Streaming Accuracy: Characterizing Early Termination in Stochastic Computing

Hsuan Hsiao
University of Toronto
julie.hsiao@mail.utoronto.ca

Joshua San Miguel
University of Wisconsin-Madison
jsanmiguel@wisc.edu

Jason Anderson
University of Toronto
janders@ece.utoronto.ca

***Abstract*—Stochastic computing has garnered interest in the research community due to its ability to implement complicated compute with very small area footprints, at the cost of some accuracy and higher latency. With its unique tradeoffs between area, accuracy and latency, one commonly used technique to minimize area and latency is to early-terminate computation. Given this, it is useful to be able to measure and characterize how amenable a bitstream is to early termination. We present *Streaming Accuracy*, a metric that measures how far a bitstream is from its most early-terminable form. We show that it overcomes limitations of prior studies, and we characterize the design space for building stochastic circuits with early termination. We then propose a new hardware bitstream generator that produces bitstreams with optimal streaming accuracy.**

I. Introduction

Stochastic computing (SC) [16], [7] is a reemerging computing paradigm that performs computation on bit-serial unary bitstreams as opposed to bit-parallel binary-encoded registers. Values in SC are represented by the probability that a bit is set in a bitstream (e.g. 0010 and 1000 both represent a value of $\frac{1}{4}$ in the unipolar format). Because of its value encoding format and its serial nature, SC is capable of performing computation with extremely small functional units (e.g. multiplication is reduced to a single AND gate). Its potential to have extremely low power and area has inspired researchers in the community to create efficient circuit implementations in various applications domains, including image processing [10], [1], [6], error-correction code decoding [15], [5] and neural networks [12], [13], [3].

While SC circuits can achieve low power and area, they trade off latency to provide similar compute accuracy as their traditional binary-encoded counterpart. SC circuits require exponentially higher latency compared to their binary-encoded counterparts. In order to bridge this performance gap, the ability to early-terminate computation greatly enhances the feasibility of SC circuits. With that in mind, the ability to *measure* the error consequences of early termination is invaluable for SC circuit designers.

Progressive precision [2] and normalized stability [19] are two metrics in SC literature that can help designers with quantifying early termination. While they each have their own respective use cases where they excel, they are insufficient if we want to know how well a bitstream is amenable to early termination at arbitrary termination points. Progressive precision evaluates a bitstream only at powers-of-2 termination points, while normalized stability requires a user-defined error threshold that is non-trivial to define. As we will show later in Section III, these limitations make the existing metrics unable to identify, among different bitstreams representing the same value, which bitstream to use to achieve the highest accuracy at arbitrary early termination points.

To aid in the ability to measure how close a bitstream is to its most early-terminable form, we propose a new metric called *streaming accuracy*. Any bitstream with a streaming accuracy of 1 achieves the highest accuracy possible at any arbitrary early termination point and thus represents the most early-terminable form. Using streaming accuracy, designers can characterize and analyze their SC circuit to see whether a component hinders or enhances the circuit's overall ability to support early termination. Early termination-based optimization and design space exploration can be carried out and evaluated in a generalized manner (i.e. without prior knowledge of any application-specific error tolerance value). Alongside streaming accuracy, we also propose the design of a hardware bitstream generator that always produces bitstreams with streaming accuracy of 1, which we call the *streaming-accurate (SA) generator*. We show later in Section V that using our SA generator yields significantly less error compared to popular Halton and LFSR-based generators when terminated early.

The contributions of this paper are as follows:

- We propose streaming accuracy, a metric that evaluates how close a bitstream is to its most early-terminable form.
- We present characterizations of how well different functional units can retain streaming accuracy and how early termination interacts with bitstream independence.
- We introduce the concept of a streaming-accurate bitstream, which achieves the lowest possible error at all partial bitstream lengths, and propose a design for a bitstream generator that generates streaming-accurate bitstreams.

II. Background and Related Work

This section gives an overview of existing metrics in literature that quantify aspects of early termination in SC, as well as how SC bitstreams are typically generated in SC circuits.

A. Early Termination Metrics

Progressive Precision. Progressive precision (PP) [2] is a property that enables a stochastic computation to trade off its accuracy with speed. If the first $l \leq L$ bits of a length-L

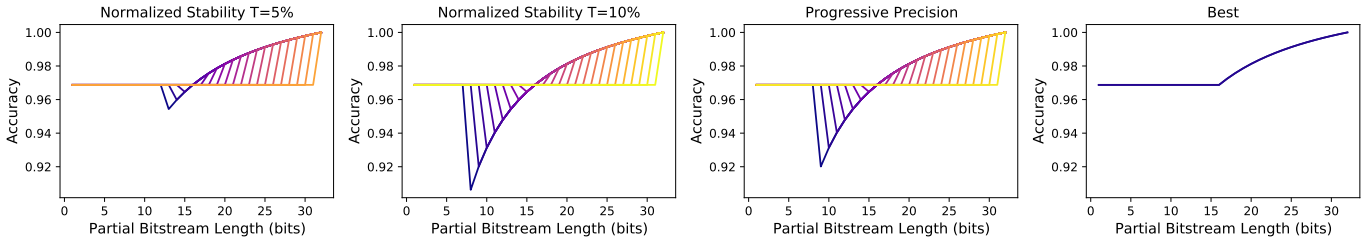

Fig. 1: Using normalized stability and progressive precision to select the best bitstream with a value of $\frac{1}{32}$ to use for early termination. Multiple bitstreams selected by the metric are plotted as different coloured lines.

computation provide a sufficiently good approximation of the desired result, the computation can be stopped early. Applying it to a bitstream X of length L, X is k-PP if the bit-error of each initial partial bitstream X_l of length $l = 2^i$ is at most k for all i. Bit-error is defined as $l \times |P_{X_l} - P^*_X|$, where P_{X_l} is the value represented by bitstream X_l, and P^*_X is the desired exact value. For example, if $X = 0111111111110000$ and $P^*_X = \frac{10}{16}$, then the initial sub-sequences of length $l = 2$, 4, 8, and 16 are $X_l = 01$, 0111, 01111111, and 0111111111110000. The bit-error for each of the partial bitstreams are 0.25, 0.5, 2, and 1 respectively, so X is 2-PP. Unlike our streaming accuracy metric, progressive precision only evaluates the bitstream at powers-of-2 partial bitstream lengths. Section III goes into detail on the limitations and differences between the two.

Normalized Stability. Wu et al. [19] define a serial bitstream to be in the stable state when the fluctuation of the bitstream value is below a user-defined error threshold. Normalized stability is a metric that measures how long a bitstream is in the stable state, within a target error threshold, normalized to the maximum achievable stable duration. Its value ranges from 0 to 1, where the closer the normalized stability is to 1, the earlier the bitstream is able to reach the stable state. For example, if the user-defined error threshold is 10%, a bitstream representing the value 0.5 that looks like $X = 1010101010101010$ is the most stable form and becomes stable after 8 bits (i.e. terminating any time after 8 bits is guaranteed to be within 10% error); it achieves the maximum stable duration of 8, and thus its normalized stability is defined as 1. Comparatively, a bitstream that looks like $Y = 0000111100001111$ has a normalized stability value of 0.5 since it is only stable after 12 bits, having only half of the stable duration of the most stable form. Unlike our streaming accuracy metric, normalized stability requires a user-defined error threshold to be useful, which can be non-trivial to define. Section III provides more detailed explanation on the limitations and differences between the two.

B. SC Bitstream Generator

In general, a length-L SC bitstream is generated by comparing the value of the desired number with random numbers from a random number generator (RNG) source over L cycles; if the random number is less than the desired number, a bit of 1 is generated for that cycle. For a bitstream length of 8 and a desired value of $\frac{4}{8}$, one would set the RNG to output random numbers between 0 and 7 and compare with 4. This will statistically generate an output bitstream where 4 out of 8 bits are set to one. Typical RNGs used in state-of-the-art circuits include linear-feedback shift registers (LFSRs) and low-discrepency sequences such as Halton [2] and Sobol [11].

III. Streaming Accuracy

As discussed previously, early termination is a crucial part of efficient stochastic computing; thus it is important for hardware designers to be able to accurately measure and characterize it. Progressive precision and normalized stability are prior metrics that can be used for this purpose, however they have some fundamental limitations.

Limitations of Prior Studies. Fig. 1 illustrates the results obtained when using normalized stability and progressive precision to select versions of bitstreams representing a value of $\frac{1}{32}$ that the metrics indicate are best for early termination. For normalized stability, error thresholds T of 5% and 10% are used (left-most two plots). In both cases, multiple bitstreams of value $\frac{1}{32}$ obtained the highest normalized stability value of 1 (20 and 25 bitstreams for 5% and 10%, respectively), and each version is plotted as a different coloured line. Compared with the actual best bitstream shown on the far right, normalized stability is unable to distinguish the best bitstream from the sub-optimal ones, potentially costing accuracy when terminating early. Similarly, for progressive precision (third plot from the left), 24 different bitstreams were identified as having the lowest k value of 0.5, each of them plotted as a different coloured line. The best bitstream for early termination is again indistinguishable from the sub-optimal ones when only looking at the value produced by progressive precision.

The limitation of progressive precision is that it evaluates only power-of-2 partial-bitstream lengths. As shown in Fig. 1, all selected bitstreams have the same accuracy as the best bitstream at power-of-2 partial bitstream lengths; however, no accuracy guarantee is given for any other partial bitstream lengths. On the other hand, normalized stability requires a user-defined error threshold to be useful, which is non-trivial to set unless the designer has sufficient application-specific knowledge. Although normalized stability can provide an estimate of which cycle it is safe to early terminate to stay within the specified error threshold, this cycle number quickly increases as the error threshold decreases. Note that normalized stability does not work with an error threshold

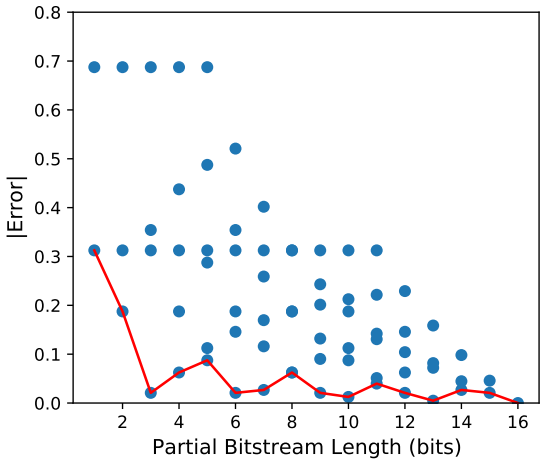

Fig. 2: Example of early termination error for a bitstream representing the value $\frac{5}{16}$. All possible $|error|$ at each partial bitstream length is shown in blue. A bitstream with $|error|$ that consists of all points marked by the red line is defined to be a streaming-accurate bitstream.

of 0, some non-zero error tolerance amount must always be specified. Overall, while these two metrics are good for their particular use cases, they are insufficient in measuring early termination in a more generalized manner.

Streaming Accuracy Definition. To be able to reason about early termination in a more generalized manner, we desire a metric that can reflect the accuracy of a bitstream at any arbitrary early termination point. For a given length-L bitstream representing a value of P_X, it can have L different early termination values P_{X_i} (given $i \leq L$), which each correspond to the value represented by its initial partial bitstream value up to the i^{th} bit of the bitstream (e.g. $\frac{\text{\# of 1s from bits 1 to } i}{i}$ in unipolar format). We look at the sum of all possible early-termination errors: $\sum_{i=1}^{L} |P_{X_i} - P_X|$, and define its raw streaming accuracy to be

$$\Phi = 1 - \frac{\sum_{i=1}^{L} |P_{X_i} - P_X|}{L} \tag{1}$$

Out of all different bitstreams representing the same value, the *best* bitstream for early termination at arbitrary points will have the highest raw streaming accuracy value.

Fig. 2 shows an example of all possible early-termination error magnitudes at different early-termination points for all bitstreams representing a value of $\frac{5}{16}$. Looking at this plot, we observe that achieving the minimum early termination error at each successive partial bitstream length (e.g. blue dots marked by the red line) requires monotonically increasing the number of 1s in its partial bitstream by at most 1. This trend holds for all other bitstream values as well. *From this analysis, we recognize that there always exists a bitstream that is optimal for early termination at arbitrary points since a partial bitstream of length $i+1$ can be constructed from a partial bitstream of length i.* For the example in Fig. 2, the bitstream that is optimal for early termination is 0100100100010010. Because of this characteristic, minimizing the error introduced at each additional bit will minimize the overall error. To construct such a bitstream, one can simply take a greedy approach of evaluating at each clock cycle whether introducing a 0 or a 1 will minimize the deviation from the full bitstream value (see Section V for our proposed hardware design that implements this). We call this bitstream the **streaming-accurate bitstream** for that value, and it is guaranteed to be the optimal early-terminable form with $\Phi = \Phi_{best}$ for that value. The streaming-accurate bitstream achieves the lowest possible error at all partial bitstream lengths (e.g. red line in Fig. 2).

To know how close a bitstream is from its optimal early-terminable form, we normalize its Φ to the Φ_{best} represented by the streaming-accurate bitstream, and define **streaming accuracy** (value-dependent form) to be

$$\text{Streaming accuracy}_{value-dependent} = \frac{\Phi}{\Phi_{best}} \tag{2}$$

A streaming accuracy of 1 indicates that the bitstream is in optimal early-terminable form, and the closer to 1 a bitstream's streaming accuracy is, the more amenable it is to be early-terminated at an arbitrary point.

Since the formulation of streaming accuracy utilizes the summation of early termination errors at different early-termination points, the Φ and Φ_{best} values are value-dependent and this can make it difficult to use when trying to aggregate streaming accuracy values across bitstreams representing different values. We define a value-independent version of streaming accuracy as

$$\text{Streaming accuracy}_{value-independent} = \frac{\Phi - \Phi_{worst}}{\Phi_{best} - \Phi_{worst}} \tag{3}$$

Φ_{worst} refers to the Φ of the worst bitstream for terminating early at arbitrary points. In this value-independent form, streaming accuracy is defined to be 1 when $\Phi_{best} = \Phi_{worst}$. For any bitstream representing a value < 0.5, the worst bitstream will be the one where all 1s are clumped at the beginning of the bitstream. Conversely, for any bitstream representing a value > 0.5, the worst will be the one where all 1s are clumped at the end of the bitstream. In this form, a streaming accuracy of 0 presents the least early-terminable form, and a streaming accuracy of 1 still represents the most early-terminable form. Any value in between indicates the early-termination characteristic of the bitstream relative to the best and the worst.

Evaluation. To see how streaming accuracy compares with progressive precision and normalized stability, we perform an experiment where we use each metric to select bitstreams deemed to be the best for early-termination and plot the accuracy afforded by their selection in Fig. 3. For each value representable with a bitstream length of 32, we sample up to 10k different bitstreams representing that value (some values have less than 10k different forms) and use each of the metrics to select the bitstream deemed best for early-termination. The accuracy of the selected bitstream at each partial bitstream length is calculated as $1 - |P_X - P_{S_i}|$, where P_X represents the value of the full bitstream, and P_{S_i} represents the value of

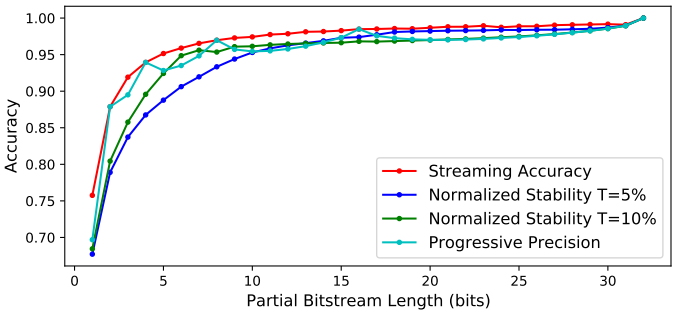

Fig. 3: Average accuracy of using partial bitstreams selected by different metrics: normalized stability, progressive precision and streaming accuracy (value-dependent).

TABLE I: Streaming accuracy retention (value-independent) of functional units.

Op	Functional Unit	SA_{out}/SA_{in}
$\times$	AND [8]	1.011
$\times$	uMUL-ST [17]	1.044
$\times$	uMUL-IS [17]	0.992
$+$	MUX with TFF select [8]	1.002
$+$	MUX with LFSR select [8]	1.000
$+$	uSADD [17]	1.050
$\div$	CORDIV [4]	0.885
$\div$	ISCBDIV [18]	0.979

the selected bitstream at partial bitstream length i. If a metric indicates multiple bitstreams to be selected as the best, the accuracy value at each partial bitstream length is averaged across all selected bitstreams. Finally the accuracy for each value is averaged across all representable values to produce the curve in Fig. 3.

From this plot, we can see that bitstreams selected using streaming accuracy consistently produce higher accuracy across different early termination points. Bitstreams selected by progressive precision obtain high accuracy at power-of-2 early-termination points but have lower accuracy at other points. Bitstreams selected by normalized stability have lower accuracy at earlier termination points and only really attain higher accuracy when terminating later (e.g. when the bitstream is considered stable). Another interesting observation with normalized stability is that the accuracy value for terminating earlier (before cycle 13) using $T = 10\%$ is actually higher than using $T = 5\%$, which is contrary to what one would expect. This further shows the limitations of normalized stability if we want to use it to measure the consequences of early termination at any arbitrary point.

IV. Characterizing Early Termination

Streaming accuracy can be valuable in characterizing SC circuits. In this section, we present two characterizations: the first evaluates various functional units proposed in the literature, and the second illustrates the relationship between streaming accuracy and bitstream independence.

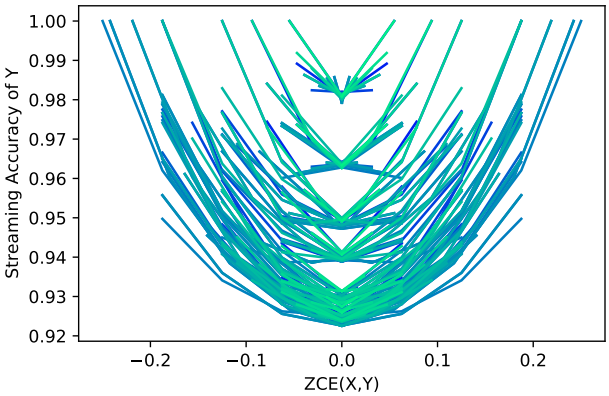

Fig. 4: Streaming accuracy (value-dependent) of different values when X is optimal streaming accuracy.

A. Streaming Accuracy Retention

When optimizing an SC circuit for early termination, it is useful for the designer to know whether passing bitstreams through different components will increase, decrease, or retain the streaming accuracy at the output. We perform this analysis on several arithmetic functional units proposed in the literature, assuming bitstream lengths of 64. For each functional unit, we measure both the input and output streaming accuracy and show the ratio between them in Table I. The inputs to this experiment consist of 100K random samples of bitstreams over all possible (X,Y) value combinations. The input streaming accuracy is computed as the average across all X and Y bitstreams used. Across different implementations of multiplication and scaled addition, we see that they generally retain and can even slightly improve the streaming accuracy of the bitstreams. On the other hand, both implementations of the divider yield a decrease in streaming accuracy of bitstreams passing through them, since they need to reorder bits in order to perform the division accurately. This characterization motivates further design space exploration of SC functional units that can achieve higher retention of streaming accuracy.

B. Relation to Bitstream Independence

Bitstream independence is often an important property for SC computations. Given two bitstreams X and Y, they are said to be independent if the output bitstream of an AND gate ($P_{X \wedge Y}$) equals the product of the two input bitstreams ($P_X P_Y$). As shown in prior work, some functional units (e.g. multiplication with an AND gate) suffer in accuracy when input bitstreams are not independent. Thus it is important to optimize SC circuits not just for streaming accuracy but also for bitstream independence. However, their relationship has not been well studied, and it is not necessarily clear whether or not these two properties are orthogonal.

We conduct experiments to characterize the relationship between streaming accuracy and bitstream independence; results are shown in Figure 4. We measure bitstream independence using ZCE [9]; a ZCE value closer to 0 implies higher independence between the bitstreams. We first set one bitstream

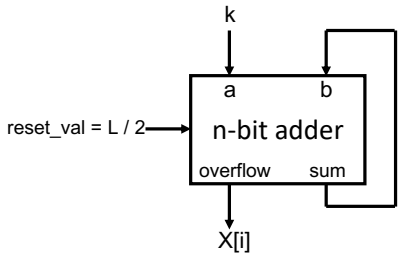

Fig. 5: Hardware for a bitstream generator that produces SA bitstream.

TABLE II: Area for generators producing bitstreams of L=64.

Generator	Area (μm^2)	Power (uW)
LFSR	75.278	26.851
Halton (base 2)	69.16	39.73
Halton (base 3)	245.252	140.796
Sobol	208.81	75.546
SA	69.16	31.306

X to be the streaming-accurate version of all possible values representable with a bitstream length of 16. For each bitstream X, we iterate through all possible values of another bitstream Y and look at all possible bitstreams for each given value of Y and plot the streaming accuracy of the Y bitstream against ZCE(X,Y). Each (X,Y) value pair shows up as a single line in the figure. As shown in the figure, streaming accuracy and bitstream independence are not orthogonal. The more independent two bitstreams are, the lower their streaming accuracy are in general. From closer analysis, we find that the reason is because forcing two bitstreams to each have high streaming accuracy reduces the number of possible alignments of 1s and 0s that they can have. This in turn leads to more similarity between the two bitstreams and thus higher likelihood that they will not be independent. This characterization motivates further research in the design of SC circuits that are jointly optimized for streaming accuracy and bitstream independence.

V. Streaming-Accurate Bitstream Generator

As mentioned in Section III, for a given bitstream length and value, we find that there exists a streaming-accurate (SA) bitstream that is most amenable to early termination. The SA bitstream has a particular form where the location of 0s and 1s are organized to be as evenly distributed as possible (e.g. the SA bitstream for $\frac{3}{7} = 0.43$ is 0101010 and $\frac{4}{7} = 0.57$ is 1010101). For bitstream lengths that are even numbers, there can exist more than one bitstream that qualifies as the SA bitstream (e.g. 01000010 and 00100010 are both SA bitstreams for $\frac{2}{8}$). In this section, we present the hardware design of a SA bitstream generator that always produces SA bitstreams, making it an ideal candidate for SC systems that support early termination.

Fig. 5 shows our proposed design of the SA bitstream generator. It is essentially an n-bit adder with preset capability, where $n = log_2 L$. For a length-L bitstream, if the value we want to generate is $\frac{k}{L}$, we set the adder's initial sum to $\frac{L}{2}$ and add k every cycle. At every cycle i, if the adder overflows, the new bit generated $X[i]$ will be a 1, otherwise it will be a 0. Note that the design shown in Fig. 5 is specific to power-of-2 bitstream lengths, which are most common in SC systems. To handle bitstream lengths of $L \neq 2^n$, we simply use an adder of size $n = ceil(log_2 L)$. Instead of simply checking the counter overflow, we emulate an arbitrary value overflow by adding a comparator to check sum $\geq L$ and outputting 1 when it is true, 0 otherwise. Every time the sum "overflows," we subtract L from the sum so the adder can keep accumulating.

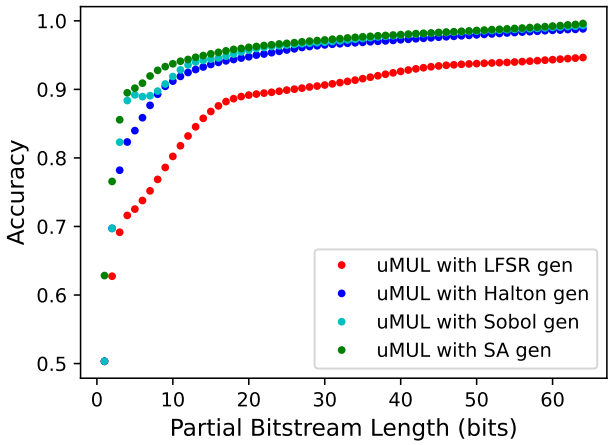

Fig. 6: Accuracy of uMUL using different generators.

A. Area and Power Evaluation

To evaluate the area and power of our SA bitstream generator, we implement it in Verilog RTL and synthesize to netlist using the area optimization options in Synopsys Design Compiler with the open-sourced NanGate FreePDK45 standard-cell library [14]. To compare against popular SC bitstream generators in literature, we implement a LFSR-based bitstream generator, two versions of the Halton-based generator and a Sobol-based generator from [17] and synthesize using the same design flow. Table II shows the area and power (at 500MHz) results for each of these SC bitstream generators when set to produce bitstreams of length 64. Our SA bitstream generator has area cost close to the other generators based on LFSR and base-2 Halton, which are among the lower area costs of existing generators. The bitstream generator using base-3 Halton sequences incurs significantly higher area cost due to the increasing cost of the binary-coded base-b counter that it requires as b increases.

B. Accuracy Evaluation

We evaluate the accuracy impact of using our SA generator vs. using the LFSR, base-2 Halton and Sobol generators at both the functional unit level and application level.

Functional Unit Evaluation. At the functional unit level, we show the accuracy of a state-of-the-art correlation-insensitive multiplier unit (uMUL-ST [17]) when the internal generator (i.e. used to generate $S^{eff}_{in,1}$ for the conditional probability calculation [17]) is set as either the LFSR, Halton, Sobol or SA

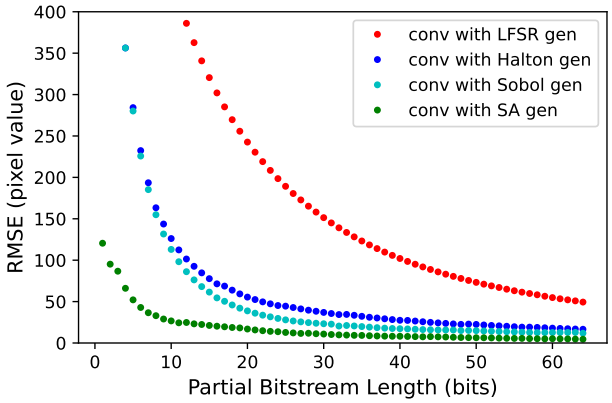

Fig. 7: RMSE of 2D convolution across 6 different images.

generator. Fig. 6 shows the accuracy of the multiplication at different early-termination points. The accuracy values plotted represent the average accuracy results from multiplying 10k random samples of (X,Y) input value pairs. Across all early termination points, the uMUL-ST unit using an LFSR-based generator achieves the lowest accuracy out of the four designs, while the uMUL-ST unit using our SA generator achieves the highest accuracy. Not only does the uMUL-ST unit using the SA generator show faster convergence to a high accuracy result, it also produces higher final accuracy on average even without terminating early.

Application Evaluation. At the application level, we show the accuracy of a 2D convolution SC system. For this experiment, our SC system takes 6 different 256×256 grayscale images and convolves each with a 3×3 weight filter that represents a Gaussian blur. All multiplications are implemented with uMUL-ST, and their products are accumulated in a parallel adder to sum up the result of nine products for each output pixel. The root-mean-square error (RMSE) with respect to a baseline computation using traditional floating-point numbers is computed at each early termination point, and the geomean across all six images for each of the different generators are shown in Fig. 7. Across all early termination points, the implementation using our SA generator achieves the lowest RMSE, while the implementation using the LFSR-based generator has the highest RMSE. This demonstrates the practical advantage of our SA bitstream generator for enabling efficient early termination in SC systems.

VI. Conclusion

In this work, we introduce streaming accuracy, a new metric that measures how close a bitstream resembles its streaming-accurate form. We show that it overcomes limitations of prior metrics that make them difficult to use when considering early termination in a generalized manner, at arbitrary termination points. We also introduce the design of a new SC bitstream generator that is guaranteed to produce bitstreams in their streaming-accurate form, and show that it improves accuracy of SC circuits compared with using an LFSR-based generator or a Halton-based generator. With SC's unique tradeoffs between area, accuracy and latency, this enhanced ability to reason about early termination can be valuable for designers to better optimize their SC systems and explore the complex design space.

References

[1] A. Alaghi and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–6.

[2] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–4.

[3] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "Vlsi implementation of deep neural network using integral stochastic computing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 10, pp. 2688–2699, 2017.

[4] T. Chen and J. P. Hayes, "Design of division circuits for stochastic computing," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 116–121.

[5] Q. T. Dong, M. Arzel, C. Jego, and W. J. Gross, "Stochastic decoding of turbo codes," *IEEE Transactions on Signal Processing*, vol. 58, no. 12, pp. 6421–6425, 2010.

[6] D. Fick, G. Kim, A. Wang, D. Blaauw, and D. Sylvester, "Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2d edge detection and noise filtering," in *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, Sep. 2014, pp. 1–4.

[7] B. R. Gaines, "Stochastic computing," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 149–156. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465505

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

[9] H. Hsiao, J. San Miguel, Y. Hara-Azumi, and J. Anderson, "Zero correlation error: A metric for finite-length bitstream independence in stochastic computing," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 260–265. [Online]. Available: https://doi.org/10.1145/3394885.3431552

[10] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, Oct 2011, pp. 154–161.

[11] S. Liu and J. Han, "Energy efficient stochastic computing with sobol sequences," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 650–653.

[12] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, "Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, 2017, p. 405–418.

[13] H. Sim and J. Lee, "Log-quantized stochastic computing for memory and computation efficient dnns," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 2019, p. 280–285.

[14] J. E. Stine *et al.*, "FreePDK: An open-source variation-aware design kit," in *IEEE MSE*, 2007, pp. 137–174.

[15] S. S. Tehrani, W. J. Gross, and S. Mannor, "Stochastic decoding of ldpc codes," *IEEE Communications Letters*, vol. 10, no. 10, pp. 716–718, 2006.

[16] J. von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components," in *Automata Studies*, C. Shannon and J. McCarthy, Eds. Princeton University Press, 1956, pp. 43–98.

[17] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. San Miguel, "uGEMM: Unary Computing Architecture for GEMM Applications," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2020.

[18] D. Wu and J. San Miguel, "In-stream stochastic division and square root via correlation," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[19] D. Wu, R. Yin, and J. San Miguel, *Normalized Stability: A Cross-Level Design Metric for Early Termination in Stochastic Computing*. New York, NY, USA: Association for Computing Machinery, 2021, p. 254–259. [Online]. Available: https://doi.org/10.1145/3394885.3431549