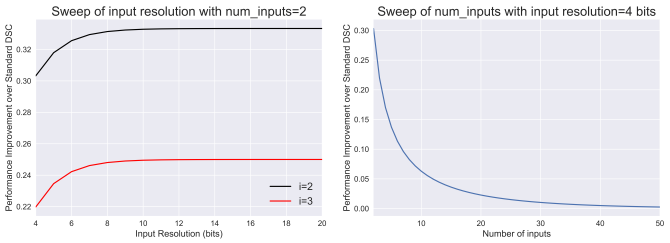

Fig. 5. Scaling of ES with explicit ordering across bit width and inputs.

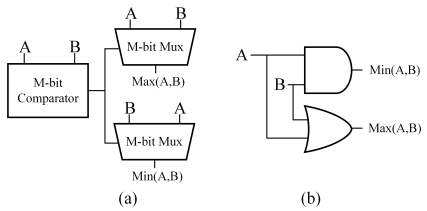

Fig. 6. Implementation of CAS block a) binary and b) unary (after [8])

.

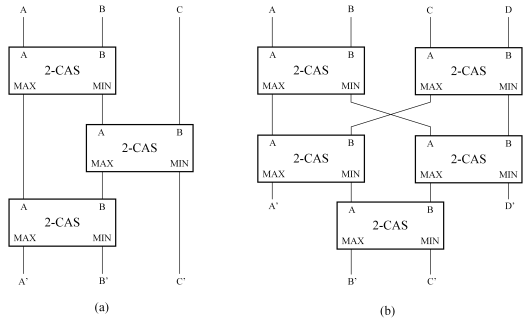

Fig. 7. Binary CAS modules for a) 3-inputs and b) 4-inputs.

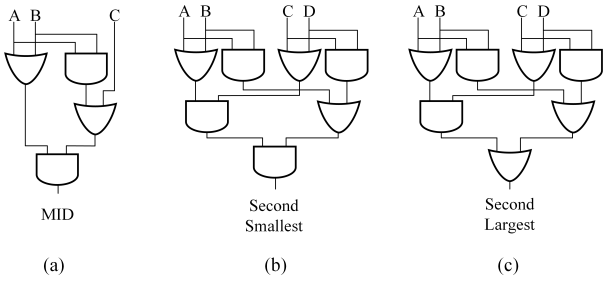

Fig. 8. Unary CAS modules for a) 3-input middle value b) 4-input second smallest value and c) 4-input second largest value.

followed by the associated unary CAS block. An example of a 2-input ES multiply with bitstream ordering can be seen in figure 9.

A. Experimental Methodology

The RTL for the designs is coded in Verilog and simulated/debugged using VCS and DVE. A unified SystemVerilog testbench with random stimulus generation, scoreboarding, error reporting, and average performance calculation across 1000 random input vectors has been developed. Synthesis is performed using Cadence RTL Compiler 2016 (RC) and is mapped to the FreePDK 45nm library. The highest synthesis effort possible is used in order to ensure the most compact designs. Area, worst case timing slack, and average power are also obtained through RC. Energy per operation is calculated by plugging corresponding values into (10).

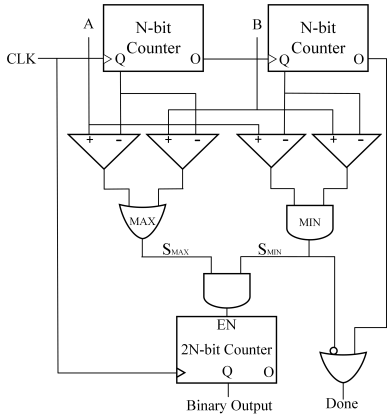

Fig. 9. N-bit DSC multiply with ES and bit stream ordering.

$$\frac{energy}{operation} = total\,power * \frac{cycles}{operation} * cycle\,time \quad (10)$$

For comparison purposes, the cycle time of 394.5 MHz was chosen since all designs would be able to converge to it based on their worst case timing slacks obtained through RC.

V. Results and Discussion

Results can be found in Table II. The method of implementing explicit ordering has no effect on performance, which is why both methods share a common row. The same can be said for the reference design (DSC), ES naïve, and CAS-ordered ES with respect to worst case timing. The binary CAS block is included in timing to show how this delay compares with the bit stream portion.

For less than 1% increase in both area and power consumption, the naïve ES architecture can show up to an impressive 50% improvement in latency compared to the traditional DSC design. However, this is the maximum improvement regardless of how the input resolution and number of inputs are scaled. Regardless, if an application is able to support a variable latency architecture then it can obtain a 50% improvement in both performance and energy efficiency at extremely little cost.

ES architectures with explicit ordering of inputs have the benefit of being able to show a performance improvement greater than the naïve implementation, which also increases with number of inputs. For example, a 4-bit multiply with 2 inputs using ES with explicit ordering sees an average performance improvement of 70%. The same 4-bit multiply with 3 inputs shows an improvement of 75%, and with 4 inputs it is 80%. However, this improvement comes at the cost of area which does not scale as well as the naïve case. A 4-bit multiply with 2 inputs using explicit ordering shows a 17% increase in area using a binary CAS and a 19% increase when ordering at the bit stream level, when compared to a traditional DSC multiply. This increase becomes 32% and 36%, respectively, with 3 inputs, and 40% and 53% with 4 inputs.

A common trait amongst all explicitly ordered ES architectures when compared to the traditional DSC design is that they all exhibit increased power dissipation. This is because the added area leads to more switching nets and thus higher dynamic power. However, even through the fractional increase in power of the ES designs compared to traditional DSC gets larger as the input resolution and number of inputs are scaled, the energy per operation is still always lower. This is because the rate at which power is growing is much smaller than the rate at which the number of cycles per operation is shrinking.

While explicit ordering at the bit stream level has higher energy and area than its counterpart using a binary CAS, its benefit comes when considering worst case timing paths. For each of the DSC multiplies, the timing path in question is always from the input SNG to the output counter. The delay of

TABLE II
RESULTS TABLE

Num inputs	2				3				4			
Bit width	4	6	8	10	4	6	8	10	4	6	8	10
Performance (averge cycles/operation)												
DSC	256	4096	65536	1048576	4096	262144	16777216	1073741824	65536	16777216	4294967296	1099511627776
ES naïve	120	2016	32640	523776	1920	129024	8355840	536346624	30720	8257536	2139095040	549218942976
ES ordered	77	1333	21717	349013	900	63504	4161600	267911424	11144	3225736	850626696	219365804168
Area (mm^2)												
DSC	463	672	898	1168	698	1037	1376	1766	934	1383	1836	2365
ES naïve	466	691	917	1171	700	1040	1380	1770	936	1387	1840	2369
ES ordered CAS	539	810	1075	1366	922	1406	1880	2366	1307	1997	2673	3362
ES ordered BS	550	820	1093	1393	958	1434	1911	2439	1462	2187	2916	3721
Worst Case Timing (ps)												
DSC/ES naïve + CAS	672	834	1034	1230	834	1127	1437	1745	1034	1437	1848	2233
ES ordered BS	708	870	1070	1266	934	1227	1537	1845	1138	1541	1952	2337
Binary CAS Block	442	692	625	826	1341	2070	2349	2465	1422	2106	2371	2535
Power (μW)												
DSC	240	338	428	515	354	500	634	777	463	655	836	1037
ES naïve	242	340	429	517	313	501	635	778	464	656	837	1038
ES ordered CAS	249	352	445	537	335	540	689	848	504	729	943	1166
ES ordered BS	263	379	479	580	444	628	795	978	640	903	1162	1432
Energy (pJ)												
DSC	160	3500	71000	1.4E+06	3700	3.3E+05	2.7E+07	2.1E+09	77000	2.8E+07	9.1E+09	2.9E+12
ES naïve	74	1800	36000	6.9E+05	1500	1.6E+05	1.3E+07	1.1E+09	36165	1.4E+07	4.5E+09	1.4E+12
ES ordered CAS	49	1200	25000	4.8E+05	770	8.7E+04	7.3E+06	5.8E+08	14000	6.0E+06	2.0E+09	6.5E+11
ES ordered BS	52	1300	26000	5.1E+05	1013	1.0E+05	8.4E+06	6.6E+08	18000	7.4E+06	2.5E+09	8.0E+11
Frequency all designs in column can converge to (GHz)												
with CAS	1.39	1.14	0.93	0.79	0.75	0.48	0.43	0.41	0.70	0.47	0.42	0.39
without CAS	1.39	1.14	0.93	0.79	1.07	0.81	0.65	0.54	0.88	0.65	0.51	0.43

the binary CAS block is considered separately. For simplicity, this work assumes that if a binary CAS block were to be used, it would be placed in the pipe stage directly before the input SNG flop and is not pipelined. Inspection of Table II shows that if the number of inputs is greater than 2 then the delay through the CAS block is always larger than the delay through the DSC multiply. The lower portion of Table II shows the cycle time that could be achieved with and without the inclusion of the binary CAS block, assuming that the CAS is not pipelined. So even through ordering at the bit stream level is larger and consumes slightly more energy than ordering with a binary CAS, the benefit lies in not having to add the extra delay to the pipe stage leading up the the DSC unit.

One final thing to keep in mind is that all of these simulations assumes a uniform probability of obtaining any input combination. In reality, this distribution is application specific. However, it is unlikely that this distribution will be skewed towards numbers in the higher end of the representable spectrum, so the average performance improvement will likely be higher than the numbers reported here.

VI. CONCLUSION

Improvements to performance and energy efficiency are crucial for improving the viability of DSC methods. As long as an application can accommodate a variable latency architecture, the naïve ES approach offers a 50% performance and energy improvement essentially for free. More performance can be gained at the cost of area by explicitly ordering inputs, especially if the given application contains operations with large numbers of inputs. Results indicate that the variable-latency architectures presented are a compelling alternative (or addition) to current state-of-the-art DSC techniques which can improve both performance and energy efficiency.

REFERENCES

[1] B. Gaines, "Stochastic computing systems," int *Advances in Informations Systems Science*, Boston, MA, Springer, 1969, pp. 1–6.

[2] D. Jenson and M. Riedel, "A deterministic approach to stochastic computing," in *IEEE/ACM Conference on Computer-Aided Design (ICCAD)*, Austin, TX, 2016.

[3] W. Qian, *Digital yet deliberately random: synthesizing logical computation on stochastic bit streams*, PhD Dissertation, University of Minnesota, 2011.

[4] M. Najafi, D. Lilja, and M. Riedel, "Deterministic methods for stochastic computing using low-discrepancy sequences," in *International Conference on Computer-Aided Design (ICCAD)*, San Diego, CA, 2018.

[5] M. Najafi, B. Li, D. Lilja, and K. Bazargan, "Accelerating deterministic bit stream computing with resolution splitting," in *International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, 2019.

[6] A. Groszewski and T. Lenz, "Deterministic stochastic computing using parallel datapaths," in *International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, 2019.

[7] Z. Wang, S. Mohajer, and K. Bazargan, , "Low latency parallel implementation of traditionally-called stochastic circuits using deterministic shuffle networks," in *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jeju, South Korea, 2018.

[8] M. Najafi, D. Lilja, M. Riedel, and K. Bazargan, , "Low-cost sorting network circuits using unary processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 8, pp. 1471–1480, 2018.