A Low-Cost FSM-based Bit-Stream Generator for Low-Discrepancy Stochastic Computing

Sina Asadi*, M. Hassan Najafi* and Mohsen Imani†
sina.asadi1@louisiana.edu, najafi@louisiana.edu, m.imani@uci.edu
*School of Computing and Informatics, University of Louisiana at Lafayette, LA, USA
†Department of Computer Science, University of California Irvine, CA, USA

***Abstract*—Low-discrepancy (LD) bit-streams have been proposed to improve the accuracy and computation speed of stochastic computing (SC) circuits. These bit-streams are conventionally generated by using a quasi-random number generator such as a Sobol sequence generator and a comparator. The high hardware cost of such number generators makes the current comparator-based generators expensive in terms of area and power cost. The hardware cost issue further aggravates when increasing the number of inputs and the precision of data. A finite state machine (FSM)-based LD bit-stream generator was proposed recently to mitigate this hardware cost. The proposed generator, however, can only generate a specific LD pattern and hence, cannot be used where multiple independent LD bit-streams are needed.**

In this work, we propose a low-cost FSM-based LD bit-stream generator that supports generation of any number of independent LD bit-streams. The proposed generator reduces the hardware area and the area-delay product up to 80% compared to those of the state-of-the-art comparator-based LD bit-stream generator while generating accurate bit-streams. We develop a parallel design of the proposed generator and show that the $8\times$ parallel implementation reduces the hardware cost on average more than 82 percent compared to the cost of the state-of-the-art parallel LD generator. Taking advantage of the provided area saving we improve the fault tolerance of the bit-stream generation unit, a vulnerable component in SC systems, by orders of magnitude. We show the effectiveness of using the proposed generator in SC-based design of convolution function as the case study.

I. Introduction

Stochastic Computing (SC) [1], [2], an unconventional computing paradigm processing random bit-streams, has been used for low-cost and noise-tolerant implementation of complex arithmetic operations [3], [4]. Orders of magnitude saving in the hardware costs compared to the conventional binary radix designs are common with SC [2]. A single AND gate, for example, can perform multiplication in the stochastic domain (see Fig. 1). Conventionally, the data is converted from binary to pseudo-random bit-streams. Computation on pseudo-random bit-streams suffers from random fluctuations [2]. Often very long bit-streams must be processed to produce acceptable results. This results in long processing time and high energy consumption which make SC designs inefficient compared to their binary counterparts.

Low-discrepancy (LD) bit-streams such as Halton- [5] and Sobol-based [6] bit-streams were proposed recently to improve the accuracy and reduce the processing time of SC. 1's and 0's are uniformly spaced in these bit-streams. Random fluctuations are removed from bit-stream generation, and deterministic and accurate bit-streams are generated. Progressive precision of Sobol sequences [6], in particular, has made them popular for LD bit-stream generation [7], [8]. The first 2^n numbers of any Sobol sequence include all possible n-bit precision values in the [0,1] interval. This allows 2^n-bit Sobol-based bit-streams to precisely represent any n-bit precision value. Fig. 2 shows the first 16 numbers of the simplest Sobol sequence and examples of LD bit-streams generated using this Sobol sequence. A '1' is generated in the bit-stream if the Sobol number is *less* than the target number.

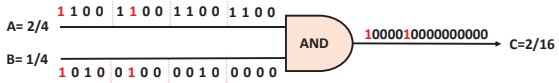

Fig. 1. Example of multiplication using stochastic bit-streams and AND gate.

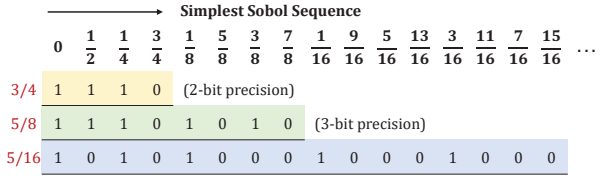

Fig. 2. Examples of LD bit-streams with different precisions generated using the simplest Sobol sequence.

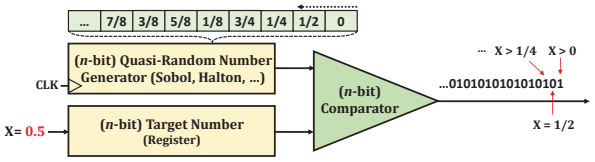

Fig. 3. Conventional comparator-based LD bit-stream generator. A '1' is generated in each cycle if the random number is less than the target number.

The conventional method for generating LD bit-streams is shown in Fig. 3. A quasi-random number (e.g., a Sobol number from a Sobol sequence) is compared to a target number using a binary comparator. The output of this comparison generates one bit of the LD bit-stream in each cycle. Quasi-random number generators, however, are costly. A 4-, 8-, and 16-bit precision Sobol sequence generator takes $2.8\times$, $4.7\times$, and $9.1\times$, respectively, more hardware area cost than the same-precision pseudo-random number generator [9]. The high hardware cost limits the potential benefits and the scalability of the conventional comparator-based LD bit-stream generator [2] [9].

Sim et al. [10] recently introduced an FSM-based LD bit-stream generator to convert data from binary to LD bit-stream. Sim's generator selects x_{n-i} or the $(n-i)^{th}$ bit of the binary input X first at cycle 2^{i-1} and thereafter every 2^i cycles. The hardware cost of Sim's generator is considerably lower than that of the comparator-based LD generator [10]. The challenge is that Sim's

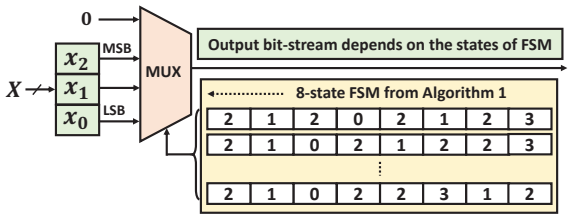

Fig. 4. Proposed FSM-based LD bit-stream generator for n=3. The FSM selects the input bits based on one of the patterns produced by Algorithm 1.

generator can only generate *one fixed LD pattern*. Hence, it cannot be used in SC designs in which multiple independent LD bit-streams are needed. This includes multi-input multipliers, scaled adders [2], and the ReSC-based designs such as the Gamma correction circuit [3], to name a few. These designs still have to employ the conventional comparator-based LD generators.

In this work, we propose a low-cost FSM-based LD bit-stream generator which supports generation of any number of LD patterns. We develop an algorithm to implement different LD patterns based on different Sobol sequences [6]. The proposed generator is able to generate any number of independent bit-streams. SC systems implemented based on the proposed generator are able to produce completely accurate results, the same as the results from the conventional binary counterparts.

Evaluating the cost of the proposed bit-stream generator in implementing stochastic multipliers with different number of inputs and precisions shows area cost reduction of up to 80% compared to the case of using the comparator-based generator. While the proposed generator is hardware efficient for common computations with 8-10 bit precision, the cost increases considerably when the precision exceeds 12 bits. To address this limitation, we integrate the proposed generator with a rotation method of processing bit-streams [11] and achieve a low-cost implementation for high-precision computations.

We further develop a parallel design for the proposed generator. Depending on the level of parallelism, the developed design provides more than 60 to 80 percent cost saving compared to the state-of-the-art parallel LD bit-stream generator [12]. We also evaluate the fault tolerance of the proposed and the comparator-based LD bit-stream generator. We show that the low-cost advantage of the proposed design allows us to use additional techniques to improve the fault-tolerance of the bit-stream generator. We show the effectiveness of using the proposed generator by implementing a SC-based convolution engine. Finally, we propose a method to further reduce the cost of bit-stream generation when generating a large number of bit-streams.

The rest of the paper is organized as follows: Section II demonstrates the proposed bit-stream generator. Section III evaluates the accuracy, hardware cost and fault tolerance of the proposed generator. Section IV shows the effectiveness of the generator for SC convolution design. Finally, Section V concludes the paper.

II. Proposed LD Bit-Stream Generator

A LD bit-stream generator converts an n-bit precision binary data into a 2^n-bit bit-stream with uniformly spaced 1's and 0's. x_i or the i^{th} bit of binary input X appears in the LD bit-stream exactly 2^i times. We connect the binary input data to the main inputs of an $(n+1)$-to-1 multiplexer (MUX). A 2^n-state FSM

Algorithm 1: Constructing an FSM based on a Sobol sequence

Input: Sobol seq (Sobol-num $[0:2^n-1]$), data-width (n)
Output: A 2^n-state FSM
for $k = 0, 1, \ldots, 2^n-1$ **do**
  if $0 \leq$ *Sobol-num(k)* $< 1/2$
    FSM output = $n-1$
  else if $1/2 \leq$ *Sobol-num(k)* $< 3/4$
    FSM output = $n-2$
  .
  .
  else if $(2^{n-1}-1)/2^{n-1} \leq$ *Sobol-num(k)* $< (2^n-1)/2^n$
    FSM output = 0
  else
    FSM output= n

is connected to the select input of the MUX to select one of the input bits at any cycle. The FSM controls the order of bit selection and the number of times each input bit is selected. Different LD bit-selection orders are needed to generate independent LD bit-streams. Fig. 4 depicts the structure of the proposed bit-stream generator for n=3. In what follows, we discuss how the bit selection orders are determined for the proposed bit-stream generator.

Prior work [6] [9] showed that, among different types of stochastic bit-streams, the Sobol sequence-based bit-streams provide the fastest convergence to the target value. Independence between different Sobol-based LD bit-streams is provided by using different Sobol sequences in converting input data into bit-stream representation [7]. Here, we propose an algorithm to determine the order of bit selection by the FSM of our bit-stream generator based on the distribution of numbers in the Sobol sequences. An independent LD bit-stream is generated by setting up the FSM using a different Sobol sequence. Note that this step is performed offline and the structure of the FSM will not change after implementation.

Algorithm 1 demonstrates the procedure. Each Sobol number from a Sobol sequence determines one state of the FSM. Assume S_k is the k^{th} number of the Sobol sequence. If $(2^{m-1}-1)/2^{m-1} \leq S_k < (2^m-1)/2^m$ (where $m = 1, 2, .., n$), x_{n-m} or the $(n-m)^{th}$ bit of binary input X should be selected by the FSM. For example, if $1/2 \leq S_k < 3/4$, m is 2 and the $(n-2)^{th}$ bit of the input data should be selected by the FSM. Assume a 4-bit binary data is to be converted to a 16-bit LD bit-stream. Fig. 5 shows the first 16 numbers of two different Sobol sequences and their corresponding selected bits (i.e., FSM states) based on Algorithm 1.

III. Evaluation

A. Accuracy

We evaluate the accuracy of the proposed bit-stream generator compared to the state-of-the-art LD bit-stream generators (Sim's design and comparator-based design) and also to the conventional comparator-based non-LD (pseudo-random) generator by exhaustively testing multiplication of two 8-bit precision data. For the non-LD generator, two different 16-bit LFSRs are used as the number sources. For Sim's design, a 256-state design of the FSM-based LD generator of [10] and a unary bit-stream generator (built from a pair of 8-bit counter and comparator) are used as elaborated in [10] to convert the two inputs. The comparator-based LD [6], [7] and the proposed FSM-based generator use the

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
Selected bit	x_3	x_2	x_3	x_1	x_3	x_2	x_3	x_0	x_3	x_2	x_3	x_1	x_3	x_2	x_3	0
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
Selected bit	x_3	x_2	x_1	x_3	x_2	x_3	x_3	x_0	0	x_3	x_3	x_2	x_3	x_1	x_2	x_3

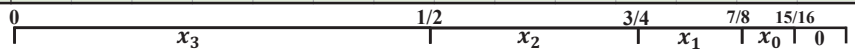

Fig. 5. First 16 numbers of two Sobol sequences and their corresponding selected bits in a 4-bit data based on Algorithm 1.

TABLE I
MAE (%) COMPARISON OF THE PROPOSED AND THE PRIOR STOCHASTIC DESIGNS WHEN MULTIPLYING TWO 8-BIT PRECISION DATA

Design \Number of Cycles	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}
Comparator-based non-LD [1]	0.05	0.15	0.26	0.39	0.58	0.79	1.20	1.67	2.32	3.32	4.72	6.62
Comparator-based LD [6] [7]	0.00	0.0003	0.0013	0.0035	0.009	0.019	0.04	0.09	0.19	0.45	0.92	1.85
Sim's FSM-based LD [10]	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	12.60	18.85	21.99
Proposed FSM-based LD	**0.00**	**0.0003**	**0.0013**	**0.0034**	**0.007**	**0.017**	**0.03**	**0.07**	**0.16**	**0.35**	**0.79**	**1.36**

first and the second Sobol sequences from the MATLAB built-in Sobol sequence generator as their LD number sources. Table I compares the mean absolute errors (MAEs) of these designs for different bit-stream lengths. We multiply the measured MAE of each design by 100 and report it as a percentage.

Due to random fluctuations in pseudo-random bit-streams and correlation between bit-streams, the non-LD design cannot provide comparable accuracy to the LD designs. The output bit-streams produced by the Sim's design has a period of 2^8 cycles and so their accuracy never improves after 2^8 cycles. Sim's design achieves a higher accuracy (i.e., a lower error rate) than the non-LD only when processing bit-streams of 2^8 (or multiples of 2^8) bits. Unary bit-streams suffer from truncation error [9]. Hence, Sim's design that converts the second input to a unary bit-stream shows poor results when truncating the bit-streams and processing bit-streams shorter than 2^8 bits (e.g., 2^7 bits). For small bit-stream lengths, both of the non-LD and the LD comparator-based designs show a better performance than Sim's design.

The authors in [7] proved that converting two n-bit precision numbers to two 2^{2n}-bit independent LD bit-streams leads to completely accurate multiplication results when logical-ANDing the generated bit-streams. The comparator-based and the proposed FSM-based LD designs implemented here both convert the input data into independent LD bit-streams. Consequently, as reported in Table I, both of these designs are able to produce completely accurate results (zero error rate) when processing for 2^{16}(=$2^{2\times8}$) cycles. They also show low error rates when processing shorter bit-streams. However, as we show in Section III-B, the proposed design has significantly lower hardware cost for computation precisions not exceeding 12 bits. We discuss that for more precise computations the proposed FSM-based design can be integrated with the rotation technique of [11] for hardware efficiency.

B. Hardware Cost

*1) **Single Bit-stream Generator**:* The hardware cost of the comparator-based LD generator depends on the required precision (bit-stream length) and is independent of the selected Sobol sequence. To generate a LD bit-stream of 2^n bit length, one n-bit Sobol sequence generator and one n-bit binary comparator are needed. Different Sobol sequences are generated by changing the values of direction vectors in the sequence generator [6]. The hardware cost of the FSM-based generator, however, not only depends on the precision but also depends on the selected LD pattern (i.e., selected Sobol sequence in Algorithm 1) [13]. This is because the structure of the FSM changes with a different bit-selection order. Fig. 6 compares the hardware area cost of the proposed FSM-based LD generator for different precisions (n=4,8,12) and LD patterns (based on ten different Sobol sequences). We synthesize the designs using the Synopsys Design Compiler v2018.06 with the 45nm FreePDK gate library [14]. As it can be seen, the proposed generator provides on average 56, 55, and 51 percent hardware area saving for 4-, 8-, and 12-bit precision bit-stream generation, respectively, compared to the comparator-based LD generator.

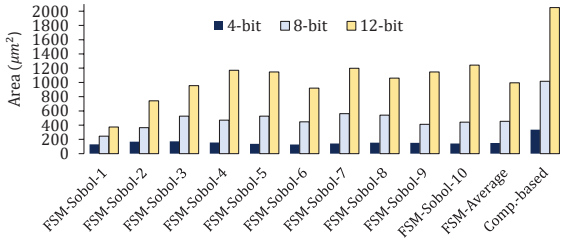

Fig. 6. Hardware area cost of the proposed LD bit-stream generator for different precisions (4, 8, and 12 bits) and LD patterns (Sobol 1-10).

TABLE II
HARDWARE AREA COST (μm^2) OF THE BIT-STREAM GENERATORS FOR PROCESSING DIFFERENT NUMBER OF INPUTS (i) AND DATA PRECISIONS (n).

Design	n=4 i=2	n=4 i=3	n=4 i=4	n=8 i=2	n=8 i=3	n=8 i=4
Comparator-based LD (Limited)	366	627	888	1087	1944	2801
This work - FSM-based LD (Limited)	**188**	**280**	**337**	**451**	**781**	**1026**
Comparator-based LD (Full) [6], [7]	1005	3740	9127	3361	13193	32406
This work - FSM-based LD (Full)	**300**	**746**	Large	Large	Large	Large
Comp-based LD + Rotation (Full) [7]	456	806	1156	1277	2324	3371
This work FSM LD + Rotation (Full)	**353**	**609**	**864**	**751**	**1495**	**2149**

*2) **Multiple Bit-stream Generators**:* The hardware area cost of the proposed bit-stream generator and the state-of-the-art comparator-based LD generator [6], [7] are compared in Table II for implementing stochastic multipliers with different number of inputs (i=$2, 3, 4$) and data precisions (n=$4, 8$). We implemented three different designs for each case: one limited-precision design that produces n-bit precision output (2^n-bit output bit-stream) and two full-precision designs that produce ($i \times n$)-bit precision output ($2^{i\times n}$-bit output bit-stream). The proposed bit-stream generator provides significantly lower hardware cost for all limited-precision designs and also for the full-precision designs with

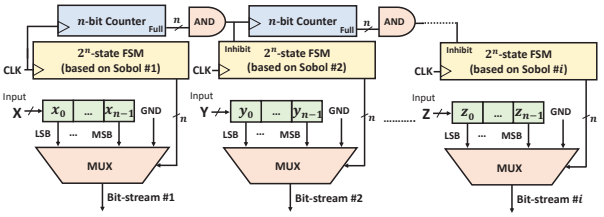

Fig. 7. Integrating the FSM-based LD bit-stream generator and the rotation method of [11] to generate i independent $2^{i\times n}$-bit bit-streams.

at most 12-bit output precision. A weakness of the proposed generator is its high hardware cost for output precisions exceeding 12 bits. This is because producing an $(i \times n)$-bit precision output requires implementing a separate $2^{i\times n}$-state FSM for each input. Although such high output precision (i.e., $\geq$ 12 bits) is rarely needed in today's common applications such as neural networks and image processing, we integrate our design with the rotation method of [11] for high-precision bit-streams.

Fig. 7 shows the structure of our generator integrated with the rotation method. This integration allows to generate i $2^{i\times n}$-bit bit-streams by using i 2^n-state FSMs (instead of i $2^{i\times n}$-state FSMs). This significantly reduces the hardware cost at no accuracy loss while producing full-precision output [9]. The rotation technique guarantees a full-precision output by rotating the bit-streams through inhibiting or stalling on powers of the stream lengths. The output of the first FSM repeats every 2^n cycles and does not rotate. Other FSMs (FSM #k=2,3,..,i) have a period of 2^n but rotate every $2^{(k-1)\cdot n}$ cycles by inhibiting. As reported in Table II, compared to the rotation-based design of the compared-based generator [7], our rotation-based design provides a lower hardware cost in all cases.

According to Table II, the proposed bit-stream generator provides up to 80% saving in the hardware area cost. Considering the fact that for the same number of processing cycles the proposed generator provides a better or the same level accuracy as the comparator-based one (see Table I), the proposed design achieves more than 80% savings in the area-delay product.

*3) **Parallel Bit-stream Generator**:* Parallelization has been used to mitigate the long latency of SC at the cost of higher hardware area and power consumption. Liu and Han [12] developed a hardware efficient parallel Sobol sequence generator that can generate multiple Sobol numbers in each clock cycle at the cost of some additional XOR gates. For an $M\times$ parallel comparator-based LD bit-stream generator, an $M\times$ parallel Sobol generator and M comparators are needed. This design reduces the number of processing cycles by a factor of M by generating M LD bit-streams of length $2^N/M$ in parallel [12]. A reasonable increase in the hardware cost but $M\times$ reduction in the latency makes the parallel design of the comparator-based LD generator an attractive alternative to the non-parallel implementation.

We also develop a parallel design for the proposed bit-stream generator. In contrast to the non-parallel design which has 2^N states, the $M\times$ parallel design has $2^N/M$ states. Each FSM state in the non-parallel design selects one bit of the input data. Each state in the $M\times$ parallel design, however, selects *at most* M bits of the binary input to generate M output bits in parallel. Fig. 8 exemplifies the case of converting a 3-bit data into an 8-bit LD

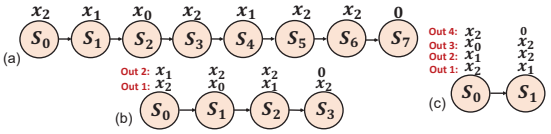

Fig. 8. An example of the FSMs for converting a 3-bit precision data to 8-bit LD bit-stream: a) non-prallel with 8 states b) 2× parallel with 4 states c) 4× parallel with 2 states. Each state is processed in one clock cycle.

TABLE III
HARDWARE AREA (μm^2) AND CRITICAL PATH LATENCY (CP) (nS) OF THE COMPARATOR-BASED AND THE FSM-BASED LD GENERATOR FOR THE CASE OF CONVERTING 8-BIT PRECISION DATA

Parallel Design	1×		2×		4×		8×	
	Area	CP	Area	CP	Area	CP	Area	CP
Comparator-based [12]	1013	0.45	1245	0.47	1658	0.47	2484	0.47
FSM-based (Sob.1)	246	0.37	266	0.35	267	0.34	250	0.30
FSM-based (Sob.2)	366	0.39	375	0.42	422	0.41	455	0.39
FSM-based (Sob.3)	526	0.42	504	0.43	512	0.40	485	0.39
FSM-based (Sob.4)	471	0.40	483	0.43	513	0.39	485	0.37
FSM-based (Sob.5)	526	0.42	479	0.41	483	0.41	487	0.39

bit-stream. Fig. 8.a shows the FSM of the non-parallel design. In this design, one input/output bit is selected/generated at any cycle. Conversion with this design takes eight cycles. The FSM of the 2× parallel design is shown in Fig. 8.b. At each cycle, two output bits are generated which reduces the number of processing cycles from eight to four. Finally, the FSM of the 4× design, shown in Fig. 8.c, produces four output bits at any cycle which reduces the processing time to only two clock cycles.

Table III compares the hardware cost of the 2×, 4×, and 8× parallel FSM-based LD generator (implemented based on the first five Sobol sequences from the MATLAB Sobol sequence generator) and the state-of-the-art parallel comparator-based LD generator [12] for the case of converting 8-bit data to 2^8-bit bit-stream. As it can be seen, both the non-parallel (1×) and the parallel designs of the FSM-based generator provide significantly lower hardware cost than their corresponding comparator-based design. The hardware cost saving provided by the FSM-based design increases when increasing the level of parallelism. On average, the 2×, 4×, and 8× design of the FSM-based generator achieves 66, 73, and 82 percent saving, respectively, compared to the corresponding comparator-based generator.

An interesting property of the proposed FSM-based design is that a higher level of parallelism can be achieved with no considerable increase in the hardware cost. In some cases, the area is even reduced with more parallelism. For instance, the non-parallel design of the FSM-based generator implemented based on Sobol sequence 1 takes an area footprint of 246 μm^2 while its 2×, 4×, and 8× parallel designs cost 266 μm^2, 267 μm^2, and 250 μm^2 area, respectively. This happens because by increasing the level of parallelism 1) the number of states decreases and 2) the same input bit is selected for more than one output bit (e.g., x_2 in the FSM of Fig. 8.c).

C. Fault-Tolerance

Fault tolerance is one of the attractive properties of SC designs [2], [15]. The processing elements of SC systems inherently tolerate high rates of soft errors (i.e., bit flips) as they process data in the stochastic domain. However, the bit-stream generators that convert binary data to stochastic bit-streams are vulnerable to bit flips as they operate in the binary domain. Here, we

TABLE IV
MAE (%) COMPARISON OF DIFFERENT LD BIT-STREAM GENERATORS WHEN INJECTING DIFFERENT RATES OF SOFT ERROR.

LD Bit-Stream Generator	Area (μm^2)	Input Width	Injected Soft Error 1%	2%	5%	10%	20%	30%
Comparator-based	337	4	3.8	5.4	7.5	9.4	13.3	17.5
	1013	8	1.1	1.6	3.0	5.3	10.2	15.1
	2054	12	0.56	1.04	2.5	5.0	10.0	15.0
Comparator-based + 3-MR	1179	4	0.18	0.68	3.0	6.0	9.6	14.0
	3349	8	0.24	0.53	1.02	2.1	5.5	11.3
	6624	12	0.06	0.12	0.38	1.4	5.3	10.6
Proposed FSM-based	165	4	3.9	4.8	7.3	11.0	17.2	21.5
	366	8	1.4	2.6	6.0	10.8	17.7	21.8
	742	12	1.3	2.5	6.0	10.8	17.7	21.7
Proposed FSM-based + 3-MR	587	4	0.11	0.42	2.1	4.9	9.9	16.0
	1266	8	0.14	0.34	1.00	2.9	9.3	16.5
	2377	12	0.05	0.18	0.95	3.5	11.3	18.2
Proposed FSM-based + 5-MR	1051	4	0.006	0.04	0.46	2.3	7.1	13.2
	2252	8	0.005	0.04	0.33	1.1	5.6	13.4
	4090	12	0.004	0.02	0.19	1.1	6.6	15.5

compare the fault tolerance of the proposed FSM-based and the comparator-based generator of [6] when converting input data with different precisions (n = 4, 8, and 12 bits) to LD bit-streams with corresponding lengths (2^4, 2^8, and 2^{12} bits). Soft errors are injected by flipping bits in the input and output bits of different components of the bit-stream generator including the storage array of the Sobol generator for the comparator-based and the sates of the FSM for the FSM-based generator. Table IV shows the results. Evidently, increasing the precision reduces the error rate (improves fault tolerance) in both the proposed and the comparator-based generators. This is because longer bit-streams are generated for higher precisions, and longer bit-streams have higher tolerance to bit flips [3], [16].

The proposed FSM-based generator shows a slightly lower tolerance to soft errors compared to the comparator-based design. This is due to the high sensitivity of FSMs to changing their state caused by bit-flips. As it can be seen in the reported numbers of Table IV, the difference between the MAEs of the two LD generators increases when increasing the fault injection rate. However, the hardware cost saving provided by the proposed generator can be exploited to improve its tolerance to soft error by using additional techniques of improving fault tolerance such as the N-modular redundancy (N-MR) [17].

For the comparator-based design we evaluate a 3-MR design by implementing three identical copies of the main components of the generator and using majority gates to vote between them. For the FSM-based design we implement a 3-MR and a 5-MR design. Table IV compares the hardware area cost and the MAE of the implemented generators for different noise injection rates. Clearly, implementing the N-MR technique has improved the fault tolerance of the generators. For example, for the case of injecting 1% soft error, the MAEs of the 5-MR implementations of the FSM-based generator are all below 0.01%, which shows over three orders of magnitude reduction in the error rate compared to the non-redundant implementation. The reported area numbers show that the 5-MR design of the FSM-based generator has a lower hardware cost than the 3-MR design of the comparator-based generator while achieving significantly lower error rates for noise injection rates below 10%. For higher injection rates, the comparator-based generator shows higher tolerance to noise.

The high hardware cost of the Sobol sequence generator in the comparator-based design makes it difficult for the designer to exploit techniques such as N-MR to improve soft error tolerance. However, supported by the area and MAE numbers reported in Table IV, the low-cost advantage of the proposed LD generator allows us to use additional techniques to improve the soft error tolerance of the bit-stream generator in the SC system.

IV. CASE STUDY: CONVOLUTION DESIGN

LD bit-streams has been recently used for accurate and energy-efficient design of SC-based convolutional neural networks (CNNs) [10], [18]–[21]. To further evaluate the effectiveness of using the proposed bit-stream generator, we compare the cost of LD bit-stream generation in SC design of convolution functions with different sizes of $k \times k$ (k =3, 5, 7, 9, and 11). 8-bit precision data is converted from binary radix to LD bit-streams and fed to the convolution design. In convolution, pairs of input data are first multiplied and then the results are accumulated. For a higher output accuracy, the state-of-the-art SC convolution designs implement the multiplication operations in the stochastic domain (using AND gates) but perform the accumulation in the binary domain using binary adders [18], [20]. Since the accumulation is performed in the binary domain, the outputs of the multiplication operations do not need to be independent. This permits to reuse two LD patterns to convert all input data to bit-stream representation. We evaluate three LD bit-stream generation approaches:

A1. Comparator-based: Each input data is compared with a Sobol number using a separate binary comparator. Two different Sobol sequences are needed to provide the two required LD patterns. To minimize the cost of generating the two Sobol sequences, the first sequence is generated by simply reversing the output bits of a binary counter [7]. The second Sobol sequence is generated by implementing the Sobol number generator described in [6]. So, the comparator-based approach consists of one 8-bit binary counter, one 8-bit Sobol generator, and $k \times k$ 8-bit comparators to convert the input data.

A2. Proposed FSM-based: Each input data is connected to the main inputs of a separate 9-to-1 MUX unit. Two 256-state FSMs, each implemented based on a different Sobol sequence, are connected to the select input of the MUX units. The two input data of each multiplication operation are connected to two separate MUX units while the select input of each MUX is fed with one of the two FSMs. So, the FSM-based design consists of two 256-state FSMs and $k \times k$ 9-to-1 MUX units.

A3. Proposed FSM + One-Hot Encoder: We also implement a third design to further reduce the cost of generating LD bit-streams for applications such as the targeted convolution that a few FSMs (in our case, two) is reused in converting a large number of inputs. In this approach, we need a separate pair of FSM and one-hot encoder for each LD pattern. Converting each input also requires a *Probability Conversion Circuit (PCC)* made of standard AND and OR gates. Fig. 9 shows the design of the PCC unit. PCC has a lower hardware cost than MUX. Hence, when converting a large number of inputs, using PCCs instead of MUXs results in a considerable hardware cost saving. Fig. 10 shows the conversion circuit for converting n input data by sharing one FSM and one one-hot encoder. For the targeted convolution design, two

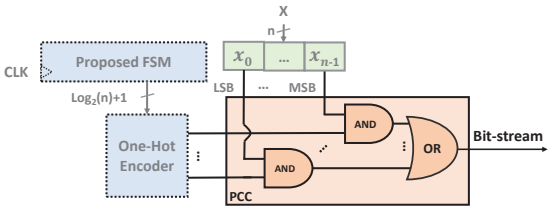

Fig. 9. The Probability Conversion Circuit (PCC).

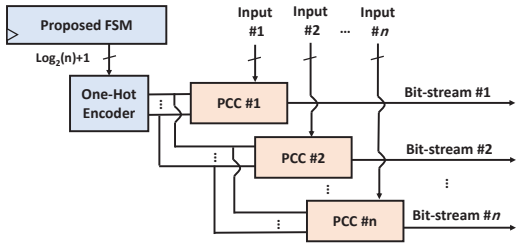

Fig. 10. Converting a large number of inputs from binary to LD bit-stream representation by sharing one one-hot encoder and one FSM.

256-state FSMs (each implemented based on a different Sobol sequence), two one-hot encoders, and $k \times k$ PCCs are needed.

Note that we do not compare with Sim's bit-stream generation approach of [10] as it generates one LD and one unary bit-stream while here we need two independent LD bit-streams. The described bit-stream generation approaches are implemented for different convolution sizes of 3×3, 5×5, 7×7, 9×9, and 11×11 using Verilog HDL and synthesized using the Synopsys Design Compiler v2018.06-SP2 with the 45nm FreePDK library [14]. The synthesis results are reported in Table V. As it can be seen, the proposed FSM + one-hot encoder design provides the minimum bit-stream generation cost by reducing the hardware area cost up to 54% compared to the comparator-based design. Critical path latency and power consumption are also reduced with this approach. Energy consumption is further decreased up to 67% compared to the comparator-based design. This reduction in hardware cost is expected to have a significant impact on the hardware efficiency of the SC-based CNNs such as the ones developed in [20], [10], [19], and [18].

V. Conclusion

LD bit-streams have shown the best performance among all types of stochastic bit-streams. The state-of-the-art LD bit-stream generators are costly and not efficient for all SC designs. In this work, we proposed a low-cost FSM-based LD bit-stream generator for SC designs that need multiple independent bit-streams. The proposed generator reduces the hardware cost up to 80% while generating accurate bit-streams. The parallel design of our bit-stream generator provides on average 66 percent area saving for the $2\times$ and 82 percent area saving for the $8\times$ parallel implementation compared to the state-of-the-art parallel LD bit-stream generator. We showed that the area saving provided by the proposed generator can be exploited to improve the fault-tolerance of the bit-stream generator, a vulnerable component in the SC systems. For noise injection rates below 10 percent, the 5-MR design of the proposed generator provides orders of magnitude reduction in the error rate at a lower hardware cost than the 3-MR comparator-based design. By evaluating the overhead cost of bit-stream generation for SC convolution design, significant area and energy consumption savings were achieved by using the proposed bit-stream generator. A new design for further cost reduction of the FSM-based LD bit-stream generator was also developed for the case of generating a large number of bit-streams.

TABLE V
Synthesis Results of the Bit-stream Generators for the Implemented Convolution Designs

LD Bit-stream Generator Method	Conv. Size	Area (μm^2)	Critical Path Latency (nS)	Power@Max Freq. (mW)	Energy per Cycle (pJ)
Comparator-based	3×3	2060	0.47	2.91	1.37
	5×5	4154	0.49	4.70	2.31
	7×7	7405	0.5	7.27	3.63
	9×9	9938	0.52	10.82	5.63
	11×11	14326	0.53	14.69	7.79
Proposed FSM-based	3×3	1455	0.42	1.73	0.73
	5×5	2943	0.44	2.79	1.23
	7×7	5175	0.46	4.22	1.94
	9×9	8151	0.47	6.09	2.86
	11×11	11871	0.47	8.57	4.02
Proposed FSM + One-Hot Encoder	3×3	1183	0.42	1.69	0.71
	5×5	2054	0.44	2.25	0.99
	7×7	3400	0.46	3.22	1.48
	9×9	5133	0.47	4.30	2.02
	11×11	7333	0.47	5.44	2.56

Acknowledgment

This work was supported in part by the Louisiana Board of Regents Support Fund no. LEQSF(2020-23)-RD-A-26, National Science Foundation grant no. 2019511, and Semiconductor Research Corporation (SRC) Task No. 2988.001.

References

[1] B. Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, pp. 37–172. Springer US, 1969.
[2] A. Alaghi *et al.* The Promise and Challenge of Stochastic Computing. *IEEE Trans. on Computer-Aided Design of Integ. Circ. and Sys.*, 37(8):1515–1531, Aug 2018.
[3] W. Qian *et al.* An Architecture for Fault-Tolerant Computation with Stochastic Logic. *IEEE Trans. on Comp.*, 60(1):93–105, Jan 2011.
[4] S. Gupta *et al.* SCRIMP: A General Stochastic Computing Architecture using ReRAM in-Memory Processing. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1598–1601, 2020.
[5] A. Alaghi and J. Hayes. Fast and accurate computation using stochastic circuits. In *DATE'14*, pp. 1–4, March 2014.
[6] S. Liu and J. Han. Energy Efficient Stochastic Computing with Sobol Sequences. In *DATE'17*, pp. 650–653, March 2017.
[7] M. H. Najafi *et al.* Deterministic Methods for Stochastic Computing Using Low-discrepancy Sequences. In *ICCAD'18*, pp. 1–8, 2018.
[8] S. Asadi and M. H. Najafi. *Accelerating Deterministic Stochastic Computing with Context-Aware Bit-Stream Generator*, pp. 157–162. New York, NY, USA, 2020.
[9] M. H. Najafi *et al.* Performing Stochastic Computation Deterministically. *IEEE Tran. on Very Large Scale Integration (VLSI) Systems*, 27(12):2925–2938, Dec 2019.
[10] H. Sim and J. Lee. A new stochastic computing multiplier with application to deep convolutional neural networks. In *the 54th DAC, 2017*, pp. 1–6, 2017.
[11] D. Jenson and M. Riedel. A Deterministic Approach to Stochastic Computation. In *Proceedings of the 35th Intern. Conf. on Computer-Aided Design*, ICCAD '16, 2016.
[12] S. Liu and J. Han. Toward Energy-Efficient Stochastic Circuits Using Parallel Sobol Sequences. *IEEE Tran. on VLSI Systems*, 26(7):1326–1339, July 2018.
[13] S. Asadi and M. H. Najafi. LDFSM: A Low-Cost Bit-Stream Generator for Low-Discrepancy Stochastic Computing: Late Breaking Results. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, DAC '20. IEEE Press, 2020.
[14] NCSU FreePDK 45nm Library. https://research.ece.ncsu.edu/eda/freepdk/freepdk45/.
[15] M. Riahi Alam *et al.* Exact In-Memory Multiplication Based on Deterministic Stochastic Computing. In *2020 IEEE Intern. Symp. on Circuits and Systems (ISCAS)*, pp. 1–5, 2020.
[16] V. T. Lee *et al.* Architecture considerations for stochastic computing accelerators. *IEEE Trans. on Computer-Aided Design of Integ. Circ. and Sys.*, 37(11):2277–2289, 2018.
[17] I. Koren and C. M. Krishna. *Fault-Tolerant Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2007.
[18] S. R. Faraji *et al.* Energy-Efficient Convolutional Neural Networks with Deterministic Bit-Stream Processing. In *DATE'19*, March 2019.
[19] S. Lee *et al.* Successive log quantization for cost-efficient neural networks using stochastic computing. In *the 56th Annual Design Automation Conference 2019*, DAC '19, 2019.
[20] V. T. Lee *et al.* Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *DATE'17*, March 2017.
[21] H. Sim *et al.* DPS: Dynamic Precision Scaling for Stochastic Computing-based Deep Neural Networks. In *DAC'18*, pp. 13:1–13:6, 2018.