In-Stream Stochastic Division and Square Root via Correlation

Di Wu
University of Wisconsin-Madison
Madison, WI, USA
di.wu@ece.wisc.edu

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

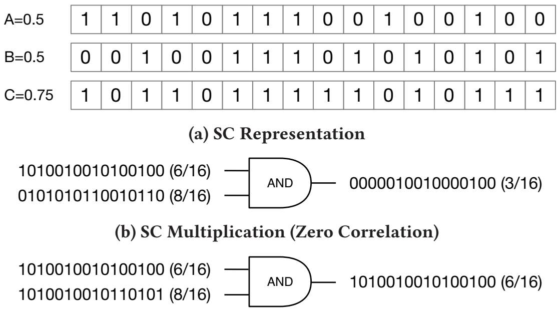
ABSTRACT

Stochastic Computing (SC) is designed to minimize hardware area and power consumption compared to traditional binary-encoded computation, stemming from the bit-serial data representation and extremely straightforward logic. Though existing Stochastic Computing Units mostly assume uncorrelated bit streams, recent works find that correlation can be exploited for higher accuracy. We propose novel architectures for SC division and square root, which leverage correlation via low-cost in-stream mechanisms that eliminate expensive bit stream regeneration. We also introduce new metrics to better evaluate SC circuits relying on equilibrium via feedback loops. Experiments indicate that our division converges 46.3% faster with both 43.3% lower error and 45.6% less area.

1 INTRODUCTION

Stochastic Computing (SC) was first proposed in 1969 [17] as a bit-serial solution for machine learning and pattern recognition, which require massive but often redundant inputs. SC data in its unipolar format generally is a Bernoulli Sequence, which is a *Bit Stream (BS)* containing uniformly distributed 0s and 1s. Its value and precision are determined by the ratio of 1s and the BS length, respectively, regardless of the element order. Fig. 1a shows an example of SC data representation. A represents the value 0.5, which is equal to the probability that each bit in BS A is 1. Note that A and B represent the same value despite unmatched bits of 1s in their sequences. Thanks to this bit-serial representation, SC can use simple logic on the input bits to calculate the output bits. Fig. 1b shows that SC multiplication can be performed with a single AND gate. Compared to binary-encoded multiplication, SC can reduce power consumption by orders of magnitude. Due to this high efficiency, SC has recently grown in interest in a wide range of domains, including Error Correcting Codes [13][4], Computer Vision [9] and Deep Learning [3]. *SC is a promising design paradigm for emerging devices that require extremely low power and area.*

However, SC's simple hardware comes at a cost of increased computation latency and non-deterministic accuracy. For example, SC multiplication (Fig. 1b) requires N cycles to multiply N-bit inputs, which is considerably more cycles than in binary multipliers. Moreover, the final result may not be consistent, since the output of an SC computation is dependent on the amount of *correlation* between the input BSs (i.e., how aligned the 1s and 0s are). Fig. 1c shows how high positive correlation[1] affects the output of the AND gate. Though the output deviates from the correct multiplication result, the insight is that the AND gate can now serve as a MIN operation of the inputs. *Instead of viewing correlation as a detriment to SC accuracy, we leverage correlation to design more accurate and efficient SC circuits for complex operations: division and square root.*



A=0.5	1	1	0	1	0	1	1	1	0	0	1	0	0	1	0	0
B=0.5	0	0	1	0	0	1	1	1	0	1	0	1	0	1	0	1
C=0.75	1	0	1	1	0	1	1	1	1	1	0	1	0	1	1	1

(a) SC Representation

(b) SC Multiplication (Zero Correlation)

(c) SC Minimum (High Positive Correlation)

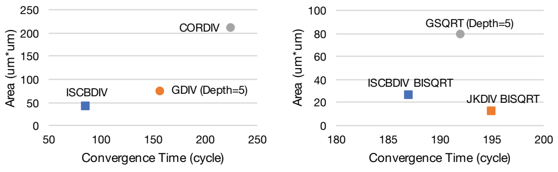
Figure 1: SC Paradigm

To achieve high accuracy in SC, researchers have proposed varying approaches. Most existing works focus on *BS generation*, a vital but expensive component of SC circuits. BS generation involves producing a random sequence of numbers and comparing them each to an input value to generate the Bernoulli Sequence [17]. To maximize accuracy, traditional techniques strive to develop high-quality Random Number Generators (RNGs) that exhibit zero correlation. Example works include low-discrepancy sequences [12] and bit scrambling [8][5]. Other researchers aim to design high-accuracy Stochastic Computing Units (SCUs) under the assumption that correlation of the inputs is zero [17][10]. However, this assumption is often difficult to maintain when computations require long chains of SCUs [4]. *BS regeneration* is necessary to retain zero correlation between intermediate SCUs [14][16], which is costly to performance, area and power. Only recently have researchers shown that it is possible to leverage correlation [15]. In this vein, *Synchronizers* have been recently proposed to enable designers to manipulate the correlation of input BSs [16]. *Our work leverages correlation to design SCUs for division and square root that are in-stream (i.e., do not require expensive BS regeneration to achieve high accuracy).*

New Correlation-Based SCUs. We propose *In-Stream Correlation-Based Division (ISCBDIV)* and *Bit-Inserting Square Root (BISQRT)*, novel in-stream architectures that exploit correlation for unipolar SC. We identify the fundamental inefficiencies of existing stochastic designs for division and square root, two important nonlinear SC operations used in many applications (e.g., machine

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317844

[1] In this example, correlation refers to the Stochastic Cross Correlation (SCC) between two BSs [1].



(a) Division

(b) Square Root

Figure 2: Comparison of Proposed SCUs to Counterparts

learning, graph processing, computer vision). Compared to linear operations [7][11], state-of-the-art implementations of nonlinear SC operations mostly follow the classical Gaines designs [17]. We introduce a fundamentally new correlation-based approach. Inspired by CORDIV [15], our proposed ISCBDIV leverages correlation to achieve higher accuracy compared to traditional uncorrelated SCUs [17]. Our insight is that for the majority of use cases, the divisor is generally larger than the dividend, so that the quotient does not exceed the legal SC data range (i.e., values from 0 to 1); otherwise the quotient saturates at 1. With this, ISCBDIV saves significant area by eliminating much of the logic overhead in CORDIV for BS regeneration and replacing it with our very simple in-stream mechanism: *Skewed Synchronizer*. In a similar vein, BISQRT leverages the insight that the result of any SC square root operation is always larger than its input. We introduce a new low-cost mechanism for inserting 1s into the input BS, thus eliminating BS regeneration in the standard Gaines design [17].

New Accuracy Metrics for Feedback-Based SCUs. We also introduce new metrics for evaluating SC accuracy in terms of correlation, bias and convergence time, since existing metrics are not well suited for SC circuits that rely on equilibrium via feedback loops.[2] In such SCUs, it takes a significant amount of time for the circuit to reach equilibrium and thus produce a stable output BS; until then, the output bits are highly fluctuating and are not yet representative of the computation's result. However, existing metrics consider accuracy with respect to the entire output BS. We argue that this yields misleading evaluations of SCU accuracy and thus propose new accuracy metrics for feedback-based designs.

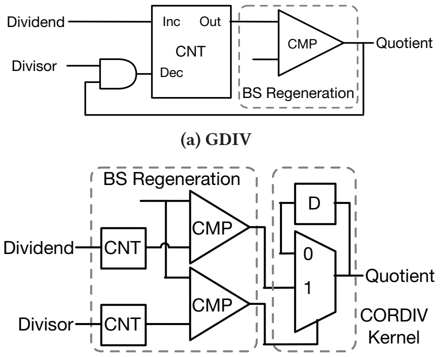
Summary of Results. We evaluate the accuracy, area, latency and power of our proposed division and square root SCUs against other state-of-the-art approaches. As shown in Fig. 2, our ISCBDIV and (two) BISQRT designs (rectangular points in figure) achieve the best tradeoff in convergence time and area savings.

2 BACKGROUND

Prior work on stochastic division start from Gaines' early proposals of SCUs [17] to more recent work on Correlated Division [15]. A high-accuracy design for division-based normalization has also been explored [6], though that is beyond the scope of this work. Unlike divider designs, there is little prior work on SC square root aside from Gaines' early proposals [17]. These classical techniques remain the state-of-the-art; our work sheds new insights on this little explored research area.

2.1 Gaines Division

Gaines Division (GDIV) [17] is shown in Fig. 3a. In this design, a feedback loop from the quotient to the divisor is used to reach an equilibrium between the increments and decrements (Inc and Dec in the figure) for the saturating counter (CNT in the figure) of depth N (i.e., N-bit). Equilibrium is reached when the probability of increment ($P_{Dividend}$) and decrement ($P_{Divisor} \times P_{Quotient}$) are equal. The quotient is then computed as: $P_{Quotient} = P_{Dividend}/P_{Divisor}$. The saturating counter can be initialized to half of the maximum count to reduce the latency to reach equilibrium. GDIV's limitations are threefold. First, its counter and comparator incur high area cost. Second, the precision of GDIV is bounded by the width of the counter. To achieve the same precision as the input, the counter width may have to be very large. Third, the final accuracy depends on the accuracy of the multiplication via the AND gate. Thus the Stochastic Cross Correlation (SCC) [1] between the divisor and quotient BSs needs to be close to zero to make the multiplication accurate. SCC relates to the ratio of paired 1s between two BSs. When the ratio of paired 1s is maximized, SCC is +1; otherwise, when minimized, SCC is −1.



(a) GDIV

(b) CORDIV

Figure 3: SC Division

2.2 Correlated Division

Correlated Division (CORDIV) leverages high positive Stochastic Cross Correlation (SCC) between the dividend and divisor [15]. It hinges on two assumptions: 1) the dividend is always smaller than the divisor, and 2) the correlation between the dividend and divisor BSs at the CORDIV Kernel (Fig. 3b) is maximized. The first assumption is practical in SC; larger dividends tend to be avoided since their quotients are likely to saturate at 1. The output $P_{Quotient} = P_{Dividend}/P_{Divisor}$ is equal to the ratio of the number of 1s in the dividend and divisor BSs: $N^1_{Dividend}/N^1_{Divisor}$.

The CORDIV architecture is shown in Fig. 3b. CORDIV has a Counter-RNG-Comparator organization to produce the positively correlated BSs for the following kernel. Thus whenever the kernel dividend bit is 1, the kernel divisor bit will likely also be 1. This allows the kernel MUX to capture all $N^1_{Dividend}$ bits. Then when the divisor bit is 0, the kernel D-flip-flop (D-FF) generates a quotient bit with a similar probability to when the divisor bit is 1. After values in saturating counters for BS regeneration become stable, CORDIV Kernel has higher accuracy than GDIV.

There are two problems for the primitive CORDIV. The first is that BS regeneration introduces significant latency overhead to reach a stable and accurate output. Provided that the initial value of 8-bit dividend saturating counter is 128, representing 0.5, and that the real dividend value is 0.75, the counter requires average 128

[2] Feedback loops are usually constructed with saturating counters [17].

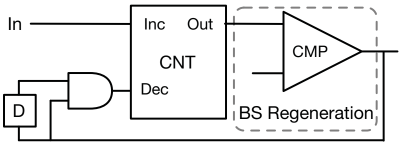


Figure 4: Gaines Square Root

cycles to reach precise 0.75, calculated as the total value increase by the average value increasing speed, i.e., $2^8 * (0.75 - 0.5) * 0.5$. Thus the quotient will be inaccurate for a long time before stabilization. The second problem is that BS regeneration brings about significant hardware overhead and limits extensive parallel deployment.

2.3 Gaines Square Root

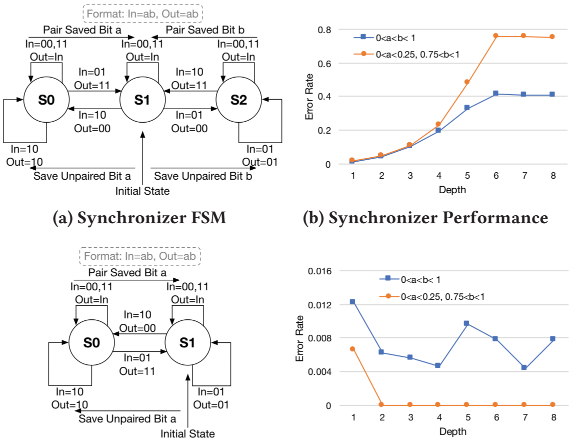
Gaines Square Root (GSQRT) is the standard SC design for square root operations [17]. As shown in Fig. 4, GSQRT is derived from GDIV. The key difference is that the Depth-N saturating counter is decremented based on the square of the output. Thus at equilibrium, the probability of increment (P_{In}) equals the probability of decrement (${P_{Out}}^2$), leading to the square root function: $P_{Out} = \sqrt{P_{In}}$. GSQRT has similar limitations as GDIV, though now the accuracy of the multiplication (AND gate) relies on low Stochastic Auto Correlation (SAC) [5] in the output BS, which is the correlation of the BS and its shifted version.

2.4 BS Generation and Regeneration

Although the accuracy increases with different techniques, problems related to latency and hardware cost arise. As uncorrelated BSs from BS generation [17] pass through long chains of SCUs, the no-correlation assumption may be broken, leading to lower accuracy and longer latency [4]. To deal with these problems, existing works perform BS regeneration for the intermittent BSs [14][16]. BS regeneration is similar to BS generation for the source input: they both compare the buffered binary data values to RNG outputs. BS regeneration uses counters to dynamically calculate values for intermittent BSs before comparison, while BS generation from source uses static pre-stored values. We refer to SCUs that require no combination of Counter-RNG-Comparator for BS regeneration as *In-Stream* SCUs. In-Stream SCUs are generally more hardware-efficient.

3 IN-STREAM CORRELATION-BASED DIVISION

This section presents our *In-Stream Correlation-Based Division (ISCBDIV)*, which is inspired by CORDIV [15]. CORDIV includes an expensive Counter-RNG-Comparator organization to regenerate correlated data for the CORDIV Kernel in Fig. 3b. To alleviate such performance and hardware overheads, we propose an in-stream mechanism (i.e., without BS regeneration) that constructs maximally correlated BSs. Synchronizer [16] is a recent technique for increasing Stochastic Cross Correlation (SCC) of two input BSs by reordering the bits to favor 1-1 pairs and suppress 1-0 or 0-1 pairs. When the two-port input is a 1-0 or 0-1 pair, Synchronizer's finite state machine uses two Depth-N (i.e., N-bit) counters (one for each input) to record the bit-1 in one input and try to match it with a later bit-1 in the other input. An example Synchronizer FSM with $N = 1$ is shown in Fig. 5a, in which bits a and b are from two BSs a and b. Given a less than b, Fig. 5b shows the L1 norm error rate with varying depth N across 10000 runs. The figure shows that the synchronizer has a lower bound on error rate and that increasing N leads to lower accuracy.



(a) Synchronizer FSM

(b) Synchronizer Performance

(c) Skewed Synchronizer FSM

(d) Skewed Synchronizer Performance

Figure 5: Synchronizer and Skewed Synchronizer

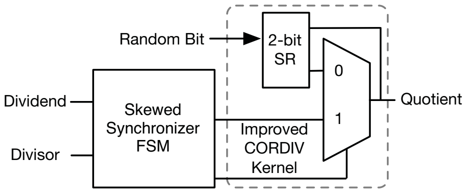
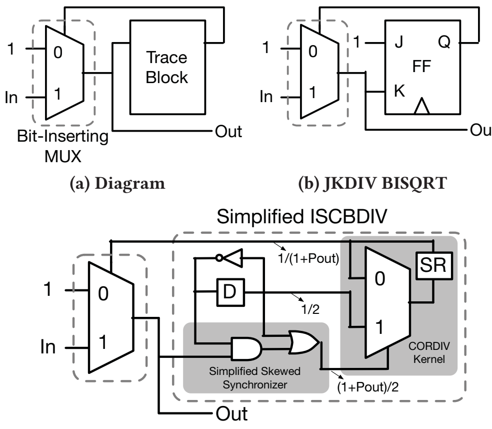


Figure 6: ISCBDIV

To achieve both high correlation and high accuracy, we design a *Skewed Synchronizer* to maximally correlate the BSs to the CORDIV Kernel in Fig. 3b. The Skewed Synchronizer leverages the assumption that the divisor is always larger than the dividend and thus only needs to reorder the dividend BS. The Skewed Synchronizer employs one Depth-N (N-bit) counter to record the 1s in the dividend when the divisor bit is 0, and then matches the saved 1 with a later 1 in the divisor BS. The FSM for the Skewed Synchronizer with $N = 1$ is shown in Fig. 5c. Bits a and b represent the bits from the smaller and larger BS, respectively. Fig. 5d shows the L1 norm error rate with varying depth N across 10000 runs. Increasing N improves the output accuracy in both cases in the figure. Specifically, when the quotient is small ($0 < a < 0.25$, $0.75 < b < 1$), the accuracy is even higher, close to 100%. This property comes from the fact that the larger BS has more 1s. Thus a 2-bit counter is sufficient to record unpaired 1s in the small BS. On the other hand, the original Synchronizer has to track 1s in both BSs and potentially trap 1s from the larger BS in the counter when bits in the smaller BS are logic 1, leading to a higher error rate (Fig. 5b).

The overall hardware architecture of the proposed ISCBDIV is presented in Fig. 6, consisting of the Skewed Synchronizer and the CORDIV Kernel. This design first converts arbitrary input BSs (with arbitrary SCC) to positively correlated BSs via the Skewed Synchronizer. During the conversion, the divisor BS is unchanged, but the dividend bits are reordered. Then an in-stream division is performed by the CORDIV Kernel. The D-FF in Fig. 3b is substituted with a 2-bit Shift Register (SR) to improve performance. This



(a) Diagram **(b) JKDIV BISQRT**

(c) ISCBDIV BISQRT

Figure 7: BISQRT Architecture

architecture only requires extra logic for the Skewed Synchronizer and to generate a single random bit. ISCBDIV is significantly more hardware-efficient than CORDIV (Fig. 3b), whose overhead from regenerating BSs includes the registers to store the binary numbers, the RNG and the two comparators.

4 BIT-INSERTING SQUARE ROOT

GSQRT calculates the square root at equilibrium via the simple equation $P_{In} = P_{Out}{}^2$, but it requires a saturating counter and a comparator to regenerate the BS (Fig. 4). To alleviate this overhead, we propose SC square root via correlation. Our *Bit-Inserting Square Root (BISQRT)* is based on the observation that the output of SC square root is always larger than the input. Thus intelligently inserting 1s into the input BS is sufficient for producing the correct output. Our design is shown in Fig. 7a. The Bit-Inserting MUX and the Trace Block constitute a feedback loop. Provided that the Trace Block has an output probability P_{Trace} of $1/(1 + P_{Out})$, the correctness of BISQRT can be verified with Equation 1. In this work, we introduce two design options to obtain the required P_{Trace}.

$$\begin{aligned} P_{Out} &= 1 \times (1 - P_{Trace}) + P_{In} \times P_{Trace} \\ &= 1 \times (1 - P_{Trace}) + P_{Out}{}^2 \times P_{Trace} \\ &= 1 - (1 - P_{Out}) \times (1 + P_{Out}) \times P_{Trace} \\ &= 1 - (1 - P_{Out}) \times (1 + P_{Out})/(1 + P_{Out}) \end{aligned} \tag{1}$$

4.1 JKDIV-Based Trace Block

JK Flip Flop (JKFF) implements the JKFF Division (JKDIV), given by $P_Q = P_J/(P_J + P_K)$ [17]. Thus setting port J to 1 results in an output probability of $P_Q = 1/(1 + P_K)$, which satisfies our desired format for P_{Trace}. From this, we build *JKDIV BISQRT*, shown in Fig. 7b, with port K of JKDIV connected to the output.

4.2 ISCBDIV-Based Trace Block

Alternatively, we can implement $P_{Trace} = 1/(1 + P_{Out})$ using a simplified ISCBDIV. This *ISCBDIV BISQRT* architecture is shown in Fig. 7c. In this design, the Skewed Synchronizer is further simplified to one AND gate and one OR gate. The dividend (MUX input port 1 in CORDIV Kernel) is a periodic BS with a probability of $1/2$, generated by the D-FF and inverter. The divisor $(1 + P_{Out})/2$ (MUX selection port in CORDIV Kernel) is generated via correlation with the Simplified Skewed Synchronizer. An important difference between original ISCBDIV and this simplified version is that the latter's output comes from the SR instead of the kernel MUX, which helps to reduce the absolute value of the correlation between input and output of Bit-Inserting MUX for the sake of output accuracy.

5 HARDWARE IMPLEMENTATION

Designs are synthesized with the Synopsys Design Compiler at $400MHz$ with TSMC $45nm$ technology. The *Area*, *Power*, *Latency*, and *Throughput Per Area (TPA)* results are summarized in Table 1. The latency here refers to 5% *Convergence Time* in Sec. 6.1. The TPA is defined as $frequency/(latency \times area)$ to estimate the computational efficiency of SCUs [12]. Note that the logic for stochastic-to-binary conversion at the output of SC systems is not considered here, as output nodes occupy less in real applications than the internal computational SCUs. The best performing divider and square root designs are bolded per column. We find that our designs simultaneously achieve less area, less power and higher throughput, making them strong candidates for low-power SC devices.

Table 1: Hardware Implementation

Design	*Area* (μm^2)	*Power* (μW)	*Latency* ($cycles$)	*TPA* ($1/(\mu m^2 \cdot s)$)
GDIV(Depth-5)	74.3	21.0	158	34,073
CORDIV	211.2	60.9	226	8,384
Proposed ISCBDIV	**40.4**	**12.5**	**86**	**115,128**
GSQRT(Depth-5)	78.3	23.5	192	26,607
Proposed JKDIV BISQRT	**11.3**	**6.3**	195	**181,529**
Proposed ISCBDIV BISQRT	25.4	12.6	**187**	84,214

6 PERFORMANCE ANALYSIS

Before evaluating the performance of our proposed SCUs, we first introduce new evaluation metrics to more systematically compare SC performance when considering SC circuits that need to reach equilibrium (due to feedback loops).

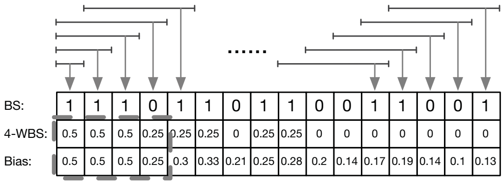
6.1 Evaluation Metrics For SC Performance

Table 2 lists our new evaluation metrics as well as existing metrics for correlation. We propose the *Window Bias (WBS)*, *Average Window Bias (AWBS)* and *p% Convergence Time (p% CTime)* to accurately evaluate SCUs with feedback.

The *Window Bias* is the accumulated error of a window in the SC BS. In existing literature, the error rate is calculated with respect to the entire BS, which is practical for simple operations like multiplication with an AND gate. However, for SCUs with feedback loops—usually constructed with saturating counters, as in GDIV and GSQRT—an equilibrium is required to produce a stable output

Table 2: Evaluation Metrics

Metric	Definition
SCC	Stochastic Cross Correlation for 2 BSs [1].
SAC	Stochastic Auto Correlation for 1 BS [5].
WBS	N-Window Bias: accumulative error for the most recent N bits; ranging from -1 to 1; expecting 0 for no error.
AWBS	Average N-Window Bias: statistical root-mean-square of multiple N-WBS values.
p% CTime	p% Convergence Time: cycle count required for SCU to achieve a stable N-WBS of less than p%; smaller is better.

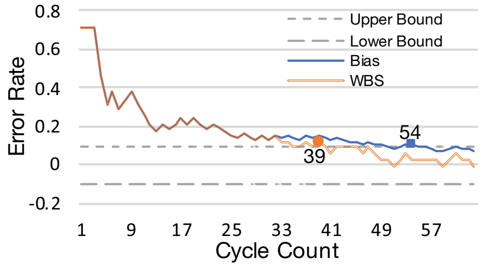


4-WBS for a 16-bit BS of value 0.5

Figure 8: Example of WBS

BS. Before equilibrium, the SCU's accuracy is still fluctuating and unstable; thus computing error rate across the entire BS unfairly quantifies the accuracy of the SCU. We propose the WBS metric to better evaluate error rates when considering equilibrium. Fig. 8 shows how to calculate the 4-WBS for a 16-bit BS with a value of 0.5 and compares the 4-WBS with the conventional error rate metric, labelled as "Bias". At each point in the BS, WBS is equal to the observed probability within the last 4-bit window minus the true value of the entire BS (0.5 in this example). Beyond the fourth bit of the BS, 4-WBS and Bias diverge. Extending this, the *Average Window Bias* is the root-mean-square of the WBS for a SCU across all input BSs and can help evaluate the average statistical accuracy.

Based on the WBS, we propose the *p% Convergence Time (CTime)* to evaluate how fast a computation can obtain a result within a bounded WBS pair, for enabling early termination [4]. It is related in part to the *Progressive Precision* [2], which defines an upper bound of the incremental error rate for a SC BS. As SC generally requires significantly more cycles to compute results than binary-encoded computing, it is worth defining a CTime metric to evaluate SC latency, particularly for SCUs with feedback loops. The p% CTime is illustrated in Fig. 9 with $p = 10$. The two horizontal dash lines indicate the upper and lower bounds for the error rate (±10%). The round and the rectangular dots are the watershed points at which the WBS and Bias, respectively, start to fluctuate stably between +10% and −10%. Thus the 10% CTime is the cycle count from the starting point to the watershed. In Fig. 9, WBS has 10% CTime of 39 cycles, with 54 for Bias, indicating that WBS uncovers more opportunity for earlier termination than Bias. CTime is a more effective metric particularly on counter-based designs.



10% CTime for 5-bit GDIV

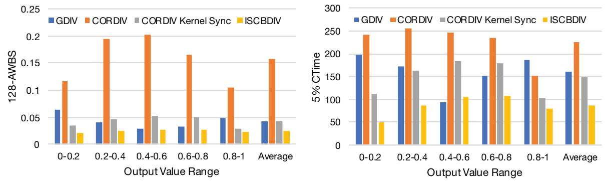
Figure 9: Example of CTime

6.2 Experimental Setup

The SC performance (AWBS and CTime) varies due to three factors. First, the type of RNG matters. We choose Sobol RNGs, which are superior to LFSR RNGs [12]. Second, BS correlation also impacts the result as shown in Fig. 1. In the following sections, we evaluate the performance with 256-bit BSs at different output ranges (i.e., range of values of result), as the input correlation may not change drastically with the BS length but rather with the output value range. Finally, the level of tolerable error is important. We choose the window size N always to be half of the period for Sobol RNGs, and $p = 5$ for CTime. Note that the window size is selected to ensure that GDIV and GSQRT with stable inputs can always converge after N cycles.

6.3 Performance of ISCBDIV

Using the metrics from Table 2, Fig. 10 shows the performance comparison of SC divisors with 10000 different Sobol RNGs. The evaluated candidates include GDIV, CORDIV, CORDIV Kernel with Synchronizer (labelled as "CORDIV Kernel Sync"), and the proposed ISCBDIV. The horizontal axis contains 5 non-overlapped output value ranges across all RNGs (e.g., the "0-0.2" category is the average for all output values within 0 and 0.2 for all RNGs), as well as their average. In terms of both AWBS and CTime, the proposed ISCBDIV outperforms all others. Compared to GDIV, the average improvements in AWBS and CTime of ISCBDIV across the output ranges are 43.3% and 46.3%, respectively. Furthermore, our design is 84.4% and 61.9% better than CORDIV in AWBS and CTime, as we are designing for in-stream, while CORDIV needs BS regeneration whose latency overhead leads to performance degradation. Note that the comparison to CORDIV Kernel with Synchronizer demonstrates the high effectiveness of the proposed Skewed Synchronizer.



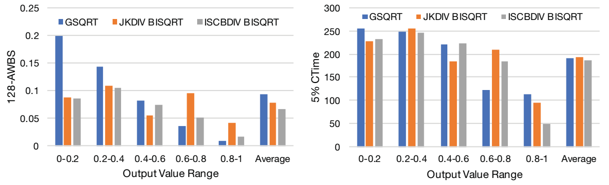
(a) 128-AWBS

(b) 5% CTime

Figure 10: ISCBDIV Performance

6.4 Performance of BISQRT

The proposed BISQRT implementation (Fig. 7a) utilizes a Bit-Inserting MUX and a Trace Block. The MUX fixes port 0 to 1 for inserting 1s into the input BS, and the Trace Block produces an output probability of $1/(1 + P_{Out})$. Fig. 11 shows the overall performance of GSQRT and our two versions of BISQRT. Compared to GSQRT, JKDIV BISQRT and ISCBDIV BISQRT converge within a similar average time but are 16.8% and 29.0% lower in AWBS. Compared to JKDIV BISQRT, ISCBDIV BISQRT is 14.6% and 4% better in AWBS and CTime.

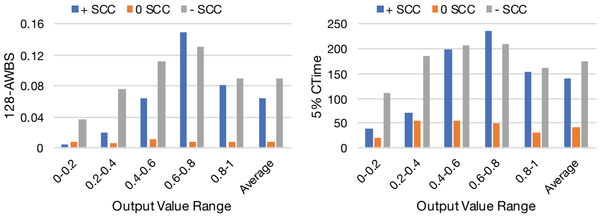


(a) 128-AWBS

(b) 5% CTime

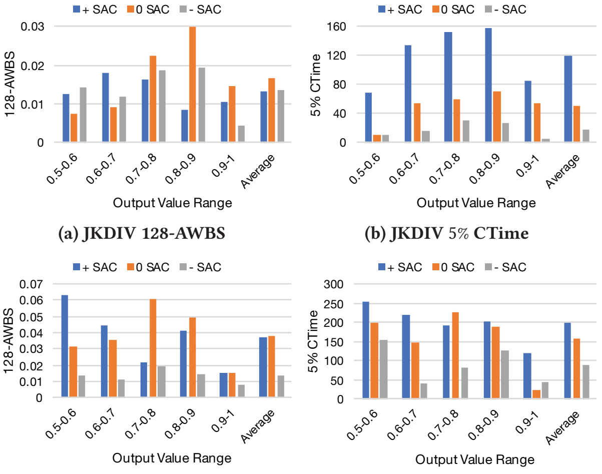
Figure 11: BISQRT Performance

We further analyze how the SAC and SCC contributes to performance. According to Table 2, we first evaluate how the SCC between port 1 and the selection port impacts the AWBS and CTime of a single MUX and how the SAC of the output BS influences the individual Trace Block, shown in Fig. 12 and 13, respectively. Based on



(a) MUX 128-AWBS (b) MUX 5% CTime

Figure 12: MUX Performance



(a) JKDIV 128-AWBS (b) JKDIV 5% CTime

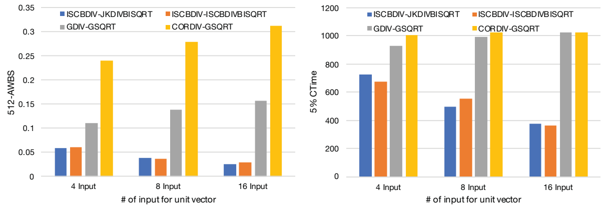
(c) Simplified ISCBDIV 128-AWBS (d) Simplified ISCBDIV 5% CTime

Figure 13: Trace Block Performance

Fig. 12a and 12b, the ideal configuration of MUX is to achieve SCC of zero; a negative SCC is the worst case on average. For JKDIV (in Fig. 13a and 13b) and simplified ISCBDIV (in Fig. 13c and 13d) based Trace Blocks, a negative SAC is favorable on average for both AWBS and CTime. In actual design, the MUX in JKDIV BISQRT has a consistent SCC of −1 across the entire output range (the worst case for MUX as claimed above), while that in ISCBDIV BISQRT has an average SCC of 0.03. For the Trace Block, SAC values are 0.11 and 0.07 in JKDIV and ISCBDIV based versions. Thus, MUX and Trace Block in ISCBDIV BISQRT achieve better SCC and SAC, leading to superior performance compared to JKDIV BISQRT in Fig. 11.

6.5 Case Study: SC Unit Vector

We implement a SC design for computing the in-stream unit vector of 1024-bit inputs. As shown in Fig. 14, this case study further evaluates the joint performance of our proposed SC division and square root. The AWBS and CTime of the SC unit vector are the average of the AWBS and CTime for each output. Among different combinations, the design with our proposed ISCBDIV and BISQRT has the best performance. For a 16-input unit vector, our design achieves 84.2% lower AWBS and 63.4% lower CTime than that with the Gaines designs. We find that CORDIV performs worst, as it takes hundreds of cycles to convert a BS to binary format before reaching equilibrium. Note that since the output of ISCBDIV is far more stable with lower error (Fig. 10), adding more inputs reduces the frequency of large errors, thus improving AWBS and CTime.



(a) 512-AWBS (b) 5% CTime

Figure 14: Unit Vector Performance

7 CONCLUSION

In this paper, we highlight the existing hardware overheads of two nonlinear SC operations—division and square root—and present more efficient designs—*ISCBDIV* and *BISQRT*—that leverage correlation. We also introduce new SC evaluation metrics to better analyze the performance of SC designs that need equilibrium via feedback loops. We evaluate our proposed designs and compare them against existing work. We observe that our ISCBDIV and BISQRT converge faster with lower error rates and less area. An additional case study with a SC unit vector further demonstrates the benefit of our designs, enhancing power-efficiency while maintaining high accuracy.

ACKNOWLEDGMENTS

We thank reviewers for their valuable feedback. This work is supported by the Wisconsin Alumni Research Foundation and the University of Wisconsin-Madison.

REFERENCES

[1] A. Alaghi et al. 2013. Exploiting correlation in stochastic circuit design. In *ICCD*.

[2] A. Alaghi et al. 2014. Fast and accurate computation using stochastic circuits. In *DATE*.

[3] A. Ren et al. 2017. SC-DCNN: Highly-Scalable Deep Convolutional Neural Network Using Stochastic Computing. In *ASPLOS*.

[4] D. Wu et al. 2016. Strategies for Reducing Decoding Cycles in Stochastic LDPC Decoders. *IEEE Transactions on Circuits and Systems II: Express Briefs* 63, 9 (Sep 2016), 873–877.

[5] F. Neugebauer et al. 2017. Building a Better Random Number Generator for Stochastic Computing. In *DSD*.

[6] K. Han et al. 2018. A Fast Converging Normalization Unit for Stochastic Computing. *IEEE Transactions on Circuits and Systems II: Express Briefs* 65, 4 (Apr 2018), 501–505.

[7] K. Kim et al. 2015. Approximate de-randomizer for stochastic circuits. In *ISOCC*.

[8] K. Kim et al. 2016. An energy-efficient random number generator for stochastic circuits. In *ASP-DAC*.

[9] M. H. Najafi et al. 2016. A Fast Fault-Tolerant Architecture for Sauvola Local Image Thresholding Algorithm Using Stochastic Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 2 (Feb 2016), 808–812.

[10] M. Parhi et al. 2015. Effect of bit-level correlation in stochastic computing. In *DSP*.

[11] P. Li et al. 2012. The synthesis of linear Finite State Machine-based Stochastic Computational Elements. In *ASP-DAC*.

[12] S. Liu et al. 2017. Energy efficient stochastic computing with Sobol sequences. In *DATE*.

[13] S. S. Tehrani et al. 2006. Stochastic decoding of LDPC codes. *IEEE Communications Letters* 10, 10 (Oct 2006), 716–718.

[14] T. H. Chen et al. 2014. Analyzing and controlling accuracy in stochastic circuits. In *ICCD*.

[15] T. H. Chen et al. 2016. Design of Division Circuits for Stochastic Computing. In *ISVLSI*.

[16] V. T. Lee et al. 2018. Correlation manipulating circuits for stochastic computing. In *DATE*.

[17] B. R. Gaines. 1969. Stochastic computing systems. In *Advances in information systems science*. Springer, 37–172.