An Optimal Variable-Latency Architecture for Deterministic Approaches to Stochastic Computing with Unary Bit Stream Preserving Properties

Oğuz Meteer *
Email: o.meteer@utwente.nl

*Department of Computer Architectures
for Embedded Systems
University of Twente, Enschede, The Netherlands

Marco J.G. Bekooij*†
Email: marco.bekooij@nxp.com

†Department of Embedded Software
and Signal Processing
NXP Semiconductors, Eindhoven, The Netherlands

***Abstract*—Classical Stochastic Computing methods perform operations on bit streams generated using random number generators. Sufficient accuracy requires long bit streams which leads to very high latency and larger area.**

Deterministic approaches to Stochastic Computing greatly reduce the latency and area, and produce fully accurate results, but still have high latency. While state-of-the-art variable-latency architectures show an up to 70% reduction in latency, they still perform operations that do not contribute to the final result.

In this paper we present an optimal variable-latency architecture in terms of minimum required cycles to obtain fully accurate results. We propose an efficient stochastic number generator that uses a down counter and compare-to-zero hardware compared to an up counter and binary comparator. This key contribution means that the latency of our design only depends on the magnitude of input values and is independent of the number of bits or inputs used. This is not the case in state-of-the-art variable-latency architectures. Also, unlike any other architecture, ours preserves the unary bit stream encoding after performing additions and multiplications, which enables more complex designs that require unary bit streams such as sorting. Our architecture thereby further improves the area, latency, and energy efficiency compared to the state-of-the-art.

***Index Terms*—Stochastic Computing, variable-latency**

I. Introduction

Contrary to traditional systems that operate on binary values, Stochastic Computing (SC) systems operate on values that are encoded as random bit streams [2]. These systems have two main benefits. The first is that complex arithmetic in traditional systems reduce to very simple logic in SC systems. The second benefit is the inherent robustness to random bit flips or Single Event Upsets (SEUs). This is due to each bit in a stochastic bit stream having the same weight unlike with binary numbers.

Although SC systems use significantly less logic to perform complex operations, they can have very high latency and may not be as energy efficient as one might expect at first glance. These systems have higher accuracy when correlation between the bit streams is lower [6]. One solution is to use longer bit streams which increases the latency and area. Another solution is to use random number generators (RNG) that produce better quality random numbers. While linear-feedback shift registers (LFSR) are typically used [9], Sobol sequence generators can produce uncorrelated sequences and reduce the bit stream length and area [5]. But compared to the energy used by the computation logic, the RNGs consume a significant amount of the total power. Finally, stochastic operations also tend to increase correlation between bit streams. One method of reducing this effect is to add randomization steps between operations [10].

These solutions are all aimed at increasing the randomness in SC systems. However recent work shows that randomness is not at all required, and proposes deterministic approaches to SC [4]. The advantages of Deterministic Stochastic Computing (DSC) is that it produces fully accurate results, has lower area, and reduces the latency exponentially. RNGs that use a large area and consume a majority of the power are replaced with simple counters. Also, the number of operations required to produce fully accurate results are known at design time.

While DSC systems are a significant improvement, the latency in these systems can still be too high to be considered in many applications. In this paper we propose an optimal variable-latency architecture that significantly improves latency, area, and energy efficiency compared to the state-of-the-art. Also, our architecture is the only DSC system in literature that preserves the unary bit stream property after performing additions, multiplications and unary sorting. This means that all bit streams stay maximally correlated and is necessary when performing unary sorting such as MIN/MAX operations. Therefore our architecture enables building more complex DSC systems.

This paper is organized as follows. In Section II we give some background on traditional and Deterministic Stochastic Computing. Then in Section III, we describe the state-of-the-art variable-latency architectures. Next we describe our proposed architecture in Section IV. Then, in Section V we compare our architecture with the state-of-the-art ones. In Section VI we show the results of our evaluation and discuss the differences. Finally we conclude the paper in Section VII.

II. Background

A. Stochastic Computing

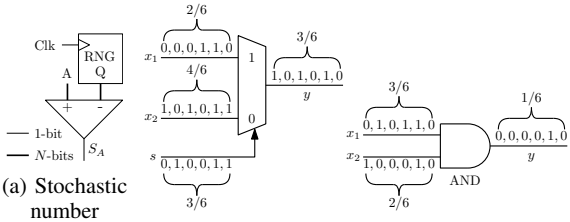
First introduced in [2], SC operates on stochastic bit streams, which have two possible coding formats: unipolar and bipolar. Unipolar formats encode a real number x in the interval $[0, 1]$ as a bit stream $X(t)$ of length L (where $t = 1, 2, ..., L$). Then the probability of each bit in X being a one is $P(X = 1) = x$. The bipolar format extends the range of x to $[-1, 1]$, and the probability of each bit in X being a one is $P(X = 1) = \frac{x+1}{2}$. While bipolar formats can encode negative numbers, the range of the unipolar format is twice as large with the same bit stream length L.

A stochastic bit stream is generated using a stochastic number generator (SNG) which comprises of a random number generator (RNG), and a binary comparator. Each clock cycle, the SNG compares an N-bit binary value A to an N-bit random number (RN) that the RNG generates, and the comparator outputs a 1 if $A > RN$, and a 0 otherwise. This is repeated until the desired bit stream length is achieved. An example of an SNG is shown in Fig. 1a. Then, SC applies simple logic gates to perform operations in terms of probabilities. Examples of scaled addition and multiplication are shown in Fig. 1b and Fig. 1c respectively. Converting bit streams back to binary values is simply done with a counter.

A major cause of accuracy loss in SC systems is due to highly correlated bit streams, i.e. the amount of overlapping 1's between bit streams. One reason that leads to increased correlation is that stochastic operations can lead to clumping together the 1's in the resulting bit stream. To show how accuracy loss occurs in SC systems due to highly correlated bit streams, we use multiplication as an example. Suppose we perform $3/6 \cdot 2/6$ where the values are represented as 111000 and 110000 respectively (i.e. maximally correlated values), the result after multiplication is 110000. This value, which represents $2/6$ is incorrect but also has clumped the 1's together. The correct result should have a single 1 in the result which represents $1/6$ (for example 000010). In order to minimize these random fluctuations and increased correlation in bit streams, their lengths need to be much longer than the resolution that they represent. Equation (1) defines the minimum bit stream length N required to generate a stochastic number within an error margin ϵ [8]:

$$N > \frac{(p)(1-p)}{\epsilon^2} \tag{1}$$

Since the $p(1-p)$ term is at most equal to 2^{-2}, the error margin ϵ must equal $2^{-(n+1)}$ where n is the binary bit length. Therefore, in order to represent a stochastic number with binary resolution of 2^{-n} bits, the bit stream length must be larger than 2^{2n} [8]. This exponential growth in bit stream length severely limits the range of usable resolutions in order to have acceptable latency. Another method employed to decrease correlation is to use randomization steps after stochastic operations to decrease the clumping of 1's in the bit stream [10].



(a) Stochastic number generator (b) Scaled addition (c) Multiplication

Fig. 1: Stochastic arithmetic operations [4].

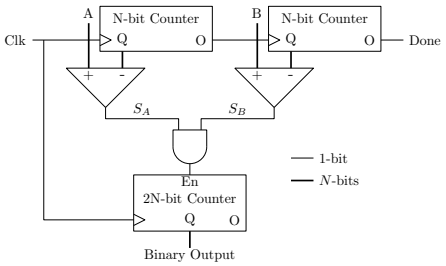


Fig. 2: N-bit clock-division DSC multiplier.

B. Deterministic Stochastic Computing

Recent work [4] shows that randomness is not a requirement in SC, and that deterministic bit streams can be used to produce fully accurate results. The key insight of this work is that a stochastic bit stream represents a probability, which is equal to the expected value of said bit stream, i.e. the *average* number of 1's and 0's. They show that independent random bit streams passively maintain the property that the average bits of stream *A* operate on the average bits of stream *B*. This is the same as convolving both bit streams.

Instead of relying on probability to maintain this property passively, they propose deterministic methods to actively enforce it, which removes the requirement that bit streams need to have low correlation between them, and produces fully accurate results. They propose three SNG types that result in convolved bit streams: *relatively prime bit lengths*, *rotation*, and *clock division*.

The relatively prime method ensures convolution by making sure that all generated bit streams are relatively prime. This limits the lengths that bit streams can have. The rotation methods explicitly rotates the bit streams, and the clock division method applies clock division to operands. These last two methods also maintain the convolution property and allow the usage of arbitrary length bit streams.

III. Variable-Latency Methods

The idea of variable-latency processing is doing less unnecessary work, i.e. when an operation does not contribute to the final result. An example of an N-bit DSC multiplier that uses clock division is shown in Fig. 2. This multiplier will always finish after 2^{2N} clock cycles regardless of input values A and B. When input values have small magnitudes,

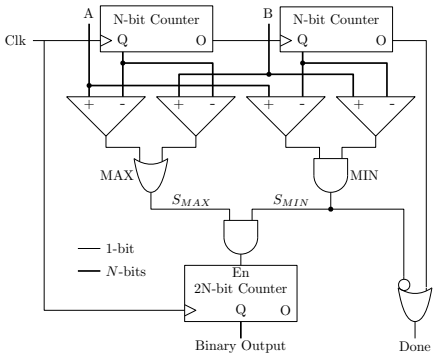


Fig. 3: N-bit DSC multiplier with bit stream sorting ES [3].

then during most clock cycles at least one of the values S_A and S_B is zero which does not contribute to the final value.

To reduce wasteful processing, variable-latency methods have been proposed which introduce so called *early shutoff* (ES) architectures [3]. When performing convolution using the clock-division method, *unary* bit streams are generated, which first contain all the 1's followed by 0's. In the example of Fig. 2, this means that when S_B becomes zero (or *Done* is asserted), we can stop since we know that all following bits of S_B will produce 0's. With the proposed ES method, assigning the value with a smaller magnitude to the B input will result in a lower latency since we can then stop earlier. This is made clear in Table I which shows how the ES hardware behaves at each clock cycle with 2 and 3 as inputs. When A is smaller than B, then S_B becomes zero on cycle 13 indicating that the computation can stop, whereas when B is smaller than A, then S_B becomes zero on cycle 9.

This method (*ES naïve*) is then improved by sorting input values, so that B always gets assigned the smaller value. This can be done either in the binary domain before the two values are assigned to A and B (*ES-CAS*), or in the bit stream domain using unary sorting [7] (*ES-BS*) as shown in Fig. 3.

While these state-of-the-art ES architectures can drastically reduce latency, they are not optimal in that only the counter of the *last* SNG stops early, while all the previous SNG counters count up to their maximum values. This becomes more apparent when N is large but all input values are small. Even with sorting, the A counter will count up to 2^N and will do that B times. For example, with $N = 8$, $A = 2$, and

TABLE I: 2-bit DSC Multiplication Using Clock-Division [3]

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A < B																
A=2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
B=3	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
A*B	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0
output cntr	1	2	2	2	3	4	4	4	5	6	6	6	6	6	6	6
B < A																
A=3	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0
B=2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
A*B	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0
output cntr	1	2	3	3	4	5	6	6	6	6	6	6	6	6	6	6

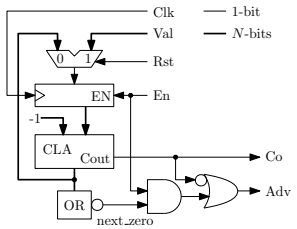


Fig. 4: Proposed stochastic number generator.

$B = 2$, the *ES ordered* architectures will take $2^8 \times 2 = 512$ clock cycles to complete where only 4 clock cycles actually contribute to the final result. When the amount of inputs increases, the overhead becomes larger as there are more counters that will count up to their maximum value.

IV. Optimal Variable-Latency Method

In this section, we propose an optimized variable-latency architecture that can significantly improve latency and energy efficiency compared to state-of-the-art. If only stochastic additions, subtractions, multiplications, and MIN/MAX operations are used, then our architecture is optimal in that only needs as many cycles as necessary to produce a fully accurate result when processing all bit streams serially. We describe how this is achieved by addressing three key problems in traditional and state-of-the-art variable-latency DSC architectures. Then we explain the design methodology of our architecture and analyze several key properties.

A. Key Improvements

1) Efficient Bit Stream Generator: Traditional SNGs in DSC systems need two adders to generate a single deterministic unary bit stream, namely the up counter and the binary comparator. We propose replacing both with a single down counter which we pre-load with the input value, and count down by one each clock cycle. We then use an OR reduction tree to detect if the next counter value is zero to reset the registers back to the input value. Comparing a value to zero is more efficient compared to a binary comparator, as it only requires a tree of OR gates instead of half adders and carry propagation logic. Our proposed SNG is shown in Fig. 4.

Merging both counters into one has a side effect that impacts designing systems using our architecture. Traditional SNGs use the carry-out signal of the previous SNG as a clock source. This is not possible with our proposed SNG, because if we preload the register with the value zero, then the carry-out signal will remain high indefinitely and thus will never toggle.

We mitigate this issue using an *enable* (EN) signal to allow the counter register to update, and generating an *advance* (Adv) signal that goes high for a single clock cycle. If the value zero is pre-loaded in the counter register then Adv remains high, which does not pose a problem since it is used in conjunction with a separate clock signal.

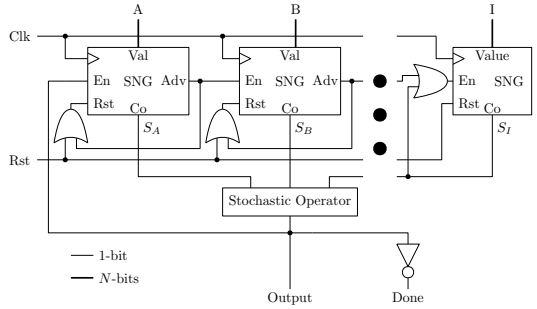


Fig. 5: SNGs in convolution configuration with I inputs.

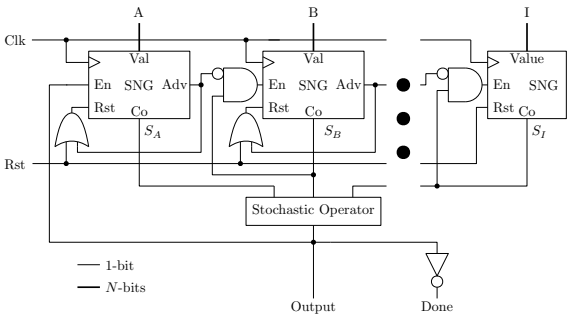


Fig. 6: SNGs in round-robin configuration with I inputs.

2) Unary Bit Stream Property Preservation: While traditional SNGs produce unary streams, they only do so for each *loop* from zero to the maximum counter value. In both examples in Table I, the A bit stream is unary in each loop of four clock cycles, generating 1100 and 1110 for the values 2 and 3 respectively, but since it has to run four times, the resulting bit stream is not unary as a whole. Our SNGs also produce the same amount of 1's but are unary as a whole.

This property is preserved when using stochastic adders and multipliers, and using unary sorting operations [7] since they do not generate 0's when all inputs are 1's like in our architecture. An exception is using an XOR gate to perform $|A - B|$, which does not generate unary streams. This is due to XOR gates producing 0's when both of their inputs are 1's. In fact, since unary sorting requires unary bit streams, our architecture is the only one in literature that allows using a sorting operation after an addition or multiplication due to the guarantee that these arithmetic operations preserve the unary bit stream property. This is highly efficient compared to traditional DSC systems, where bit streams after multiplication or addition would have to be converted to binary, and then back to bit streams for them to be unary again so that sorting could be done.

3) Universally Applied Early Shutoff: In state-of-the-art variable-latency architectures, early shutoff is only applied to the last SNG of each stochastic operation, where our architecture applies it to every SNG by resetting each if the next value is going to produce a zero. This can significantly improve the latency as each SNG produces only 1's and only as many as the magnitude of its input value, making it optimal.

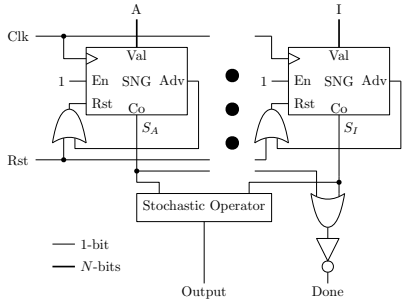


Fig. 7: SNGs in parallel configuration with I inputs.

Thus, swapping input values between SNGs does not effect latency, meaning that sorting of input values is not required.

B. Architecture Configuration

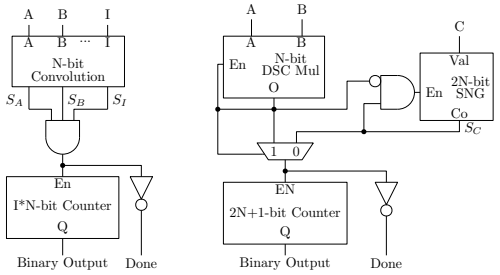
In this section, we analyze the three main configurations of using the SNGs: *convolution*, *round-robin* and *parallel*. They dictate when each SNG in a group of SNGs is allowed to run and are the foundational building blocks to implement stochastic operations. Table II shows an example with three SNGs and how they behave in these three configurations.

1) Convolution: Any I-input stochastic operation using the convolution configuration can be built using I SNGs. Every SNG except the last one is reset by ORing a global reset and its own Adv signal, whereas the last SNG only uses a global reset signal. The first SNG is enabled by the output of the stochastic operator, and each subsequent SNG is enabled by the Adv signal of the previous SNG. The last SNG is enabled by ORing the previous Adv signal and its own carry-out. If the stochastic operator produces 1's when all inputs are 1's, then the output is guaranteed to be unary. In that case, a convolution is finished when the stochastic operator produces a zero. Otherwise the done signal is generated by NORing the outputs of each SNG. Fig. 5 shows an example of an I-input convolution operation.

2) Round-Robin: The design of an I-input stochastic operation using the round-robin configuration is very similar to the convolution one. The difference lies in how each of the I inputs is enabled, where the first input is enabled by the

TABLE II: Example of counter values of SNGs A, B and C in convolution, round-robin and parallel configurations. SNGs generate a 1 if their counter value is not 0 (shown as shaded).

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Convolution															
A=3	3	2	1	3	2	1	3	2	1	3	2	1	0	0	0	0
B=1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
C=4	4	4	4	3	3	3	2	2	2	1	1	1	0	0	0	0
	Round-robin															
A=3	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0
B=1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
C=4	4	4	4	4	4	3	2	1	0	0	0	0	0	0	0	0
	Parallel															
A=3	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0
B=1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C=4	4	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0



(a) Multiplier (b) MAC Unit

Fig. 8: Simplified example designs using our proposed architecture. Clock and reset signals are not shown for simplicity.

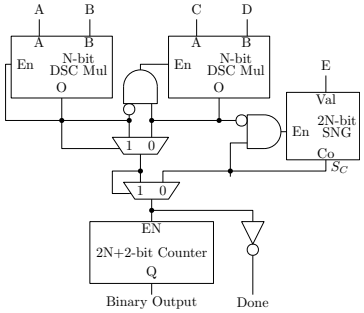


Fig. 9: Simplified example of an N-bit MAC unit with two multipliers using our proposed architecture. Clock and reset signals are not shown for simplicity.

output of the stochastic operator like with convolution. All subsequent inputs are enabled by ANDing the inverted Adv signal of the previous SNG with their own carry-out.

If all inputs of the round-robin operation are unary, and the stochastic operator produces 1's when all inputs are 1's, then the output is also guaranteed to be unary since it simply runs all unary input sources one after another. The stochastic operation is then finished when it produces a zero. However, if there is at least one non-unary source, then the done signal must be generated by NORing together the inputs of the non-unary source and the outputs of the other sources. Fig. 6 shows an example of an I-input round-robin operation.

3) Parallel: Using the parallel configuration, each SNG is free-running until all of them are finished. Since the latency of each SNG is variable, the result of a stochastic operator can generate 0's. Therefore the done signal should be generated by ORing the outputs of all SNGs, which gives us a unary bit stream as long as the SNG with the highest latency.

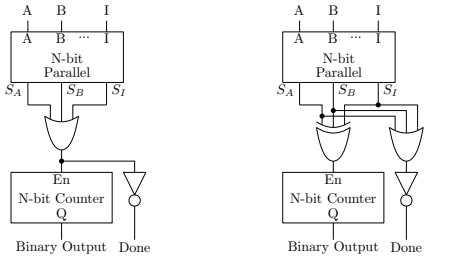
C. Design Methodology

Creating DSC systems using our proposed architecture requires some changes to the design methodology of traditional and variable-latency DSC systems.

1) Multiplier: A multiplier uses the convolution configuration and an AND gate as the stochastic operation. A simplified example of an I-input, N-bit multiplier is shown in Fig. 8a. Since all the SNGs only produce 1's and therefore the AND gate also only produces 1's, it means that the output of a multiplier is guaranteed to be unary.

2) Adder: In traditional DSC systems, a stochastic scaled adder is implemented as an I-input multiplexer (MUX) as shown in Fig. 1b. All stochastic operations that are connected to the adder run in parallel. A counter, driven by those stochastic operations, in its turn drives the multiplexer. Scaling is done by simply generating an I times longer bit stream and concatenating the bit streams of each input of the adder.

In our architecture, a stochastic scaled adder is implemented using the round-robin configuration and a 2-input MUX as the stochastic operation. The first operand of the MUX is connected to the '1' and select ports, and the second is connected to the '0' port. An example of a MAC unit using a 2-input MUX is shown in Fig. 8b.



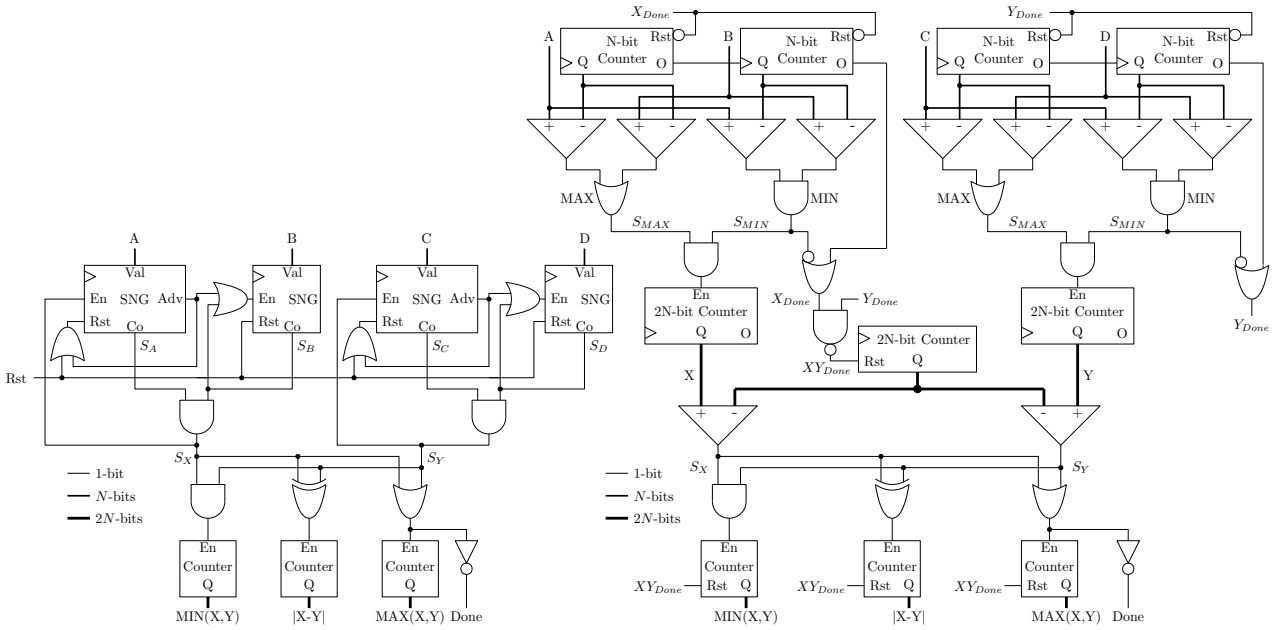
(a) Maximum Value (b) Absolute Difference

Fig. 10: Simplified example designs using our proposed architecture. Clock and reset signals are not shown for simplicity.

Adders with more inputs are implemented by cascading multiple 2-input MUXs recursively. With each extra operand, another MUX is added, the output of the previous MUX is connected to the '1' and select ports, and the output of the extra operand is connected to the '0' port. This makes it so that our architecture does not require a counter for driving a MUX, simplifying the design. An example of a MAC unit with two multipliers , which requires adding three operands is shown in Fig. 9. Due to the used round-robin configuration, each source is only active during its turn and disabled otherwise, lowering the power usage.

3) Unary Sorting: When bit streams are maximally correlated, then the MIN and MAX operators can be performed with a single AND and OR gate respectively [7], and is implemented using the parallel configuration. Both operators preserve the unary bit stream encoding property, and can be used after other unary sorting operators. Fig. 10a shows a simplified example design of finding the maximum value.

4) Absolute Difference: A single XOR gate can calculate $|A - B|$ given that both bit streams are maximally correlated [1]. However, it generates 0's when both inputs are 1's, and thus violates the unary bit stream preserving property. An example is $1000 \oplus 1111 = 0111$, which generates a reversed unary bit stream. In our architecture, it is implemented using the parallel configuration. A simplified example design calculating the absolute difference is shown in Fig. 10b.



(a) Our architecture (b) ES-BS architecture [3]

Fig. 11: Examples of a multiply-sort design. Clock and some reset signals are not shown for simplicity.

V. Evaluation

In this section we evaluate a DSC multiplier and DSC multiply-sort unit implemented using three architectures: *ES-BS* and *ES-CAS* [3] which perform unary bit stream sorting and pipelined Compare-and-Swap binary sorting before the SNGs respectively, and our proposed architecture. The multiply-sort designs are shown in Fig. 11. All implementations were synthesized using Synopsys Design Compiler with $high$ synthesis and mapping effort. The TSMC 40nm LP standard cell library was used with a typical-typical corner case and $V_{dd} = 1.1V$. 1000 uniformly distributed random numbers were generated and used as stimulus to obtain average performance, power, and energy figures for all implementations using gate-level simulations. Correctness was verified by comparing the test bench results to a Python script performing the same operations.

VI. Results

Synthesis, power, and energy results for the multiplier and multiply-sort implementations are shown in Table III and Table IV respectively. We report area numbers for synthesis results at the highest frequency that each implementation can run at, and for a frequency that all implementations can run at comfortably, which is 1 GHz and 625 MHz for the multiplier and multiply-sort implementations respectively.

A. Multiplier

Compared to the state-of-the-art variable-latency architectures, the results show that for each multiplier configuration our proposed design has at least a 24.4%, 50.5% and 69.4% lower latency for two, three and four inputs respectively. The difference in latency also grows as the number of inputs increases. These differences are due to our architecture being optimal in that each SNG only counts for minimum amount of cycles compared to the other designs that perform an early shutoff for the last SNG.

Looking at timing and area numbers, the worst case timings of our design is between 15.0% higher and 20.0% lower for two and three inputs compared to the *ES-BS* and *ES-CAS* reference designs, but is between 5.1% and 23.3% lower for four inputs. Area wise at maximum clock frequency, our design is between 4.4% and 33.0% smaller than the reference designs, and synthesized for 1 GHz, our design is between 15.1% and 42.8% smaller. This is due to our architecture not needing any N-bit binary comparators and sorting hardware. As the number of inputs grows with the *ES-CAS* and *ES-BS* designs, the sorting hardware increases in complexity because not only the smallest and largest input values have to be found, but also the value in between and the second smallest and second largest for three and four inputs respectively [3].

Finally, when looking at the power and energy usage, we see that the power usage of our designs is comparable to the *ES-BS* design, and between 9.9% and 34.7% less than the *ES-CAS* design. This difference is mostly attributed to the *ES-CAS* design having additional pipeline registers compared to the other designs. Since the latency of our design is lower, we see that it uses between 24.2% and 71.4% less energy than the *ES-BS* design, and between 33.3% and 77.0% less than the *ES-CAS* design.

TABLE III: Multiplier Implementation Results

#inputs	2				3			4	
Bit width	4	6	8	10	4	6	8	4	6
Average #cycles/operation									
ES-BS	77	1367	22516	345463	900	63903	4201842	11228	3318482
ES-CAS	77	1367	22516	345463	900	63903	4201842	11228	3318482
Proposed	57	1033	16993	259864	424	30806	2081495	3226	1015674
Worst case timing (ps)									
ES-BS	420	460	510	590	470	510	550	530	590
ES-CAS	400	470	530	590	470	700	750	650	730
Proposed	460	490	510	610	470	530	600	500	560
Area (μm^2) @ Maximum Clock Frequency									
ES-BS	647	1022	1256	1369	1023	1629	2184	1430	2116
ES-CAS	625	938	1050	1273	938	1522	2077	1545	2240
Proposed	463	754	979	1217	761	1151	1426	1059	1500
Area (μm^2) @ 1 GHz									
ES-BS	352	537	709	886	593	915	1209	889	1374
ES-CAS	395	598	789	985	702	1086	1625	1013	1559
Proposed	299	445	591	749	445	665	890	590	892
Total power (μW) @ 1 GHz									
ES-BS	337	454	567	680	454	628	801	579	811
ES-CAS	383	529	671	814	540	803	1224	718	1073
Proposed	347	459	578	718	461	628	799	576	805
Total energy (nJ) @ 1 GHz									
ES-BS	26	621	12767	234915	409	40131	3365675	6501	2691289
ES-CAS	29	723	15108	281207	487	51314	5143055	8062	3560731
Proposed	20	470	9669	178007	195	19346	1663115	1858	817618
Frequency that all designs in column can converge to (GHz)									
	2.17	2.04	1.89	1.64	2.13	1.43	1.33	1.54	1.37

B. Multiply-Sort

We have implemented a multiply-sort design using only the *ES-BS* architecture and our proposed architecture shown in Fig. 11b and Fig. 11a respectively. We have not implemented it using the *ES-CAS* architecture since it had the worst results in terms of power and energy efficiency in the multiplier design.

In terms of latency, our design requires between 55.6% and 55.9% less clock cycles than the *ES-BS* design. Timing wise, our design has a lower worst case timing between 52.0% and 55.2%. Looking at area numbers, our design is between 41.1% and 47.2% smaller when synthesized for maximum clock frequency, and between 40.4% and 51.7% smaller when synthesized for 625 MHz. Finally, our design uses between 16.4% and 31.6% less power than the *ES-BS* design, and combined with the much lower latency, the energy usage of our design is between 62.9% and 69.8% lower.

The main reason for these significant differences is that sorting and absolute difference operations require unary bit streams to produce correct results. However, traditional DSC systems almost never produce unary bit streams after multiplication (see Table I for an example). Therefore, a conversion to the binary domain and then back to the bit stream domain is necessary to produce unary bit streams. This extra conversion step can only take place after the multiplications are finished, which is the main source of higher latency, area and power usage of the *ES-BS* design. It also shows that our architecture enables more complex designs without any costly conversion overhead.

VII. Conclusion

In this paper we proposed an optimized variable-latency architecture for deterministic approaches Stochastic Computing systems, that can significantly improve latency compared to state-of-the-art. When using stochastic additions, absolute difference, multiplications and unary sorting, it is also optimal in that only needs as many clock cycles as necessary to produce a fully accurate result. Also, our architecture does not require any sorting of input values as is necessary with state-of-the-art variable-latency architectures. We have also proposed an efficient stochastic number generator that uses a down counter and compare-to-zero instead of the traditionally used up counter and binary comparator, which leads to a smaller design. This results in the latency of our proposed stochastic number generator to only depend on the magnitudes of the input values that are processed, so that far less clock cycles are wasted on operations that do not contribute to the final result. We implemented a multiplier and multiply-sort unit and have shown that our architecture can significantly improve latency, area, and energy efficiency compared to state-of-the-art variable-latency architectures.

Acknowledgment

This work is part of the research program Perspectief ZERO with project number P15-06 Project 3, which is (partly) financed by the Dutch Research Council (NWO).

References

[1] Armin Alaghi, Cheng Li, and John P. Hayes. “Stochastic circuits for real-time image-processing applications”. In: *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2013, pp. 1–6. DOI: 10.1145/2463209.2488901.

[2] Brian R Gaines. “Stochastic computing systems”. In: *Advances in information systems science*. Springer, 1969, pp. 37–172.

TABLE IV: Multiply-Sort Unit Implementation Results

Bit width	4	6	8	10
Average #cycles/operation				
ES-BS	196	3421	54341	885799
Proposed	87	1514	24016	390927
Worst case timing (ps)				
ES-BS	1050	1160	1250	1350
Proposed	470	530	600	640
Area (μm^2) @ Maximum Clock Frequency				
ES-BS	1979	2917	3854	4726
Proposed	1144	1665	2271	2495
Area (μm^2) @ 625 MHz				
ES-BS	1224	2055	2768	3708
Proposed	730	1081	1437	1790
Total power (μW) @ 625 MHz				
ES-BS	578	882	1119	1483
Proposed	483	657	836	1015
Total energy (nJ) @ 625 MHz				
ES-BS	113	3017	60808	1313640
Proposed	42	995	20077	396791
Minimum frequency of all designs in column (GHz)				
	0.95	0.86	0.8	0.74

[3] Alexander J. Groszewski and Earl E. Swartzlander. "A Variable-Latency Architecture for Accelerating Deterministic Approaches to Stochastic Computing". In: *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. 2019, pp. 608–613. DOI: 10.1109/IEEECONF44664.2019.9048881.

[4] Devon Jenson and Marc Riedel. "A Deterministic Approach to Stochastic Computation". In: *Proceedings of the 35th International Conference on Computer-Aided Design*. ICCAD '16. Austin, Texas: Association for Computing Machinery, 2016. ISBN: 9781450344661. DOI: 10.1145/2966986.2966988. URL: https://doi.org/10.1145/2966986.2966988.

[5] Siting Liu and Jie Han. "Toward Energy-Efficient Stochastic Circuits Using Parallel Sobol Sequences". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.7 (2018), pp. 1326–1339. DOI: 10.1109/TVLSI.2018.2812214.

[6] M. Hassan Najafi, David J. Lilja, and Marc Riedel. "Deterministic Methods for Stochastic Computing Using Low-Discrepancy Sequences". In: *Proceedings of the International Conference on Computer-Aided Design*. ICCAD '18. San Diego, California: Association for Computing Machinery, 2018. ISBN: 9781450359504. DOI: 10.1145/3240765.3240797. URL: https://doi.org/10.1145/3240765.3240797.

[7] M. Hassan Najafi et al. "Low-Cost Sorting Network Circuits Using Unary Processing". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.8 (2018), pp. 1471–1480. DOI: 10.1109/TVLSI.2018.2822300.

[8] Weikang Qian. "Digital yet Deliberately Random: Synthesizing Logical Computation on Stochastic Bit Streams". AAI3466985. PhD thesis. USA, 2011. ISBN: 9781124808987.

[9] Weikang Qian et al. "An Architecture for Fault-Tolerant Computation with Stochastic Logic". In: *Computers, IEEE Transactions on* 60 (Feb. 2011), pp. 93–105. DOI: 10.1109/TC.2010.202.

[10] Menghui Xu et al. "Stochastic Belief Propagation Polar Decoding With Efficient Re-Randomization". In: *IEEE Transactions on Vehicular Technology* 69.6 (2020), pp. 6771–6776. DOI: 10.1109/TVT.2020.2979610.