Context-Aware Bit-stream Generator for Deterministic Unary Processing

Sina Asadi and M. Hassan Najafi
sina.asadi1@louisiana.edu, najafi@louisiana.edu
School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, USA

***Abstract*—Deterministic methods of processing bit-streams have been proposed to produce completely accurate results with stochastic logic. Real-valued numbers in the [0,1] interval are converted to unary bit-streams and processed using relatively prime stream length, clock division, and rotation methods. Long latency is the main issue with these deterministic methods. To process m n-bit precision numbers, bit-streams of $2^{m \times n}$ bits must be generated. In this work, we propose a context-aware deterministic bit-stream generator to improve the performance of these methods. The proposed design improves the performance up to 90% compared to the conventional architecture.**

***Index Terms*—Stochastic computing, unary computing, deterministic bit-stream processing, bit-stream generator, context-aware design, hardware accelerator.**

I. Overview

Stochastic computing (SC) [1], [2], [9] has been used for low-cost and noise-tolerant implementation of arithmetic functions. Complex operations can be implemented using simple logic gates (e.g., multiplication operation can be implemented using a single AND gate). Input data is represented by uniformly distributed random (i.e., interleaved) or unary (i.e., first all '1's followed by all '0's or vise versa) bit-streams. The ratio of the number of ones to the length of the bit-stream determines the bit-stream value in this paradigm. For example, 10100 is a representation of 0.4 in SC. While this unconventional representation of data is not compact compared to the conventional weighted binary radix representation, it insures the computation against soft errors (i.e., bit flips). All digits have equal weight and a single bit-flip results in a small error in the represented value.

Inaccuracy of processing bit-streams, however, has been the main issue with the conventional stochastic designs. Random fluctuations in generating bit-streams and correlation between bit-streams lead to results that are only approximately correct. Some deterministic methods of processing bit-streams [3], [6] were introduced recently to produce completely accurate results with SC constructs. Relatively prime bit-stream lengths [6], clock dividing bit-streams, and rotation of bit-streams [3] are the three methods that guarantee deterministic and accurate processing of bit-streams. These methods were initially proposed based on unary bit-streams. Fig. 1 exemplifies the clock division and the rotation methods based on unary bit-streams. Authors in [7] enhanced the performacne of the three deterministic methods by generating pseudo-random but accurate deterministic bit-streams. More recently, authors in [8] proposed two fast-converging deterministic methods of processing bit-streams based on low-discrepancy (LD) bit-streams. They used Sobol sequence-based LD bit-streams to produce high-quality deterministic results.

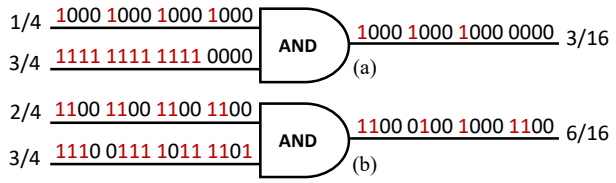

Fig. 1. Examples of deterministic bit-stream-based multiplication using (a) clock division (b) rotation methods [3].

The pseudo-random linear feedback shift register (LFSR)-based [7] and the LD Sobol-based [8] deterministic methods converge to acceptable results faster than the initial unary-based methods. However, a higher power consumption in generating pseudo-random numbers and a higher hardware-cost in generating Sobol numbers make the unary-based methods more efficient for the cases that completely accurate results are expected. For such cases, the processing time of different methods is the same. Therefore, the unary-based methods that are constructed based on counters have a lower power consumption and a lower hardware cost. The unary-based methods, further, work well with an analog interpretation of the bit-streams, where the value is encoded as the fraction of time the signal is high and represented using time-encoded pulse signals [5], [6].

A common property to all these deterministic methods of processing bit-streams is that producing completely accurate result requires generating and processing bit-streams for $2^{m \times n}$ cycles where m is the number of inputs and n is the precision of data. For example, processing two 8-bit precision numbers requires generating 2^{16}-bit bit-streams in 2^{16} cycles. The processing time increases exponentially by increasing the number of inputs and the precision of data. This long processing time further translates to very high energy consumption (energy = power $\times$ time), making deterministic bit-stream processing energy-inefficient for applications that completely accurate results are expected.

The conventional bit-stream generator used in the deterministic methods generates and processes bit-streams regardless of the values of the input data. Whether the input is $128/256$ (=$1/2$) or $13/256$, the same length bit-stream is generated. While $13/256$ requires at least 256 bits to be precisely represented, $128/256$ can be represented accurately using a

short stream of only 2 bits (e.g. 10 or 01). In this work, we show that context-aware generation of bit-streams can significantly improve the processing time of the deterministic methods. We propose a control unit to enhance the bit-stream generator of the deterministic designs. The proposed controller determines the minimum precision required for each input data and dynamically adjusts the system to work for the minimum number of operation cycles sufficient to produce accurate result.

II. Deterministic Bit-stream Generator

To process data with the deterministic bit-stream methods we need to first convert the data from the conventional binary to the bit-stream representation. In this section, we first discuss the current binary-to-bit-stream converter used in the structure of the three deterministic methods. We then discuss our proposed context-aware bit-stream generator.

A. Conventional Design

Fig. 2 shows the conventional structure of a binary-to-bit-stream converter (a.k.a. stochastic number generator (SNG) or converter module [3]) currently used in the deterministic methods of processing bit-streams. An increasing/decreasing number from an up/down counter is compared to a constant number (based on the input data) and the output of comparison produces one bit of the bit-stream in each cycle. A one is generated at the output of the comparator if "*Number Source*" $<$ "*Constant Number*". A zero is generated otherwise.

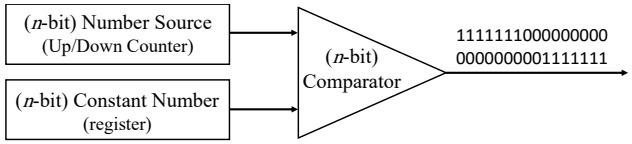

Fig. 2. Conventional Binary-to-Unary Bit-stream Converter

Fig. 3 depicts the conventional architecture of a deterministic bit-stream processing system based on the clock division method of [3]. m inputs are converted from binary-radix to bit-stream representation using m converter modules. The system runs for $2^{m\times n}$ cycles to produce the correct (completely accurate) result. The processing is stopped by sending a *stop signal* to the bit-stream-to-binary converter (*BBC*) unit. As shown in Fig. 3, the *stop signal* is produced at no additional hardware cost by using the same counters used in the converter modules. The signal turns to 1 when the system operates for exactly $2^{m\times n}$ cycles.

B. Proposed Context-Aware Design

1) Structure for Completely Accurate Results: The conventional architecture generates bit-streams regardless of the value of the input data. Multiplying 10/16 (1010 in binary radix) and 8/16 (1000 in binary radix) with the conventional architecture requires the same number of cycles as multiplying 9/16 and 15/16. In both cases, the system runs for 2^{4+4}(=256) cycles to produce the correct result. The deterministic methods of processing bit-streams, however, can produce the correct result

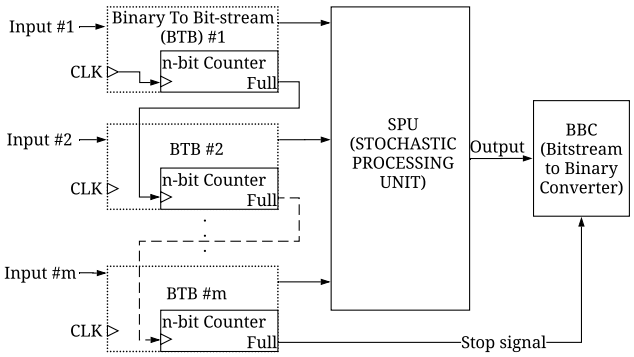

Fig. 3. Conventional architecture of a deterministic bit-stream processing system based on the clock division method of [3].

of multiplying 10/16 and 8/16 in only 2^{3+1} (=16) cycles if effectively change 10/16 to 5/8 and 8/16 to 1/2, and represent each input using its minimum required bit-stream length (i.e., 8 and 2 bits, respectively). In this case, running the system for a larger number of cycles than 16 cycles wastes the time and most importantly the energy resources of the system.

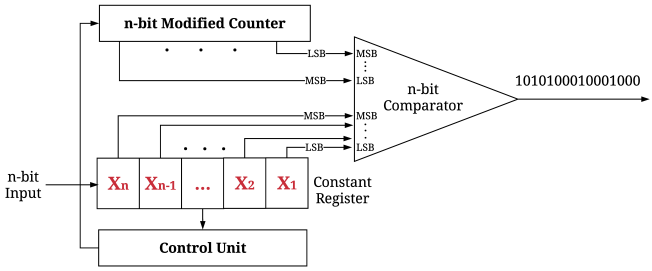

Fig. 4. Proposed bit-stream generator with Control Unit (CU) and Modified Counter (MC)

Fig. 4 shows our proposed context-aware bit-stream generator. The control unit (*CU*) reads the input data from the constant register and determines the minimum bit-width necessary to precisely present the input value. Fig. 6 shows how CU produces the control signal for A=10/16 (or 1010 in binary radix). *CU* sends 011 to a modified counter (*MC*), forcing it to work as a 3 bit counter (counting from 0 to 7). Fig. 5 shows the structure of a 4-bit *MC*. The output bits of *MC* are connected to the comparator in reverse order, generating a Sobol number in each cycle[1]. The comparator, therefore, compares the effective bits of the constant register (e.g., 101 in 1010 and 1 in 1000) to a new Sobol number in each cycle. This results in converting the input value to a fast converging LD bit-stream, instead of generating a unary bit-stream, but with the same hardware cost as a unary stream generator. LD bit-streams are preferred to unary bit-streams as they enjoy the progressive precision of random bit-streams while produce deterministic and accurate result [8]. Fig. 7 shows the general structure of an n-bit *CU*.

Fig. 8 depicts our proposed context-aware architecture for a deterministic bit-stream processing system. The system oper-

[1]The simplest Sobol sequence can be generated by simply reversing the output bits of a counter [4]

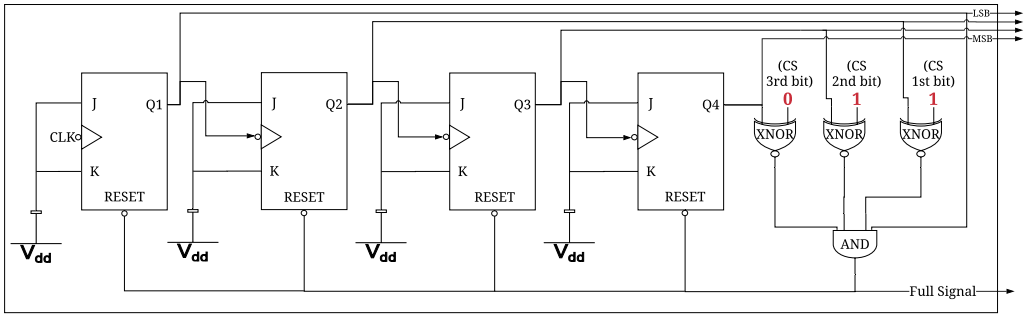

Fig. 5. 4-bit Modified Counter

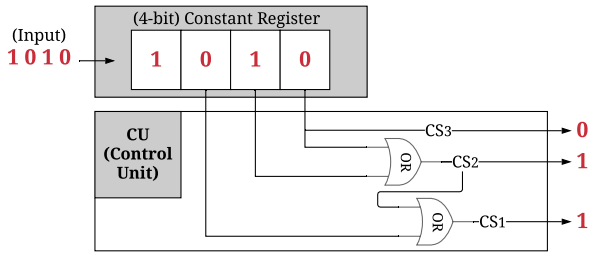

Fig. 6. A 4-bit Control Unit generating control signals for 10/16

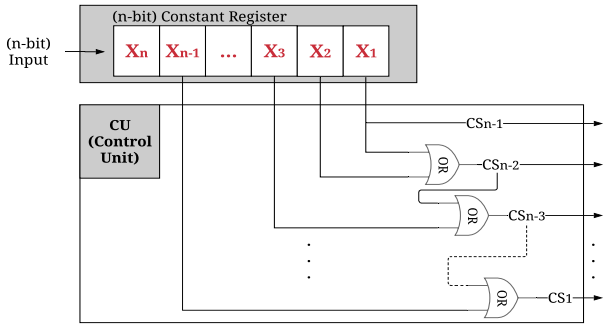

Fig. 7. n-bit Control Unit

ates for $2^{Q_1+Q_2+...+Q_m}$ cycles to produce the correct result, where Q_i is the required precision for input i and m is the number of independent inputs in the system.

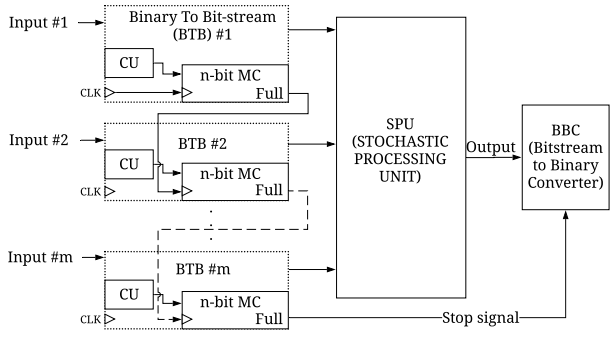

Fig. 8. Proposed Architecture with Control Unit (CU) and Modified Counter (MC)

2) Structure for Error Tolerant Applications: In binary radix representation, least significant bits (LSBs) have less impact on the accuracy of computations than most significant bits (MSBs). This impact further reduces when the bit-width of data increases. The difference between two 8-bit precision numbers A=10000000 and B=10000001 is only on the LSB. A represents 0.5 while B represents 0.5039. The absolute difference between these two values is only 0.0039, a negligible difference for many applications. If the application can tolerate such small rates of inaccuracy, it is feasible to further reduce the processing time of deterministic bit-stream processing for many input cases. For example, if setting the LSB of B=10000001 to 0, the input data can be precisely represented using a stream of only 2 bits rather than a stream of 2^8 bits. This will significantly reduce the number of cycles required in the deterministic bit-stream processing at the cost of a negligible accuracy loss. Fig. 9 shows a modified *CU* for a system that processes 4-bit precision inputs and can tolerate small rates of inaccuracy. Assume two 4-bit inputs, A=1001 and B=1001, are to be multiplied. With the proposed design, the number of clock cycles decreases from 2^8 to 2^2 at the cost of 0.06 error rate. This rate of error can further decrease as the bit-width increases.

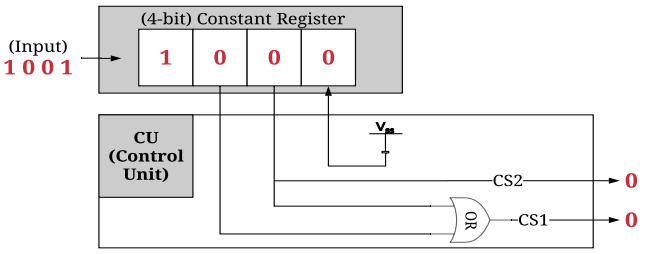

Fig. 9. Modified 4-bit Control Unit for error tolerant applications

III. Evaluation

The overhead of the proposed design is some additional logic gates due to adding the *CU* and *MC* units. As shown in Fig. 7, *CU*s include some standard OR gates. *MC*s also contain some additional XNOR gates compared to the regural counters to enumerate the maximum number which *CU* determines. The overhead cost, however, is insignificant compared to the total cost of the the system. Table I shows the overhead of the proposed design for different input precisions in terms of the required additional logic gates. As can be seen, for the 16-bit precision design, the proposed context-aware design has an overhead of 14 OR and 15 XNOR gates.

TABLE I
OVERHEAD OF PROPOSED DESIGN.

Input Precision	# of LSBs set to 0	# of OR Gates	# of XNOR Gates	Saved Components
4-bit	0	2	3	0
	1	1	2	1 Flip Flop, 1 AND Gate
8-bit	0	6	7	0
	1	5	6	1 Flip Flop, 1 AND Gate
	2	4	5	2 Flip Flop, 2 AND Gate
	3	3	4	3 Flip Flop, 3 AND Gate
	4	2	3	4 Flip Flop, 4 AND Gate
16-bit	0	14	15	0
	1	13	14	1 Flip Flop, 1 AND Gate
	2	12	13	2 Flip Flop, 2 AND Gate
	3	11	12	3 Flip Flop, 3 AND Gate
	4	10	11	4 Flip Flop, 4 AND Gate
	5	9	10	5 Flip Flop, 5 AND Gate
	6	8	9	6 Flip Flop, 6 AND Gate

TABLE II
PERFORMANCE IMPROVEMENTS WITH THE PROPOSED DESIGN.

# of Inputs	Input Precision	Conventional Design # of cycles	Proposed Design Percentage of reduced cycles
2 Inputs	4-bit	2^8	54.85
	8-bit	2^{16}	55.55
	16-bit	2^{32}	55.55
3 Inputs	4-bit	2^{12}	69.67
	8-bit	2^{24}	70.36
	16-bit	2^{48}	70.37
4 Inputs	4-bit	2^{16}	79.62
	8-bit	2^{32}	80.24
	16-bit	2^{64}	80.25
5 Inputs	4-bit	2^{20}	86.30
	8-bit	2^{40}	86.82
	16-bit	2^{80}	86.83

By accepting some inaccuracies in the computation, not only the overhead but also the overall area occupancy of the system decreases compared to the conventional architecture. This additional saving is due to further reduction in the size of the *MC*s. For clarity, we provide an example. Assume the data bit-width is 16 and the application can tolerate up to 2 percent error rate in the result. This allows the system to ignore up to 6 LSBs (set them to 0) for a 2-input multiplier design. The modified design for such an error tolerant application has an overhead of 8 OR and 9 XNOR gates and can work with a 10-bit counter instead of a 16-bit one. For some rates of error, the hardware savings from reducing the size of the counter will be higher than the overhead cost of additional gates added to the system. Therefore, the proposed design can save some hardware cost compared to the conventional architecture.

Tables II and III report performance improvements when using the proposed designs. For each case (i.e., a different number of inputs and a different precision) we find the number of processing cycles required for the conventional design and also the percentage of the reduced number of cycles for all possible input values for the proposed design. As can be seen, by increasing the number of inputs (m) and the precision of data (n) a higher reduction in the processing time is achieved. A lower processing times also translates into a lower energy consumption. The proposed context-aware designs make the deterministic bit-stream processing more appealing for applications that expect high accuracy processing and also for error tolerant applications.

TABLE III
PERFORMANCE IMPROVEMENT WITH THE PROPOSED DESIGN FOR ERROR TOLERANT APPLICATIONS. THE REPORTED ERROR RATE IS THE MAXIMUM ERROR FOR THE CASE OF MULTIPLYING THE INPUTS.

# of Inputs	Input Precision	Conventional Design # of cyles	# of LSBs set to 0	Proposed Design Percentage of Reduced cycles	Error Rate (%)
2 Inputs	4-bit	2^8	1	88.18	11
	8-bit	2^{16}	1	88.88	0.7
			2	97.21	2
			3	99.3	5
			4	99.82	11
	16-bit	2^{32}	1	88.88	3.05E-03
			2	95.65	9.15E-03
			3	98.68	2.14E-02
			4	99.64	4.58E-02
			5	99.9	9.46E-02
			6	99.97	0.1921654
3 Inputs	4-bit	2^{12}	1	95.93	15
	8-bit	2^{24}	1	96.29	1
			2	99	3
			3	99.84	7
			4	99.97	16
	16-bit	2^{48}	1	96.29	4.58E-03
			2	99.095	1.37E-02
			3	99.84	3.20E-02
			4	99.97	6.86E-02
			5	99.9971	0.1418353
			6	99.9996	0.2881052

IV. CONCLUSION

In this work, we proposed a context-aware architecture to improve the performance of the deterministic bit-stream processing systems. The proposed design employs a control unit to extract the minimum precision required to precisely represent each input data. The result is a considerable improvement in the processing time and so energy consumption at a reasonable hardware cost overhead.

REFERENCES

[1] A. Alaghi and J. P. Hayes. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):92:1–92:19, 2013.

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

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

[4] S. Liu and J. Han. Energy Efficient Stochastic Computing with Sobol Sequences. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 650–653, March 2017.

[5] M. H. Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani. An Overview of Time-Based Computing with Stochastic Constructs. *IEEE Micro*, 37(6):62–71, November 2017.

[6] 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*, 25(5):1644–1657, May 2017.

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

[8] M. H. Najafi, D. J. Lilja, and M. Riedel. Deterministic Methods for Stochastic Computing Using Low-discrepancy Sequences. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '18, pages 51:1–51:8, New York, NY, USA, 2018. ACM.

[9] W. Qian, X. Li, M. Riedel, K. Bazargan, and D. Lilja. An Architecture for Fault-Tolerant Computation with Stochastic Logic. *Computers, IEEE Trans. on*, 60(1):93–105, Jan 2011.