The Hypergeometric Distribution as a More Accurate Model for Stochastic Computing

Timothy J. Baker and John P. Hayes
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, 48109 USA
{bakertim, jhayes}@umich.edu

Abstract— **A fundamental assumption in stochastic computing (SC) is that bit-streams are generally well-approximated by a Bernoulli process, i.e., a sequence of independent 0-1 choices. We show that this assumption is flawed in unexpected and significant ways for some bit-streams such as those produced by a typical LFSR-based stochastic number generator (SNG). In particular, the Bernoulli assumption leads to a surprising overestimation of output errors and how they vary with input changes. We then propose a more accurate model for such bit-streams based on the hypergeometric distribution and examine its implications for several SC applications. First, we explore the effect of correlation on a mux-based stochastic adder and show that, contrary to what was previously thought, it is not entirely correlation insensitive. Further, inspired by the hypergeometric model, we introduce a new mux tree adder that offers major area savings and accuracy improvement. The effectiveness of this study is validated on a large image processing circuit which achieves an accuracy improvement of 32%, combined with a reduction in overall circuit area.**

Keywords—Approximate computing, stochastic computing, hypergeometric distribution, error analysis

I. Introduction

Stochastic computing (SC) [2][6] is a paradigm of approximate computing that employs pseudo-random bit-streams known as stochastic numbers (SNs) to perform computation. SC has attracted attention recently because it offers large area and power reduction for applications that require massive amounts of computation and are error-tolerant. Examples include image processing [12], digital filter design [10] and artificial neural networks [4][5]. The major challenges of SC are managing accuracy and computation time, and generating SNs efficiently.

In the unipolar format, the value X of an N-bit SN $\mathbf{X}$ is p_x, the probability that any bit of $\mathbf{X}$ is a 1. A unipolar SN, therefore, takes values in the unit interval $[0,1]$, and X can be estimated by dividing the number of 1s in $\mathbf{X}$ by N. For computations that require negative values, the bipolar format is used. In this case, X is interpreted as $2p_x - 1$, so SNs take values in the $[-1,1]$ interval. Arithmetic on SNs can be performed using very small circuits. For example, Fig. 1a shows a multiplier circuit for two unipolar stochastic numbers. Here, two SN generators (SNGs) are used to produce SNs $\mathbf{X}$ and $\mathbf{Y}$ which are inputs to an AND gate with output $\mathbf{Z}$. Probability theory tells us that if the bits of $\mathbf{X}$ and $\mathbf{Y}$ are uncorrelated, then $p_z = p_x \times p_y$ which implies that $Z = X \times Y$. Thus, the AND gate serves as a multiplier for unipolar SNs. As Fig. 1c shows, the AND's complexity is greatly exceeded by that of its SNGs.

Due to the inherent randomness of SC, the measured value of Z tends to fluctuate around its expected value of $X \times Y$, so SC is most suited to applications that tolerate small errors. Understanding how the expected error of a stochastic circuit varies with input value(s) and bit-stream length allows designers to answer questions such as: "Given a desired level of accuracy, what bit-stream length should be used?" Estimating the associated errors can be done analytically or through simulation [15], but the results can be quite different, as we now show.

Consider the stochastic multiplier in Fig. 1a with the SN lengths set to N. Assuming independence among the bits within and between $\mathbf{X}$ and $\mathbf{Y}$, as in the usual Bernoulli model of SNs, the standard deviation σ_Z of Z is

$$\sigma_Z = \sqrt{XY(1-XY)/N} \tag{1}$$

which is plotted as the yellow surface in Fig. 2a with $N = 255$ [4][15]. Using simulation, σ_Z can be estimated accurately as

$$\sigma_Z = \sqrt{\sum_{i=1}^{R} \frac{(Z_i - Z^*)^2}{R}} \tag{2}$$

where R is the number of simulation runs, Z_i is the measured output value during simulation run i, and $Z^* = XY$ is the expected output value. To achieve high accuracy, R was set to 10,000 and for each pair of values X, Y, we simulated the circuit with traditional SNGs built around linear feedback shift registers (LFSRs), as in Fig. 1c. The results are plotted as the blue surface in Fig. 2a. Despite the extremely small size of this circuit, Fig. 2a shows a large disagreement between theory and practice which stems from assumptions made about *LFSR SNs*, i.e., SNs generated by LFSR-based SNGs.

Disagreement between statistics derived from the Bernoulli model and from LFSR SNs has been noted before [10][14][16]. The goal of this work is to explore this disagreement in depth and develop a better model for LFSR SNs. As explained in Sec. III, the new model treats SN bits as trials of a hypergeometric random variable, in contrast to classic SC theory which treats

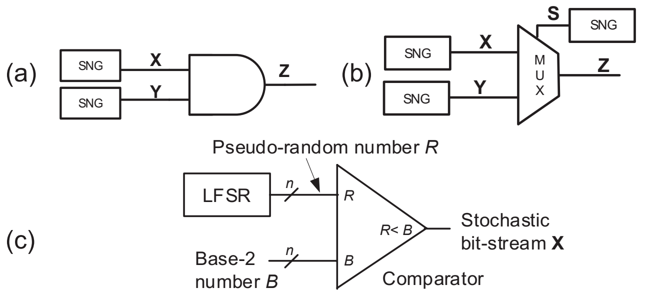

Fig. 1. Stochastic computing elements. (a) multiplier circuit; (b) adder circuit based on a multiplexer (mux); (c) stochastic number generator (SNG).

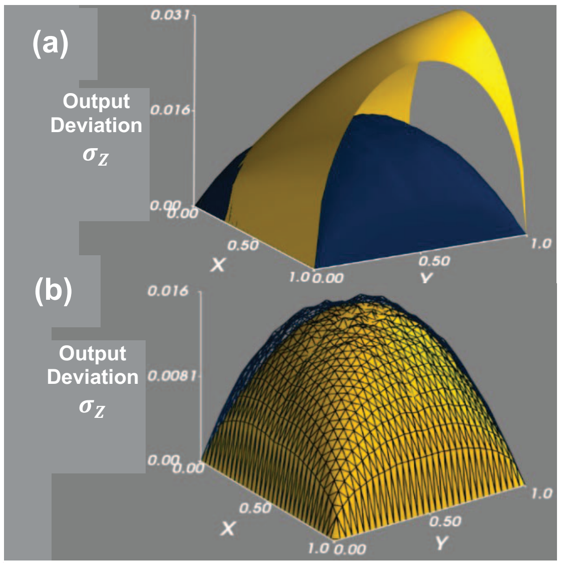

Fig. 2. Output deviation σ_z of the stochastic multiplier circuit plotted against input values X and Y: (a) theoretical σ_z assuming **X** and **Y** are Bernoulli SNs (yellow surface), and experimental σ_z from simulation (blue surface); (b) theoretical σ_z assuming **X** and **Y** are hypergeometric SNs (yellow surface) and experimental σ_z from simulation (blue mesh).

the bits as independent Bernoulli trials. Using this new model to estimate σ_Z for the stochastic multiplier circuit results in the yellow surface in Fig. 2b, which is a much better match for the simulation results shown as a blue mesh in Fig. 2b.

The main contributions of this work are:

- A more accurate model for LFSR-generated SNs that uses the hypergeometric distribution in place of the traditional Bernoulli model.
- An application of the hypergeometric model which shows that mux-based adders are not entirely correlation insensitive, as is commonly believed.
- Introduction of a new mux tree design inspired by the hypergeometric model that improves accuracy while reducing area compared to traditional designs.
- Validation of the preceding results on a large image processing circuit.

The paper is organized as follows: Sec. II gives background information on SC, while Sec. III introduces the hypergeometric model for LFSR SNs and applies it to various stochastic circuits. Sec. IV then validates our results through simulation of an SC application circuit, and Sec. V draws some conclusions.

II. Background

Next, we briefly review SC and its underlying assumptions.

A. Stochastic Computing Elements

The two most fundamental SC operations are multiplication and addition. Sec. I describes the stochastic multiplier shown in Fig. 1a. Addition between two SNs must either be scaled or saturated due to the confinement of (unipolar) SN values to the unit interval. Scaled addition is preferred and is typically implemented by a low-cost multiplexer (mux) circuit, as in Fig. 1b. S is set to 0.5 so that in each clock cycle, the mux propagates either a bit from **X** or a bit from **Y** with equal probability; hence $p_z = 0.5p_x + 0.5p_y$, i.e., $Z = 0.5(X + Y)$.

SNGs are used to convert numbers from the traditional binary domain of base-2 numbers to the SC domain. The most common SNG design is shown in Fig. 1c. It consists of an n-bit LFSR of maximum period [7] and a comparator. Over $2^n - 1$ clock cycles, the LFSR state traverses, in a pseudo-random order, all state values in the interval $[1, 2^n - 1]$. During each clock cycle, the LFSR state is compared with B to produce a bit of the output SN **X**. As the number of clock cycles increases, estimates for p_x tends towards $(B - 1)/2^n$. Hence, by specifying B, an SN with a desired value can be generated. Note that other types of SNGs such as SBoNG [16] are known which produce SNs with better randomness properties but at higher area cost; these are not considered in this paper.

The compactness of stochastic multipliers and adders offers an opportunity for massively parallel datapaths in applications that involve many multiplications and additions, such as matrix-vector products of the kind encountered in neural networks (NNs). However, the high cost of SN generation can offset the benefits of small arithmetic circuits, and long bit-streams may be required to reach an acceptable level of accuracy. SNG design and accuracy control are, therefore, two important and active areas of research in SC. For example, sharing RNSs between SNGs can reduce circuit area, but will correlate the generated SNs and generally lead to correlation-induced errors [1][16]. To mitigate such errors and maintain the benefits of sharing RNSs, techniques like circular shifting [10] have been developed to reduce correlation between SNs that share an RNS.

In some cases, circuit properties allow SNG sharing without introducing correlation errors. For example, it has been observed [1][10] that correlation between the data inputs to the mux adder does not affect output error. Thus, the SNGs for **X** and **Y** in Fig. 1b can share an RNS without causing inaccuracies. An intuitive explanation for this feature of the mux adder is that, in each clock cycle i, the output bit z_i is either x_i or y_i and there is no interaction between x_i and y_i. However, we will show that prior analysis does not hold under circumstances common to existing stochastic designs. In fact, we find that correlation between data inputs *can* influence the output error, and correlation can even be exploited to reduce the error of mux-based circuits.

B. Modeling Stochastic Numbers

Traditionally, SNs have been modeled as a series of Bernoulli trials [5][6]. We will refer to such SNs as *Bernoulli SNs*. This model is mathematically convenient and has led to many useful designs including sequential circuits that compute functions that are otherwise poorly approximated in the SC domain [5][13]. Some SNGs, such as SBoNG, produce SNs that are very well modeled as Bernoulli SNs.

However, the Bernoulli model fails to accurately describe some SNs such as LFSR SNs, or those produced from some sequential designs [5][16]. For LFSR SNs, the Bernoulli model leads to an overestimation of the output errors for combinational computing elements as illustrated by Fig. 2a. We will see in Sec. IV that this overestimation is easily overlooked in circuit-wide analysis where the poorly understood error of sequential elements is underestimated.

Accurate error models are essential for automated design tools that determine, for example, the bit-stream length required to meet specified accuracy or precision demands [15]. Poor models will result in bit-stream lengths that are either too long or too short. A simulation-based approach has the advantage that it can provide accurate error estimates for complicated circuits, but it has the disadvantage of potentially long computation time. For example, the authors of [14] describe situations in SC where an analytic approach can take less than a minute to estimate output errors, whereas simulation takes over a month. The difference in speed makes accurate analytic models desirable and is another motivation for this work.

III. The Hypergeometric Model

This section introduces a new model for SNs based on the hypergeometric distribution.

A. *SN Bits as Hypergeometric Trials*

The large discrepancy between the plots of Fig. 2a suggests the need for a new model of LFSR SNs. Consider **X** to be one such SN. Upon closer inspection, we observe that the LFSR state determines **X**'s next bit, but each LFSR state is only visited once during its state sequence. In other words, LFSR states are visited in a pseudo-random order *without* replacement. Therefore, bits within **X** are not independent. For example, if a 0 is generated for **X** in the first clock cycle, the probability of generating a 0 in the next clock cycle decreases since the LFSR state that caused the 0 to be generated will not be repeated. This pattern of generating bits emulates trials of a hypergeometric random variable (RV) which we refer to as hypergeometric trials [8].

A hypergeometric RV A is defined by three parameters: n the number of trials, K the total number of success states in the population of interest, and N the population size; for short we use the notation $A \sim \text{Hypergeometric}(N, K, n)$. For each trial, a state is drawn from the population without replacement and A's value is the number of success states drawn during the n trials. If states are instead drawn with replacement, A would be a binomial RV with probability of success K/N and n trials; for short, $A \sim \text{Binomial}(K/N, n)$. To contrast hypergeometric and binomial RVs consider two RVs: $A \sim \text{Hypergeometric}(N, K, n)$ and $B \sim \text{Binomial}(K/N, n)$. A's expectation, and variance are

$$\mathbb{E}[A] = \frac{K}{N} n \tag{3}$$

$$\sigma_A^2 = n \frac{K}{N} \frac{(N-K)}{\text{N}} \frac{N-n}{N-1} \tag{4}$$

respectively. For B, these formulas are similar: $\mathbb{E}[B] = \frac{K}{N} n$ and $\sigma_B^2 = n \frac{K}{N} \frac{(N-K)}{N}$. The only difference in these statistics is that the variance for the hypergeometric RV has a factor $\frac{N-n}{N-1}$ which decreases as n, the number of trials, increases.

Returning to the n-bit LFSR SNG with binary input B that generates an N-bit SN **X**, we observe that the bits of **X** can be modeled as trials of $A \sim \text{Hypergeometric}(2^n - 1, B - 1, N)$. Consequently, we have that $X = A/N$, $\mathbb{E}[X] = \frac{B-1}{2^n-1}$ and, in the common case that N is set to $2^n - 1$, $\sigma_X^2 = 0$ ($N = 2^n$ is also often used, but here we consider $N = 2^n - 1$ for clarity). X's expected value is the same as it would be if **X** were modeled as a Bernoulli SN, but with variance zero. Other work has also noted that LFSRs can generate SNs that have no variance in their values [10], and here we explore the consequences of this fact.

B. *Accuracy of AND Multiplier*

The simulation of Fig. 2a used 8-bit LFSRs to generate 255-bit SNs. Using the hypergeometric SN model, we find that $\sigma_X = \sigma_Y = 0$. Then using the methodology presented in [14] which, for a given circuit, determines the output variance in terms of the inputs' expected values and variances, we obtain

$$\sigma_Z = \sqrt{\frac{XY(1-Y)(1-X)}{N-1}} \tag{5}$$

which is plotted as the yellow surface in Fig. 2b. The blue mesh in Fig. 2b again shows the result of using simulation to estimate σ_Z accurately, as defined by (2). Comparing Figs. 2a and 2b, we find there is a much stronger agreement about σ_Z between theory and experiment when using the hypergeometric rather than the Bernoulli model for LFSR SNs. Guided by this and subsequent results, we propose *hypergeometric SNs*, i.e., SNs whose bits are treated as hypergeometric trials, as a more accurate alternative to the Bernoulli SN model for LFSR SNs.

In general, the hypergeometric model is useful for finding the variance of LFSR SNs. This variance can then be combined with the variance propagation formulas of [14] to predict the output error of a stochastic circuit as we have just shown for the multiplier circuit. The methodology in [14] is applicable to any combinational SC circuit with uncorrelated inputs; for other circuits, equations for variance must be derived case by case.

C. *Accuracy of Mux Adder*

Consider the mux circuit of Fig. 1b where n-bit LFSR SNGs are used to generate $N = (2^n - 1)$-bit SNs. Assuming no correlation is present and combining the hypergeometric model with the formulas from [14], the standard deviation of Z is

$$\sigma_Z = \sqrt{\frac{S(1-S)\big(X(1-X)+Y(1-Y)\big)}{N-1}} \tag{6}$$

In the Bernoulli SN case, a single LFSR can be shared between **X**'s and **Y**'s SNGs without affecting σ_Z. But does this result hold if **X**, **Y** and **S** are modeled as hypergeometric SNs? In other words, will the expression for σ_Z in (6) change if **X** and **Y** are correlated? The methodology in [14] cannot be applied to cases with correlated SNs, so we derive σ_Z by hand. For clarity, we will continue to ignore quantization error which depends on the quantization scheme and in many cases is much smaller than random fluctuation error, i.e., variance-based error [17].

Without loss of generality, let $X \leq Y$. Since **X** and **Y** are SNs generated by a maximum-length LFSR sequence, we have $\sigma_X^2 = \sigma_Y^2 = 0$, which implies there are exactly XN and YN 1s in **X** and **Y**, respectively. Furthermore, when an RNS is shared between two SNGs, the generated SNs become maximally correlated in that they overlap as much as possible [1]. This implies that, for $x_i y_i$, the pattern 11 occurs exactly XN times, 10 occurs exactly zero times, and 01 occurs exactly $(Y - X)N$ times. In accordance with these patterns, define three SNs: $\mathbf{A} = \mathbf{X} \wedge \mathbf{Y}$, $\mathbf{B} = \overline{\mathbf{X}} \wedge \mathbf{Y}$ and $\mathbf{C} = \mathbf{X} \wedge \overline{\mathbf{Y}}$ where $\wedge$ is the AND operator and $\overline{}$ is NOT. The mux output can then be expressed as $\mathbf{Z} = \mathbf{A} \vee (\mathbf{B} \wedge \mathbf{S}) \vee (\mathbf{C} \wedge \overline{\mathbf{S}})$ where $\vee$ is the OR operator.

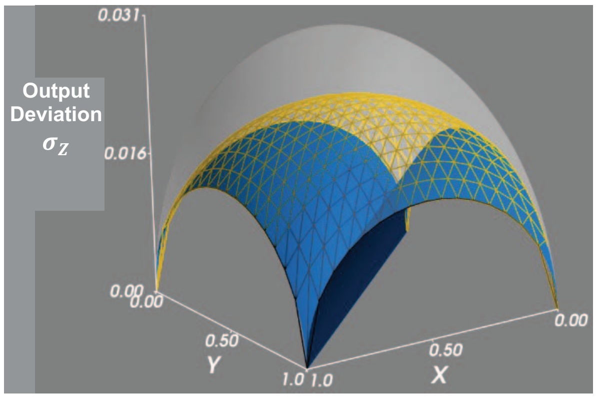

Fig. 3. Output deviation of mux adder vs. data input values X, Y and $S = 0.5$ for varying degrees of correlation between **X** and **Y**: maximally correlated (blue surface), uncorrelated (yellow mesh) and maximally anti-correlated (translucent white surface).

The aforementioned frequencies of the $x_i y_i$ bit patterns imply that $A = X$, $B = (Y - X)$, $C = 0$ and $\sigma_A^2 = \sigma_B^2 = \sigma_C^2 = 0$. Furthermore, $\sigma^2_{C \wedge \bar{S}} = 0$ because $C = \sigma_C^2 = 0$. We then have $\mathbb{E}[Z] = (1 - S)X + SY$ from the linearity of expectation, while the variance of Z is

$$\sigma_Z^2 = \sigma_{B \wedge S}^2 \tag{7}$$

which is the variance of the output of an AND gate with uncorrelated inputs **B** and **S** where $\sigma_B^2 = \sigma_S^2 = 0$. Therefore, $\sigma_{B \wedge S}$ can be found in the same manner as (5). Finally, we have

$$\sigma_Z = \sqrt{\frac{S(1-S)(Y-X)\big(1-(Y-X)\big)}{N-1}} \tag{8}$$

which never exceeds σ_Z for the uncorrelated case (6) and tends to zero as Y and X approach one another in value. For the maximally anti-correlated case (**X** and **Y** overlap as little as possible), a similar approach shows that

$$\sigma_Z = \sqrt{\frac{S(1-S)(U(1-U)+V(1-V)+2UV)}{N-1}} \tag{9}$$

where $U = X - \max(0, X + Y - 1)$ and $V = Y - \max(0, X + Y - 1)$. In this case, σ_Z is never less than, and can be much higher than, σ_Z when **X** and **Y** are uncorrelated. In sum, if the entire LFSR sequence is used and we model LFSR SNs as hypergeometric SNs, we find that correlation improves accuracy and anti-correlation decreases accuracy for the mux adder. Fig. 3 visualizes σ_Z for different levels of correlation between **X** and **Y** when $S = 0.5$ and **X**, **Y** and **S** are hypergeometric SNs. These plots match those produced by simulating the mux with LFSR SNs and estimating σ_Z with (2). Most noteworthy is the large difference in errors for the correlated and anti-correlated cases.

Fig. 4a illustrates the intuition behind this result. When **X** and **Y** are correlated, they overlap more and the output becomes more certain, while when **X** and **Y** are anti-correlated, they overlap less, and the output becomes less certain. In other words, the mux processing becomes more deterministic (less random) in the correlated case. When **X** and **Y** are Bernoulli SNs, this determinism doesn't increase accuracy because the variance in X and Y propagates to the output. But when $\sigma_X^2 = \sigma_Y^2 = 0$, as in the case of LFSR and hypergeometric SNs, the deterministic

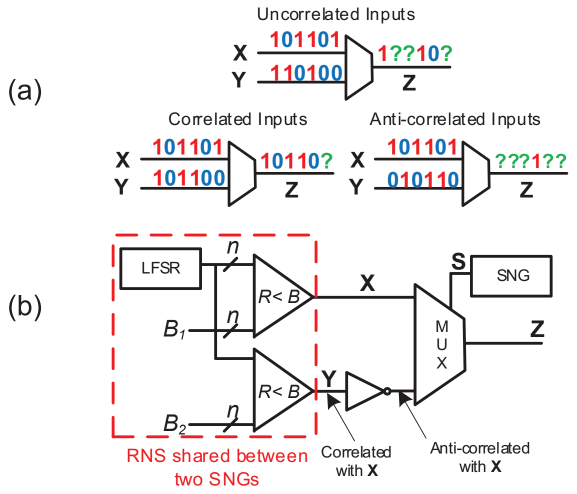

Fig. 4. (a) Comparison of mux behavior with varying levels of input correlation. An output bit denoted '?' is uncertain and depends on the select input. (b) Subtracter circuit with shared LFSR SNG.

mux processing produces a more accurate result. In general, if σ_X^2 and σ_Y^2 are small but nonzero, correlation may still improve accuracy for the same reason. This analysis provides motivation to correlate mux data inputs, not only to save circuit area (through sharing RNSs), but also to improve accuracy.

An example of such accuracy improvement is seen in [10], where the authors study stochastic digital filter design. Their filters contain many muxes that are indirectly connected to LFSR SNGs via XOR multiplier gates. This multiplication step adds some variance to the mux inputs. LFSRs are shared among the SNGs in various ways that introduce some correlation to the mux inputs. Table 4 of [10] gives error simulation data, which show that, in most cases and on average, basic LFSR sharing produces smaller errors than the no-share case. This accuracy improvement is mentioned, but not explored, in [10] presumably because the paper's focus is on other aspects of SNG design. However, it is a fairly clear example of correlation's effect on mux circuits, even when the inputs have some variance in their values.

We can now optimize existing circuits such as the subtracter circuit [12] in Fig. 4b. When an RNS is shared amongst SNs, they become correlated, but the inverter causes **Y** to become anti-correlated with **X** before they enter the mux. This anti-correlation causes a significant decrease in accuracy as shown in Fig. 3, which can be fixed by inverting the LFSR state before it enters **Y**'s comparator. This change causes **Y** to initially be anti-correlated with **X**, then, after passing through the inverter and before entering the mux, **Y** becomes correlated with **X**. This can come at little to no hardware overhead by changing **Y**'s comparator's design. We refer to this updated circuit as the correlation adjusted mux subtracter or "CAM subtracter" for short. Assuming all values of X and Y are equally likely, the CAM subtracter improves the accuracy by 39% on average compared to the original subtracter circuit.

D. New Balanced MUX Tree Design

Next we generalize the foregoing results to large muxes. A 2^m-way mux is typically designed as a balanced tree composed of $2^m - 1$ two-way muxes, where m is the depth of the tree. An

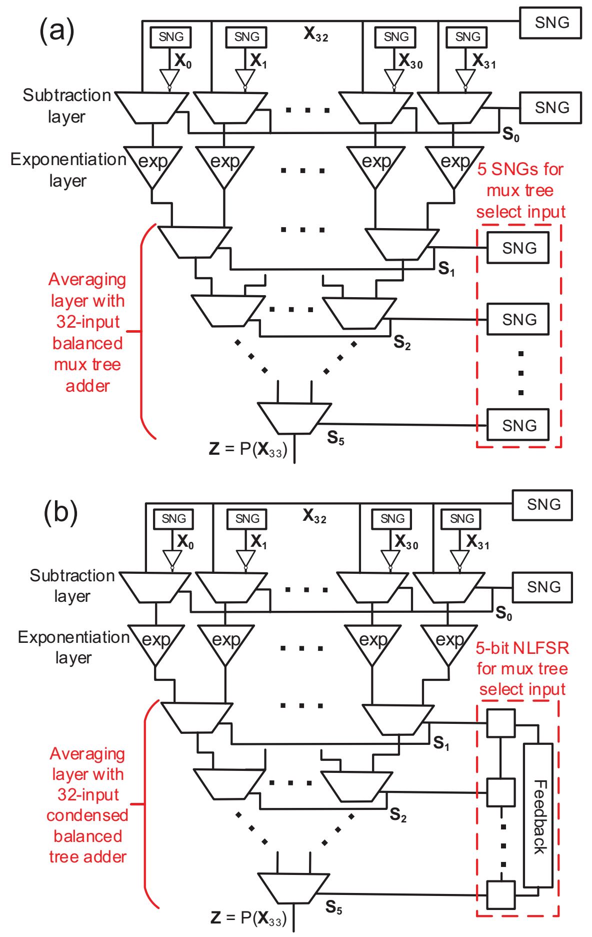
(a)
SNG
X_0
X_1
X_32
X_30
X_31
Subtraction layer
Exponentiation layer
exp
5 SNGs for mux tree select input
S_0
S_1
S_2
S_5
Averaging layer with 32-input balanced mux tree adder
Z = P(X_33)
(b)
5-bit NLFSR for mux tree select input
Feedback
Averaging layer with 32-input condensed balanced tree adder

Fig. 5. KDE circuits: (a) traditional [12]; (b) hypergeometric-inspired.

example is shown in the lower half of Fig. 5a with $m = 5$. In SC, a 2^m-way mux tree can be used to compute a weighted sum of its 2^m data inputs. We consider the common case where all select input values are set to 0.5, so the mux tree computes the average of the data inputs [12]. To save power and area, muxes in the same layer of the mux tree can share an LFSR SNG for their select inputs without causing inaccuracies [10].

A key and very desirable feature of the hypergeometric distribution is that as more of the population is sampled, i.e., more states of the LFSR sequence are traversed, uncertainty goes to zero. For example, in the two-way mux case (Fig. 1b) with $S = 0.5$ and $\sigma_S = 0$, exactly half of the outputs bits are propagated from $\mathbf{X}$ and the other half from $\mathbf{Y}$. This uniform sampling simplifies the circuit behavior and intuitively increases accuracy compared to the case where S is noisy ($\sigma_S \gg 0$).

Despite a similar LFSR-based construction, the traditional 2^m-way mux tree does not maintain the two-way mux's uniform data-sampling property. The reason is that the tree's m-bit select input S is a concatenation of m *independent* LFSR SNs. This makes it possible for the individual LFSR bit-streams to overlap in ways that cause S values to repeat, Hence, some data inputs will be selected more often than others due to random fluctuations in S. The chance of this happening decreases as bit-stream length increases, so a mux tree produces fairly accurate results. But can the mux tree be redesigned to eliminate the chance of non-uniform sampling and better emulate the hypergeometric behavior of a two-way mux?

To answer this question, we propose the new mux tree design in the lower half of Fig. 5b. It consists of a single non-linear FSR (NLFSR) in place of the mux's 5 LFSR SNGs. This n-bit NLFSR is constructed from an n-bit LFSR of maximum period $2^n - 1$, an $(n-1)$-input NOR gate and a 2-input XOR gate, as in [3]. The NLFSR has the same state sequence as the underlying LFSR, but with the all-0 state appended, which is necessary to sample all data inputs. In Fig. 5b, the NLFSR's 5-bit state serves as the select input S to the mux tree and the non-repeating property of the NLFSR sequence implies that each data input will be sampled uniformly. We call this mux tree design a "condensed balanced tree adder" (CBT adder for short).

To measure the accuracy improvements offered by CBT adders, we tested 16-, 32- and 64-input trees with 63- and 255-bit LFSR and Bernoulli SNs of random values. For each configuration, 5,000 simulation runs were used, and the root mean squared errors (RMSEs) of the traditional tree adder and CBT adder were calculated. For both correlated and un-correlated Bernoulli SNs, the CBT design achieved 16% error reduction compared to the traditional tree design. For correlated and uncorrelated LFSR SNs, the CBT design achieved 23% and 16% error reduction, respectively. The results were similar across the various numbers of data inputs and SN lengths.

Besides improving accuracy, the use of CBT adders in mux trees greatly reduces area. For an adder with 2^n-bit input SNs and a 2^m-way mux tree, the usual mux tree (Fig. 5a) requires m n-bit LFSRs, and m n-bit comparators, whereas a CBT adder requires just a single m-bit NLFSR. Its accuracy improvement and area reduction make the CBT adder a compelling new design for averaging stochastic numbers.

IV. Case Study Applications

A. Image Processing Case Study

Several major applications of SC involve summations with large numbers of inputs, Fig. 5a shows an example: an image processing circuit known as a kernel density estimation (KDE) circuit [12]. It takes a pixel's value from 33 consecutive video frames to calculate the probability density function

$$P(X_{32}) = \frac{1}{32}\sum_{t=0}^{31} e^{-4|X_{32}-X_t|} \tag{10}$$

where X_t is the pixel's value during frame t. The output of (10) can then be compared to a threshold value to determine if a pixel is a foreground or a background pixel.

The first layer of the KDE circuit subtracts a pixel's value X_{32} from its value during the previous 32 frames, $X_0, \ldots, X_{31}$. The second layer uses a sequential SC element to perform absolute exponentiation [13] on each difference. Finally, a mux tree averages the 32 exponentiations according to (10). Prior SC theory predicts that one LFSR can be shared by the SNGs for $\mathbf{X}_0, \ldots, \mathbf{X}_{31}$ without affecting accuracy. We refer to the circuit in Fig. 5a with such sharing as the traditional KDE (TKDE) circuit. The new results presented here predict that a CAM subtracter will increase the subtraction layer's accuracy if LFSR SNs are used. Further, our results predict that the mux tree can be replaced with a CBT adder to save circuit area and improve

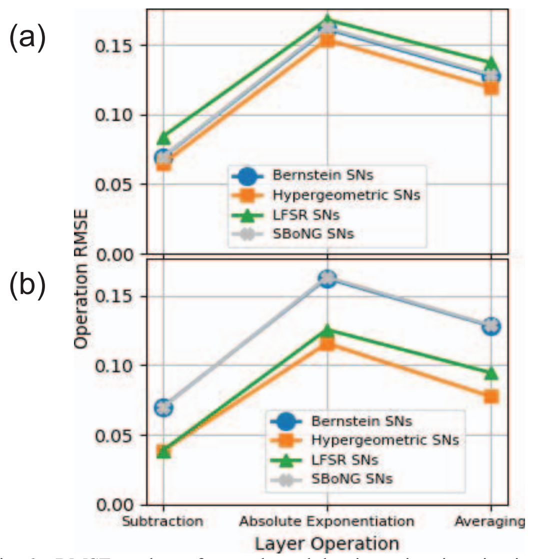

Fig. 6. RMSE per layer for two kernel density estimation circuits: (a) TKDE; (b) HKDE.

accuracy. We refer to the TKDE circuit with these two changes as the hypergeometric-inspired KDE (HKDE) circuit (Fig. 5b).

We evaluated the two KDE designs with the MPEG-4 test sequence "hall monitor," a benchmark used in video processing [18]**.** For both cases, we simulated a variety of SNs: Bernoulli and hypergeometric SNs (using software RNSs), LFSR SNs, and SNs generated using SBoNG. The size of the LFSR and SBoNG states were set to 8 bits and a 5-bit NLFSR was used for the CBT adder's select input. SNs of length $N = 255$ were used in all cases. Plotted in Fig. 6 is the RMSE between the simulated value and target value (based on (10)) for each circuit layer.

For the TKDE circuit, the subtraction layer's RMSE is about the same for each type of SN because the inputs to the mux are anti-correlated and thus result in high error in all cases. For the HKDE circuit, the error for this layer decreases substantially for hypergeometric and LFSR SNs because the CAM subtracter is used. However, the error does not decrease for SBoNG SNs or Bernoulli SNs because correlation does not affect mux errors for these SNs [1][10]. Error results for LFSR SNs generally match those of hypergeometric SNs but tend to be higher due to correlations present in the consecutive LFSR states that would not be present in a true hypergeometric RV. Overall, we achieve a significant output accuracy improvement of 32% for LFSR SNs by using the HKDE design in place of the TKDE design.

To estimate area cost, we use gate count (GC) where the GC of a single logic gate is the number of inputs to that gate divided by 2 [16]. The GC of a circuit is the sum of the GC of all its logic gates. When using LFSR SNGs, the HKDE circuit's mux tree GC and overall GC are, respectively, 71% and 12% less than the corresponding TKDE circuit's GCs. The area improvement is due to the CBT adder which has a single 5-bit NLFSR in place of five 8-bit LFSRs and comparators.

B. *Other Applications*

The analysis presented in this paper applies to many well-known combinational SC designs, especially those centered around muxes. These include digital filters [10] and the reconfigurable SC architecture ReSC [17]. Using correlated LFSR SNs instead of more Bernoulli-like SNs (such as SBoNG SNs) results in significant accuracy improvement for these designs. Such improvements could result in a smaller ReSC circuit or in shorter bit-streams to meet specified accuracy goals.

In general, the hypergeometric model is suitable for SNs produced from SNGs built around any uniform, non-repeating RNS such as a LFSR. Another notable RNS of this type is the rule 90-150 hybrid cellular automata that has been shown to be a competitive alternative to the LFSR for SN generation [5][9].

V. Conclusions

In this study, we proposed a new model for LFSR SNs based on the hypergeometric distribution. This model intuitively fits these SNs due to the non-repeating way in which LFSRs produce random numbers. We showed how the new model leads to more accurate error estimates than analysis with the Bernoulli model for several classes of LFSR-based circuits. Most importantly, we showed that combinational circuit error is lower when using LFSR SNs which implies shorter bit-streams suffice to obtain a specified level of accuracy or precision. Finally, we introduced the CAM subtracter and CBT adder which significantly improve accuracy and area over traditional designs.

References

[1] Alaghi, A. and J.P. Hayes, "Exploiting correlation in stochastic circuit design," *Proc. ICCD*, 39-46, 2013.

[2] Alaghi, A., W. Qian and J.P. Hayes. "The promise and challenge of stochastic computing," *IEEE Trans. CAD*, **37:** 1515-1531, 2018.

[3] Arazi, B. "On the synthesis of de-Bruijn sequences", *Information and Control,* **49**, 81-90, 1981.

[4] Braendler, J.D., T. Hendtlass and P. O'Donoghue. "Deterministic bit-stream digital neurons," *IEEE Trans. Neural Nets.*, **13**, 1514–1525, 2002.

[5] Brown, B.D. and H.C. Card. "Stochastic neural computation. I. Computational elements," *IEEE Trans. Computers*, **50**, 891-905, 2001.

[6] Gaines, B.R. "Stochastic computing systems," *Advances in Information Systems Science*, **2**: 37-172, 1969.

[7] Golomb, S.W. *Shift Register Sequences*, Aegean Park Press, 1981.

[8] Grinstead, C.M. and L.J. Snell. *Grinstead and Snell's Introduction to Probability*, version of 4 July 2006, American Math. Soc., 2006.

[9] Hortensius, P.D., R.D. McLeod and H.C. Card. "Parallel random number generation for VLSI systems using cellular automata," *IEEE Trans Computers*, **38**, 1466-1473, 1989.

[10] Ichihara, H. et al. "Compact and accurate digital filters based on stochastic computing," *IEEE Trans. Emerging Topics in Comp.*, **7**, pp. 31-43, 2019.

[11] Kim, K. et al. "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," *Proc. DAC,* 1-6, 2016.

[12] Li, P. et al. "Computation on stochastic bit streams digital image processing case studies," *IEEE Trans. VLSI*, **22**, 449-462, 2014.

[13] Li, P. et al. "The synthesis of linear finite state machine-based stochastic computational elements," *Proc. ASPDAC*, 757-762, 2012.

[14] Ma, C., S. Zhong and H. Dang, "Understanding variance propagation in stochastic computing systems," *Proc. ICCD*, 213-218, 2012.

[15] Neugebauer, F., I. Polian & J.P. Hayes. "Framework for quantifying and managing accuracy in stochastic circuit design," *Proc. DATE,* 1- 6, 2017.

[16] Neugebauer, F., I. Polian & J.P. Hayes. "Building a better random number generator for stochastic computing," *Proc. DSD*, 1-8, 2017.

[17] Qian, W. et. al. "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Computers*, **60**, 93-105, 2011.

[18] Sikora, T. "The MPEG-4 video standard verification model," *IEEE Trans. Circuits and Systems for Video Technology*, **7**, 19-31, 1997.