Toward Energy-Efficient Stochastic Circuits using Parallel Sobol Sequences

Siting Liu, Jie Han, *Senior Member, IEEE*

Abstract—Stochastic computing (SC) often requires long stochastic sequences and, thus, a long latency to achieve accurate computation. The long latency leads to an inferior performance and low energy efficiency compared to most conventional binary designs. In this paper, a type of low-discrepancy sequences, the Sobol sequence, is considered for use in SC. Compared to the use of pseudorandom sequences generated by linear feedback shift registers (LFSRs), the use of Sobol sequences improves the accuracy of stochastic computation with a reduced sequence length. The inherent feature in Sobol sequence generators enables the parallel implementation of random number generators with an improved performance and hardware efficiency. In particular, the underlying theory is formulated and circuit design is proposed for an arbitrary level of parallelization in a power of 2. In addition, different strategies are implemented for parallelizing combinational and sequential stochastic circuits. The hardware efficiency of the parallel stochastic circuits is measured by energy per operation (EPO), throughput per area (TPA) and run time. At a similar accuracy, the $8\times$ parallel stochastic circuits using Sobol sequences consume approximately 1% of the EPO of the conventional LFSR-based non-parallelized circuits. Meanwhile, an average of 70 (up to 89) times improvements in TPA and less than 1% run time are achieved. A sorting network is implemented for a median filter (MF) as an application. For a similar image processing quality, a higher energy efficiency is obtained for an $8\times$ parallelized stochastic MF compared to its binary counterpart.

Index Terms—Stochastic computing, low-discrepancy sequences, Sobol sequences, parallel Sobol sequence generator.

1 Introduction

Stochastic computing (SC) is an unconventional computing paradigm proposed in the 1960s [1]. In a typical SC system, the operands are encoded by random binary bit streams or stochastic sequences. Arithmetic operations are performed by bit-wise operations on the stochastic sequences. Different from the complex logic circuits required for implementing binary arithmetic, basic SC elements are very simple for arithmetic operations. For instance, an AND gate can perform multiplication in the unipolar representation, whereas a binary array multiplier costs considerable hardware resources. SC is also highly fault-tolerant against bit-flip errors because a bit flip in a long stochastic sequence has little effect on the computed result. However, a bit flip on the most significant bit of a binary number can dramatically change the result.

To encode a number x in SC, it is firstly converted to a probability p within $[0, 1]$. Then, p is encoded into a stochastic sequence by using a stochastic number generator (SNG). A conventional SNG is shown in Fig. 1. The random number generator (RNG) is usually implemented by a linear feedback shift register (LFSR). An N-bit LFSR generates pseudorandom numbers from 1 to $2^N - 1$ with a period of $2^N - 1$, or from 0 to $2^N - 1$ with a period of 2^N if the all-zero state is added [2]. These pseudorandom numbers can be considered as uniformly distributed random numbers between 0 and $2^N - 1$. The probability, p, is represented by an N-bit binary number normalized by 2^N. Then, by comparing the pseudorandom number with the binary representation of p, a stochastic sequence is generated; a random bit is '1' if $2^N \cdot p$ is larger than the random number or '0' otherwise. The probability of each bit being '1' in the stochastic sequence is then approximately p.

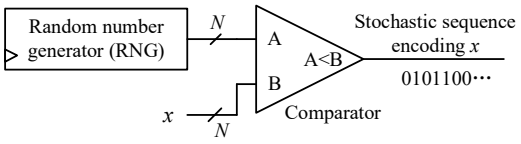


Fig. 1. A stochastic number generator (SNG).

In the unipolar representation, p is directly assigned as the probability, so the unipolar representation deals with values in $[0, 1]$. In the bipolar representation, $p = (x + 1)/2$, such that a value x in $[-1, 1]$ is encoded. Other coding schemes are rarely used due to the complicated conversion scheme. Thus, they are not discussed in this paper.

Although an SC circuit can be simple, its performance is undermined by the required sequence length [3]. Since each bit is generated in a clock cycle, it takes L clock cycles to fully generate and process a stochastic sequence with L bits. As a result, the energy consumption increases proportionally with L and the throughput decreases in an inversely proportional manner with L; therefore, a large L leads to a low energy efficiency. The accuracy of SC can be improved by increasing the sequence length, but the error only decreases with $O(1/\sqrt{L})$ [4].

In [5], the Halton sequence is introduced for use in SC. This low discrepancy (LD) sequence requires a shorter length for achieving the same accuracy compared to LFSR-

• *Siting Liu and Dr. Jie Han are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB , T6G 1H9, Canada.*

Manuscript received September 4, 2017.

generated pseudorandom sequences. When several independent sequences are required, however, the generation of Halton sequences relies on the use of counters with different radices, thus a base conversion becomes necessary for a binary circuit. The base conversion imposes additional hardware overhead on the stochastic circuit. In [6], the Sobol sequence is introduced to replace LFSR-generated sequences to improve the efficiency of an SC circuit. It is shown that in most cases, the use of Sobol sequences leads to a better energy efficiency with a similar accuracy compared to the use of LFSR-generated sequences. However, the improvement is not as significant when compared to the use of Halton sequences.

In this paper, the inherent feature of Sobol sequence generation is exploited for an efficient parallel implementation of the generator. In the proposed $2^m\times$ $(m = 0, 1, 2, \dots)$ parallel generator, only a few extra XOR gates are required to implement the parallelization. Both parallel combinational and sequential circuits are then designed for stochastic computation. With $8\times$ parallelization, a circuit using Sobol sequences consumes approximately 1% of the energy consumption of an LFSR-based circuit with more than 49 times of the throughput per area (TPA) to achieve a similar accuracy. A stochastic median filter is implemented for removing noise in images. At a similar quality, the parallel stochastic Sobol design achieves a higher energy efficiency than its binary counterpart.

The remainder of this paper is organized as follows. Section 2 introduces the background. The parallel Sobol sequence generator design is discussed in section 3. Parallel stochastic circuit designs are elaborated in section 4. The hardware evaluation and comparison are presented in section 5 for several basic stochastic circuits. In section 6, the proposed parallel Sobol circuits are used in a stochastic sorting network to implement a median filter. Section 7 concludes the paper.

2 Review

2.1 Low-discrepancy (LD) Sequences

LD sequences were first proposed to accelerate the convergence process of Monte-Carlo (MC) integration [7], [8]. MC integration requires S-dimensional (or S-independent) random sequences to estimate an S-dimensional numerical integration. Using random sequences with a sufficient length, MC integration can provide an estimate of the result for a numerical integration. It has been shown that a lower discrepancy in the random samples leads to a smaller error in MC integration [7]. An SC circuit can be considered as an MC problem. It is shown in [5] that a stochastic circuit using Halton sequences as LD sequences produces a smaller error than a circuit using pseudorandom sequences.

Discrepancy is a measure indicating how evenly a random sequence is distributed in the sample space. For a random sequence P, it can be quantitatively measured by the star discrepancy $D^*(P)$. For a random sequence with L random points, it is given by [5], [7]

$$D^*(P) = \max_B \left| \frac{A(B;P)}{L} - \lambda(B) \right|, \tag{1}$$

where B is any s-dimensional region in the form $\prod_{i=1}^{s}[0, u_i)$ within an s-dimensional unit cube $\prod_{i=1}^{s}[0, 1]$; $A(B; P)$ is a function counting the number of points satisfying $P \in B$, and $\lambda(B)$ is the Lebesgue measurement of B: it is the length of B if $s = 1$ or the area of B if $s = 2$.

The condition for being an LD sequence is that $D^*(P)$ reaches a convergence speed of $O(\log(L)^{s-1}/L)$. Thus, a longer sequence (with a larger L) and/or fewer independent sequences (with a smaller s) imply a smaller error in an SC circuit. For a small s and a large L, the error in MC integration asymptotically converges to $O(1/L)$, whereas it is approximately $O(1/\sqrt{L})$ for using pseudorandom sequences. Thus, the stochastic circuits using LD sequences can produce more accurate results with shorter sequences.

Several methodologies have been developed to generate different types of LD sequences, including Halton, Sobol and Faure sequences. Software-based generation methods have been developed, but few have been implemented in hardware.

2.2 Sobol Sequence Generation

A direction vector array (DVA) $\{V_k\}$ $(k = 0, 1, \dots, N-1)$ is a group of intermediate variables; these variables can be generated by using primitive polynomials [9]. At least $\lceil \log_2 L \rceil$ direction vectors (DVs) are required for generating a Sobol sequence with a length of L. Uncorrelated Sobol sequences can be generated using different DVAs derived from various primitive polynomials. An algorithm for generating Sobol sequence with a DVA is elaborated in [9] and is briefly summarized in Fig. 2. In Fig. 2, $\{R_i\}$ $(i = 0, 1, 2 \dots, L-1)$ is a Sobol sequence with length L, and "LSZ" stands for "least significant zero".

At each iteration of the loop, the ith quasirandom number, R_i, is XOR-ed with one of the DVs, V_k, to produce R_{i+1}. The index k is determined by the position of the LSZ in the binary form of i. For example, if $i = 11$, i is firstly converted to the binary representation $(1011)_2$. Then, the LSZ of i is at bit 2, thus $k = 2$. Accordingly, V_2 in the DVA is XOR-ed with R_{11} to produce R_{12}. The LSZ detection is of complexity $O(\log N)$ using shift-and-count in a software implementation, whereas a priority encoder can detect the LSZ in hardware. The truth table of a 4-to-2 priority encoder for the LSZ detection is shown in Table 1. "X" stands for "don't care".

For the algorithm in Fig. 2, a hardware Sobol sequence generator is proposed in [10], as shown in Fig. 3. The counter counts i in the for-loop. The priority encoder is used to detect the LSZ. The obtained index k is then passed to the component storing the values in DVA for retrieving V_k. The XOR gates and D flip-flops (FFs) are used to perform

1: $R_0 = 0$; ▷ Initialization
2: **for** $i = 0$ to $L - 2$ **do**
3: $k =$LSZ position of i; ▷ Detection of LSZ
4: $R_{i+1} = R_i \oplus V_k$;
5: **end for**;
6: **return** $\{R_n\}, n = 0, 1, \dots, L - 1$

Fig. 2. Sobol sequence generation algorithm.

TABLE 1
Truth table of a 4-to-2 priority encoder for LSZ detection

Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
X	X	X	0	0	0
X	X	0	1	0	1
X	0	1	1	1	0
0	1	1	1	1	1

$R_{i+1} = R_i \oplus V_k$ for each iteration of the for-loop. An N-bit generator can produce non-repeated Sobol sequences with a length of $L = 2^N$. Parallelization can be implemented on the Sobol sequence generator in Fig. 3, however only a maximum level of $4\times$ is achieved in [10].

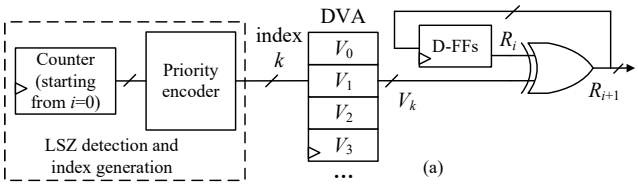
3 Parallel Sobol Sequence Generator

A solution to the long-latency problem is to parallelize the computation by duplicating the SNGs and stochastic circuits. However, the energy efficiency may not be improved because the power consumption would be increased such that the energy consumption would remain nearly the same. Additionally, the hardware cost would also be increased. However, the Sobol sequence generation is inherently parallelizable so that better energy efficiency can be obtained by implementing a high degree of parallelism.

3.1 Formulation

In what follows, the unique feature of Sobol sequence generation is exploited for parallelization. Specifically, the LSZs of continuous non-negative integers follow a regular pattern. For the ease of interpretation, let $L(i)$ indicate the LSZ position of i in its binary format ($i = 0, 1, \dots$). The LSZs of continuous integers are listed in Table 2. Following the algorithm in Fig. 2, the pattern of the LSZs is explored to generate multiple or multi-dimensional Sobol sequences.

As shown in Table 2, $L(i)$ is "nearly periodic" with a period of 8, except for the i's with a remainder of 7 when divided by 8, i.e., $i \equiv 7 \mod 8$. Next, we show that $L(i)$



Working example of a Sobol sequence generator

CLK	*i*	*k*	V_k	R_i	R_{i+1}
reset	0	0	00000000	00000000	00000000
1	1	1	10000000	00000000	10000000
2	2	0	11000000	10000000	01000000
3	3	2	10000000	01000000	11000000
4	4	0	11100000	11000000	00100000
5	5	1	10000000	00100000	10100000
…	…	…	…	…	…

*The D-flip flops are reset for an extra clock cycle to compensate the clock cycle required to fetch the DV.

(b)

Example of a DVA

k	V_k
0	10000000
1	11000000
2	11100000
3	11110000
4	11111000
5	11111100
…	…

(c)

Fig. 3. (a) A Sobol sequence generator, adapted from [10]. (b) An example is given by using (c) the designated DVA. The example reflects actual hardware operation instead of a mathematical model.

TABLE 2
LSZ positions of continuous non-negative integers

i	0	1	2	3	4	5	6	7
$L(i)$	**0**	**1**	0	**2**	0	1	0	**3**
i	8	9	10	11	12	13	14	15
$L(i)$	0	1	0	2	0	1	0	**4**
i	…							
$L(i)$	…							
i	8*j*	8*j*+1	8*j*+2	8*j*+3	8*j*+4	8*j*+5	8*j*+6	8*j*+7
$L(i)$	0	1	0	2	0	1	0	*L*(*j*)+3

is "nearly periodic" with a period of 2^m ($m = 0, 1, 2, \dots$), except for the i's that $i \equiv (2^m - 1) \mod 2^m$. It is proven that the LSZ for the residue class modulo 2^m is

$$L(i) = \begin{cases} L(j) + m & \text{when } i \equiv (2^m - 1) \mod 2^m, \\ & \text{for } j = \lfloor i/2^m \rfloor. \\ L(l) & \text{when } i \equiv l \mod 2^m, \\ & l = 0, 1, \dots, 2^m - 2. \end{cases} \tag{2}$$

An example of (2) is shown in Fig. 4. The detailed mathematical proof is provided as follows.

Lemma 3.1. *$L(i) = k$ is equivalent to [10]:*

$$i \equiv (2^k - 1) \mod 2^{k+1}, (k = 0, 1, \dots). \tag{3}$$

Fig. 5 shows an example for Lemma 3.1.

Corollary 3.1.1. *For a number $i = 2^m \cdot j + 2^m - 1, (i, j, m \in \mathbb{Z}_{\geq 0})$ or $i \equiv 2^m - 1 \mod 2^m$, $L(i) = L(j) + m$.*

Proof. Let $L(j) = k$. Per Lemma 3.1, we have $j \equiv 2^k - 1 \mod 2^{k+1}$, that is, $j = h \cdot 2^{k+1} + 2^k - 1$, $h \in \mathbb{Z}_{\geq 0}$, so that $i = 2^m \cdot j + 2^m - 1 = 2^m \cdot (h \cdot 2^{k+1} + 2^k - 1) + 2^m - 1 = h \cdot 2^{m+k+1} + 2^{m+k} - 1$. Therefore,

$$\begin{aligned} i &\equiv (h \cdot 2^{m+k+1} + 2^{m+k} - 1) \mod 2^{k+m+1} \\ &\equiv (2^{m+k} - 1) \mod 2^{k+m+1}. \end{aligned} \tag{4}$$

By applying Lemma 3.1, we obtain $L(i) = m + k = L(j) + m$. □

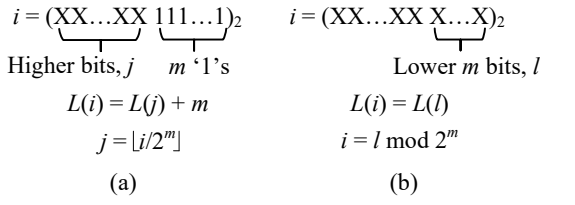


Fig. 4. LSZ position for the residue class modulo 2^m: (a) case 1: the lower m bits are all '1's; (b) case 2: the lower m bits are not all '1's.

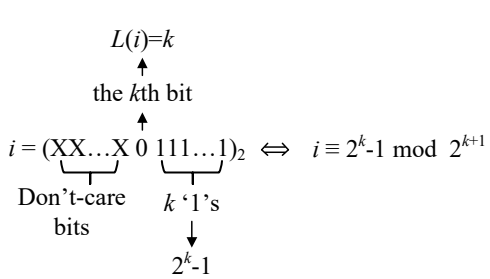


Fig. 5. An illustration of Lemma 3.1.

Corollary 3.1.2. *For a number* $i \equiv l \mod 2^m, (i, m, l \in \mathbb{Z}_{\geq 0}, l \neq 2^m - 1)$, $L(i) = L(l)$.

Proof. Since $i \equiv l \mod 2^m$ and $l \neq 2^m - 1$, $0 \leq l \leq 2^m - 2$, and i can be represented by $i = 2^m \cdot j + l$, $j = 0, 1, \ldots$. Let $L(l) = k$. First, it is clear that $k < m$, which can be proved by contradiction. Per Lemma 3.1, we also have: $l \equiv 2^k - 1 \mod 2^{k+1}$. The LSZ of i can be obtained by

$$i \equiv (2^m \cdot j + l) \mod 2^{k+1}. \tag{5}$$

Since $m > k$, and $m, k \in \mathbb{Z}_{\geq 0}$, so $m \geq k + 1$. Then, the first term on the right hand side of (5) can be removed due to $(2^m \cdot j) \equiv 0 \mod 2^{k+1}$. (5) becomes

$$\begin{aligned} i &\equiv l \mod 2^{k+1} \\ &\equiv 2^k - 1 \mod 2^{k+1}. \end{aligned} \tag{6}$$

Again, we obtain $L(i) = k = L(l)$ with the application of Lemma 3.1. □

By Corollaries 3.1.1 and 3.1.2, (2) is explained.

3.2 Parallelized Sobol SNG and probability estimator

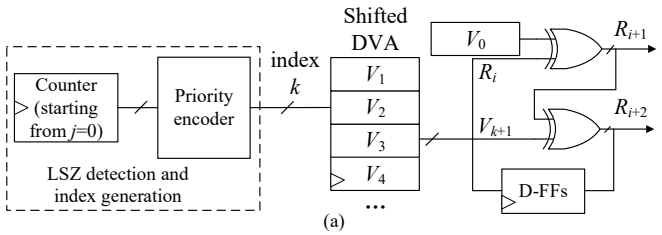
As per Table 2, it is clear that $L(i) = L(0) = 0$ for every even number i, because an even number ends with '0' in the binary format, which is the second case of (2) when $m = 1$. Therefore, the first DV, V_0, can be pre-loaded instead of being computed for every other clock cycle as shown in Fig. 3, i.e., to perform $R_{i+1} = R_i \oplus V_0$. As per (2), when i is an odd number, the same LSZ detection and index generation unit can be used, with the counter counting $j = \lfloor i/2 \rfloor$, ($i = 1, 3, 5, \ldots$ so $j = 0, 1, 2, \ldots$) instead of counting i. The '$+m$' term (or '+1' in this case) in (2) can be offset by shifting the index of DVA instead of changing the LSZ detection and index generation unit as in [10]. Thus, the index of DV V_k, k, loaded from the shifted DVA yield $k = L(j) + 1$, which is in accordance with the first case in (2). The XOR gate at the bottom is used to perform $R_{i+2} = R_{i+1} \oplus V_k$. Accordingly, a $2\times$ parallel Sobol sequence generator is designed as shown in Fig. 6.

A $4\times$ parallel Sobol sequence generator can similarly be constructed for $m = 2$. When $i \equiv l \mod 4$ ($l = 0, 1, 2$), as per (2), the LSZ position for i, $L(i)$, yields $L(i) = L(l)$. Equivalently,

$$L(i) = \begin{cases} 0 & \text{when } i \equiv 0 \mod 4, \\ 1 & \text{when } i \equiv 1 \mod 4, \\ 0 & \text{when } i \equiv 2 \mod 4. \end{cases} \tag{7}$$

Accordingly, V_0 and V_1 are preloaded to perform the XOR operations. When $i \equiv 3 \mod 4$, i.e., $i = 3, 7, 11, \ldots$, the shifted DVA produces V_k with $k = L(i) = L(j) + 2$, where $j = \lfloor i/4 \rfloor$. The counter in Fig. 7 is used for counting j ($j = 0, 1, \ldots$). A $4\times$ parallel Sobol sequence generator is designed and the diagram is shown in Fig. 7.

Similarly, an arbitrary level of $2^m\times$ ($m = 3, 4, \ldots$) parallelization can be implemented by exploring the regular pattern of the LSZ positions. Only several additional XOR gates are required to implement the parallelization. When multiple Sobol sequences are required, a second DVA different from the existing one is inserted, and so are the XOR gate array and the D-FFs. The LSZ detection and index generation components can be shared since the LSZs are the same for different Sobol sequence generations [10]. A generator for two uncorrelated Sobol sequences is shown in Fig. 8.



Working example of a 2× parallel Sobol sequence generator

CLK	i	k	R_i	V_0	R_{i+1}	V_{k+1}	R_{i+2}
reset	0	0	00000000	10000000	10000000	00000000	10000000
1	1	1	00000000	10000000	10000000	11000000	01000000
2	2	0	01000000	10000000	11000000	11100000	00100000
3	3	2	00100000	10000000	10100000	11000000	01100000
4	4	0	01100000	10000000	11100000	11110000	00010000
5	5	1	00010000	10000000	10010000	11000000	01010000
…	…	…	…	…	…	…	…

*The D-flip flops are reset for an extra clock cycle to compensate the clock cycle required to retch the DV.
*The DVA used is the same as the one in Fig. 3(c).

(b)

Fig. 6. (a) Proposed $2\times$ parallel Sobol sequence generator. The D-FFs at the final stage are used for recursively generating the Sobol sequence. (b) A working example shows how the generator works.

An SNG is composed of an RNG and a comparator. Similar to Fig. 1, a Sobol SNG can be implemented by an N-bit Sobol sequence generator and a comparator. For a parallel SNG, 2^m comparators are required to implement $2^m\times$ parallelization, and 2^m stochastic sequences encoding the same value are generated. Because the circuit for generating additional Sobol sequences is small (using a few XOR gates), the hardware cost of the comparators will dominate, especially when the level of parallelization is high. A $2^m\times$ parallel Sobol SNG is shown in Fig. 9.

The probability estimator (PE) can be implemented by an accumulative parallel counter (APC) as proposed in [11]. The APC can take multiple stochastic sequences at one clock cycle and obtain the total number of '1's in parallel stochastic sequences. The diagram is shown in Fig. 10.

4 Parallel Stochastic Circuits

4.1 Basic Computing Elements

To compare the hardware efficiency of using different types of random sequences in SC, several basic stochastic elements

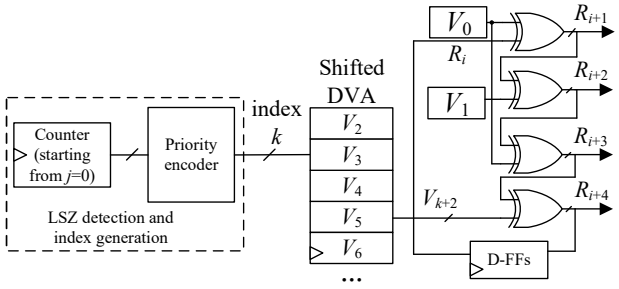


Fig. 7. Proposed $4\times$ parallel Sobol sequence generator.

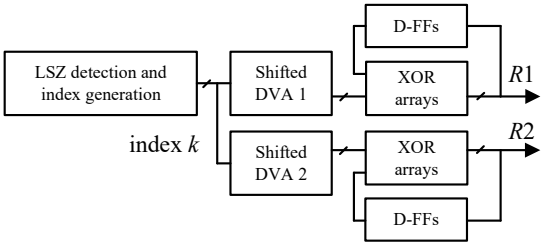


Fig. 8. Two uncorrelated Sobol sequences are generated by the same LSZ detection and index generation component, but different DVAs.

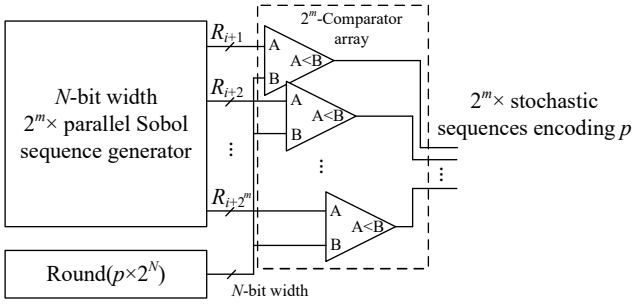


Fig. 9. A $2^m\times$ parallel Sobol SNG.

are considered: (a) an AND gate implementing a multiplier; (b) a multiplexing circuit computing the Bernstein polynomial [12] as a high-dimensional case; (c) a divider based on stochastic integrator (with an up/down counter) as a stochastic sequential element. The schematics of the circuits are shown in Fig. 11.

In Fig. 11(a), given uncorrelated stochastic sequences encoding x_1 and x_2, the probability of the output of the AND gate is $y = x_1x_2$ in the unipolar representation. The multiplexing circuit in Fig. 11(b) is used to calculate an Nth order Bernstein polynomial $f(x) = \sum_{i=0}^{N} z_i B_{i,N}(x)$, where $B_{i,N}(x) = \binom{N}{i} x^i (1-x)^{N-i}$. The selection signal is produced by a binary adder summing up the independent stochastic bit streams encoding x. A stochastic divider employs the converged value of the up/down counter to estimate the quotient of two numbers [13]. As shown in Fig. 11(c), when an equilibrium state is reached, the probabilities of counting-up and counting-down are equal.

4.2 Parallel Computing Elements

4.2.1 Parallel combinational elements

The implementation of parallel stochastic combinational elements is straightforward. In general, 2^m duplicates of the original stochastic circuit can implement $2^m\times$ parallelization. Fig. 12 shows a $4\times$ unipolar stochastic multiplier, which is implemented by four duplicates of the AND gate.

Similarly, a parallel stochastic Bernstein polynomial circuit can be implemented by duplications.

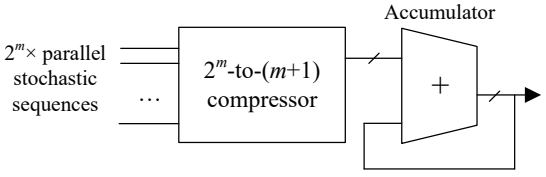


Fig. 10. An accumulative parallel counter (APC) adapted from [11].

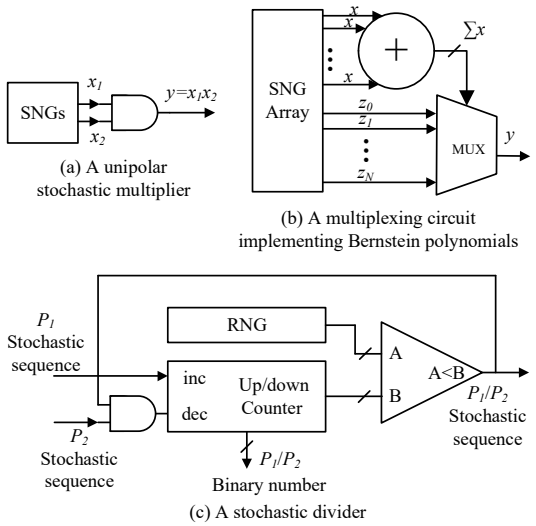


Fig. 11. Basic stochastic elements for hardware efficiency evaluation.

4.2.2 Parallel sequential elements

Stochastic sequential elements mainly consist of two categories: finite state machine (FSM)-based and stochastic integrator-based. For the FSM-based circuits, the functionalities are based on the theory of Markov chains, that is, the current state of the FSM is only directly related with its last state. This type of circuits requires that each bit in the input stochastic sequence is independently generated from one clock cycle to another in a temporal manner. However, the bits in a Sobol sequence are generated from the previous bits, so it violates the independence requirement and will not produce accurate results. For example, a stochastic tanh (Stanh) circuit using a Sobol sequence creates a hard-threshold function instead of an S-shaped curve as shown in Fig. 13. However, the use of Sobol sequences improves the accuracy of stochastic integrator-based circuits. The stochastic divider is considered as an illustrative example.

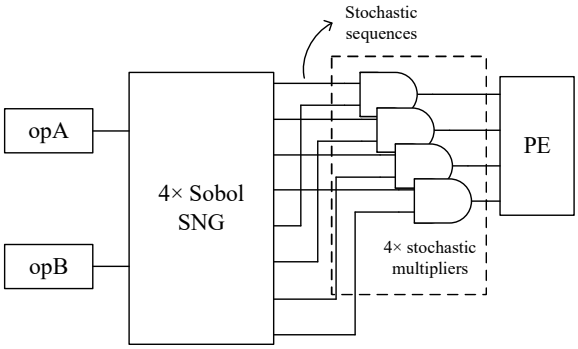


Fig. 12. A $4\times$ unipolar stochastic multiplier. The SNG generates parallel stochastic sequences for the multiplier and multiplicand, opA and opB, respectively. PE stands for a probability estimator.

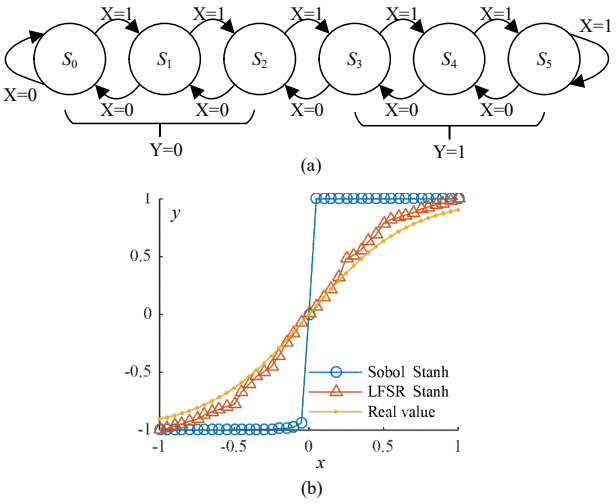


Fig. 13. Using Sobol sequences in the FSM-based Stanh circuit produces inaccurate results: (a) the state transition graph of Stanh, (b) the results of $\tanh(3x)$ computed by using Sobol and LFSR-generated sequences. x is encoded by the stochastic sequence X, while y is encoded by the sequence Y.

Because it does not increase the convergence speed of a stochastic divider by simply duplicating the circuit, the parallelization is implemented by doubling the input sequences of the up/down counter to accelerate the computation as shown in Fig. 14. To help understand the underlying theory, the mathematical model of a stochastic divider is analyzed as follows.

In the stochastic divider in Fig. 11(c) [13], the computation of the quotient relies on the convergence of the stochastic integrator until its equilibrium state is reached. Let $p_{1,i}$, $p_{2,i}$ and q_i be the ith bit in the stochastic sequences encoding P_1, P_2 and P_1/P_2. Let the integer stored in the N-bit counter be Y_i. The probability value carried by the counter is then $y_i = Y_i/2^N$, where N is the bit width of the up/down counter. The AND gate serves as a stochastic multiplier, and the output of the AND gate is $p_{2,i} \cdot q_i$. The up/down counter is updated by the rule [14]:

$$Y_{i+1} = Y_i + p_{1,i} - p_{2,i} \cdot q_i. \tag{8}$$

If the initial value stored in the counter is Y_0, the value of Y_k at an arbitrary kth clock cycle is obtained by accumulating (8) for $i = 0, 1, \ldots, k-1$ as

$$Y_k = Y_0 + \sum_{i=0}^{k-1} (p_{1,i} - p_{2,i} \cdot q_i). \tag{9}$$

Additionally, the sequence $\{q_i\}$ is generated by comparing the number stored in the counter and the uniformly distributed random number generated by the RNG, in a similar manner as an SNG, so the expectation of q_i is $\mathbb{E}[q_i] = y_i = Y_i/2^N$. Then, the expectation of Y_k is given by

$$\mathbb{E}[Y_k] = Y_0 + \sum_{i=0}^{k-1} (P_1 - P_2 y_i). \tag{10}$$

Substituting Y_k and Y_0 by y_k and y_0, (10) becomes

$$\mathbb{E}[y_k] = y_0 + \frac{1}{2^N} \sum_{i=0}^{k-1} (P_1 - P_2 y_i). \tag{11}$$

The Euler method is a first-order iterative algorithm solving an ordinary differential equation (ODE). For an ODE $\frac{\mathrm{d}y(t)}{\mathrm{d}t} = f(t)$, a one-step solution is calculated from the previous estimate

$$\hat{y}_{i+1} = \hat{y}_i + h f(t_i), \tag{12}$$

where h is the step size and $t_i = hi$. $\hat{y}_i$ is the Euler numerical solution at the ith step. Given an initial condition of an ODE y_0, the kth step numerical solution is calculated by accumulating (12) through $i = 0, 1, \ldots, k-1$, such that

$$\hat{y}_k = y_0 + h \sum_{i=0}^{k-1} f(t_i). \tag{13}$$

By comparing (11) and (13), it can be seen that the stochastic integrator of the divider provides an unbiased estimate to the Euler solution of the ODE:

$$\frac{\mathbf{d}y(t)}{\mathbf{d}t} = P_1 - P_2 y(t), \tag{14}$$

with a step size of $h = 1/2^N$. By solving (14) analytically, the convergence process of the counter is approximated by

$$y(t) = \frac{P_1}{P_2} - \frac{1}{P_2}(P_1 - P_2 y_0) e^{-P_2 t}, \tag{15}$$

where t is discretized to the number of clock cycles, i.e., $t_i = h \cdot i = i/2^N$, $i = 0, 1, \ldots$ and y_0 is set to the initial value of the counter. As t approaches infinity, the exponential term approaches 0 so that $y(t)$ converges to the quotient of P_1 and P_2. The convergence process is governed by an exponential function, and the speed of convergence is determined by the exponent.

The convergence process of the proposed parallel stochastic divider in Fig. 14 can similarly be evaluated. The expectation of the "INC" input parallel stochastic sequences can be obtained from the distribution of $\sum P_1$ in Table 3, as $\mathbb{E}[\text{INC}] = 0 \times (1 - P_1)^2 + 2P_1(1 - P_1) + 2 \times P_1^2 = 2P_1$. Similarly, $\mathbb{E}[\text{DEC}] = 2P_2 y(t)$. Then the expectation of the value stored in the counter at the kth clock cycle is given by

$$\mathbb{E}[Y_k] = Y_0 + \sum_{i=0}^{k-1} (2P_1 - 2P_2 y_i). \tag{16}$$

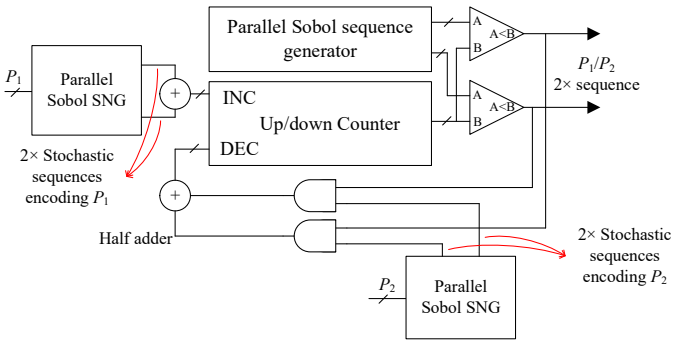


Fig. 14. A $2\times$ parallel stochastic divider.

TABLE 3
Probability distribution of $\sum P_1$.

$\sum P_1$	0	1	2
probability	$(1-P_1)^2$	$2P_1(1-P_1)$	P_1^2

Due to (16), the parallel stochastic divider actually solves

$$\frac{\mathbf{d}y(t)}{\mathbf{d}t} = 2P_1 - 2P_2 y(t). \tag{17}$$

The solution for (17) is

$$y(t) = \frac{P_1}{P_2} - \frac{1}{P_2}(P_1 - P_2 y_0)e^{-2P_2 t}. \tag{18}$$

Compared to (15), the exponent is doubled, so the convergence process of the $2\times$ parallel divider design is twice as fast as the one without using any parallelization.

4.2.3 Convergence time of a stochastic divider

The time when the value of $y(t)$ converges, referred to as the convergence time, can be estimated by (15) or (18). Since the value stored in the counter is an N-bit number, the resolution is $1/2^N$ for encoding a probability in $[0, 1]$. When the absolute value of the exponential term in (15) (or (18)) is smaller than the resolution of the counter, the state of the counter is considered converged. The convergence time can then be estimated. Additionally, both the divisor and dividend are shifted to the left by the same number of bits such that the MSB of the divisor is '1'. By doing so, the divisor, P_2, is amplified, so that the term containing the exponential expression in (15) or (18) approaches to 0 faster, while the quotient is not changed. Assume that the convergence time is t_{conv}, an inequality is composed to find t_{conv} for the original stochastic divider as per (15),

$$\left|(P_1 - P_2 y_0)\frac{1}{P_2}e^{-P_2 t_{conv}}\right| < \frac{1}{2^N}. \tag{19}$$

By solving the inequality, t_{conv} is estimated to be

$$t_{conv} \geq \frac{1}{P_2}(N\log 2 + \log\left|\frac{P_1 - P_2 y_0}{P_2}\right|) \tag{20}$$

This approach is used for estimating the run time of a stochastic divider. Also, it can be applied to any stochastic integrator-based circuits, whose convergence follows an exponential function.

5 Experiments and Results

5.1 Metrics

The performance of the SC elements are examined by energy per operation (EPO), throughput per area (TPA) [15] and run time. The root-mean-squared error (RMSE) is used to measure the accuracy.

Given the sequence length L and level of parallelization P for a stochastic arithmetic operation, the EPO for an SC element can be calculated by

$$\text{EPO} = \text{Total Power} \times T_{clk} \times L/P, \tag{21}$$

where T_{clk} is the clock period and power is measured at the corresponding T_{clk}. $P = 1$ when no parallelization is applied. Similarly, the EPO of an SNG is measured by the energy consumption for generating one bit. Throughput is used to measure how much information a system can process during a unit time. The TPA of an SNG is measured by the number of stochastic bits generated in a unit time per unit area. The TPA of a stochastic circuit is measured by the number of computation results produced in a unit time per unit area. The run time is considered as $t_c \times L/P$ for producing one result for a combinational circuit, where t_c is the critical path delay. The sequence length of a stochastic divider is determined by the convergence time. Subsequently, the TPA is given by

$$\text{TPA} = 1\text{ bit}\times P/t_c/\text{area}, \tag{22}$$

for an SNG and

$$\text{TPA} = 1/(t_c \times L/P)/\text{area}, \tag{23}$$

for a stochastic arithmetic circuit.

The run time, T, is evaluated by

$$T = t_c \times L/P. \tag{24}$$

The critical path delay, area and power consumption are first obtained by the Synopsis Design Compiler with a 28nm industrial process. The same temperature and process corners are applied to all the circuits. The EPO, TPA and run time are then computed. The RMSE is obtained by using 10,000 random trials for each circuit.

5.2 Evaluation of the proposed Sobol SNG

5.2.1 Accuracy

The accuracy of an 8-bit Sobol SNG is compared with an 8-bit LFSR-based SNG to show how accurate a real number is encoded by using those two different types of SNGs with different sequence lengths. The numbers to be encoded are randomly chosen. For a different sequence length, the ratio of '1's in the stochastic sequences generated by the two SNGs are calculated. The difference between the ratio and the number to be encoded is measured by RMSE and the results are shown in Fig. 15.

The accuracy of a stochastic sequence generated by a Sobol SNG is consistently higher than that generated by an LFSR-based SNG. Note that the parallelization does not affect the accuracy since the total number of 1's remains the same, but with a faster generation rate.

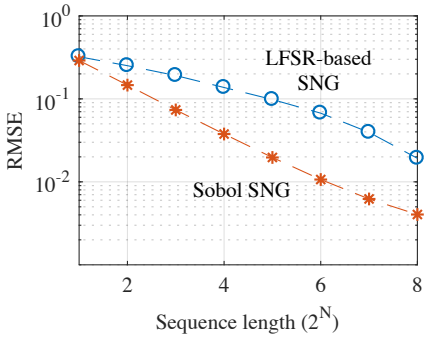


Fig. 15. Accuracy of the Sobol SNG and LFSR-based SNG.

5.2.2 Hardware efficiency

A Sobol SNG consists of a sequence generator and comparators. A $2^m\times$ parallelization requires approximately $(2^m-1)\times N$ more XOR gates than a single Sobol SNG, where N is the bit width of the generator. The number of comparators required is the same as the level of parallelization, i.e., 2^m. Table 4 shows the components required in a $2^m\times$ parallel Sobol SNG.

The hardware efficiency of a Sobol SNG is measured in EPO,TPA and generation time. For the Sobol SNG, different levels of parallelization are implemented. The results are shown in Fig. 16 for an 8-bit SNG. The performance of an 8-bit LFSR-based SNG is used as a reference.

When no parallelization is applied, as can be seen in Fig. 16, the Sobol SNG has a lower hardware efficiency compared with the LFSR-based SNG. However, the EPO and TPA of a Sobol SNG increase with the level of parallelization because of its small hardware cost.

5.3 Stochastic Combinational Circuits

5.3.1 Accuracy

For the AND-based stochastic multiplier, two independent stochastic sequences are required. For a third-order Bernstein polynomial circuit, at least 4-dimensional or 4 independent stochastic sequences are required. The accuracy comparisons for the stochastic multiplier and Bernstein polynomial circuit are shown in Fig. 17, in which another type of LD sequences, Halton sequence, is also considered. The Halton sequence generators are implemented by inversely-mapped counters using different bases [5].

As shown in Fig. 17, for the same sequence length, the accuracy of the results produced by using Sobol sequences are mostly higher than those obtained using LFSR-generated and Halton sequences. For the stochastic multiplier, it is also observed that as the sequence length increases, the RMSE of the LD sequences-based circuits decreases faster than that of the LFSR-based design. However, due to the increased dimension of the sequences, the RMSE of the Bernstein polynomial circuit using LD sequences does not converge as fast as the multiplier.

5.3.2 Hardware Efficiency

The hardware efficiency of stochastic multipliers are shown in Figs. 18, 19 and 20. For the same accuracy, a lower EPO, a larger TPA and a shorter run time indicate a more efficient hardware design. From Figs. 18, 19 and 20, most Sobol-based designs show a higher efficiency than Halton and LFSR-based designs. It is due to the shorter sequence length required in a Sobol-based design to achieve a similar RMSE.

TABLE 4
Hardware cost for a $2^m\times$ parallel Sobol SNG

Parallelism	LSZ detection & DVA	XOR gates	Comparators
$1\times$	1	N	1
$2\times$	1	$2N$	2
$4\times$	1	$4N$	4
...	...	...	...
$2^m\times$	1	2^mN	2^m

Also, the parallelization in Sobol sequence generation makes a design more efficient because of the small hardware cost to implement the parallelization. The Sobol- and Halton-based multipliers consume a similar energy when no parallelization is applied, whereas a parallel Sobol design is more energy efficient than the Halton-based design. If parallelization is implemented for the Halton- or LFSR-based designs, the EPO and TPA would not change much due to the extra power consumption and hardware cost, albeit with a reduction in run time.

The stochastic multiplier using 2^8-bit Sobol sequences has a similar RMSE with a design using 2^{12}-bit LFSR-generated sequences. The EPO of the $8\times$ parallel Sobol multiplier costs only 1.44% of the energy of the LFSR-based design with 49.80 times of the TPA. The Bernstein polynomial circuits are also evaluated, and the results show a similar trend as the multipliers.

5.3.3 Curse of dimensionality

The benefits using LD sequences can diminish as the dimension of the sequences increases [7], which is referred to as the "curse of dimensionality". To investigate how the number of dimensions affects the accuracy and the hardware efficiency of a Sobol-based design, n-dimensional random sequences are employed to implement $(n-1)$th order Bernstein polynomial with the same sequence length. 10,000 Monte Carlo simulation runs are carried out, with the coefficient of the Bernstein polynomial randomly chosen. As shown in Fig. 21(a), the RMSEs of LFSR-based designs oscillate around 0.04 throughout all dimensions. Although the RMSE of a Sobol-based design increases, it does not exceed that of an LFSR-based design for up to 20 dimensions. The RMSE of a Halton-based design is slightly larger than that of the Sobol counterpart.

The EPO, TPA and run time are reported in Figs. 21(b), (c) and (d). As can be seen, a $4\times$ parallel Sobol-based design always results in the lowest EPO, the largest TPA and the shortest run time. The EPO of the Halton-based

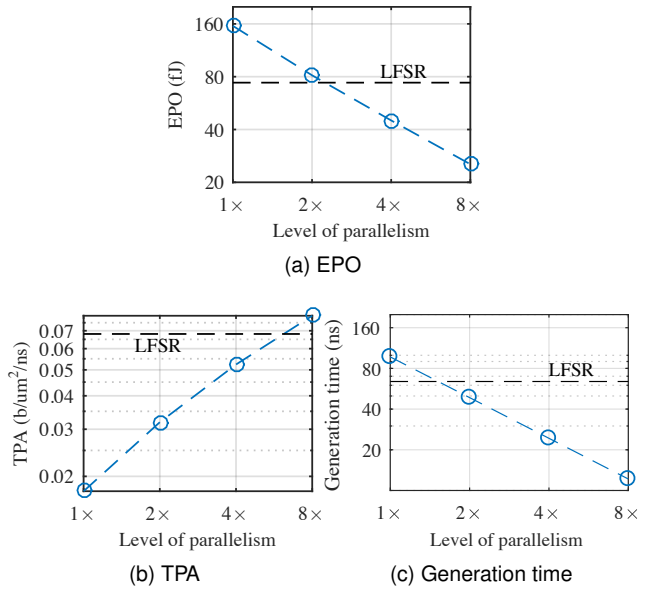


Fig. 16. Hardware measurements of SNGs.

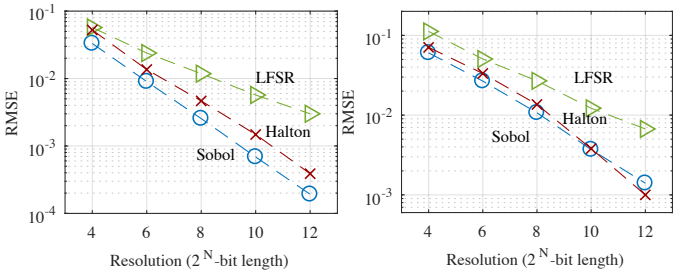


Fig. 17. Accuracy of combinational stochastic circuits using Sobol, Halton and LFSR-generated sequences.

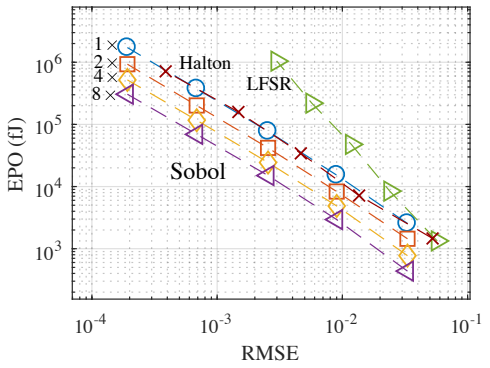


Fig. 18. EPO of stochastic multipliers using 2^{12}-bit, 2^{10}-bit, 2^8-bit, 2^6-bit and 2^4-bit length from left to the right.

design increases quickly with the order of the polynomial, because a larger counter is required in the larger base for a higher-dimensional Halton sequence. The EPO of the Sobol-based design increases rather slowly because the LSZ detection and index generation unit is shared to generate multi-dimensional Sobol sequences as shown in Fig. 8.

5.4 Stochastic Sequential Circuits

5.4.1 Verification of the proposed parallel stochastic divider

The parallel stochastic divider design proposed in Section 4 is first verified by using hardware simulation. An 8-bit counter is used and different levels of parallelization are applied. The result produced by using LFSRs is also considered for comparison as shown in Fig. 22. The predicted

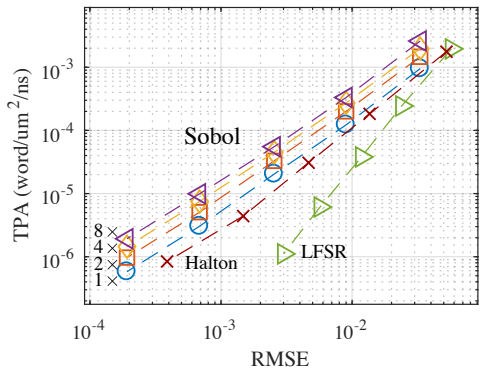


Fig. 19. TPA of stochastic multipliers using 2^{12}-bit, 2^{10}-bit, 2^8-bit, 2^6-bit and 2^4-bit length from left to the right.

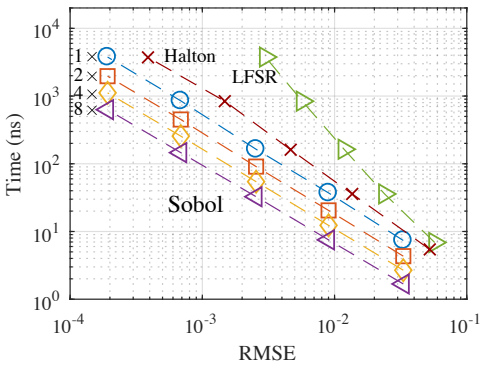


Fig. 20. Run time of stochastic multipliers using 2^{12}-bit, 2^{10}-bit, 2^8-bit, 2^6-bit and 2^4-bit length from left to the right.

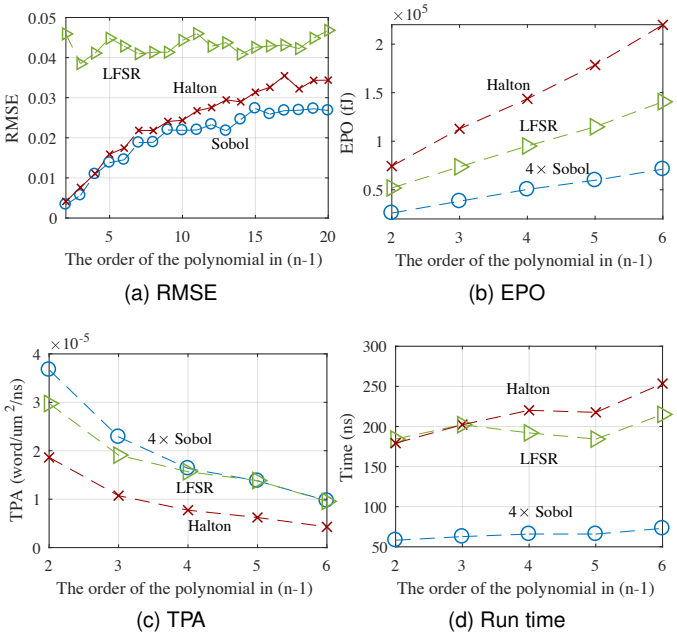


Fig. 21. Accuracy, EPO, TPA and run time of an (n-1)th order Bernstein polynomial circuit using different random sequences with the same sequence length. An n-dimensional random sequence is required to implement an (n-1)th order Bernstein polynomial circuit.

convergence time is calculated by using (19) with $P_1 = 0.8$, $P_2 = 0.9$ and $y_0 = 0.5$.

From Fig. 22, all the parallel divider designs converge to the accurate result with different speeds. The Sobol design using $4\times$ parallelization is the fastest, followed by the designs using $2\times$ parallelization and no parallelization. The result produced by using LFSRs fluctuates significantly and leads to an ambiguous result, as shown in Fig. 22. However, the results produced by Sobol sequences are more stable. Additionally, the predicted convergence time fits well with the simulation results with less glitches for Sobol sequences.

5.4.2 Accuracy

The accuracy is measured with different levels of parallelization and different bit widths for the up/down counter. The results are shown in Fig. 23.

As shown in Fig. 23, the parallelization has little effect on the accuracy of Sobol-based dividers, and the RMSEs of

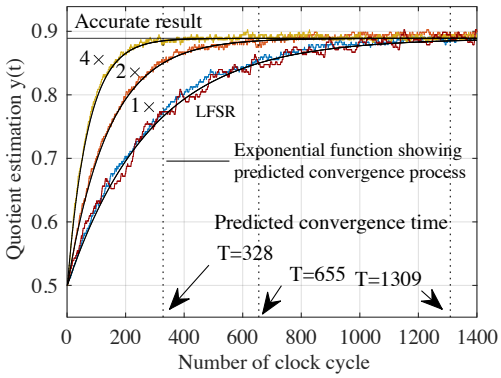


Fig. 22. Convergence processes of stochastic dividers using Sobol sequences with $1\times$, $2\times$ and $4\times$ parallelization (colored lines), computing $0.8 \div 0.9$. $y(t)$ is initialized with $y_0 = 0.5$. The red bold curve with larger glitches is produced by a stochastic divider using LFSR-generated sequences.

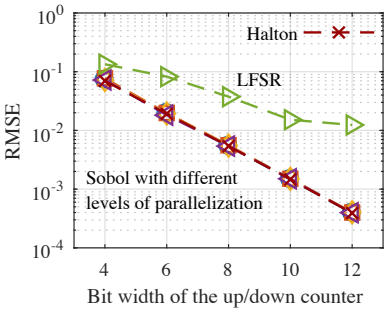


Fig. 23. Accuracy of stochastic dividers using Sobol sequences with $1\times$, $2\times$, $4\times$ and $8\times$ parallelization with different sizes of the up/down counter. The RMSEs for using Halton and LFSR-generated sequences without parallelization are also compared.

those designs almost overlap with one another. The RMSE of a Sobol-based design decreases linearly with the bit width of the up/down counter and it is consistently smaller than that of an LFSR-based design. A Halton-based divider shows a similar accuracy as its Sobol counterpart.

5.4.3 Hardware Efficiency

For different operands, the convergence time (in the number of clock cycles) of the stochastic divider is different. An average convergence time is estimated for randomly chosen operands and the average EPO, TPA and run time are obtained by using the average convergence time. Different levels of parallelization are applied to evaluate the improvement in performance and hardware efficiency. The results are shown in Figs. 24, 25 and 26.

As can be seen, as the level of parallelization increases, the energy and hardware efficiency improve. However, the improvement of a higher level parallelization is not as significant as at a lower level parallelization. It is evident in Fig. 25, that the TPAs of $8\times$ parallel stochastic dividers are close to $4\times$ ones, whereas the TPAs of $2\times$ parallel stochastic dividers are well separated from the ones without parallelization. It occurs because the number of comparators increases linearly with the level of parallelization, as shown in Table 4. Thus, they dominate the energy and hardware cost of the design.

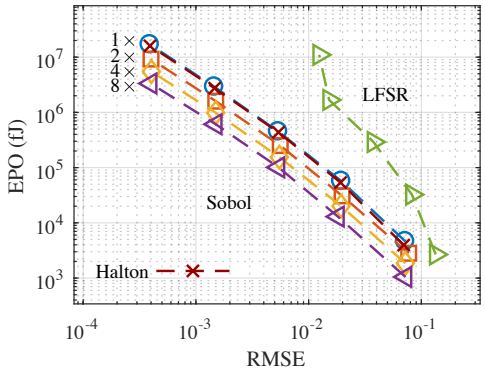


Fig. 24. EPO of stochastic dividers using Sobol sequences with $1\times$, $2\times$, $4\times$ and $8\times$ parallelization with 12-bit, 10-bit, 8-bit, 6-bit and 4-bit up/down counter from left to the right. The LFSR-based design without parallelization is also shown.

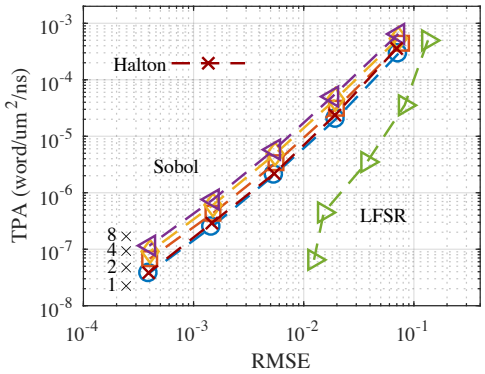


Fig. 25. TPA of stochastic dividers using Sobol sequences with $1\times$, $2\times$, $4\times$ and $8\times$ parallelization with 12-bit, 10-bit, 8-bit, 6-bit and 4-bit up/down counter from left to the right. The LFSR-based design without parallelization is also shown.

For the Halton-based designs, they have a similar EPO, TPA and run time as their Sobol counterparts without parallelization. However, the Sobol-based design benefits from the efficient implementation of parallelization, so it outperforms its Halton counterpart in all considered metrics.

For the LFSR-based designs, it takes at least two more bit widths to achieve a similar accuracy as the Sobol-based

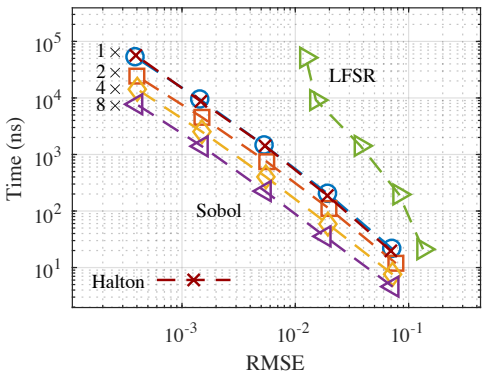


Fig. 26. Run time of stochastic dividers using Sobol sequences with $1\times$, $2\times$, $4\times$ and $8\times$ parallelization with 12-bit, 10-bit, 8-bit, 6-bit and 4-bit up/down counter from left to the right. The LFSR-based design without parallelization is also shown.

designs. Therefore, it results in a longer run time and a lower hardware efficiency. When a 12-bit LFSR-based design is compared with an 8-bit Sobol design, the $8\times$ parallel Sobol-based design costs 0.92% of the energy consumption and produces 89 times of the TPA of an LFSR-based design. Additionally, the accuracy of an 8-bit Sobol-based divider is higher than that of the 12-bit LFSR-based divider.

The basic stochastic elements are not compared with their binary counterparts because it either favors the SC design without considering the SNGs and PEs or otherwise, it imposes a large overhead on the stochastic arithmetic elements. Instead, the hardware efficiency is compared at the application level in the next section.

6 Application

6.1 A sorting network and median filter design

A sorting network rearranges a list of values in an ascending or descending order. It is widely used in modern computer systems for file-matching, data searching and filtering, among other applications [16]. A sorting network can be implemented in hardware by comparators and multiplexers for comparing and swapping the values. However, the hardware cost is very high for large volumes of input data. The depth of a sorting network is defined as the largest number of comparators an input goes through the network. It is in the order of $O(n\log(n)^2)$ [17], where n is the size of the input data. If the compare-and-swap is executed in parallel, the run time is proportional to the network depth. Here, a stochastic circuit is used to implement the sorting network with an energy-efficiency compare-and-swap unit.

The design of a basic stochastic unit is shown in Fig. 27. The underlying principle can be explained by considering the AND gate as an example. The probability of its output sequence is given by:

$$\begin{aligned} P[P_{out}=1] &= P[(P_X > RN) \wedge (P_Y > RN)] \\ &= P[\min\{P_X, P_Y\} > RN]. \end{aligned} \tag{25}$$

Thus, the probability of the output sequence is the lesser value of P_X and P_Y. Similarly, the OR gate can be shown to obtain the larger value of P_X and P_Y:

$$\begin{aligned} P[P_{out}=1] &= P[(P_X > RN) \vee (P_Y > RN)] \\ &= P[\max\{P_X, P_Y\} > RN]. \end{aligned} \tag{26}$$

The output stochastic sequence can be used for further comparing and swapping with the values in another stochastic sequence generated from the same RNG such that (25) is satisfied. Hence, a sorting network can be constructed by using just one RNG generating the stochastic sequences for all the input data and the AND and OR gates for comparing and swapping the stochastic sequences.

A median filter (MF) is often used to reduce the salt-and-pepper noise in images without losing the edges. It replaces each pixel in an image with the median value of the surrounding pixels. A 3×3 MF can be implemented by a sorting network [18]. A design is shown in Fig. 28 with the stochastic sorter as its basic unit.

Since the input sequences are required to be generated by the same RNG as per (25), only one RNG is used for the first layer of the sorters as shown in Fig. 29. Passing through the first layer of sorters, the output sequences are still the original stochastic sequences with the same permutations of '0's and '1's for a certain value, but with a different order. It means that the output sequences can be directly used as the input sequences in the next compare-swap layers instead of being re-generated by an SNG. As a result, only one RNG is required for the stochastic MF. Additionally, the low-discrepancy characteristic is maintained after stages of computations. Both Sobol sequences and pseudorandom sequences can be used for this scheme.

6.2 Experimental results of the median filter

This stochastic MF design is tested on the 8-bit grey image "cameraman" polluted by salt & pepper noise with a density of 0.1. A conventional binary design is also considered for comparison. Different levels of approximation are applied to the binary design by truncating the bit width. Since there are in total 8 layers of comparing and sorting, the binary MF takes 8 clock cycles to produce the final result. The original and polluted images are depicted in Fig. 30 and the filtering results are shown in Fig. 31.

It can be seen that both binary and stochastic circuits can filter the noise with a high quality. However, when the sequence length is reduced to 16 bit for a stochastic design or the bit width is reduced to 4 for a binary design, some severe distortion is present. For the stochastic MF using LFSR-generated sequences, the image can be either darker or brighter than the original image due to random fluctuations. There is a loss of detail in Figs. 31(f) and (g) for 16-bit and 32-bit stochastic designs using LFSRs.

The quality of the output image is measured by the peak-signal-to-noise-ratio (PSNR). The results are illustrated in Fig. 32. Because of the random noise and random fluctuations in LFSR-based stochastic circuits, 100 Monte-Carlo simulation runs are carried out to measure the mean and

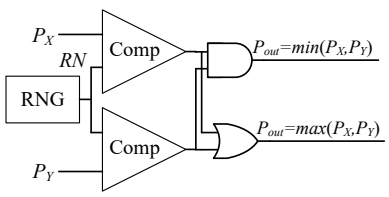


Fig. 27. Stochastic sorter circuit. The stochastic sequence encoding a larger value is moved downward.

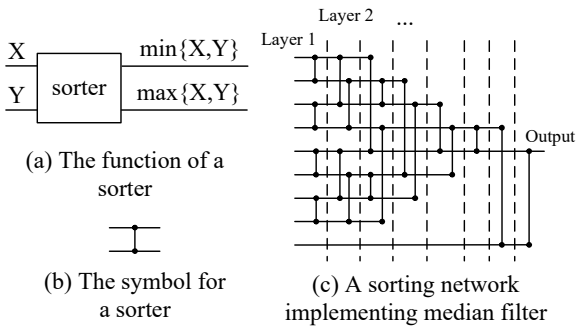


Fig. 28. The basic unit for implementing a median filter.

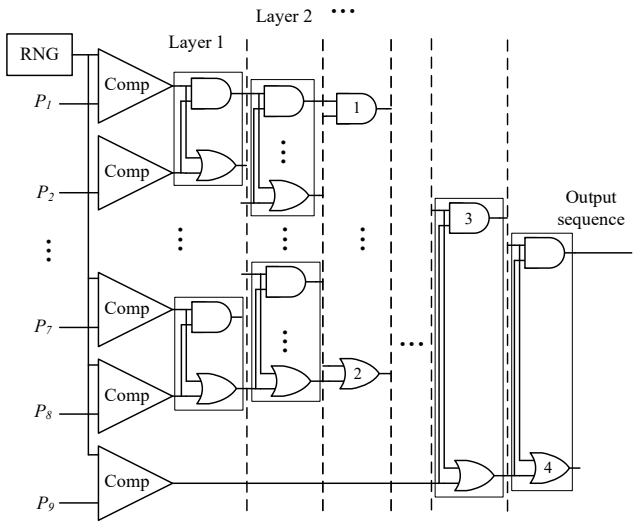


Fig. 29. Stochastic median filter based on the sorting network in Fig. 28(c). Since the output stochastic sequences from gates 1, 2, 3 and 4 are not used, they can be removed for saving hardware. Each sorter unit is marked by a rectangle.

standard deviation of the PSNRs. For both the Sobol and binary designs, the deviation is purely introduced by the randomly added noise rather than the design itself due to their deterministic nature, thus the deviation is very small. As can be seen, the LFSR-based stochastic design results in the lowest PSNR because of the randomness in the generation of the random sequences. The Sobol-based design has overall the best PSNR even over the binary design, because a simple truncation in the binary design introduces a bias and the resulting value is equal to or smaller than its actual value. Since a base-2 Halton sequence is the same as a one-dimensional Sobol sequence, they produce results with a similar accuracy. The efficiency of the hardware implementation is again measured by EPO, TPA and run time. The results are shown in Figs. 33, 34 and 35.

As can be seen from Fig. 33, the energy efficiency is improved by using parallel Sobol-based designs. The EPO of the $8\times$ parallel stochastic MFs are smaller than its binary counterparts except for the 8-bit binary design. For the same resolution, the hardware efficiency in terms of EPO, TPA and run time for the binary circuits is proportional to the bit width N, whereas it is proportional to 2^N for the stochastic circuits. Therefore, the slope of the EPO curve in Fig. 33 for the binary circuits is less steep than those for the stochastic circuits.

However, the TPA and run time are still the shortcomings of a stochastic MF. The stochastic MF is only faster with 16-bit sequences and $8\times$ parallelization than 4-bit binary design. Due to the extra hardware cost of the comparators and the increased critical path delay, the TPA of a Sobol MF design is not significantly improved by parallelization. However, all parallel Sobol designs consistently show advantages over the LFSR-based designs for all considered metrics with a better image processing quality. The Halton-based designs require a similar EPO as $2\times$ parallel Sobol designs, while it is not as efficient as Sobol-based designs with $4\times$ or a larger parallelization.

(a) Original image (b) Polluted image

Fig. 30. Original and polluted "cameraman" images.

7 Conclusion

The use of parallel Sobol sequences in stochastic computing achieves higher energy efficiency, higher throughput and shorter run time than the use of conventional LFSR-generated sequences and the Halton sequence, another type of LD sequences. The parallelization exploits the regular patterns of the LSZ positions of continuous non-negative integers. As a result, it only imposes a small hardware overhead of a few XOR gates.

The proposed stochastic circuits using parallel Sobol sequences are utilized in a sorting network and a MF. The Sobol-based MF shows a higher energy efficiency than its binary counterparts and it consistently outperforms an LFSR-based MF with a higher filtering quality.

References

[1] B. R. Gaines, *Stochastic Computing Systems*. Boston, MA: Springer US, 1969, pp. 37–172.

[2] P. K. Gupta and R. Kumaresan, "Binary multiplication with PN sequences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 603–606, 1988.

[3] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s, p. 92, 2013.

[4] J. Han, H. Chen, J. Liang, P. Zhu, Z. Yang, and F. Lombardi, "A stochastic computational approach for accurate and efficient reliability evaluation," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1336–1350, 2014.

[5] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *Proceedings of the conference on Design, Automation & Test in Europe (DATE)*. IEEE, 2014, p. 76.

[6] S. Liu and J. Han, "Energy efficient stochastic computing with Sobol sequences," in *Proceedings of the conference on Design, Automation & Test in Europe (DATE)*. IEEE, 2017.

[7] H. Niederreiter, *Random Number Generation and quasi-Monte Carlo Methods*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.

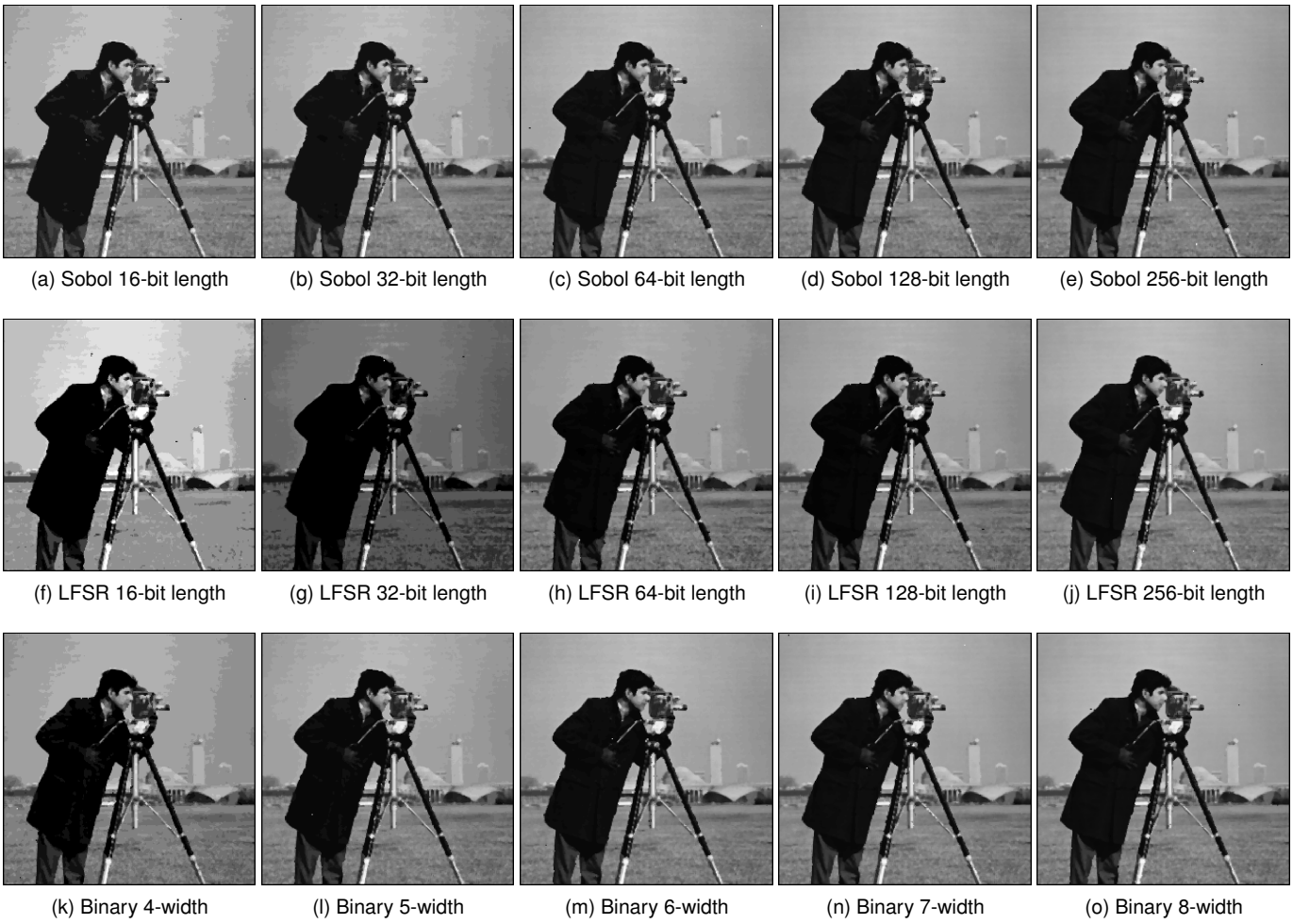
[8] D. P. Kroese, T. Taimre, and Z. I. Botev, *Handbook of Monte Carlo methods*. John Wiley & Sons, 2013, vol. 706.

[9] P. Bratley and B. L. Fox, "Algorithm 659: Implementing Sobol's quasirandom sequence generator," *ACM Trans. Math. Softw.*, vol. 14, no. 1, pp. 88–100, Mar. 1988.

[10] I. L. Dalal, D. Stefan, and J. Harwayne-Gidansky, "Low discrepancy sequences for Monte Carlo simulations on reconfigurable platforms," in *International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, 2008, pp. 108–113.

[11] P.-S. Ting and J. P. Hayes, "Stochastic logic realization of matrix operations," in *Digital System Design (DSD), 17th Euromicro Conference on*. IEEE, 2014, pp. 356–364.

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



(a) Sobol 16-bit length (b) Sobol 32-bit length (c) Sobol 64-bit length (d) Sobol 128-bit length (e) Sobol 256-bit length

(f) LFSR 16-bit length (g) LFSR 32-bit length (h) LFSR 64-bit length (i) LFSR 128-bit length (j) LFSR 256-bit length

(k) Binary 4-width (l) Binary 5-width (m) Binary 6-width (n) Binary 7-width (o) Binary 8-width

Fig. 31. Experimental results using the stochastic and binary implementations of a median filter.

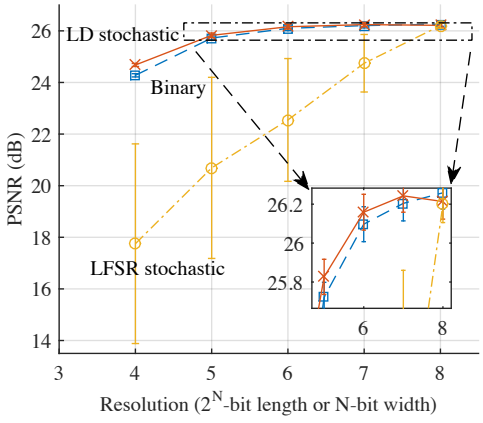


Fig. 32. PSNR comparison of different median filter implementations.

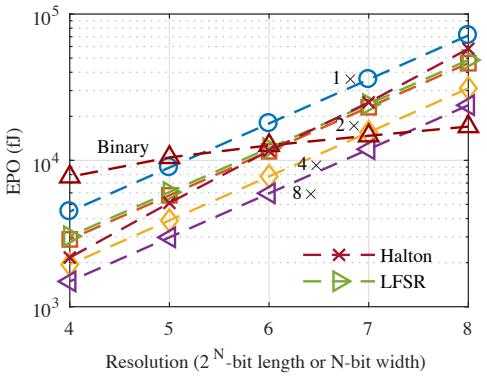


Fig. 33. EPO comparison of different median filter implementations.

[13] B. D. Brown and H. C. Card, "Stochastic neural computation. I. computational elements," *IEEE Transactions on computers*, vol. 50, no. 9, pp. 891–905, 2001.

[14] N. Saraf, K. Bazargan, D. J. Lilja, and M. D. Riedel, "IIR filters using stochastic arithmetic," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[15] R. Wang, J. Han, B. F. Cockburn, and D. G. Elliott, "Stochastic circuit design and performance evaluation of vector quantization for different error measures," *IEEE Trans. on VLSI Systems*, vol. 24, no. 10, pp. 3169–3183, Oct 2016.

[16] D. E. Knuth, *The art of computer programming: sorting and searching*. Pearson Education, 1998, vol. 3.

[17] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. ACM, 1968, pp. 307–314.

[18] S. Wolfram, *A New Kind of Science*. Wolfram Media, 2002. [Online]. Available: http://www.wolframscience.com

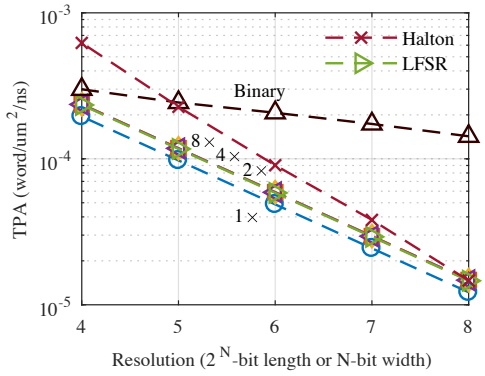


Fig. 34. TPA comparison of different median filter implementations.

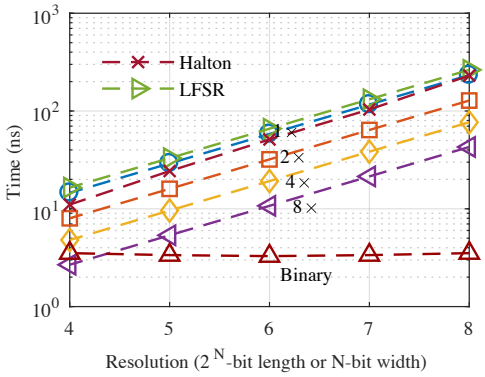


Fig. 35. Run time comparison of different median filter implementations.

Siting Liu received the B.Eng. and M.Eng. degree in electrical engineering and automation from Harbin Institute of Technology, Harbin, Heilongjiang, China, in 2012 and 2014, respectively. He is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada. His current research interest is stochastic computing.

Dr. Jie Han received the B.Sc. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. degree from Delft University of Technology, The Netherlands, in 2004. He is currently an associate professor in the Department of Electrical and Computer Engineering at the University of Alberta, Edmonton, AB, Canada. His research interests include approximate computing, stochastic computation, reliability and fault tolerance, nanoelectronic circuits and systems, novel computational models for nanoscale and biological applications.

Dr. Han and coauthors received the Best Paper Award at the International Symposium on Nanoscale Architectures 2015 (NanoArch 2015) and Best Paper Nominations at the 25th Great Lakes Symposium on VLSI 2015 (GLSVLSI 2015) and NanoArch 2016. He was nominated for the 2006 Christiaan Huygens Prize of Science by the Royal Dutch Academy of Science. His work was recognized by Science, for developing a theory of fault-tolerant nanocircuits (2005).

He is currently an associate editor for IEEE Transactions on Emerging Topics in Computing (TETC) and IEEE Transactions on Nanotechnology. He served as a General Chair for GLSVLSI 2017 and the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2013), and a Technical Program Committee Chair for GLSVLSI 2016 and DFT 2012.