Low-Latency Deterministic Multiplier for Stochastic Computing

Anwar K. Hussein and N. Sertac Artan*
Department of Electrical and Computer Engineering, New York Institute of Technology, New York, NY
emails: {ahusse14, nartan}@nyit.edu
*Corresponding Author

***Abstract*—The cost, power consumption, and availability of large-scale computing resources dampen the progress in computing. Stochastic Computing (SC) aims to mitigate this issue with more efficient primitives for approximate computing. Yet, high latency and low accuracy prevent the adoption of SC. SC accuracy can be improved by replacing Random Number Generators with generators of Low Discrepancy (LD) sequences such as Sobol. The large area requirement of these generators are mitigated with Finite-State Machine (FSM) based methods. FSM-based multipliers led to significant gains in accuracy and latency. Nevertheless, FSM-based methods are still impractical, as multiplication still takes up to 2^{2N} cycles. Another approach for improving latency is resolution splitting (RS). However, RS does not take advantage of the properties of the operands. In this paper, we propose to merge these two leading approaches for reduced latency. The speed advantage of the proposed approach is demonstrated with an image-filtering task. The proposed approach speeds up multiplications up to $604\times$ compared to conventional SC, and a $2.35\times$ compared to the state-of-the-art SC at the cost of increased area.**

I. Introduction

Stochastic computing (SC) allows the execution of arithmetic functions with simple and area-efficient circuits. Operations such as multiplication and addition can be performed using an AND gate and a multiplexer, respectively [1]. In SC, numbers are expressed in terms of the probability of the occurrence of 1s in a bit-stream. In stochastic numbers, each bit has the same unit weight (similar to unary numbers). This representation with a bit-stream with equally weighted bits makes SC resilient to errors and reduces hardware cost.

High processing cycles and inaccuracy in computation are the major drawbacks of conventional stochastic circuits. The inaccuracy in the stochastic circuits is primarily caused by the randomness of the bit order due to the random number generators (RNG), which are used to generate the stochastic bit-streams. The most commonly used RNG in conventional stochastic computing (CSC) is Linear Feedback Shift Register (LFSR) [2].

TABLE I: Comparison of Stochastic Computing Methods.

Method	**Number Source**	**Cycles**
Conventional Stochastic	LFSR, Counter	2^N
Deterministic	rotation, clk division	2^N
Resolution Splitting (C = 2)	rotation, clk division	$2^{N/2}$
FSM-based serial	FSM+MUX	2^N
FSM-based parallel	FSM+MUX	$2^N/N$

Deterministic stochastic computing (DSC) minimizes or eliminates the inaccuracies at the cost of increased latency [3]–[5]. DSC guarantees that for a binary operator (e.g., multiplication of two numbers), each bit in one of the operands is evaluated against every bit of the other operand. This eliminates the randomness in the bit order and thus eliminates errors. DSC achieves this by rearranging and replicating the bit-streams using three different techniques: (1) relatively prime length, (2) rotation, and (3) clock division [3]. However, this replication also significantly increases the total processing time.

Another solution to increase the accuracy of SC is to use low-discrepancy sequences such as Sobol or Halton sequences for generating stochastic numbers instead of the LFSRs used in conventional SCs [6]. To accurately represent a binary number with a stochastic number, the bit-stream generator has to run for the full processing cycle, which depends on the precision of the given number.

To represent an N-bit number, 2^N processing cycles are required and to multiply two N-bit numbers with DSC, 2^{2N} cycles are required. To reduce the number of cycles for multiplication, Sim and Lee have proposed a down-counter-based stochastic multiplier, where the operation can be completed before reaching the full processing time in most cases [7]. Their approach also uses a Finite State Machine (FSM) for LD sequence generation. In their approach, referred here as *FSM-based Serial*, they used a down counter to generate one of the operands. This down counter also determines the total number of cycles. In other words, it decides when to end the multiplication based solely on the value of one of the operands. Note that, in their approach, the early completion of the operation is achieved by preserving the full accuracy of the result. Here, we call this as *early completion technique*. The operand generated by this down counter is a left flushed unary number. Additionally, Sim and Lee also proposed a parallel approach to further speed up the calculation, referred in this paper as *FSM-based Parallel*.

To address the high processing cycle time in SC, another parallel method has been proposed by Najafi et al. called resolution-splitting [8]. Latency exponentially increases in SC when the precision of the operands increases. In resolution-splitting, SC is performed by reducing the precision of the

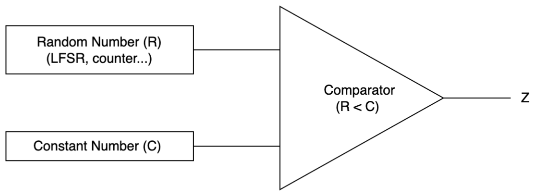

Fig. 1: Stochastic bit-stream generation using a Random Number Generator (RNG) such as a Linear Feedback Shift Register (LFSR).

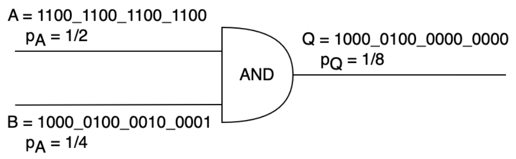

Fig. 2: Stochastic multiplier using an AND gate.

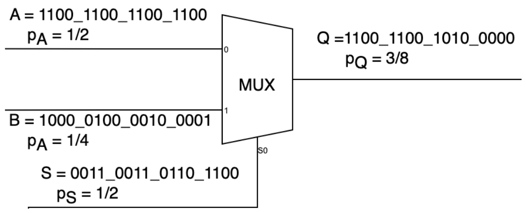

Fig. 3: Stochastic weighted adder using a single multiplexer.

operands by splitting the operation into reduced precision operands. This led to much shorter bit-streams, and running operations on these bit-streams is more efficient. The intermediate results produced by these operations are then combined to produce the final result. The number of operands after the split is determined by a coefficient, C, which is a multiple of 2. Table I shows the time complexity of different SC methods discussed in this paper. The number source column indicates the bit generator used for the respective methods.

This paper is built upon the following observations: First, the resolution splitting improves the processing speed by reducing operand size, which is independent of the operand value, and improves worst-case performance. On the other hand, the early completion technique in the FSM-based methods does not affect the worst-case behavior. Conversely, it improves the average case. Thus, early completion is beneficial for practical applications, where most operations do not require the worst-case because not all operands are 2^{N-1}. These two gains are orthogonal, and exploiting them together is potentially beneficial. In this paper, we propose a method taking advantage of both resolution splitting and early completion for speeds higher than both approaches. We demonstrated that our method can significantly improve the performance of practical applications using the example of image filtering. The area cost of our method is higher than resolution splitting but on par with the FSM-Parallel method.

The main contributions of this paper are 1) A parallel, early completion approach with high performance in worst-case and average-case. 2) Application of the proposed method in image filtering. The rest of this paper is organized as follows: Section II gives a brief background to stochastic computing. In Section III, our proposed method is explained in depth. Section IV includes an evaluation of our proposed method. Section V summarizes our conclusions.

II. Background

A. Basic Concept

Stochastic Computing is a probabilistic computing method [1], where numbers in the range [0,1] are represented with the probability of the occurrence of 1s in a bit-stream. The bit-stream is called a stochastic number (SN). The precision of the representation is a direct function of the length of the bit-stream: The longer the bit-stream, the closer the expected value to the desired value.

For example, the SN $x = 0100\ 1000$ is an 8-bit sequence, with a probability of $2/8 = 0.25$ for the occurrence of bit value 1, thus it represents the number 0.25. Unipolar SNs represent only positive numbers and have a range of [0,1]. Positive numbers in a wider range can be normalized to the [0,1] range. To represent both positive and negative numbers, Bipolar SNs are used, whose range is [-1,1]. More specifically, an SN with N_1 1s and N_0 0s represents the number x_u, and x_b as a unipolar SN, and bipolar SN, respectively, where

$$x_u = \frac{N_1}{N_1 + N_0} \tag{1}$$

$$x_b = \frac{N_1 - N_0}{N_1 + N_0} \tag{2}$$

SN can be generated by comparing the output of RNGs [6], [9], and the binary value of the number to be generated as shown in Fig. 1. Bit-streams generated using RNGs such as LFSRs lack accuracy due to random fluctuation of bits in the stream. For an N-bit binary number, up to 2^N bits are required to represent it as an SN. Arithmetic functions such as multiplication and addition on SNs are performed using simple gates (such as an AND gate and a multiplexer, respectively, as shown in Fig. 2 and Fig. 3). Due to this property, SC is area-efficient. SC schemes are ideal for applications that are limited by hardware area. In addition, applications using stochastic circuits benefit from lowered cost in making the circuits, as the number of logic used in SC is far less than the conventional binary applications.

B. Deterministic Processing

Pseudo Random and Quasi Random number generators (RNG) play an integral role in generating stochastic numbers [6]. RNGs such as LFSRs can be used when the application is error-tolerant. However, quasi-random number generators such as counter-based and bit-shuffling-based number generators

TABLE II: Processing cycle comparison of SC multipliers using bit splitting for 8-bit operands.

Split Coefficient	Processing Cycles	
1	2^{2N}	2^{16}
2	2^{N}	2^{8}
4	$2^{N/2}$	2^{4}

are necessary for an accurate result. Recent work shows that bit-streams do not necessarily have to be random and introduced a deterministic way of computing [3,4,5]. Deterministic SC uses SN operands with relatively prime lengths to achieve highly accurate results. Accuracy is achieved by maintaining independence between bit-streams, where each bit from one operand sees each of the bits in the other operand exactly once. [3]. Relatively prime length, clock division, and rotation are among the deterministic ways of generating bits. These methods use a simple bit shuffling technique to achieve highly accurate results.

C. Parallel Computing

Parallel computing schemes have been shown to reduce the latency of SC that is caused by the high number of processing cycles. In these schemes, an arithmetic operation is performed after splitting the bit-streams. In [10] a stochastic multiplier with parallel paths has been proposed using a deterministic computing scheme.

Najafi et al.'s resolution splitting method [8] shows a parallel computing scheme that splits the operands using a splitting coefficient C and performs operation processing on the partial products in a parallel fashion. For example, given two 8-bit binary numbers $X = 1001\ 1011$ and $Y = 1011\ 1001$ and a splitting coefficient of 2, parallel computing can be performed on the partial products, $X0 = 1001$, $X1 = 1011$, $Y0 = 1011$, $Y1 = 1001$. The two operands can be expressed as follows:

$$X = X_0 \cdot 2^{N/2} + X_1,\ Y = Y_0 \cdot 2^{N/2} + Y_1 \tag{3}$$

The product is the sum of the partial products of the split operands.

$$X \cdot Y = (X_0 \cdot Y_0) \cdot 2^N + (X_0 \cdot Y_1 + X_1 \cdot Y_0) \cdot 2^{N/2} + X_1 \cdot Y_1 \tag{4}$$

D. Low Discrepancy Sequences

Low discrepancy (LD) or quasi-random sequences comprise bit-streams with equally distributed 1s and 0s [6]. A fast converging result can be achieved when using LD instead of pseudo-random sequences that are generated by LFSRs and other methods. This means processing cycles can be reduced and the accuracy of the result is higher. The most commonly used LD sequences are the Sobol and Halton sequences [11]. Although accurate results can be achieved using Sobol and Halton sequences, the hardware overhead of the bit-stream generation remains higher compared to LD sequences such as FSM-based generators, which use a simple bit shuffling scheme and a down counter to save processing time.

FSM-based circuits generate LD sequences by using 2^N state FSMs. The FSMs perform the bit shuffling to generate the quasi-random bit-stream that results in equally distributed 1s and 0s. The core principle of the bit shuffling, which was introduced by Sim and Lee [7], is that given a binary number $x = x_{N-1}x_{N-2}\ldots x_0$, the stochastic stream X includes, first, x_{N-i} bits appear 2^{i-1} cycle and 2^i cycle till the stream gets to a length of 2^N. The output of the FSM is then used as a select bit of a multiplexer which selects between the input bits.

FSM-based multipliers, unlike other stochastic multipliers, do not need 2 separate RNGs when multiplying 2 numbers. Instead, the bit-stream of one of the operands is fed into an AND gate and the second operand is fed to the down counter. The down counter determines the run duration based on the value of the operand that was fed to it. This guarantees convergence to the result without processing the entire bit-stream. As a result, energy consumption will be lowered. The number of cycles needed for multiplying two numbers is $2^N \cdot y$, where y is the second operand. For instance, if y has a value of 1, the processing will take 2^N cycles as opposed to 2^{2N} in conventional and deterministic cases. For example, if you have an 8-bit number x, and you multiply it with 1, the bit-stream needed to be generated to get an accurate result is 2^8. In the conventional stochastic cases, we will need to wait for the full 2^{2N} cycles, in this case, 2^{16} clock cycles to get the product.

FSM-based multipliers suffer from a high processing cycle as the precision of the operands increases. To mitigate this, Sim and Lee propose a bit-parallel approach [7]. This method involves the division of stochastic streams into N segments and multiplication performed by using a counter. The counter counts the number of 1s in each segment and produces an accurate result. For example, when multiplying two 8-bit numbers, the 256 stream is divided into 32 segments. The up counter then counts the number of ones in each segment until the down counter counts to 0.

III. Our Proposed Method

A. Overview

In this section, we propose combining resolution splitting with the FSM-based LD sequence generator to reduce the processing latency of SC for both the average and the worst cases.

B. Multiplier with Early Completion

Our base multiplier structure is the FSM-based serial design by Sim and Lee [7]. One of the operands are generated by an FSM-based generator. The other operand is generated as a unary number with the help of a down counter. The down counter takes one of the operands, y as its input, and starts counting down from the initial value of $2^N \cdot y$. This initial value also determines the bit-stream length. The multiplication operation continues until the down counter reaches zero. The second operand is fed to an FSM-based LD sequence generator. The output of the down counter and the bit generator are fed to an AND gate, which determines whether the up counter (i.e., accumulator) is incremented in a given cycle. When the down counter reaches zero, the up counter will have the product of the two operands.

This approach from Sim and Lee reduces the average latency of operation. If y is less than $2^N - 1$, the multiplication completes sooner than other LD-based SCs. The smaller the value of y, the shorter will be the latency. However, this approach does not improve the worst-case. On the other hand, by combining this approach with the resolution splitting, we were able to reduce both worst-case and average-case behavior (Section III-C).

C. Average and Worst Case Latency Improvements

Parallel computing methods such as the resolution splitting proposed by Najafi et al. [8] enable reductions in latency in the worst case. However, these methods do not take advantage of the operand values to further speed up the operations in the average case. As discussed in the previous section, the early completion techniques can finish the multiplication early based on the operand values and do not cause any loss in accuracy. Here, we combine these two methods to get the best outcome for both average and worst cases. In resolution splitting, each operation is divided into operations with smaller operands and computes partial products. Once the partial products are computed, they are combined into the final product via an adder/shifter network. A splitting coefficient, C, a power of 2, determines the number of operand splits. The block diagram of an 8-bit multiplier using resolution splitting on FSM-based serial multipliers is shown in Fig. 4 with $C = 2$. The four individual multiplier modules are responsible for multiplying the partial products. One of the partial products is multiplied by 2^8 and the sum of the other two is multiplied by 2^4 following (4) with $N = 8$. Using an appropriate adder tree, the latency can be further reduced using a higher C [8].

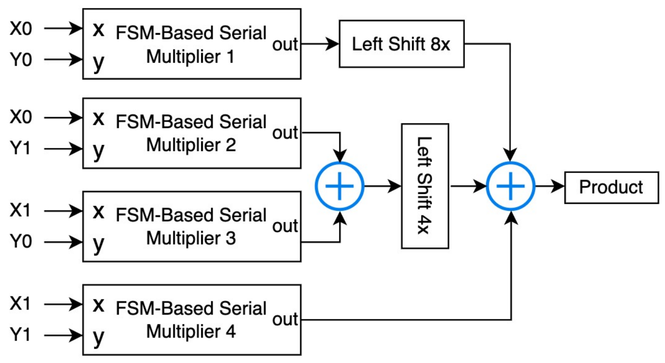
X0
Y0
x
y
FSM-Based Serial Multiplier 1
out
Left Shift 8x
X0
Y1
FSM-Based Serial Multiplier 2
X1
Y0
FSM-Based Serial Multiplier 3
X1
Y1
FSM-Based Serial Multiplier 4
Left Shift 4x
Product

Fig. 4: Our proposed multiplier combines FSM-based serial multipliers with a resolution splitting method (C=2).

IV. Evaluation

A. Application: Image Filtering

To evaluate the performance of the proposed multiplier for a practical application, we compared it to various multipliers for image filtering using Matlab. In image processing, numerous filters are applied to enhance an image. Common applications of these filters include noise reduction and quality enhancement. The convolution operation between an image and a filter (i.e., kernel) is given by:

$$g(x, y) = \sum_{i=1}^{3} \sum_{j=1}^{3} f(x + i - 1, y + j - 1) \cdot k(i, j) \quad (5)$$

where: $g(x, y)$ is the filtered pixel at (x, y), $f(\alpha, \beta)$ is a pixel, $k(i, j)$ is the (i, j)th coefficient in the kernel. The 3×3 Gaussian Blur Kernel with a standard deviation (σ) of 2 is used for noise reduction here (6).

$$\text{Gaussian blur kernel: } \begin{bmatrix} 0.078 & 0.082 & 0.078 \\ 0.082 & 0.089 & 0.082 \\ 0.078 & 0.082 & 0.078 \end{bmatrix} \quad (6)$$

We used the Cameraman image of size 256×256 with noise as shown in Fig. 5a.

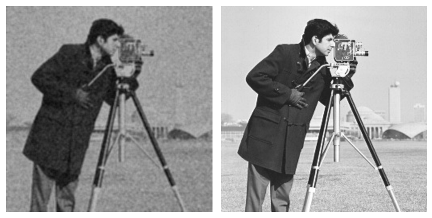

Fig. 5: Example image Cameraman with size 265×256.

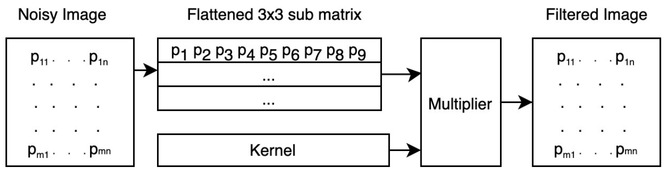

Fig. 6: Image filtering operation.

In this paper, we used the image filtering operation illustrated in Fig. 6. The image is zero-padded at all edges to accommodate the filtering of the border pixels. Then, the kernel is applied to each 3×3 sub-matrix in the padded image using (6). The 9 multiplication operations for each sub-matrix are carried out with different multipliers, and the sums are accumulated to generate each filtered pixel. The resulting filtered image is shown in Fig. 5b. The speed-ups for 4 multipliers normalized to the latency of the conventional stochastic multiplier are shown in Table III.

The proposed method shows speed-ups of $604\times$ compared to the conventional stochastic, and $2.35\times$ compared to the next fastest approach (LFSR-based multiplier). In addition, the proposed method shows over $8\times$ speed-up compared to the FSM-based parallel method.

TABLE III: Comparison of Multiplier Speed-ups.

Method	**Speed-up**
Conventional Stochastic	$1\times$
FSM-based Serial	$9\times$
FSM-based Parallel	$74\times$
LFSR-based with split (C=2)	$256\times$
Proposed method (C=2)	$604\times$

B. Resource Utilization

To evaluate the hardware area of the proposed result, we have implemented an 8-bit multiplier module with a split coefficient, $C = 2$ in Verilog. The design is synthesized for FPGA using AMD/Xilinx Vivado 2023.2. Table IV shows the synthesis result of different stochastic multipliers and the Booth binary multiplier as a reference.

Our results are shown in Table IV. The proposed method has an area comparable to the fastest FSM-based Parallel method. However, its area is larger than the resolution splitting method by 74% more LUTs, and 18% more FFs. Nevertheless, our method is much faster as shown above compared to both methods, which can justify the area overhead for suitable applications such as image filtering.

TABLE IV: Comparison of FPGA resource utilization of 8-bit Binary and Stochastic Multipliers.

Method	**LUT**	**FF**
Booth Binary Multiplier	264	-
Conventional Stochastic	12	35
FSM-based Serial	36	37
FSM-based Parallel	91	75
LFSR-based resolution splitting ($C = 2$)	53	61
Proposed method ($C = 2$)	92	72

V. Conclusions

In this paper, we combined two orthogonal methods for latency reduction in stochastic computing and achieved accurate results with significantly lower latency. Although the average-case improvements do not offer any improvements in the worst-case, which is a concern for instance in applications with heightened security, they are beneficial for many applications. We showed image filtering as an example viable application, where the latency is determined by the filter coefficients, which are taken from practical values.

Acknowledgment

This work is partially funded by a New York Institute of Technology (NYIT) Institutional Support for Research and Creativity (ISRC) Grant.

References

[1] C. Winstead, "Tutorial on stochastic computing," *Stochastic Computing: Techniques and Applications*, pp. 39–76, 2019.

[2] J. H. Anderson, Y. Hara-Azumi, and S. Yamashita, "Effect of lfsr seeding, scrambling and feedback polynomial on stochastic computing accuracy," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1550–1555.

[3] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–8.

[4] M. H. Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani, "Time-encoded values for highly efficient stochastic circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1644–1657, 2017.

[5] M. H. Najafi, D. J. Lilja, M. Riedel, and K. Bazargan, "Power and area efficient sorting networks using unary processing," in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 125–128.

[6] H. Hsiao, J. Anderson, and Y. Hara-Azumi, "Generating stochastic bitstreams," *Stochastic Computing: Techniques and Applications*, pp. 137–152, 2019.

[7] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *Proceedings of the 54th annual design automation conference 2017*, 2017, pp. 1–6.

[8] M. H. Najafi, S. R. Faraji, B. Li, D. J. Lilja, and K. Bazargan, "Accelerating deterministic bit-stream computing with resolution splitting," in *20th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2019, pp. 157–162.

[9] P. Y. Chawke and R. Kshirsagar, "Design of 8 and 16 bit lfsr with maximum length feedback polynomial using verilog," in *Proceedings of 13th IRF International Conference*, 2014.

[10] Y. Zhang, S. Liu, J. Han, Z. Lin, S. Wang, X. Cheng, and G. Xie, "An energy-efficient binary-interfaced stochastic multiplier using parallel datapaths," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[11] S. Liu and J. Han, "Toward energy-efficient stochastic circuits using parallel sobol sequences," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1326–1339, 2018.