Deterministic Methods for Stochastic Computing using Low-Discrepancy Sequences

M. Hassan Najafi
University of Louisiana at Lafayette
Lafayette, Louisiana
najafi@louisiana.edu

David J. Lilja
University of Minnesota
Minneapolis, Minnesota
lilja@umn.edu

Marc Riedel
University of Minnesota
Minneapolis, Minnesota
mriedel@umn.edu

ABSTRACT

Recently, deterministic approaches to stochastic computing (SC) have been proposed. These compute with the same constructs as stochastic computing but operate on deterministic bit streams. These approaches reduce the area, greatly reduce the latency (by an exponential factor), and produce completely accurate results. However, these methods do not scale well. Also, they lack the property of progressive precision enjoyed by SC. As a result, these deterministic approaches are not competitive for applications where some degree of inaccuracy can be tolerated. In this work we introduce two fast-converging, scalable deterministic approaches to SC based on low-discrepancy sequences. The results are completely accurate when running the operations for the required number of cycles. However, the computation can be truncated early if some inaccuracy is acceptable. Experimental results show that the proposed approaches significantly improve both the processing time and area-delay product compared to prior approaches.

KEYWORDS

Stochastic computing, deterministic computing, scalable design, fast converging process, low-discrepancy sequences

ACM Reference Format:
M. Hassan Najafi, David J. Lilja, and Marc Riedel. 2018. Deterministic Methods for Stochastic Computing using Low-Discrepancy Sequences. In *IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN (ICCAD '18), November 5–8, 2018, San Diego, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3240765.3240797

1 INTRODUCTION

Stochastic Computing (SC) [3, 5, 9, 11, 13] is a well-established paradigm for low-cost, noise-tolerant approximate computing. Recent work has shown that the same constructs used for computation on stochastic bitstreams can be used for computation on deterministic bitstreams, if these bitstreams are generated in specific ways [6][8]. The results are completely accurate with no inaccuracy caused by random fluctuation or correlation.

Three different approaches have been suggested: using relatively prime stream lengths [8], clock dividing the streams, and rotating the streams [6]. So called unary bitstreams are used – stream of bits with a sequence of 1s followed by a sequence of 0s, where the fraction of 1's to the bitstream length represents the value that is being computed on. Operations must run for the product of the length of the input bitstreams to produce the correct result. For instance, when multiplying two n-bit precision input values represented by two 2^n-length bitstreams, the input streams are connected to the inputs of an AND gate. (Recall that multiplication is performed using a simple AND gate in SC.) The operation must run for 2^{2n} cycles to produce the correct output. (This is, in fact, exponentially fewer cycles that would be required with stochastic bitstreams [6]).

While unary stream-based deterministic methods are able to provide completely accurate results, they do not offer progressive precision [2, 10]. The output converges to the expected correct value slowly. This slow convergence makes the deterministic approaches inefficient for applications that can tolerate some inaccuracy (e.g., image processing and neural network applications). Limited scalability is a significant drawback of the approaches described in [8] and [6]. See the discussion in [3]. The required number of cycles and the cost of generating bitstreams both increase significantly with an increasing number of inputs. These parameters also increase with increasing precision of the input data.

Recently Najafi and Lilja [10] proposed a "down-sampling" method for the three deterministic approaches introduced in [8] and [6], which improves the progressive precision of these deterministic approaches. The strategy is to modify the structure of the stream generators by using pseudo-random stochastic bitstreams. When run for the full length, i.e., for 2^{2n} cycles when multiplying two n-bit precision inputs, the new method produces completely accurate results. When slight inaccuracy is acceptable, it provides significant improvement in the processing time and the energy consumption.

In this paper, we introduce two fast-converging scalable deterministic approaches for SC based on low-discrepancy (LD) sequences [12]. LD sequences such as Halton [1] and Sobol [7] have been previously used for improving the speed of SC circuits. These provide a lower error rate with fixed processing time compared to conventional linear feedback shift register (LFSR)-based pseudo-random sequences.

Our strategy is as follows. First we show that computation on LD-based stochastic bitstreams can be completely accurate. We introduce a LD-based deterministic method that converges quickly and produces completely accurate results. We then integrate the rotation method of [6] with the LD Sobol-based stochastic bitstreams. The result is both area-delay efficient and scalable. We show that our new LD-based deterministic methods converge significantly faster to the expected output than the prior methods. We evaluate

ICCAD '18, November 5–8, 2018, San Diego, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5950-4/18/11…$15.00
https://doi.org/10.1145/3240765.3240797

the scalability of the proposed methods. We show that the second of our two methods scales well with increasing number of inputs.

This paper is structured as follows: In Section 2, we present background information on SC including the conventional as well as deterministic approaches. In Section 3, we describe our proposed deterministic methods based on LD sequences. In Section 4, we evaluate our method. We compare the scalability of the proposed methods to the prior methods on the operation of multiplication, varying both the precision and number of inputs. Finally, in Section 5, we present conclusions.

2 BACKGROUND

2.1 Stochastic Computing

SC is a computing paradigm in which numbers in the [0, 1] interval are presented using streams of random (or unary) bits and processed using simple logic. The input data is encoded by the probability of obtaining a one versus a zero. A random or pseudo-random number is compared to a constant number (based on input data) and the output of the comparison produces one bit of the bitstream in each cycle (see Figure 1). Pseudo-random number generators such as LFSRs are often used as the source of generating random numbers in SC systems [13].

Multiplication is the most common stochastic operation. Its hardware implementation is orders of magnitude simpler than that of a conventional binary radix counterpart. A single AND gate performs multiplication in stochastic domain. The performance of an AND gate used as a stochastic multiplier depends on the degree of correlation between the input bitstreams. To produce accurate results, the input bitstreams must be completely independent, so completely uncorrelated. Conventionally this independence is provided by using randomness (or pseudo-randomness) when generating bitstreams. Commonly, LFSRs are used. However, this approach suffers from random fluctuations; this results in computation that is only approximately correct [13].

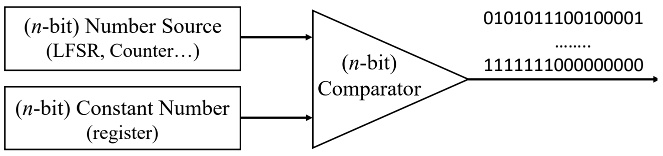


Figure 1: Structure of a stochastic stream generator.

2.2 Deterministic Approaches to SC

Recent work on SC has shown that computation on stochastic bitstreams can be performed deterministically [6] [8]. By properly structuring input bitstreams, completely accurate results can be produced with no random-fluctuation or correlation errors. The required independence between input bitstreams is provided by using relatively prime stream lengths [8], rotation, or clock division [6]. Logical computation is performed on unary streams – that is to say sequences of 1s followed sequences 0s. To generate a unary stream, an increasing value from a counter is compared to a constant value corresponding to the target value in the [0, 1] interval. To produce completely accurate results with these deterministic approaches, the operation must run for an exact number of clock cycles: the product of the length of the input bitstreams [6]. Due to the nature of unary representation, however, running the operation for fewer cycles leads to a poor result with an error out of the acceptable error bound. In the following, we see three examples of deterministic multiplication with unary bitstreams (for the detail of each method please refer to [6]):

Example 1. Relatively prime length method with unary streams:

1/3 = 100100100100
3/4 = 111011101110
3/12 = 100000100100

Example 2. Clock division method with unary streams:

1/4 = 1000 1000 1000 1000
3/4 = 1111 1111 1111 0000
3/16 = 1000 1000 1000 0000

Example 3. Rotation method with unary streams:

1/4 = 1000 1000 1000 1000
3/4 = 1110 0111 1011 1101
3/16 = 1000 0000 1000 1000

Note that the essential property of these deterministic methods is that they pair every bit of one bitstream with every bit of the other exactly once [6]. This property applies regardless of the distribution of the 1's and 0's in the bitstreams. The bitstreams can in fact be randomized. Najafi and Lilja [10] proposed a modified version of these three deterministic methods by bringing randomization *back* into the bitstream representation. Their method offers better progressive precision. The computation is still deterministic and completely accurate. However, when using pseudo-randomized bitstreams, the computation converges to the correct value faster than with unary streams. Truncating the output stream by running for fewer clock cycles still produces high quality result. In the following, we see three examples of deterministic multiplication with pseudo-randomized bitstreams:

Example 4. Relatively prime length method with pseudo-randomized bitstreams:

1/3 = 100100100100
3/4 = 101110111011
3/12 = 100100100000

Example 5. Clock division method with pseudo-randomized bitstreams:

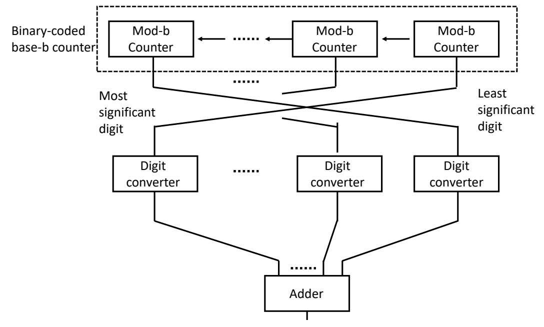
1/4 = 0010 0010 0010 0010
3/4 = 1111 0000 1111 1111
3/16 = 0010 0000 0010 0010

Example 6. Rotation method with pseudo-randomized bitstreams:

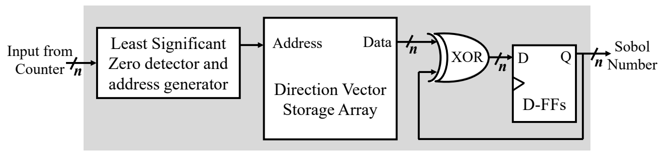
1/4 = 0100 0100 0100 0100
3/4 = 1101 1110 0111 1011
3/16 = 0100 0100 0100 0000

2.3 Low-Discrepancy Sequences in SC

Low discrepancy (LD) sequences were traditionally used to accelerate the convergence in Monte-Carlo simulations [4]. Recent work on SC [1][7] utilized these sequences in improving the speed of computation on stochastic bitstreams. With LD sequences, 1's and 0's in the stochastic streams are uniformly spaced, so the streams do



(a)


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

(b)

Figure 2: (a) Halton sequence generator [1] (b) Sobol sequence generator [4, 7].

not suffer from random fluctuations. The bitstreams can quickly and monotonically converge to the target value, producing acceptable results in a much shorter time [1].

Alaghi and Hayes proposed the use of LD Halton sequences for SC [1]. A Halton sequence generator consists of a binary-coded base-b counter, where b is a prime number. For d independent input streams in a SC system, d counters with different prime bases must be used. For instance, in the simplest case of multiplying two stochastic bitstreams using an AND gate, one base-2 and one base-3 counter is required. The order of the counter's output digits are reversed and the reordered digits are converted to equivalent binary numbers. The structure of the Halton sequence generator proposed in [1] is shown in Figure 2.a. Stochastic bitstreams generated using Halton-based sequences can significantly improve the processing time of SC for achieving the same accuracy compared to the conventional LFSR-based pseudo-random bitstreams. However, the base conversion required in the structure of this sequence generator results in additional hardware overhead [7].

Liu and Han [7] recently proposed another LD-based stochastic stream generator based on Sobol sequences. Compared to Halton sequence generator, generation of Sobol sequences does not need the additional base-conversion hardware. The Sobol sequence generator, instead, consists of an address generator that detects the position of the least significant zero, a storage array storing the values of the direction vectors as intermediate variables for sequence generation, and a pair of XOR gate and D-type flip-flop for recursively generating random numbers. The structure of the Sobol sequence generator, shown in Figure 2.b, is proposed in [4] and used in [7] for generating LD stochastic bitstreams. Different Sobol sequences can be generated by changing the values of the direction vectors.

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

Table 1: Comparing different sources of generating numbers (3-bit precision) for stochastic stream generator

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

Table 1 compares three different sources of generating numbers for the stochastic stream generator of Figure 1. A counter is being used to generate unary streams. An LFSR is being used to generate pseudo-random bitstreams. A Sobol sequence generator, on the other hand, is used to generate a LD-based stochastic stream. As can be seen, the first 2^n numbers in a Sobol sequence can be used to precisely present all possible n-bit precision numbers in [0, 1] interval.

3 PROPOSED FAST-CONVERGING DETERMINISTIC APPROACHES TO SC

In this section, we propose two new deterministic methods for computation with stochastic bitstreams. We first describe the proposed methods and their hardware structures, and then evaluate their accuracy and hardware costs compared to prior state-of-the-art work.

3.1 First Method

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

In the following, we see an example of multiplying two 2-bit precision input values using the first proposed method. The first input value is converted to a bitstream representation using the simplest Sobol sequence (Sobol seq. 1 in Figure 3). The second input value is converted using the second Sobol sequence from the MATLAB built-in Sobol sequence generator (Sobol seq. 2 in

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

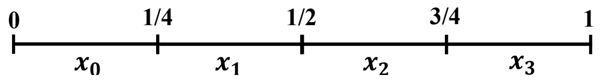


Figure 3: First 16 numbers of the first four Sobol sequences from MATLAB built-in Sobol sequence generator, and the category of each one based on their position in the [0, 1] interval.

Figure 3). Note that, when converting to a bitstream representation, a one is generated if the Sobol number is *less* than the input target number.

Example 7. Deterministic 2-bit precision multiplication using the first proposed method:

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

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

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

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

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

The result of multiplying two input values, represented by two bitstreams, is deterministic and completely accurate if every bit of one bitstream meets every bit of the other stream exactly once [6]. As shown in Figure 3 for $n = 2$, for any pair of two different Sobol sequences, every label u ($u = 0, 1, 2, 3$) in x_u ($x = a, b, c, d$) meets every label t ($t = 0, 1, 2, 3$) in y_t ($y = a, b, c, d$) exactly once if considering the first 2^4 numbers of each sequence. So, the result of multiplying two 2-bit precision numbers by ANDing their 2^4-bit stream representation, generated based on two different Sobol sequences, is deterministic and completely accurate.

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

Figure 4.a shows the structure of the sources of generating Sobol sequences for the first proposed method. These are used as the number sources in the stochastic stream generator of Figure 1. Note that the simplest Sobol sequence is simply the reverse of the output bits of a binary counter and so we generate the first Sobol sequence by hard wiring the output bits of a counter at no extra hardware cost.

3.2 Second Method

The second method is based on the prior deterministic methods introduced in [6]. Inspired from the idea of using pseudo-randomized bitstreams with the three state-of-the-art deterministic approaches [10], we propose to integrate the LD-sequences with the previously proposed deterministic methods. In [10] maximal period pseudo-random sources (i.e., maximal period LFSRs) are used to generate

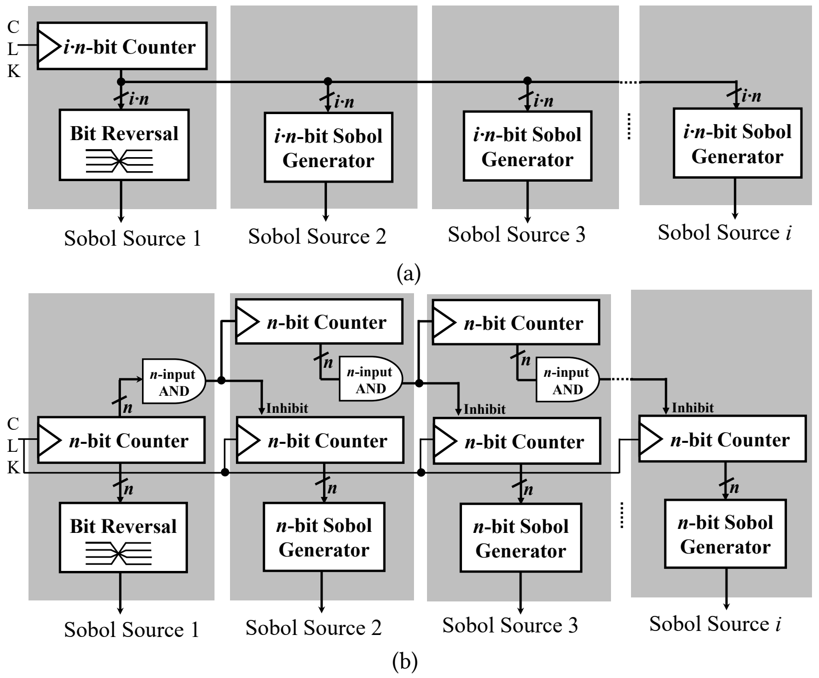


Figure 4: Structures of the sources of generating Sobol sequences based on (a) first proposed method (b) second proposed method.

deterministic accurate bitstreams. The important point is that the period of the pseudo-random source should be equal to the length of the bitstream. By using such a source to generate random numbers, the input value could be converted into a pseudo-random but completely accurate stochastic representation. Instead of pseudo-random sources, in this work we use LD sequence generators.

In contrast to our first method, for the second method, the precision of the sequence generator is equal to the precision of the input data. For example, for multiplication of two n-bit precision inputs data, two n-bit LD sequence generators are required. In case of using LD Halton sequences which are generated based on prime numbers, the relatively prime length method of [8] must be used to guarantee the required independence between the bitstreams. The Sobol sequences, on the other hand, must be integrated with the clock division or the rotation method of [6]. The operations then must continue for the product of the length of the bitstreams to produce deterministic complete accurate results.

The authors in [10] showed that the rotation method has a faster convergence property and is more energy efficient than the clock division deterministic method. So, for the rest of the paper, for the second proposed method, we integrate the LD Sobol sequences with the rotation method. While we limit our reported results to LD Sobol sequences and the rotation approach, the proposed idea can similarly be applied to LD Halton sequences and the relatively prime length method.

The rotation method of [6] guarantees a deterministic accurate output by rotating the bitstreams through inhibiting or stalling on powers of the stream lengths. Figure 4.b shows the structure of the sources of generating Sobol sequences for the second proposed method based on the rotation method. The first Sobol source repeats every 2^n cycles but do not rotate. Other Sobol sources (source k=2, 3, ..., i) have a period of 2^n but rotate every $2^{(k-1)\cdot n}$ cycles by inhibiting. Additional counters in the structure of the second method control these inhibits. We will show that due to using n-bit Sobol generators, instead of expensive $i \cdot n$-bit generators, the structure of second proposed method has a lower hardware cost than that of the first porposed method.

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

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

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

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

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

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

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

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

Table 2: Mean Absolute Error (%) comparison of the prior approximate and deterministic approaches to SC and the proposed deterministic approach when multiplying two 8-bit precision stochastic streams with different numbers of operation cycles.

Design Approach	Area(μm^2)	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}
Conv. Approx. SC [5, 13]	781	0.05	0.15	0.26	0.39	0.58	0.79	1.20	1.67	2.32	3.32	4.72
Deter. Rotation Unary [6]	492	0.00	3.10	4.84	6.15	7.08	7.66	7.99	8.17	8.26	33.1	51.8
Deter. Rotation Pseudo-Random [10]	536	0.00	0.09	0.16	0.24	0.35	0.47	0.60	0.71	0.82	2.56	4.26
This work 1- Deter. Sobol	3361	0.00	0.0003	0.0013	0.0035	0.009	0.019	0.041	0.092	0.190	0.451	0.921
This work 2- Deter. Rotation Sobol	1277	0.00	0.0013	0.0033	0.0075	0.014	0.031	0.059	0.112	0.190	0.451	0.921

3.3 Accuracy Evaluation

For accuracy comparison of the proposed methods with prior works, we exhaustively tested multiplication of two 8-bit precision input data in [0, 1] interval from a large set of random input values for the conventional approximate SC [5][13] and for the prior pseudo-random rotation approach [10], and on every possible input value for the unary-stream based deterministic approach [6] and on the proposed methods. Table 2 compares the mean absolute errors (MAEs) of the conventional approximate SC (using two different 16-bit LFSRs as the number generators), the prior deterministic unary-stream based rotation approach (using two 8-bit counters as the number generators), the prior deterministic pseudo-random based rotation approach (using two different 8-bit LFSRs as the number generators), the first introduced method based on Sobol sequences (using two 16-bit Sobol Sequence generators), and the second introduced Sobol-sequence based rotation approach (using two 8-bit Sobol generators). Note that for the two required Sobol sequences in the proposed methods we use the simplest Sobol sequence and the second Sobol sequence from the MATLAB Sobol sequence generator.

As can be seen in Table 2, similar to the deterministic approaches proposed in [6] and [10], our methods could produce completely accurate results in 2^{16} cycles. Due to using LD-based bitstreams, however, the MAE of the computation is significantly lower than that of the prior approximate and deterministic approaches when truncating the bitstreams (running the operation for fewer cycles). For example, when running the multiplication operation for 2^{15} cycles (processing 2^{15}-bit streams), the proposed methods have shown a MAE of around 10^{-3}, which is 100X lower than the MAE of the deterministic pseudo-random rotation method of [10] and 3000X lower than that of the deterministic unary-stream based rotation approach of [6]. Thus, our methods show a much better progressive precision property and converge to correct result much faster than prior methods. Note that for stream lengths of less than 2^n (here, 2^8) the two proposed methods are essentially the same (because rotation begins after 2^n cycles) and so, they show the same accuracy for these stream lengths.

3.4 Cost Comparison

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

4 SCALABILITY EVALUATION

Limited scalability has been an important challenge of prior deterministic methods of SC. As the authors in [3] discuss, when many mutually independent stochastic bitstreams are needed, the hardware cost significantly increases with the number of inputs. Stochastic bitstreams generated via LD sequences has a faster convergence than the pseudo-random sequences and of course than the unary counter-based sequences. However, the benefits of using

Table 3: Hardware area cost (μm^2) of the bitstream generator for different data precision and number of inputs (N=Input Data Precision - i=Number of Inputs)

Design Approach	N=4 i=2	N=4 i=3	N=4 i=4	N=8 i=2	N=8 i=3	N=8 i=4	Stream Generator Structure
Conv. Approx. SC [5, 13]	397	821	1394	781	1622	2799	i i*N-bit LFSRs + i N-bit Comparator
Deter. Rotation Unary [6]	224	342	459	492	754	1016	1 N-bit Counter + i-1 N-bit CounterEN + i N-bit Comp
Deter. Rotation Pseudo [10]	262	411	560	536	832	1127	1 N-bit LFSR + i-1 N-bit LFSREN + i N-bit Comp
This work 1- Deter. Sobol	1005	3740	9127	3361	13193	32406	1 i*N-bit Counter + i-1 i*N-bit Sobol Gen+ i N-bit Comp
This work 2- Deter. Rotation Sobol	456	806	1156	1277	2324	3371	i-1 N-bit Count+i-1 N-bit (CountEN+Sobol Gen)+i N-bit Comp

LD sequences diminish as the number of inputs increases, because the cost of generating them is much higher than pseudo-random number generation. In this section, we evaluate the scalability of the proposed methods compared to prior methods and show that the second proposed method which integrates the LD sequences with the rotation approach has the best scalability compared to prior deterministic and also conventional approximate SC.

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

The MAE of the implemented multipliers for different stream lengths (different operation cycles) are presented in Figures 5 and 6. As can be seen in Figure 5, for the 4-bit precision multipliers, the proposed methods show a better progressive precision property than prior deterministic methods and their computation accuracy scales with increasing the number of inputs. Only the conventional approximate SC approach shows a better scalability than the second proposed method but it lacks the ability to generate completely accurate results. For the 2-input and 3-input multipliers with 8-bit precision (Figure 6) we achieved the best accuracy performance by using the two proposed methods. Both methods converge to the expected correct value very fast and scale well with increasing the number of inputs.

Finally, we show the area-delay product of the implemented 8-bit precision multipliers for different MAEs in Figure 7. We first exhaustively tested each design approach with a large set of input values and found the average processing time of each one to achieve a specific MAE rate. We then multiplied the processing time with the corresponding design hardware area cost to produce the area-delay product. As can be seen in Figure 7, the second proposed method (red lines) has the lowest area-delay product between different approximate and deterministic state-of-the-art methods which shows its superiority to prior methods and its scalability when the number of inputs increases.

5 CONCLUSION

Poor progressive precision and limited scalability are the main challenges of the recently proposed deterministic approaches to SC. In this work we proposed two fast-converging scalable deterministic approaches for processing bitstreams based on low-discrepancy sequences. The first proposed approach provides the best accuracy for a fixed processing time while the second approach has the lowest area $\times$ delay product. Both methods produce completely accurate results if running the operation for a specific number of cycles. Although the hardware area cost of the proposed methods is higher

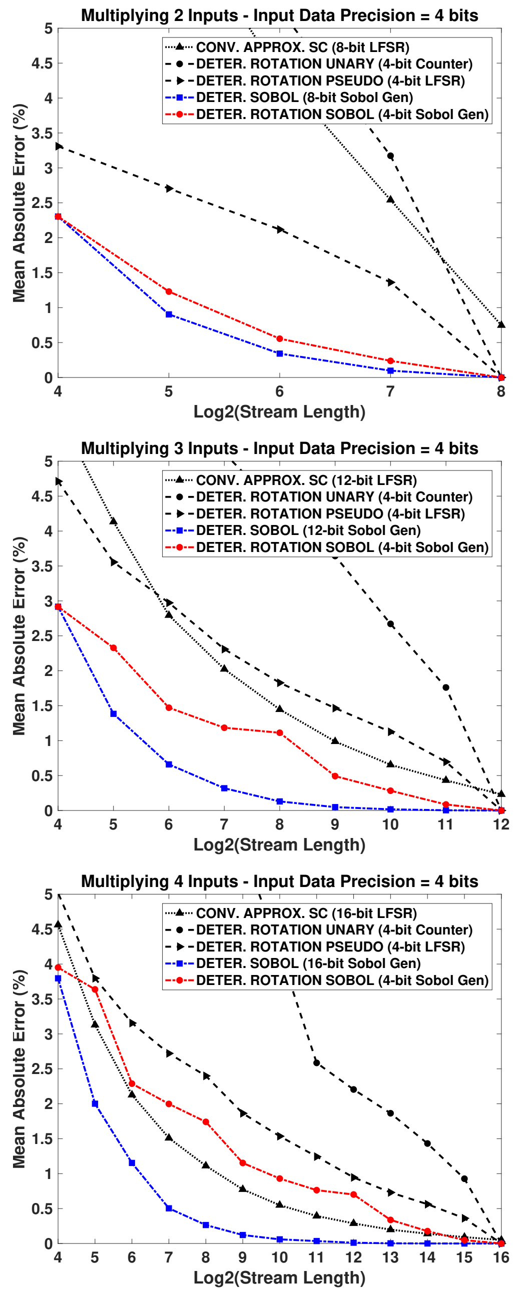
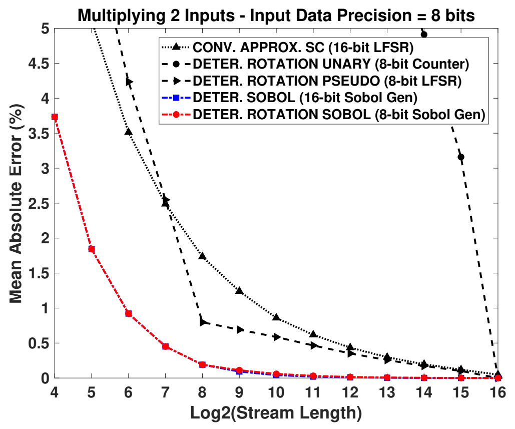


Figure 5: MAEs of 4-bit precision multipliers for different stream lengths.

than prior methods, a significantly better progressive precision makes them a better choice for applications that can tolerate slight inaccuracy (e.g., image processing and neural network applications). In future work, we will evaluate the efficiency of the proposed method on more complex examples from real applications. We will explore different optimization techniques to further reduce the hardware cost.



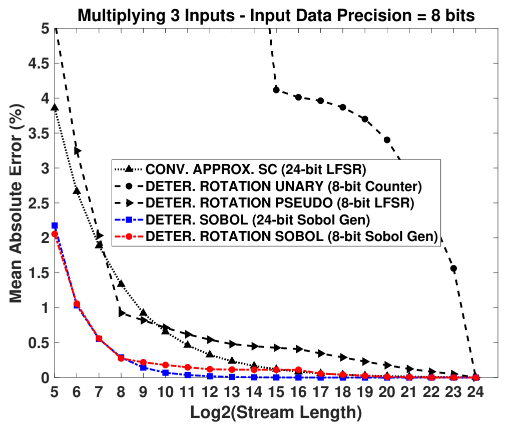
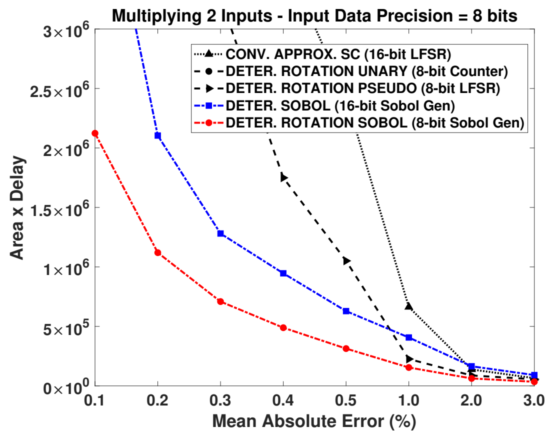


Figure 6: MAEs of 8-bit precision multipliers for different stream lengths.



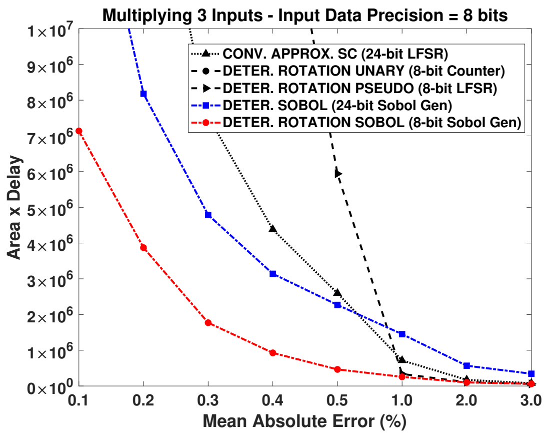


Figure 7: Area×Delay of 8-bit precision multipliers for different MAEs. Note that the Area×Delay numbers for the DETER. ROTATION UNARY method were much larger than other method and out of the range shown in the figure)

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grant no. CCF-1438286. 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] A. Alaghi and J. 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.

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

[3] A. Alaghi, W. Qian, and J. P. Hayes. The Promise and Challenge of Stochastic Computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(8):1515–1531, Aug 2018.

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

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

[6] D. Jenson and M. Riedel. A Deterministic Approach to Stochastic Computation. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, pages 102:1–102:8, New York, NY, USA, 2016.

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

[8] M. H. Najafi and S. Jamali-Zavareh and D. J. Lilja and M. D. Riedel and K. Bazargan and R. Harjani. Time-Encoded Values for Highly Efficient Stochastic Circuits. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 25(5):1–14, 2017.

[9] M. H. Najafi, P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel. A Reconfigurable Architecture with Sequential Logic-Based Stochastic Computing. *J. Emerg. Technol. Comput. Syst.*, 13(4):57:1–57:28, June 2017.

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

[11] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan. Polysynchronous Clocking: Exploiting the Skew Tolerance of Stochastic Circuits. *IEEE Transactions on Computers*, 66(10):1734–1746, Oct 2017.

[12] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* Society for Industrial and Applied Mathematics, 1992.

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