Eliminating a Hidden Error Source in Stochastic Circuits

Paishun Ting and John P. Hayes
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109, USA
{paishun, jhayes}@umich.edu

***Abstract*—Stochastic computing (SC) computes with probabilities using random bit-streams and standard logic circuits. Its advantages are ultra-low area and power, coupled with high error tolerance. However, due to its randomness features, SC's accuracy is often low and hard to control, thus severely limiting its practical applications. Random fluctuation errors (RFEs) in SC data are a major factor affecting accuracy, and are usually addressed by increasing the bit-stream length *N*. However, increasing *N* can result in excessive computation time and energy consumption, counteracting the main advantages of SC. In this work, we first observe that many SC designs heavily rely on constant inputs, which contribute significantly to RFEs. We then investigate the role of constant inputs in SC, and propose a systematic algorithm CEASE to eliminate them by introducing memory into the target circuits. We provide analytical and experimental results which demonstrate that CEASE is optimal in terms of minimizing RFEs.**

I. Introduction

Stochastic computing (SC) is an unconventional digital logic technique that interprets signals as probability values embedded in random 0-1 bit-streams called stochastic numbers (SNs). The value of an SN is usually estimated by the fraction of 1s in the bit-stream. For example, the bit-stream **X** = 010011 is a 6-bit SN with (unipolar) value 0.5, since three of its six bits are 1, i.e., the probability of a 1 appearing in **X** is 0.5, which we denote by $p_X(1) = X = 0.5$. SC performs arithmetic on probabilities by transforming a set of SNs. The scaled adder in Fig. 1*a* illustrates SC's basic mechanism. The multiplexer (MUX) takes two variable SNs **X** and **Y**, and an independent constant SN **R** of value $R = 0.5$ as inputs, and outputs the SN **Z**. The probability of a 1 occurring in **Z** is the probability of (**X**, **R**) = (1, 0) plus the probability of (**Y**, **R**) = (1, 1). With $R = 0.5$, it is easily seen that $Z = 0.5(X + Y)$, a scaled sum of X and Y. The scaling ensures that Z lies in the probability range [0, 1].

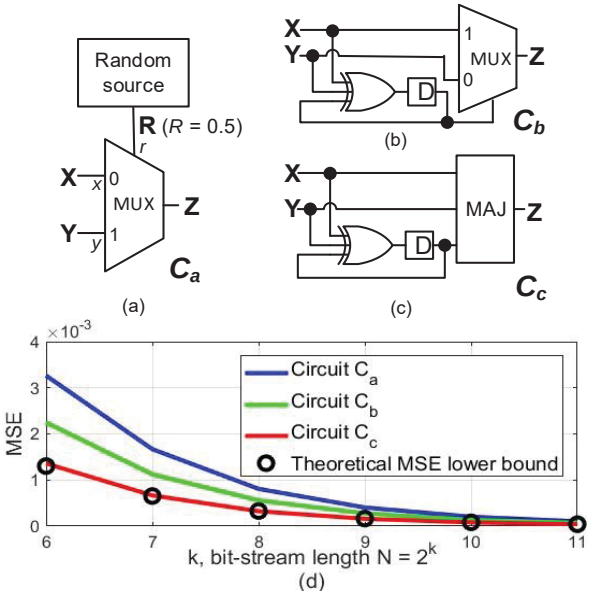

Figure 1. Three stochastic implementations of scaled addition: (a) conventional MUX-based design C_a with a constant input $R = 0.5$; (b) ad hoc sequential design C_b with no constant input; (c) sequential design C_c produced by CEASE; (d) error comparison of the three designs.

978-1-5386-0362-8/17/$31.00 ©2017 IEEE

SC's advantages of error tolerance, low power, and low area cost show great promise in applications like image processing [1], ECC decoding [7], and machine learning [5] However, it suffers from two major error sources: undesired correlations and random fluctuations. Correlation is due to inadequate randomness among the SNs being processed. It may be tackled via decorrelation circuitry that can add significantly to area cost [14]. *Random fluctuation errors* (*RFEs*) occur when the SN length N is too small, or the quality of the randomness sources is poor [4]. Fig. 2 shows how three SNs generated by the circuits in Fig. 1 can fluctuate around their exact value 0.5 as N changes. As we will see, while these circuits implement the same scaled addition function, they have very different RFE levels. RFEs can be reduced by increasing N, but this can produce very long run times and hence high energy consumption. To avoid such problems, SC usually compromises accuracy, thereby narrowing the range of applications to which it can be successfully applied.

Some prior work integrates deterministic or otherwise artificially correlated bit-stream formats into SC design to reduce RFEs [4][8][9][15]. However, this comes with major drawbacks such as expensive hardware to (re)generate the special number formats, excessively long bit-streams, lack of general synthesis methods, etc., depending on the bit-stream format employed. It is also far easier to maintain the bit-stream randomness required by conventional SC. Furthermore, there are many applications to which conventional SC is better suited. For example, the retina-implant chip described in [1] avoids costly SN generators by converting light signals directly into conventional random bit-streams. In such cases, increasing SN length appears to be the most practical way to decrease RFEs, but it has the problems mentioned previously.

In this paper, we introduce and investigate a new design methodology to reduce RFEs in conventional SC, and improve

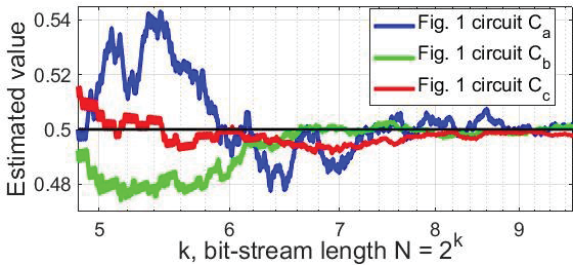

Figure 2. Typical random fluctuations in three SNs with the same exact value 0.5 as bit-stream length N increases.

accuracy without compromising computation time and energy consumption while maintaining desirable SC features such as error tolerance. It is based on the observation that most SC designs, from the simple scaled adder in Fig. 1*a*, to complex stochastic circuits generated by major synthesis methods like STRAUSS [2] and ReSC [13], heavily rely on the use of constant SNs to achieve good function approximations. These SNs not only increase hardware overhead due to their need for random sources but, as we show in this paper, also turn out to be a major source of error-inducing RFEs.

A common way to quantify SC errors is *mean squared error* (*MSE*), which is the accuracy metric used in this work. The MSE of an N-bit SN $\mathbf{Z}$ is defined as:

$$\mathrm{MSE}(\mathbf{Z}, N) = \mathbb{E}[(\hat{Z}^{(N)} - Z)]^2 \quad (1)$$

where $\mathbb{E}(\cdot)$ denotes the expectation function, and $\hat{Z}^{(N)}$ denotes the estimated value of the N-bit SN $\mathbf{Z}$ generated by a stochastic circuit. Eq. (1) computes the average squared deviation of an SN's estimated value from its exact or desired value.

In this paper, we show that it is possible to remove all error-inducing input constants by resorting to sequential SC designs. We also devise a systematic method which we call ***C**onstant **E**limination **A**lgorithm for **S**uppression of **E**rrors* (*CEASE*) for constant removal. While a function may have various circuit implementations without constant inputs, CEASE circuits provide a guarantee of optimality on RFE reduction. Figs. 1*b-c* depict two sequential scaled adder designs after eliminating constant SNs; the former was designed in ad hoc fashion, while the latter was generated by CEASE. (It happens to include a subcircuit that implements the majority function MAJ.) Fig. 1*d* plots the (sampled) MSEs of all three scaled adders against bit-stream length $N = 2^k$. It can be seen that the CEASE design is the most accurate. Furthermore, it meets the theoretical lower bound on MSE for RFEs indicated by the small circles.

The main contributions of this paper are:

- Clarifying the role of constants in SC design, and showing that they are a major contributor to RFEs.
- The CEASE algorithm to systematically remove constant SNs by transferring their role to memory.
- Proving that CEASE provides a guarantee of optimality on RFE reduction.

The paper is organized as follows. Sec. II briefly introduces SC, and examines the role of constant SNs. Sec. III details CEASE and analyzes its performance. Sec. IV presents experimental results, while Sec. V draws some conclusions.

II. Stochastic Computing

We first review relevant concepts of stochastic circuits and the functions they implement. We then investigate the role of constant input SNs in SC. A combinational stochastic circuit C implements a class of arithmetic functions that depend on the Boolean function f realized by C, as well as the SN values applied to C and their joint probability distribution.

Example 1: Combinational SC Adder. The adder in Fig. 1*a*, has three SNs $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{R}$ applied to its inputs x, y, r. It implements the Boolean function $f(x, y, r)$, which outputs a 0 bit except when $f(1, 0, 0) = f(0, 1, 1) = f(1, 1 ,0) = f(1, 1, 1) = 1$. The probability that the circuit's output is 1 is thus the probability that one of the input patterns 100, 011, 110 or 111 occurs. We can then write

$$\begin{aligned} Z &= p_{\mathcal{X}}(1,0,0) + p_{\mathcal{X}}(0,1,1) + p_{\mathcal{X}}(1,1,0) + p_{\mathcal{X}}(1,1,1) \\ &= \textstyle\sum_{\boldsymbol{b}}[f(\boldsymbol{b}) p_{\mathcal{X}}(\boldsymbol{b})] \end{aligned} \quad (2)$$

where $\boldsymbol{b}$ denotes a 3-bit input binary vector.

In general, suppose C implements the Boolean function $z = f(x_1, x_2, \ldots, x_n)$. Let $\mathcal{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n\}$ with values $\{X_1, X_2, \ldots, X_n\}$ be the set of SNs applied to the inputs $x_1, x_2, \ldots, x_n$. The stochastic function realized by C has the following form [3]:

$$Z = F(f, p_{\mathcal{X}}) = \textstyle\sum_{\boldsymbol{b}}[f(\boldsymbol{b}) p_{\mathcal{X}}(\boldsymbol{b})] \quad (3)$$

where $p_{\mathcal{X}}(\boldsymbol{b})$ is the joint probability distribution of the input SNs, and the summation is over all combinations of the n-bit input vector $\boldsymbol{b}$. Eq. (3) indicates that a stochastic function is a linear combination of the probability terms $p_{\mathcal{X}}(\boldsymbol{b})$ with binary coefficients $f(\boldsymbol{b})$ taking 0-1 values.

Eq. (3) has the most general form of a stochastic function. However, many stochastic circuits, including those synthesized by STRAUSS and ReSC heavily use constant input SNs to define the both target function's value and its precision. For example, in Fig. 1*a*, the input SN $\mathbf{R}$ with a fixed value 0.5 is generated by a random source not controllable by the user of the circuit, hence it is a constant SN. The user can only control the values of the variable SNs $\mathbf{X}$ and $\mathbf{Y}$. In such cases, we can separate the input SNs into two disjoint subsets $\mathcal{X} = \{\mathcal{X}_V, \mathcal{X}_C\}$, where $\mathcal{X}_V$ and $\mathcal{X}_C$ denote variables and constants, respectively [6]. A stochastic function of $p_{\mathcal{X}_V}$ can then be derived from Eq. (3) by replacing the $\mathcal{X}_C$ with appropriate constant values.

Example 1 (cont.): Returning to the MUX-based adder, let $\mathcal{X} = \{\mathcal{X}_V, \mathcal{X}_C\}$, where $\mathcal{X}_V = \{\mathbf{X}, \mathbf{Y}\}$ and $\mathcal{X}_C = \{\mathbf{R}\}$ with $R = 0.5$, as in Fig. 1*a*. If $\mathbf{R}$ is independent of $\mathcal{X}_V$, then on substituting $p_{\mathbf{R}}(1) = R = 0.5$ into Eq. (2), we get

$$\begin{aligned} Z &= 0.5[p_{\mathcal{X}_V}(1,0) + p_{\mathcal{X}_V}(0,1) + p_{\mathcal{X}_V}(1,1) + p_{\mathcal{X}_V}(1,1)] \\ &= 0.5[p_{\mathbf{X}}(1) + p_{\mathbf{Y}}(1)] = 0.5(X + Y) \end{aligned} \quad (4)$$

which is the expected addition function of X and Y.

Looking closely at Eq. (4), we can see that it is a linear combination of probability terms with *non-binary coefficients*, in contrast to Eq. (3) which allows only the binary coefficients 0 and 1. This suggests that constant inputs allow a stochastic function to have coefficients that are any rational numbers in the range [0, 1], which denotes the unit interval. The following theorem generalizes this observation.

Theorem 1: The stochastic function implemented by a combinational circuit with input SNs $\mathcal{X} = \{\mathcal{X}_V, \mathcal{X}_C\}$ has the form

$$Z = F(p_{\mathcal{X}_V}) = \textstyle\sum_{\boldsymbol{b}_V}\left[g(\boldsymbol{b}_V) \cdot p_{\mathcal{X}_V}(\boldsymbol{b}_V)\right] \quad (5)$$

where the $g(\boldsymbol{b}_V) \in [0,1]$ are constants that depend on f and $p_{\mathcal{X}_C}$. (For brevity, this dependency is dropped from Eq. (5).) A proof of this theorem can be found in Appendix.

Theorem 1 reveals some interesting facts about the impact of constant inputs on stochastic functions. It implies that, at the expense of extra constant inputs, the class of implementable

functions can be greatly broadened. For example, a combinational circuit cannot implement the stochastic function $Z = 0.5(X + Y)$ with just two inputs **X** and **Y**, since this function also requires the non-binary coefficient 0.5. This scaled add function is combinationally implementable however, as demonstrated in Example 1, by supplying an extra constant input SN **R** with value 0.5. In general, when a target function is not already in the form of Eq. (5), it has to be converted to that form by introducing suitable constants. Moreover, the function's approximation accuracy highly depends on the number of constants used, as can be seen in both the STRAUSS and ReSC synthesis algorithms. A circuit with good approximation accuracy therefore typically comes with many RFE-inducing constant inputs. For instance, the STRAUSS implementation of $Z = \frac{7}{16} - \frac{1}{4}X - \frac{9}{16}X^2$ derived in [2] employs four constants, each of value 0.5.

To eliminate constant-induced RFEs, we propose ***C**onstant **E**limination **A**lgorithm for **S**uppression of **E**rrors* (*CEASE*), a systematic algorithm for removing constants while keeping their functional benefits. Specifically, CEASE transforms a target combinational circuit into a functionally equivalent stochastic sequential circuit with no constant inputs and with reduced RFEs. CEASE also offers a guarantee of optimality on RFE reduction, thereby providing a big improvement in accuracy.

III. Constant Elimination

This section details CEASE, the proposed constant SN elimination algorithm, along with an analysis of its accuracy.

Example 1 (cont.): We re-examine the MUX-based adder of Fig. 1*a*, shown again in Fig. 3. To implement the scaled addition

$$Z = 0.5(X + Y) \tag{6}$$

accurately, the expected number of 1s in **Z** should be half the number of 1s in the input SNs **X** and **Y**. Let subscript i denote the bit of an SN appearing in clock cycle i. When both $\mathbf{X}_i$ and $\mathbf{Y}_i$ are 1, the corresponding output bit $\mathbf{Z}_i$ will be 1; this is exact since a single 1 should be produced in **Z** whenever the circuit receives two 1s. Similarly, when both $\mathbf{X}_i$ and $\mathbf{Y}_i$ are 0, $\mathbf{Z}_i$ will have the exact value 0. When only one of the two inputs is 1, i.e., $\mathbf{X}_i\mathbf{Y}_i$ = 10 or 01, Eq. (6) implies that ideally $\mathbf{Z}_i$ should be 0.5. However, $\mathbf{Z}_i$ obviously cannot directly output "0.5" from a logic circuit that computes with 0-1 values. This representational dilemma is effectively solved by the extra constant SN **R** whose stochastic value 0.5 ensures that *on average* $\mathbf{Z}_i = 1$ whenever two copies of 10 or 01 are received. In other words, a single 1 is expected to be produced in response to every two applications of 10 or 01. This single 1 spread over two cycles thus contributes 1/2 to **Z**.

The fact that using additional constants produces extra RFEs can also be seen from Fig. 3, where the constant $\mathbf{R}_i$ is used to

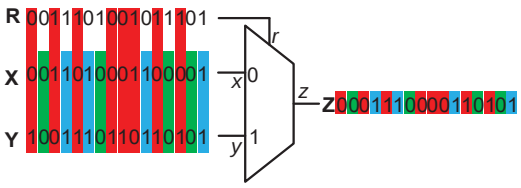

Figure 3. The MUX-based stochastic scaled adder. On receiving 11 (blue) or 00 (green), the output is 1 or 0, respectively. On receiving 01 or 10 (red), the output is 1 with probability 0.5.

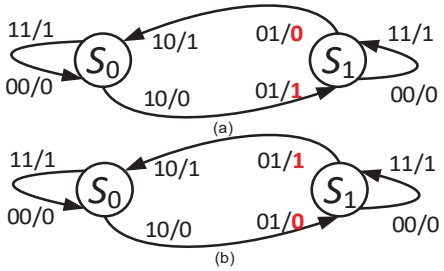

Figure 4. STGs for the sequential scaled adders corresponding to (a) ad hoc design C_b of Fig. 1*b*, and (b) CEASE design C_c of Fig. 1*c*. The differences between the two STGs are marked in red in the figure.

select inputs whenever $\mathbf{X}_i\mathbf{Y}_i$ = 10 or 01 (marked in red). Notice here that in the red cycles, four 1s should appear in every eight output bits. However, since this is only true *on average*, there may be variations due to the probabilistic nature of **R**. In this example, there are only three 1s instead of four in the eight output bits selected by **R**, producing a 1/16 error in the output value. The key to eliminating **R** (and the RFEs it introduces) is to enable the circuit to *remember* every two applications of 10 or 01, which implies changing it from combinational to sequential. This makes it possible for the circuit to output *exactly* one 1 for every two applications of input pattern 01 or 10.

A. Functions Implemented by CEASE Circuits.

The fact that it is impossible for combinational logic to output a non-binary value without the use of constant inputs is reflected in Eq. (3) where only binary coefficients are allowed. CEASE circumvents this issue and at the same time reduces RFEs by constructing an equivalent sequential circuit. The idea behind CEASE is to introduce memory elements to count and remember non-binary values. It constructs a sequential circuit that accumulates such values to be output later. When an accumulated value exceeds one, then a 1 is outputted.

Example 2: Sequential SC Adder. The state-transition graph (STG) of the scaled adder C_c produced by CEASE (Fig. 1*c*) is shown in Fig. 4*b*. Like the combinational adder in Fig. 3, C_c outputs a 1 when 11 is received, and a 0 when 00 is received. The difference is that when a 10 or 01 is received, C_c remembers this information by going from state s_0 to state s_1, and outputting a 0. When another 10 or 01 is received, the circuit's implicit counter will, in effect, overflow by returning to state s_0, and outputting a 1. In this way, it is guaranteed that *exactly* one 1 is

Input:	Target stochastic function F^*
Output:	An SC finite-state machine approximating F^*
Step 1.	Approximate F^* as F using Eq. (5) with rational coefficients $\boldsymbol{g} = \{g(\boldsymbol{b}_1), g(\boldsymbol{b}_2), \ldots, g(\boldsymbol{b}_m)\}$ in [0, 1].
Step 2.	Find the lowest common multiple q of the denominators of $\boldsymbol{g}$. Let $\boldsymbol{a} = \{a(\boldsymbol{b}_1), a(\boldsymbol{b}_2), \ldots, a(\boldsymbol{b}_m)\} = q \cdot \boldsymbol{g}$.
Step 3.	Construct a modulo-q counter MC with states $s_0, s_1, \ldots, s_{q-1}$.
Step 4.	Modify MC so that on receiving the pattern $\boldsymbol{b}_i$, it jumps forward $a(\boldsymbol{b}_i)$ states. At each clock cycle, it outputs 1 if MC overflows, otherwise it outputs 0.
Step 5.	Synthesize MC using any suitable conventional synthesis technique.

Figure 5. Algorithm CEASE for constant elimination.

generated whenever *exactly* two copies of 10 or 01 are received. Hence, the RFEs introduced by constant SNs are completely removed by CEASE.

In general, CEASE takes a target arithmetic function and approximates it as in Eq. (5) to generate a sequential circuit implementing the approximated function without constant SNs. The resulting circuit resembles a counter that keeps a running sum of each non-binary input value of interest. Whenever an accumulated sum exceeds 1, the circuit outputs 1 and resets the counter to the overflow amount. A pseudo-code algorithm summarizing CEASE is given in Fig. 5.

Example 2 (cont.): Consider again the scaled addition $Z = 0.5(X + Y)$. Eq. (4) implies that $Z = 0.5p_{\mathcal{X}_V}(1,0) + 0.5p_{\mathcal{X}_V}(0,1) + p_{\mathcal{X}_V}(1,1)$. Therefore, the coefficient set $\boldsymbol{g}$ is $\{g(0,0) = 0, g(0,1) = 1/2, g(1,0) = 1/2, g(1,1) = 2/2\}$. Since all the coefficients are rational, the first step of CEASE is skipped. The lowest common multiple q of the denominators in $\boldsymbol{g}$ is the number of count states needed. Since q = 2 here, we need a two-state counter. Furthermore, $\boldsymbol{a} = q \cdot \boldsymbol{g} = \{0, 1, 1, 2\}$. Therefore, the counter is designed such that every time the pattern $\mathbf{X}_i\mathbf{Y}_i$ = 10 or 01 is applied, the counter adds 1 to its state. The pattern $\mathbf{X}_i\mathbf{Y}_i$ = 11 adds 2 to the counter's state. When the counter overflows, a 1 is sent to the output; otherwise the output is set to 0. This confirms that Fig. 4*b* is indeed the STG of an exact scaled adder, with C_c in Fig. 1*c* being one of its possible circuit implementations.

Another viewpoint on the validity of the scaled adder in Fig. 4*b* is its behavior under steady-state probability distribution. It is not hard to see that the long-term probabilities of staying in state s_0 and s_1 are equal, i.e., $p_S(s_0) = p_S(s_1) = 1/2$, since the state transition behavior of this circuit is symmetric. The probability of outputting a 1 when $S = s_0$ is $p_{\mathcal{X}_V}(1,1)$, and that probability is $p_{\mathcal{X}_V}(1,1) + p_{\mathcal{X}_V}(0,1) + p_{\mathcal{X}_V}(1,0)$ when S = s_1. Hence, the overall probability of outputting a 1 is

$$Z = p_S(s_0)p_{\mathcal{X}_V}(1,1) + p_S(s_1)\,[p_{\mathcal{X}_V}(1,1) + p_{\mathcal{X}_V}(0,1) + p_{\mathcal{X}_V}(1,0)]$$
$$= 0.5[p_{\mathbf{X}}(1) + p_{\mathbf{Y}}(1)] = 0.5(X + Y)$$

which is indeed the scaled addition function. Not surprisingly, a CEASE circuit implements a stochastic function in the form of Eq. (5), as stated in the following easily-proven theorem.

Theorem 2: Any circuit generated by CEASE implements a stochastic function in the form of Eq. (5), the class of functions combinationally implementable with constant inputs.

B. Accuracy of CEASE Circuits.

We now consider the role of CEASE in RFE reduction.

Theorem 3: Given a stochastic function $Z = F(p_{\mathcal{X}_V})$ in the form of Eq. (5) with rational coefficients, suppose the members of $\mathcal{X}_V$ are Bernoulli bit-streams, but correlations among them are unknown. Then the following holds for all integers $N > 0$:

$$MSE(\mathbf{Z}_C, N) \lesssim MSE(\mathbf{Z}^*, N) \quad (7)$$

where $\mathbf{Z}_C$ is the output SN generated by a CEASE circuit implementing F, while $\mathbf{Z}^*$ is the output of any other circuit.

The notation $\lesssim$ in (7) indicates that inequality holds up to a rounding error. Depending on the rounding policy, rounding may produce up to 1-bit error when the length of SNs is unable to represent certain values exactly. For example, an SN of odd length N cannot represent 1/2 exactly, but one of length $N \pm 1$ can. Theorem 3 states that among all possible implementations of F, CEASE produces a result with the least MSE. A proof of Theorem 3, which requires some advanced concepts from statistics, is outlined in the Appendix.

Theorem 3 can be understood intuitively from the fact that CEASE's precise counting process guarantees exactness as discussed above, and hence minimizes MSEs. For comparison, consider the circuit C_b in Fig. 1*b*, whose STG is given in Fig. 4*a*. One can easily see that C_b, while constructed in ad hoc fashion, also computes scaled addition like the CEASE circuit C_c in Fig. 1*c* whose STG is in Fig. 4*b*. Suppose the following artificially constructed SNs are applied to C_b and C_c:

$$\mathbf{X}_{art} = 010101010101\ (X = 6/12 = 0.5)$$
$$\mathbf{Y}_{art} = 101010101010\ (Y = 6/12 = 0.5)$$

The expected output value should be $0.5(X_{art} + Y_{art}) = 0.5$, which is exactly what C_c will give. However, feeding these two input SNs to C_b's STG in Fig. 4*a* initialized to state s_0 will produce the output $\mathbf{Z}_b$ = 111111111111 (Z_b = 12/12 = 1), a 100% error! The accuracy difference between the two designs is due to the fact that CEASE *guarantees* to output a 1 whenever two copies of 10 or 01 are received, whereas the ad hoc design does not. CEASE-generated designs also retain the high tolerance of stochastic circuits to transient errors (bit-flips) affecting the variable inputs. An occasional transient or soft error can cause a relatively small miscount of the applied input patterns, which can then result in a similarly small output error. For instance, if $\mathbf{X}_{art}$ is changed to 010101010000 due to two 1-to-0 bit-flips, the output value produced by C_c will become 5/12, which is a good estimate of the exact output value 0.5 = 6/12.

It's also worth mentioning that as a side effect of removing constant inputs, CEASE reduces potential correlation errors induced by such inputs. However, undesired correlations among variable inputs must be tackled separately using decorrelation methods such as [14].

A scaled sequential adder constructed in ad hoc fashion around a T flip-flop is given in [11] and shown by simulation to be more accurate than the standard combinational design. The STG of that adder is exactly the same as that in Fig. 4*b* for the adder constructed by CEASE. This confirms the high accuracy claimed for the T-flip-flop-based adder, an important factor in the success of the neural network implementation in [11].

IV. Experimental Results

This section examines the performance of CEASE on some representative published circuits. It also assesses the accuracy of CEASE using randomly generated stochastic circuits.

A. Multi-linear Polynomial

CEASE can be applied to SN formats other than unipolar as well, since it deals directly with probabilities rather than their interpretation. Suppose, for example, that CEASE is applied to the circuit C_{ST} synthesized by STRAUSS [2] and outlined in Fig. 6*a*. C_{ST} uses the inverted bipolar (IBP) SN format to handle negative values, and realizes the following stochastic function:

$$\tilde{Z} = \frac{7}{16} - \frac{1}{8}\left(\tilde{X}_1 + \tilde{X}_2\right) - \frac{9}{16}\tilde{X}_1\tilde{X}_2 \quad (8)$$

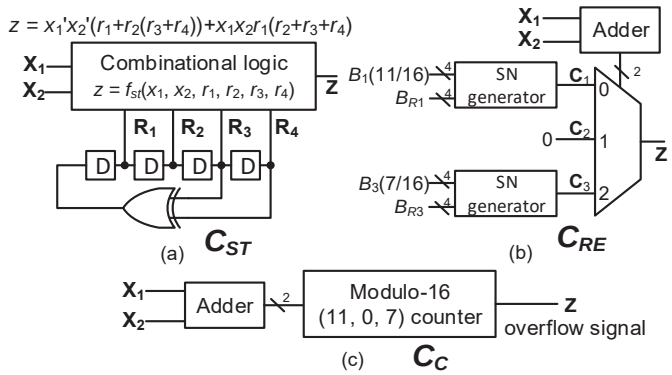

Figure 6. Three implementations of Eq. (8): (a) STRAUSS design C_{ST}, (b) ReSC design C_{RE}, and (c) CEASE design C_C.

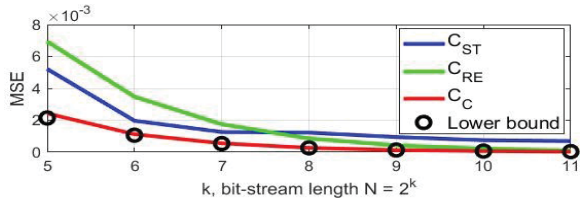

Figure 7. MSE comparison for the circuits in Fig. 6.

where $\tilde{\mathbf{X}}_1$ and $\tilde{\mathbf{X}}_2$ are independent IBP SNs with the same value. This STRAUSS design heavily relies on constant SNs, as it employs four constants $\mathbf{R}_1$, $\mathbf{R}_2$, $\mathbf{R}_3$, $\mathbf{R}_4$, each of value 0.5. Another implementation C_{RE} of the same function $\tilde{Z}$ synthesized by ReSC [13] is given in Fig. 6*b*; it relies on the constants $\mathbf{C}_1$, $\mathbf{C}_2$ and $\mathbf{C}_3$ to provide the same level of accuracy. To implement Eq. (8) using CEASE, we first derive the corresponding unipolar stochastic function from the relation $\tilde{X} = 1 - 2X$, where $X = p_{\mathbf{X}}$ is the unipolar SN value corresponding to the IBP value $\tilde{X}$. On replacing $\tilde{Z}$, $\tilde{X}_1$ and $\tilde{X}_2$ by their unipolar counterparts in Eq. (8) and re-arranging, we obtain

$$Z = \frac{11}{16} - \frac{11}{16}X_1 - \frac{11}{16}X_2 + \frac{18}{16}X_1X_2 \qquad (9)$$

Since X_1 and X_2 are independent, the term X_1X_2 can be written as $p_{\mathbf{X}_1}(1)\, p_{\mathbf{X}_2}(1) = p_{\mathcal{X}_V}(1,1)$, where $\mathcal{X}_V = \{\mathbf{X}_1, \mathbf{X}_2\}$. Furthermore, we can "demarginalize" the marginal probabilities by using $X_1 = p_{\mathcal{X}_V}(1,0) + p_{\mathcal{X}_V}(1,1)$ and $X_2 = p_{\mathcal{X}_V}(0,1) + p_{\mathcal{X}_V}(1,1)$. Replacing X_1, X_2 and X_1X_2 in Eq. (9) with these probabilities yields a unipolar stochastic function to which we can apply CEASE.

$$Z = F(f, p_{\mathcal{X}_V}) = \frac{11}{16} \cdot p_{\mathcal{X}_V}(0,0) + \frac{7}{16} \cdot p_{\mathcal{X}_V}(1,1) \qquad (10)$$

Eq. (10) is the unipolar or probability interpretation of Eq. (8) with coefficients in [0, 1]. This fact can also be directly seen from the ReSC design C_{RE} in Fig. 6*b*, which outputs 11/16 and 7/16 when the input pattern is 00 and 11, respectively.

A CEASE design C_C implementing Eq. (8) in the IBP domain and Eq. (10) in the unipolar domain is given in Fig. 6*c*. This is a constant-free sequential circuit built around a modulo-16 counter, which adds 11 or 7 to its count state on receiving a 00 or 11, respectively, and it remains in the same state on receiving a 01 or 10. Whenever the counter overflows, a 1 is produced at the output, and the counter is reset to the amount of the overflow. C_C requires four flip-flops for its 16-state counter. C_{ST} shown in Fig. 6*a*, requires four constant SNs that are generated by a 4-tap LFSR, which also needs four flip-flops. However, C_{ST} has the limitation that each tap of the LFSR does not produce a constant with value exactly 0.5, because it does not loop through the all-0 state, resulting in the constant 8/15 instead of 0.5. To eliminate this small error, C_{ST} would require random sources that are more accurate and probably costlier than a 4-bit LFSR. C_{RE}, besides its expensive SN generators, also needs two high-quality 4-bit random sources (omitted in Fig. 6*b*) for B_{R1} and B_{R3}.

An MSE comparison of the above three circuits is given in Fig. 7. Here we use MATLAB's *rand* function to generate high-quality random numbers for the ReSC design C_{RE}. The STRAUSS design C_{ST} does not converge to the correct value due to the error introduced by the LFSR's missing all-0's state; this error may be removed by replacing the LFSR with higher-quality random number sources. The CEASE circuit C_C, on the other hand, consistently provides the best accuracy among all the designs, and its MSEs match the theoretical lower bound predicted by Theorem 3. This implies that C_C can compute in far less time, and hence with better energy efficiency, than the other designs. For example, C_C achieves an MSE of 0.002 with N = 32 bits, while the ReSC design C_{RE} needs approximately 128 bits for the same accuracy.

B. Complex Matrix Multiplication

Fig. 8*a* shows a stochastic circuit with 12 constants implementing complex matrix multiplication [12]. It has four outputs, each of which depends on three constant inputs, all of which can be eliminated by CEASE. Here we show the accuracy improvement after applying CEASE to the sub-circuit spanned by $\mathbf{Z}_i^1$, one of the circuit's four primary outputs. The resulting STG has four states, which require two flip-flops to implement. The CEASE circuit is similar in structure to that in Fig. 6*c*.

An MSE comparison of the circuit in Fig. 8*a* and the CEASE circuit is shown in Fig. 8*b*, which again shows that CEASE improves accuracy effectively and at the same time matches the theoretical MSE lower bound.

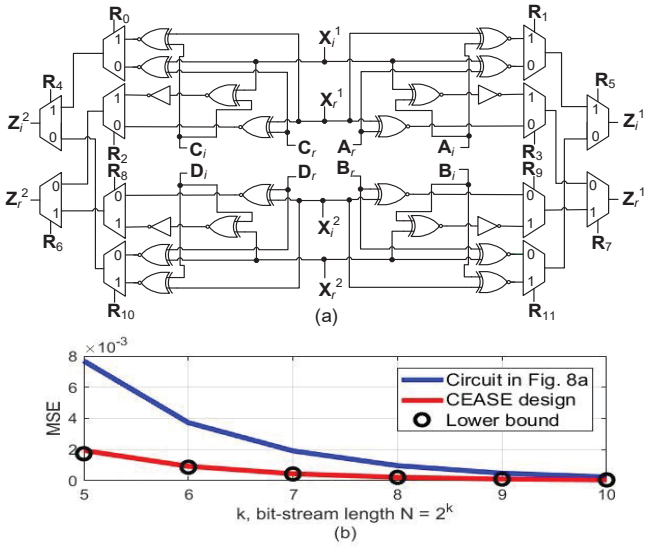

Figure 8. (a) Stochastic circuit implementing complex matrix multiplication [12]. (b) MSE comparison between the circuit in (a) and the circuit generated by CEASE.

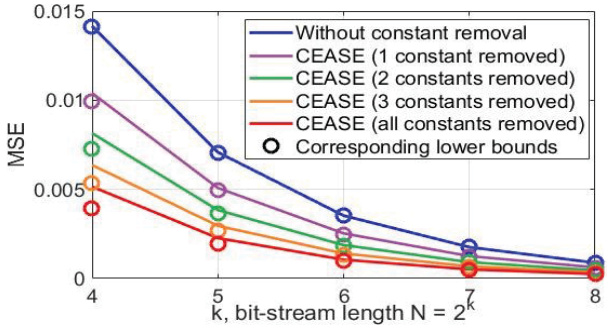

Figure 9. MSE comparison for random circuits with four constant and two variable input SNs. The lower bounds are computed by treating the unremoved constants as variables.

C. Random Circuits

In the absence of benchmark stochastic circuits, we use randomly generated circuits to further estimate the performance of CEASE. Specifically, we first generate 100,000 random functions in the form of Eq. (5) that are implementable using four-constant, two-variable stochastic circuits, where the constants all have value 0.5 and the variable inputs are fed with random values. We then apply CEASE to the circuits implementing these random functions. Fig. 9 plots the average MSEs of these circuits against bit-stream length. We also allow CEASE to remove some or all the constants. As can be seen in Fig. 9, the MSEs depend on the number of constants removed, with the lowest MSEs achieved by removing all the constants. The results match the theoretical lower bounds, with slight deviations caused by rounding very short SNs.

V. Conclusions

We have clarified the role of constant SNs in stochastic circuits, and shown that, while such constants are essential in practical SC design, they are an unexpected source of significant amounts of random fluctuation errors. We further demonstrated that constant inputs can be completely eliminated by employing sequential stochastic circuits. A systematic algorithm CEASE was devised for efficiently removing constants in this way. We proved analytically the optimality of CEASE in terms of RFE reduction. Experimental results were presented which confirm that with fixed computation time (and hence fixed energy consumption), constant-free sequential designs of the kind generated by CEASE can greatly improve the accuracy of SC.

Acknowledgment: This work was supported by Grant CCF-1318091 from the U.S. National Science Foundation.

VI. Appendix

A. Proof of Theorem 1.

By classifying SN inputs into variable and constant parts as in $\boldsymbol{\mathcal{X}} = \{\boldsymbol{\mathcal{X}}_V, \boldsymbol{\mathcal{X}}_C\}$, Eq. (3) can be re-written as:

$$Z = F(f, p_{\boldsymbol{\mathcal{X}}_V, \boldsymbol{\mathcal{X}}_C}) = \sum_{\boldsymbol{b}_V, \boldsymbol{b}_C}\left[f(\boldsymbol{b}_V, \boldsymbol{b}_C) p_{\boldsymbol{\mathcal{X}}_V, \boldsymbol{\mathcal{X}}_C}(\boldsymbol{b}_V, \boldsymbol{b}_C)\right] \quad (11)$$

Using the properties of conditional probability, we can re-write $p_{\boldsymbol{\mathcal{X}}_V, \boldsymbol{\mathcal{X}}_C}(\boldsymbol{b}_V, \boldsymbol{b}_C)$ as $p_{\boldsymbol{\mathcal{X}}_C|\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_C|\boldsymbol{b}_V) \cdot p_{\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_V)$, where the term $p_{\boldsymbol{\mathcal{X}}_C|\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_C|\boldsymbol{b}_V)$ is a function of $\boldsymbol{b}_C$ and $\boldsymbol{b}_V$. Eq. (11) then becomes

$$Z = \sum_{\boldsymbol{b}_V, \boldsymbol{b}_C}\left[f(\boldsymbol{b}_V, \boldsymbol{b}_C) p_{\boldsymbol{\mathcal{X}}_C|\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_C|\boldsymbol{b}_V) \cdot p_{\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_V)\right]$$

$$= \sum_{\boldsymbol{b}_V}\left[p_{\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_V) \cdot \sum_{\boldsymbol{b}_C}\left[f(\boldsymbol{b}_V, \boldsymbol{b}_C) \cdot p_{\boldsymbol{\mathcal{X}}_C|\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_C|\boldsymbol{b}_V)\right]\right] \quad (12)$$

The summation $\sum_{\boldsymbol{b}_C}\left[f(\boldsymbol{b}_V, \boldsymbol{b}_C) \cdot p_{\boldsymbol{\mathcal{X}}_C|\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_C|\boldsymbol{b}_V)\right]$ is over all combinations of $\boldsymbol{b}_C$, and hence $g(\boldsymbol{b}_V) = \sum_{\boldsymbol{b}_C}\left[f(\boldsymbol{b}_V, \boldsymbol{b}_C) \cdot p_{\boldsymbol{\mathcal{X}}_C|\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_C|\boldsymbol{b}_V)\right]$ does not depend on $\boldsymbol{b}_C$, so we can re-write Eq. (12) as $Z = F(p_{\boldsymbol{\mathcal{X}}_V}) = \sum_{\boldsymbol{b}_V}\left[g(\boldsymbol{b}_V) \cdot p_{\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_V)\right]$ which is linear in $p_{\boldsymbol{\mathcal{X}}_V}(\boldsymbol{b}_V)$ with all coefficients $g(\boldsymbol{b}_V)$ in the range [0,1]. The dependency of $F(p_{\boldsymbol{\mathcal{X}}_V})$ on f and $p_{\boldsymbol{\mathcal{X}}_C}$ is implicit via $g(\boldsymbol{b}_V)$ only.

B. Proof Outline of Theorem 3.

Let N be the SN length, and let N_i be the number of bit-pattern $\boldsymbol{b}_i$ received by a CEASE circuit C. Suppose C has q states, and $[a_1, a_2, \ldots, a_m] = q[g_1, g_2, \ldots, g_m]$ are the numbers of states that C jumps forward on receiving bit pattern $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_m$, respectively. Hence, the total number of states that C will jump forward after receiving the N-bit SNs will be $\sum_{i=1}^{m} a_i N_i = \sum_{i=1}^{m} q g_i N_i$. The number of 1s in the output $\mathbf{Z}$ is $\left\lfloor \frac{1}{q}\sum_{i=1}^{m} q g_i N_i \right\rfloor = \lfloor \sum_{i=1}^{m} g_i N_i \rfloor = \sum_{i=1}^{m} g_i N_i - \epsilon$, where $\epsilon \in [0, 1)$ is an offset term that takes into account the floor operation. The estimated value of $\mathbf{Z}$ is $\hat{Z} = \frac{1}{N}[\sum_{i=1}^{m} g_i N_i - \epsilon] = \sum_{i=1}^{m} g_i \frac{N_i}{N} - \frac{\epsilon}{N} = \hat{Z}_u - \frac{\epsilon}{N}$, where $\frac{\epsilon}{N} \in [0, \frac{1}{N})$ is the rounding error which, in the worst case, can only cause less than a 1-bit difference in $\mathbf{Z}$. $\hat{Z}_u = \sum_{i=1}^{m} g_i \frac{N_i}{N}$, on the other hand, is an unbiased estimate of Z which achieves the Cramér–Rao bound, a lower bound on MSE for an unbiased estimator [10]. (The proof that $\hat{Z}_u$ achieves this bound is omitted due to space limitations.) Summarizing, we conclude that C has the minimum MSE among all designs up to a rounding error.

References

[1] Alaghi, A. et al. "Stochastic circuits for real-time image-processing applications." *Proc. DAC*, Art.136, 2013.
[2] Alaghi, A. and Hayes, J.P. "STRAUSS: spectral transform use in stochastic circuit synthesis." *IEEE Trans. CAD*, 34, pp.1770-1783, 2015.
[3] Alaghi, A. and Hayes, J.P. "On the functions realized by stochastic computing circuits." *Proc. GLSVLSI*, pp.331-336, 2015.
[4] Braendler, D. et al. "Deterministic bit-stream digital neurons." *IEEE Trans. Neural Nets.*, 13, pp. 1514-1525, 2002.
[5] Brown, B. D. and Card, H. "Stochastic neural computation I." *IEEE Trans. Computers*, 50, pp.891-905, 2001.
[6] Chen, T.-H. and Hayes, J.P. "Equivalence among stochastic logic circuits and its application to synthesis." *IEEE Trans. Emerging Topics in Computing*, 2016. IEEE Xplore early access article.
[7] Gaudet, V.C. and Rapley, A.C. "Iterative decoding using stochastic computation." *Electron. Letters*, 39, pp.299-301, 2003.
[8] Gupta, P.K. and Kumaresan, R. "Binary multiplication with PN sequences." *IEEE Trans. ASSP*, 36, pp. 603-606, 1988.
[9] Jenson, D. and Riedel, M. "A deterministic approach to stochastic computation." *Proc. ICCAD*, pp. 1-8, 2016.
[10] Keener, R. W. *Theoretical Statistics: Topics for a Core Course*. Springer, 2010.
[11] Lee, V. T. et al. "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing." *Proc. DATE*, 2017.
[12] Paler, A. et al. "Approximate simulation of circuits with probabilistic behavior." *Proc. DFTS,* pp.95-100, 2013.
[13] Qian, W. et al. "An architecture for fault-tolerant computation with stochastic logic." *IEEE Trans. Comp.*, 60, pp.93-105, 2011.
[14] Ting, P.-S. and Hayes, J.P. "Isolation-based decorrelation of stochastic circuits." *Proc. ICCD*, pp.88-95, 2016.
[15] Vahapoglu, E. and Altun, M. "Accurate synthesis of arithmetic operations with stochastic logic."*Proc. ISVLSI*, pp.415-420, 2016.