IET Computers & Digital Techniques

Special Issue: Defect and Fault Tolerance in VLSI and Nanotechnology Systems

IET Journals
The Institution of Engineering and Technology

Removing constant-induced errors in stochastic circuits

ISSN 1751-8601
Received on 22nd March 2018
Revised 26th October 2018
Accepted on 20th November 2018
E-First on 19th March 2019
doi: 10.1049/iet-cdt.2018.5017
www.ietdl.org

Paishun Ting[1] ✉, *John P. Hayes*[1]

[1]*Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 48109, USA*

✉ *E-mail: paishun@umich.edu*

Abstract: Stochastic computing (SC) computes with probabilities using randomised bit-streams and standard logic circuits. Its major advantages include 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, the authors first observe that many SC designs heavily rely on constant inputs, which contribute significantly to RFEs. They then investigate the role of constant inputs in SC and propose a systematic algorithm constant elimination algorithm for suppression of errors to eliminate the constant-induced RFEs by introducing memory elements into the target circuits. The resulting optimal modulo-counting (OMC) circuits remove all constant inputs and, at the same time, minimise RFEs. Analytical and experimental results are presented demonstrating other aspects of OMC circuits, including their initialisation and autocorrelation properties, as well as their optimality in terms of minimising RFEs.

1 Introduction

Stochastic computing (SC) is an unconventional digital design technique that interprets signals as probability values embedded in randomised 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 $\boldsymbol{X} = 010011$ is a six-bit SN with (unipolar) value 0.5, since three of its six bits are 1, i.e. the probability of a 1 appearing anywhere in $\boldsymbol{X}$ is 0.5. We denote the value of $\boldsymbol{X}$ by $p_{X^{(t)}}(1) = X = 0.5$. $\boldsymbol{X}^{(t)}$ denotes the tth bit of $\boldsymbol{X}$. Note that when clear from the context, we drop the superscript t for brevity, and simply write $p_{X^{(t)}}(1) = p_{\boldsymbol{X}}(1)$. SC performs arithmetic on probabilities by transforming a set of SNs.

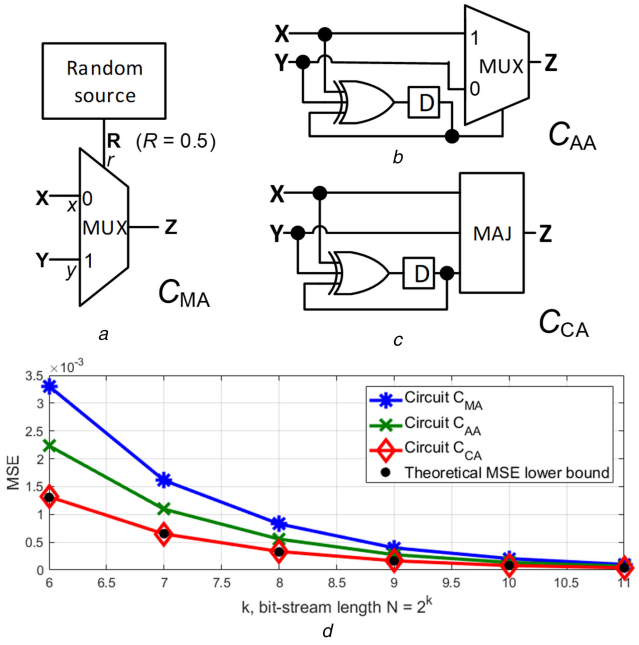

Fig. 1 *Three stochastic implementations of scaled addition*
(a) Conventional MUX-based design C_{MA} with a constant input $R = 0.5$, ***(b)*** *Ad hoc* sequential design C_{AA} with no constant input, ***(c)*** Sequential design C_{CA} produced by CEASE, ***(d)*** Error comparison of the three designs

The scaled adder C_{MA} in Fig. 1*a* illustrates SC's basic mechanism. The multiplexer (MUX) takes two variable SNs $\boldsymbol{X}$ and $\boldsymbol{Y}$, and an independent constant SN $\boldsymbol{R}$ of value $R = 0.5$ as inputs and outputs the SN $\boldsymbol{Z}$. In any clock cycle t, the probability of $\boldsymbol{Z}^{(t)} = 1$ is the probability of $(\boldsymbol{X}^{(t)}, \boldsymbol{R}^{(t)}) = (1, 0)$, plus the probability of $(\boldsymbol{Y}^{(t)}, \boldsymbol{R}^{(t)}) = (1, 1)$. With $R = 0.5$, it is easily seen that $Z = 0.5(X + Y)$, a scaled sum of X and Y. The scaling here ensures that Z lies in the probability range or unit interval [0, 1].

SC's advantages of low power, low area cost and tolerance for soft errors show considerable promise in applications such as image processing [1], error-correcting code decoding [2], and machine learning [3]. However, SC suffers from various error sources including undesired correlations and random fluctuations. Correlation is due to inadequate independence among the SNs being processed. It may be tackled via decorrelation circuitry, which can add significantly to area cost [4]. Random fluctuation errors (RFEs) occur when the SN length N is too small or when the quality of the randomness sources is poor [5]. 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 later, while these circuits implement the same scaled addition function, they have quite different RFE levels. RFEs can be reduced by increasing N, but this can lead to 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.

Several prior works reduce or eliminate correlation errors and RFEs by introducing some degree of determinism into the design of stochastic-type circuits [5–8]. For example, low-discrepancy quasi-random sources such as Halton sequences can be used to generate accurate but deterministic bit-streams [9]. While deterministic bit-streams can improve accuracy, there are many situations where conventional SC is preferable. For example, the input pixels of the retina-implant chip described in [1] are continuously sensed from the environment and are thus already stochastic and dynamically changing over time. More generally, when connecting SC components, it is far easier to maintain the bit-stream randomness required by conventional SC than to enforce

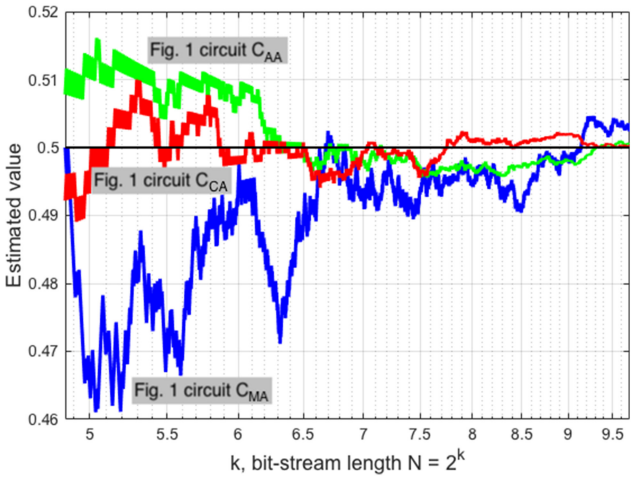

Fig. 2 *Typical random fluctuations in three SNs with the same exact value 0.5 as bit-stream length N increases*

rigid deterministic patterns. Furthermore, converting stochastic bit-streams to deterministic ones and vice versa can significantly increase hardware cost and system latency. For these reasons, we only consider SC with random number representations.

The main contribution of this study is a new design methodology to reduce RFEs in conventional SC, and improve accuracy without compromising computation time or 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 such as Spectral Transform Use in Stochastic circuit Synthesis (STRAUSS) [10] and reconfigurable architecture based on stochastic logic (ReSC) [11], 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 here, also turn out to be a significant source of 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 $\boldsymbol{Z}$ is defined as

$$\mathrm{MSE}(\boldsymbol{Z}, N) = \mathbb{E}\left[\left(\hat{Z} - Z\right)\right]^2, \tag{1}$$

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

We show that it is possible to remove all RFE-inducing input constants by resorting to a new class of sequential SC designs. We devise a systematic method, which we call constant elimination algorithm for suppression of errors (CEASE) [12] for constant removal. While a function may have various circuit implementations with no constant inputs, CEASE-generated circuits provide a guarantee of optimality on RFE reduction. Figs. 1*b* and *c* depict two sequential scaled adder designs C_{AA} and C_{CA} after eliminating constant SNs; the former was designed in *ad hoc* fashion, while the latter is based on CEASE. (It happens to include a sub-circuit that implements the majority function MAJ.) Fig. 1*d* plots the (sampled) MSEs of all three scaled adders in Fig. 1 against bit-stream length $N = 2^k$. It can be seen that the CEASE-based design C_{CA} is the most accurate. Furthermore, as we will demonstrate, C_{CA} meets the theoretical lower bound on MSE for RFEs indicated by the small circles. Note here that this MSE bound is the lower bound for 'conventional' SC that deals with Bernoulli bit-streams. The CEASE design C_{CA} achieves this bound by removing all the inherent RFEs induced by constants that control the circuit's function. The remaining RFEs are from user-supplied inputs which are not controllable. Notice too that if the user applies accurate inputs (such as low-discrepancy bit-streams) that have no RFEs, the CEASE circuit will correspondingly generate exact outputs up to rounding errors. In other words, CEASE is compatible with non-stochastic bit-streams and can provide further accuracy improvements.

The main contributions of this study are as follows:

- 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 function to memory.
- A new class of optimal modulo-counting (OMC) circuits which implement CEASE and minimise RFEs.
- An analysis of the properties of OMC circuits.

The paper is organised as follows. Section 2 briefly introduces SC and examines the role played by constant SNs in stochastic circuits. Section 3 introduces the CEASE method and OMC circuits, and analyses their performance. Section 4 examines additional properties of OMC circuits. Section 5 discusses applications of these circuits, while Section 6 draws some conclusions.

2 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 realised by C, as well as the SN values applied to C and their joint probability distribution.

Example 1: Combinational SC adder. The adder C_{MA} in Fig. 1*a* has three SNs $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{R}$ applied to its inputs x, y, r, respectively. It implements the Boolean function $f_{\mathrm{mux}}(x, y, r)$, which outputs a 0 bit except when $f_{\mathrm{mux}}(1, 0, 0) = f_{\mathrm{mux}}(0, 1, 1) = f_{\mathrm{mux}}(1, 1, 0) = f_{\mathrm{mux}}(1, 1, 1) = 1$. 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. This enables us to write

$$\begin{aligned} Z &= p_{\mathcal{X}}(1,0,0) + p_{\mathcal{X}}(0,1,1) + p_{\mathcal{X}}(1,1,0) + p_{\mathcal{X}}(1,1,1) \\ &= \sum_{\boldsymbol{b}} [f_{\mathrm{mux}}(\boldsymbol{b}) p_{\mathcal{X}}(\boldsymbol{b})], \end{aligned} \tag{2}$$

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

In general, suppose a combinational circuit C implements the Boolean function $z = f(x_1, x_2, \ldots, x_n)$. Let $\mathcal{X} = \{\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n\}$ 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 realised by C has the following form [13]:

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

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 $\boldsymbol{b}$. Equation (3) indicates that a stochastic function is a linear combination of the probability terms $p_{\mathcal{X}}(\boldsymbol{b})$ with coefficients $f(\boldsymbol{b})$ that take 0–1 values. Equation (3) generalises (2) from the special case of a scaled adder to an arbitrary combinationally implementable stochastic function.

Many stochastic circuits, including those synthesised by STRAUSS and ReSC, heavily use constant input SNs to define both the target function's value and its precision. For example, in Fig. 1*a*, the input SN $\boldsymbol{R}$ 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 $\boldsymbol{X}$ and $\boldsymbol{Y}$. In such cases, we can separate the input SNs $\mathcal{X}$ into two disjoint subsets $\mathcal{X} = \{\mathcal{X}_{\mathrm{V}}, \mathcal{X}_{\mathrm{C}}\}$, where $\mathcal{X}_{\mathrm{V}}$ and $\mathcal{X}_{\mathrm{C}}$ denote variables and constants, respectively [14]. A stochastic function of $p_{\mathcal{X}_{\mathrm{V}}}$ can then be derived from (3) by replacing $\mathcal{X}_{\mathrm{C}}$ with appropriate constant values, as the following example illustrates.

Example 1: (cont.): Returning to the MUX-based adder C_{MA} in Fig. 1*a*, let $\mathcal{X} = \{\mathcal{X}_{\mathrm{V}}, \mathcal{X}_{\mathrm{C}}\}$, where $\mathcal{X}_{\mathrm{V}} = \{\boldsymbol{X}, \boldsymbol{Y}\}$ and $\mathcal{X}_{\mathrm{C}} = \{\boldsymbol{R}\}$ with $R = 0.5$. If $\boldsymbol{R}$ is independent of $\mathcal{X}_{\mathrm{V}}$, then on substituting $p_{\boldsymbol{R}}(1) = R = 0.5$ into (2), we get

$$\begin{aligned} Z &= 0.5\left[p_{\mathcal{X}_{\mathrm{V}}}(1,\ 0) + p_{\mathcal{X}_{\mathrm{V}}}(0,\ 1) + p_{\mathcal{X}_{\mathrm{V}}}(1,\ 1) + p_{\mathcal{X}_{\mathrm{V}}}(1,\ 1)\right] \\ &= 0.5[p_{\boldsymbol{X}}(1) + p_{\boldsymbol{Y}}(1)] = 0.5(X + Y)\,. \end{aligned} \tag{4}$$

which is the expected scaled addition function of X and Y.

Looking more closely at (4), we can see that it is a linear combination of probability terms with *non-binary coefficients*, in contrast to (2) and the more general (3) which allow only the binary coefficients 0 and 1 for their probability terms. This suggests that constant inputs enable a stochastic function to have coefficients that are any rational numbers in the range [0, 1], which denotes the unit interval. The following theorem generalises this observation.

Theorem 1: The stochastic function realised by a combinational circuit with input SNs $\mathcal{X} = \{\mathcal{X}_{\mathrm{V}}, \mathcal{X}_{\mathrm{C}}\}$ is

$$Z = F\left(p_{\mathcal{X}_{\mathrm{V}}}\right) = \sum_{\boldsymbol{b}_{\mathrm{V}}} \left[\mathrm{g}(\boldsymbol{b}_{\mathrm{V}}) \cdot p_{\mathcal{X}_{\mathrm{V}}}(\boldsymbol{b}_{\mathrm{V}})\right], \tag{5}$$

where the $g(\boldsymbol{b}_{\mathrm{V}})$s are rational constants in the range [0, 1] that depend implicitly on f and $p_{\mathcal{X}_{\mathrm{C}}}$.

A proof of this theorem can be found in the 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 $\boldsymbol{X}$ and $\boldsymbol{Y}$, 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 $\boldsymbol{R}$ of value 0.5. In general, when a target function is not already in the form of (5), it has to be converted to that form by introducing suitable constants. The accuracy with which the resulting function approximates the target function F 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 typically comes with many RFE-inducing constant inputs. For instance, the STRAUSS implementation of $Z = (7/16) - (1/4)X - (9/16)X^2$ derived in [10] employs four constant inputs $\boldsymbol{R}_1$, $\boldsymbol{R}_2$, $\boldsymbol{R}_3$, and $\boldsymbol{R}_4$, each of value 0.5. These constants require costly randomness sources to generate them, and are, as we will demonstrate, a significant source of RFEs.

To eliminate constant inputs and their associated errors, we propose CEASE, a systematic algorithm for removing constants while keeping their functional benefits. Specifically, CEASE transforms a target combinational function into an equivalent stochastic finite-state machine (FSM) 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.

3 Constant elimination

This section details CEASE [12], the proposed constant SN elimination algorithm, along with an analysis of its basic accuracy.

Example 1: (cont.): We re-examine the MUX-based adder $C_{\mathrm{MA}}$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 $\boldsymbol{Z}$ should be half the number of 1s in the input SNs $\boldsymbol{X}$ and $\boldsymbol{Y}$. For a given cycle t, when both $\boldsymbol{X}^{(t)}$ and $\boldsymbol{Y}^{(t)}$ are 1, the corresponding output bit $\boldsymbol{Z}^{(t)}$ will be 1;

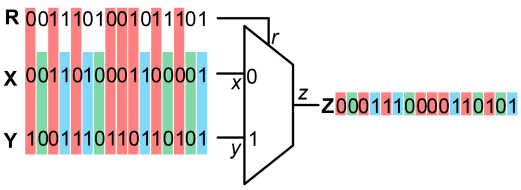

Fig. 3 *MUX-based stochastic scaled adder C_{MA}. On receiving 11 or 00, the output is 1 or 0, respectively. On receiving 01 or 10, the output is 1 with probability 0.5*

this is exact since a single 1 should be produced in $\boldsymbol{Z}$ whenever the circuit receives two 1s. Similarly, when both $\boldsymbol{X}^{(t)}$ and $\boldsymbol{Y}^{(t)}$ are 0, $\boldsymbol{Z}^{(t)}$ will have the exact value 0. When only one of the two inputs is 1, i.e. $\boldsymbol{X}^{(t)}\boldsymbol{Y}^{(t)} = 10$ or 01, (6) implies that ideally, $\boldsymbol{Z}^{(t)}$ should be 0.5. However, $\boldsymbol{Z}^{(t)}$ 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 $\boldsymbol{R}$ whose stochastic value 0.5 ensures that on average a single 1 is generated in $\boldsymbol{Z}$ whenever two copies of 10 or 01 are received. In other words, a 1 and a 0 are 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 $\boldsymbol{Z}$.

The fact that constants produce extra RFEs can be seen from Fig. 3, where the constant $\boldsymbol{R}^{(t)}$ is used to select inputs whenever $\boldsymbol{X}^{(t)}\boldsymbol{Y}^{(t)} = 10$ or 01. Notice here that all of these 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 $\boldsymbol{R}$. In this example, there are only three 1s instead of four in the eight output bits selected by $\boldsymbol{R}$, producing a 1/16 error in the output value. The key to eliminating $\boldsymbol{R}$ (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 and one 0 for every two applications of input patterns 01 or 10.

3.1 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 (3) where only binary coefficients are allowed. CEASE circumvents this issue and at the same time reduces RFEs by constructing an equivalent FSM. The idea behind CEASE is to introduce memory elements to count and remember non-binary values; the resulting FSM accumulates such values to be output later. When an accumulated value exceeds one, a 1 is sent to the output.

Example 2: Sequential SC adder. The state-transition graph (STG) of the CEASE-generated scaled adder C_{CA} (Fig. 1*c*) is shown in Fig. 4*b*. Like the combinational adder in Fig. 3, C_{CA} 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_{CA} 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 generated whenever *exactly* two copies of 10 or 01 are received. Hence, the RFEs introduced by constant SNs are completely removed.

Fig. 5 gives a pseudo-code summary of CEASE. In general, CEASE takes as its input a target arithmetic function F in the form of (5) and generates an FSM M_C realising F without constant inputs. If F is not already in the form of (5), it must be approximated by using an SC synthesiser such as STRAUSS. M_C can be realised in various ways. We refer to any circuit that implements M_C as an *optimal modulo counting* (*OMC*) circuit C_{OMC}. An OMC circuit implements M_C by keeping a running sum of each non-binary input value of interest. Whenever the accumulated sum exceeds one, the circuit outputs a 1 and stores the overflow amount.

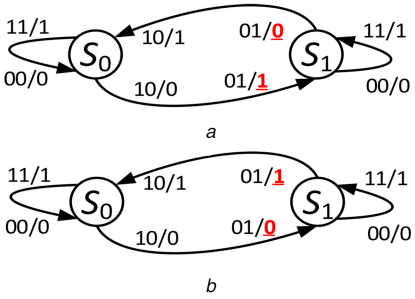

Fig. 4 *STGs for the sequential scaled adders corresponding to* ***(a)*** *Ad hoc* design C_{AA} of Fig. 1*b*, ***(b)*** CEASE-generated design C_{CA} of Fig. 1*c*. The differences between the two STGs are underlined in figure

Input:	A stochastic function F in the form of Eq. (5)
Output:	An FSM M_C that implements F
Step 1.	Find the least common multiple q of the denominators of F's coefficients $\boldsymbol{g} = \{g(\boldsymbol{b}_1), g(\boldsymbol{b}_2), \ldots, g(\boldsymbol{b}_m)\}$.
Step 2.	Compute $\boldsymbol{a} = \{a(\boldsymbol{b}_1), a(\boldsymbol{b}_2), \ldots, a(\boldsymbol{b}_m)\} = q \cdot \boldsymbol{g}$.
Step 3.	Construct a modulo-q counter *CNT* with states $s_0, s_1, \ldots, s_{q-1}$.
Step 4.	Modify *CNT* 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 *CNT* overflows, otherwise it outputs 0. The resulting FSM is M_C.

Fig. 5 *Algorithm CEASE for constant elimination*

Example 2: Sequential SC adder (cont.) Consider again the scaled addition function $Z = 0.5(X + Y)$. Equation (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 $\{g(0,0), g(0,1), g(1,0), g(1,1)\} = \{0/2, 1/2, 1/2, 2/2\}$. The least 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 $\boldsymbol{X}^{(t)}\boldsymbol{Y}^{(t)} = 10$ or 01 is applied, the counter adds 1 to its state. The pattern $\boldsymbol{X}^{(t)}\boldsymbol{Y}^{(t)} = 11$ 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 for the FSM of an exactly scaled adder, with the circuit C_{CA} in Fig. 1*c* being one of its many possible OMC implementations.

The three SC synthesisers mentioned in this paper, STRAUSS, ReSC and CEASE, all have similar algorithmic complexity. Each requires a functional approximation if the target function is not directly implementable by its methods. In the CEASE case, the function must be in the form of (5). Then, an SC design that implements the approximating function will be generated. This step converts (5) to a specific modulo counter, and can usually be done extremely fast compared to the function approximation step.

Another viewpoint on the validity of the scaled adder in Fig. 4*b* is its behaviour under steady-state probability distribution. It is easy to see that the long-term probabilities of staying in states s_0 and s_1 are equal, i.e. $p_S(s_0) = p_S(s_1) = 1/2$, since the state transition behaviour 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

$$\begin{aligned} Z &= p_S(s_0)p_{\mathcal{X}_V}(1, 1) + p_S(s_1) \\ &\quad \left[p_{\mathcal{X}_V}(1, 1) + p_{\mathcal{X}_V}(0, 1) + p_{\mathcal{X}_V}(1, 0)\right] \\ &= 0.5[p_X(1) + p_Y(1)] = 0.5(X + Y), \end{aligned}$$

which is indeed the scaled addition function.

Not surprisingly, a CEASE-generated FSM M_C implements a stochastic function in the form of (5) accurately (up to an unavoidable rounding error). Depending on the rounding policy, rounding may produce a 1-bit error when the SN length 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. The following theorem, whose proof is given in the Appendix, formalises this assertion.

Theorem 2: For an n-input, q-state CEASE-generated FSM M_C with N-bit SNs, the number of 1s in its output SN $\boldsymbol{Z}$ is

$$N^{\boldsymbol{Z},1} = \sum_{i=1}^{m} \frac{a_i}{q} N_i + \tilde{\epsilon}, \tag{7}$$

where N_i is the number of bit-pattern $\boldsymbol{b}_i$ received by M_C, while a_i is the number of states M_C advances when a $\boldsymbol{b}_i$ pattern is received and $i \in \{1, 2, \ldots, m = 2^n\}$. The residual (non-outputted) error due to rounding is $\tilde{\epsilon} = \frac{a_0}{q} - \epsilon$, where a_0 is such that s_{a_0} is the initial state of M_C and $\varepsilon \in [0, 1)$. Furthermore, $\boldsymbol{Z}$'s value is

$$Z = \sum_{i=1}^{m} \frac{a_i}{q} p_{\mathcal{X}_V}(\boldsymbol{b}_i) + \frac{1}{N}\mathbb{E}(\tilde{\epsilon}), \tag{8}$$

where $p_{\mathcal{X}_V}(\boldsymbol{b}_i)$ is the probability of M_C receiving the input pattern $\boldsymbol{b}_i$.

Next, we give an intuitive explanation of (7) and (8). In (7), the number of 1s in the output $N^{\boldsymbol{Z},1}$ reflects the number of times that the modulo counter goes beyond 1 and overflows. The modulo counter accumulates the value (a_i/q) for each pattern $\boldsymbol{b}_i$ it receives, and there are N_i occurrences of the pattern $\boldsymbol{b}_i$. Therefore, $(a_0/q) + \sum_{i=1}^{m}(a_i/q)N_i$ is the total value accumulated in the modulo counter at the end, where (a_0/q) accounts for the value initially stored in the counter and can be adjusted by configuring the counter's initial state. Since the number of 1s in $\boldsymbol{Z}$ must be an integer, $N^{\boldsymbol{Z},1}$ must be $\sum_{i=1}^{m}(a_i/q)N_i + (a_0/q)$. In (7), this floor operation is captured by the term $\tilde{\epsilon}$, which depends on the initial state of the counter and must be less than one, because the residual (non-overflow) value stored in the counter must be less than one. To go from (7) to (8), first note that the value of $\boldsymbol{Z}$ is the expected number of 1s in $\boldsymbol{Z}$ divided by N. Consequently, $Z = \mathbb{E}\left(\sum_{i=1}^{m}(a_i/q)(N_i/N) + (\tilde{\epsilon}/N)\right) = \sum_{i=1}^{m}(a_i/q)\mathbb{E}(N_i/N) + (\mathbb{E}(\tilde{\epsilon})/N) = \sum_{i=1}^{m}(a_i/q)p_{\mathcal{X}_V}(\boldsymbol{b}_i) + (1/N)\mathbb{E}(\tilde{\epsilon})$, since $\mathbb{E}(N_i/N)$, the expected fraction of the pattern $\boldsymbol{b}_i$ in the input lines is the probability that $\boldsymbol{b}_i$ occurs.

In Theorem 2, a_i is the number of states M_C advances on receiving $\boldsymbol{b}_i$ and q is the total number of states in M_C. This clearly implies that $a_i \leq q$ for all i, so in (8) the coefficients $(a_i/q) \in [0,1]$ are indeed rational numbers in the unit interval. Comparing (8) with (5), we see that M_C implements a stochastic function in the form of (5), the class of functions combinationally implementable with constant inputs. These functions are exact up to an unavoidable rounding error $(1/N)\mathbb{E}(\tilde{\epsilon})$, which, in the worst case, can only cause a 1-bit difference in $N^{\boldsymbol{Z},1}$.

3.2 Accuracy of CEASE

We now consider the role of CEASE in RFE reduction. The next theorem says that among all possible FSMs implementing F, CEASE produces a result with the smallest MSE, i.e. the output SN produced by a CEASE-derived OMC circuit fluctuates least.

Theorem 3: Given a stochastic function $F(p_{\mathcal{X}_V})$ in the form of (5), suppose the members of $\mathcal{X}_V$ are Bernoulli bit-streams with unknown correlations. Then for all positive integers N

$$\mathrm{MSE}(\boldsymbol{Z}_C, N) \lesssim \mathrm{MSE}(\boldsymbol{Z}, N), \tag{9}$$

where $\boldsymbol{Z}_C$ is the SN produced by a CEASE-generated OMC circuit that implements F, while $\boldsymbol{Z}$ is the SN produced by any other design that implements F.

The notation $\lesssim$ in (9) indicates that inequality holds up to a rounding error. 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, and hence minimises MSEs. For comparison, consider the *ad hoc* design C_{AA} in Fig. 1*b*, whose STG is given in Fig. 4*a*. One can easily see that C_{AA} also computes scaled addition like the OMC circuit C_{CA} in Fig. 1*c* whose STG is in Fig. 4*b*. Suppose the following artificially constructed SNs are applied to both C_{AA} and C_{CA}:

$$\boldsymbol{X}_{art} = 010101010101 (X = 6/12 = 0.5),$$

$$\boldsymbol{Y}_{art} = 101010101010 (Y = 6/12 = 0.5).$$

The expected output value should be $0.5(X_{art} + Y_{art}) = 0.5$, which is exactly what C_{CA} will give. However, feeding these two input SNs to C_{AA} initialised to state s_0 will produce the output $\boldsymbol{Z} =$ 111111111111 ($Z = 12/12 = 1$) – a 100% error! The accuracy difference between the two designs is due to the fact that the OMC circuit *guarantees* to output a 1 whenever two copies of 10 or 01 are received, whereas the *ad hoc* design does not. Also, a closer look at $\boldsymbol{X}_{art}$ and $\boldsymbol{Y}_{art}$ reveals that these two bit-streams are highly correlated (in a negative sense) since their 1s never overlap. It is well known that the MUX-based adder is immune to the correlation between its two variable inputs. This desirable property is preserved in the CEASE design C_{CA} and is confirmed by the fact that it accurately computes the scaled sum given the highly correlated inputs $\boldsymbol{X}_{art}$ and $\boldsymbol{Y}_{art}$.

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 causes a relatively small miscount of the applied input patterns, which can then result in a similarly small output error. For instance, if $\boldsymbol{X}_{art}$ is changed to 010101010000 due to two 1-to-0 bit-flips, the output value produced by C_{CA} will become 5/12, which is a good estimate of the exact output value $0.5 = 6/12$.

It is 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 those in [4].

A scaled sequential adder constructed in *ad hoc* fashion around a T flip-flop is given in [15] and shown by simulation to be more accurate than the standard combinational design. The STG of that adder is exactly the same as the CEASE-generated one shown in Fig. 4*b*, implying that this T-flip-flop-based design is also an OMC circuit. This confirms the high accuracy claimed for the T-flip-flop-based adder, a key factor in the success of the neural network implementation in [15].

4 Further analysis of CEASE

This section analyses some important characteristics of CEASE FSMs, including the impact of their initial state, as well as their interaction with autocorrelated bit-streams.

4.1 Impact of initialisation

Like any other FSMs that compute with finite precision, a CEASE design can also have a rounding error due to the representational limitation of finite-length SNs, as briefly described in Section 3. For example, a 4-bit SN can only contain 0, 1, 2, 3 or 4 logical 1s. It cannot contain, say, 2.5 logical 1s to represent the value $2.5/4 = 0.625$ exactly. If the exact value is 0.625, the number of 1s in the SN must be rounded to an integer closest to 2.5, such as 2 or 3. In the CEASE case, this rounding error is reflected by the term $\tilde{\epsilon}$ in (7).

In [15], the authors show that the final value their ad-hoc OMC adder rounds to depend on the adder's initial state and the rounding direction: truncation (rounding down) or rounding up. Next, we generalise their observation to OMC circuits beyond stochastic adders and demonstrate analytically that for an arbitrary OMC circuit, different and desirable rounding policies can be achieved by carefully adjusting the circuit's initial state.

Consider a CEASE FSM M_C with q states. Initialising M_C to state s_0, the first state of the associated modulo counter, is equivalent to truncating the fraction part of the expected number of 1s in the output bit-stream $\boldsymbol{Z}$. This is because M_C only produces a 1 in the output line whenever its modulo counter overflows. Therefore, when M_C finishes a computation in a state $s \neq s_0$, the residual value stored in the modulo counter is discarded. Consider again the CEASE-generated OMC adder C_{CA} whose STG is given in Fig. 4*b*. Here we initialise C_{CA} to its first state s_0, and apply the following two inputs; $\boldsymbol{X} = 00011010$ and $\boldsymbol{Y} = 01100110$. Since $X = 3/8$, and $Y = 4/8 = 1/2$, the expected value for the output $\boldsymbol{Z}$ is $0.5(X + Y) = 0.5 \times (7/8) = 3.5/8$, implying that ideally, the number of 1s in $\boldsymbol{Z}$ should be $N^{\boldsymbol{Z},1} = 3.5$, which is obviously not achievable with digital logic circuits. If we work out the stochastic computation, it is not hard to see that in this case, $\boldsymbol{Z} = 00101010$, which contains three 1s, and the machine terminates at state s_1. The residual value 0.5 stored in the modulo counter state s_1 is consequently discarded, and the actual output value is thus truncated to 3/8.

Suppose we now change the adder C_{CA}'s initial state to s_1, and leave the other conditions unchanged. In this case, $\boldsymbol{Z} = 01010110$, which contains four 1s, and C_{CA} terminates at state s_0. Clearly, C_{CA} now rounds the expected value 3.5/8 to 4/8. This is because initialising C_{CA} to state s_1 implicitly introduces an extra value of 0.5 into the circuit. This circuit will then produce a 1 whenever the residual value is 0.5.

In general, initialising M_C to the state $S^{(0)} = s_{a_0}$ implements the following rounding policy with respect to $N^{\boldsymbol{Z},1}$: 'discard a residual value if it is $< 1 - (a_0/q)$, and add a carry of 1 otherwise'. This follows from the fact that $N^{\boldsymbol{Z},1} = \sum_{i=1}^{m} (a_i/q) N_i + (a_0/q)$ (see Section 3.1). If the residual value of $\sum_{i=1}^{m} (a_i/q) N_i$ is $< 1 - (a_0/q)$, then even after adding the contribution from the initial state (a_0/q), the overall residual value will still be discarded by the floor operation.

Summarising, due to the fact that the precision of a digital system is finite, rounding errors are unavoidable. The rounding policy of a CEASE-generated circuit can be adjusted by configuring its initial state. For an OMC circuit with q states, initialising it to state s_0 is equivalent to the rounding policy of truncating the fraction part of $N^{\boldsymbol{Z},1}$. On the other hand, initialising the circuit to a 'middle state,' i.e. to $s_{q/2}$, is equivalent to rounding $N^{\boldsymbol{Z},1}$ to the closest integer.

4.2 Autocorrelation and CEASE

Autocorrelation quantifies the correlation between two elements in a single data sequence. The autocorrelation coefficient, AC(t, s), for an SN $\boldsymbol{X}$ describes the similarity between $\boldsymbol{X}^{(t)}$ and $\boldsymbol{X}^{(s)}$. For example, applying the definition of Pearson correlation coefficient [16] to the AC in the SC context, we obtain $\mathrm{AC}(t, s) = \left(\left(\mathbb{E}\left[\left(\boldsymbol{X}^{(t)} - X\right)\left(\boldsymbol{X}^{(s)} - X\right)\right]\right)/\sigma^2\right)$, where $\sigma^2 = \mathbb{E}[(\boldsymbol{X}^{(i)} - X)^2]$ is the variance of $\boldsymbol{X}^{(i)}$. Thus AC(t, s) measures the similarity between the probabilities of the bits at two time steps t and s. If $\boldsymbol{X}$ is Bernoulli, i.e. if each bit of $\boldsymbol{X}$ is independently generated, then $\boldsymbol{X}$'s AC is two-valued with $\mathrm{AC}(t, t) = 1$, while $\mathrm{AC}(t, s) = 0$ for all $t \neq s$ [17].

Most sequential stochastic circuits are designed to work with input SNs that satisfy certain temporal independence constraints such as Bernoulli properties. Undesired autocorrelations can sometimes degrade the accuracy of such circuits dramatically. On the other hand, sequential stochastic circuits themselves can introduce autocorrelations into their output SNs, making them non-Bernoulli [3, 18]. The preceding facts imply that if two independently-designed sequential stochastic circuits are cascaded,

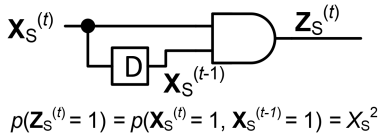

Fig. 6 *Conventional sequential stochastic squarer*

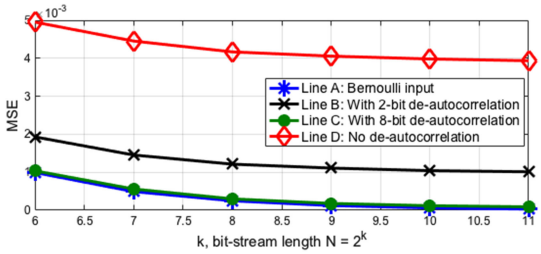

Fig. 7 *MSE versus bit-stream length for the squarer in Fig. 6 for inputs with different autocorrelation levels*

the downstream circuit's accuracy may drastically degrade due to autocorrelations introduced by the predecessor circuit. Since CEASE produces sequential designs, we are interested in the connections between CEASE designs and autocorrelation. It turns out that CEASE has some desirable properties related to autocorrelation, which we examine in this section.

First, we illustrate via an example how undesired autocorrelation affects the accuracy of a sequential stochastic circuit. Fig. 6 shows a stochastic squarer C_S, which is the standard sequential circuit for computing $Z_S = X_S^2$ [18]. To compute the value of $\boldsymbol{Z}_S$ at any cycle t, this squarer uses its D flip-flop to hold $\boldsymbol{X}_S^{(t-1)}$, which is then multiplied with $\boldsymbol{X}_S^{(t)}$ using the AND gate. The AND gate thus computes $p(\boldsymbol{X}_S^{(t-1)} = 1, \boldsymbol{X}_S^{(t)} = 1) = p_{\mathcal{X}_S^{(t)}}(1) \times p_{\mathcal{X}_S^{(t-1)}}(1) = X_S^2$, which implicitly assumes independence between $\boldsymbol{X}_S^{(t)}$ and $\boldsymbol{X}_S^{(t-1)}$. Since the output probability has to be correct for all t, it is not hard to see that C_S requires all the adjacent bits in its input $\boldsymbol{X}_S$ to be independent. In other words, it requires $\boldsymbol{X}_S^{(i)}$ and $\boldsymbol{X}_S^{(j)}$ to be uncorrelated, for all i, j, where $i = j + 1$. Fig. 7 plots MSE rate against bit-stream length for C_S using 0.5-valued inputs with different autocorrelation levels. Specifically, Line A shows the error rate for a Bernoulli input, which converges towards zero quickly as bit-stream length increases. Line D, on the other hand, shows the error rate when the input is produced by the sequential OMC adder in Fig. 1*c*. This input SN thus exhibits undesired autocorrelation, resulting in C_S having obvious errors. Each of the two lines B and C shows an error rate for an autocorrelated input that is de-autocorrelated using a shuffling de-autocorrelator, which we will discuss shortly.

Many existing sequential stochastic circuits, including the preceding squarer example, can suffer from accuracy degradation if inputs are autocorrelated [3, 18]. However, this is not true for a CEASE FSM M_C, which can, therefore, be called *autocorrelation-insensitive*. This means that M_C is immune to autocorrelation-induced accuracy degradation. To see this, recall from (8) that M_C implements a stochastic function of the form $\sum_{i=1}^{m}(a_i/q)p_{\mathcal{X}_V}(\boldsymbol{b}_i) + (1/N)\mathbb{E}(\tilde{e})$, which depends only on the joint probability of all the inputs at a given cycle, but not on the joint probability of any individual inputs across different cycles. Theorem 4 summarises the above discussion.

Theorem 4: For a CEASE FSM M_C, autocorrelation in any of its input SNs does not reduce M_C's accuracy.

Theorem 3 compares all possible implementations for a target function F and asserts that when the inputs are Bernoulli, a CEASE design achieves the optimal result. When the inputs are not Bernoulli, a CEASE design's accuracy remains the same, as Theorem 4 states. However, there are no guarantees that this CEASE design remains optimal. In other words, there may be other designs that achieve a lower error rate than CEASE, when the input SNs are not Bernoulli.

Having discussed the behaviour of CEASE-generated OMC circuits with autocorrelation in their inputs, we next investigate the effect these circuits have on autocorrelation in their outputs. This issue is worth looking into, especially when an SN $\boldsymbol{Z}$ generated by an OMC circuit is to be processed by some downstream sequential stochastic circuits that are sensitive to autocorrelation. Indeed, directly feeding $\boldsymbol{Z}$ to any sequential circuit may degrade accuracy. Line D in Fig. 7 shows the error rate plotted against bit-stream length for the squarer of Fig. 6 using a bit-stream generated by the OMC adder C_{CA} in Fig. 1*c* as the squarer's input. Here the scaled adder C_{CA} adds the two input SNs of value 0.5 and produces its output SN, also of value 0.5. We see that the MSE does not converge to zero, regardless of the bit-stream length. This clearly shows that the autocorrelation injected by C_{CA} degrades the accuracy of the downstream squarer.

It is very hard, in general, to analyse the impact of inputs with undesired autocorrelations on an arbitrary sequential stochastic circuit. In this work, we focus on analysing a well-defined class of stochastic FSMs called shift-register-based (SRB) [18]. An SRB circuit is a definite FSM, i.e. it has finite-input memory of length m implying that its behaviour is completely determined by its m most recent input bits. The squarer shown in Fig. 6 is a classic example of an SRB circuit since its output value is determined by the two most recent input bits. Many stochastic circuits turn out to be SRB, including those generated by state-of-the-art SC synthesisers [10, 11]. A useful aspect of SRB designs is that they can be described by a clocked Boolean function (CBF) [4] in which the parameters are possibly delayed Boolean values of the inputs. For instance, the squarer C_S is described by the CBF $z_S^{(t)} = f_S(x_S^{(t)}, x_S^{(t-1)}) = x_S^{(t)} \cdot x_S^{(t-1)}$.

To see how the autocorrelation introduced by an OMC circuit affects a downstream SRB circuit, we describe the latter using the CBF f_S, from which its stochastic function F_S can be constructed. The joint probability terms in F_S can then be analysed using the law of total probability (LTP) [16], which allows a probability term to be decomposed into several sub-terms, each with a distinct event. For example, suppose A and B_1, B_2,…, B_k are events with B_i's being disjoint and the union of B_i's being the entire sample space. Then LTP says

$$p(A) = \sum_{i=1}^{k} p(A, B_i) = \sum_{i=1}^{k} p(A \mid B_i) p(B_i).$$

By applying LTP and conditioning on the states of the preceding OMC circuit, F_S can be analysed. We illustrate this via the squarer example.

Example 3: Squarer. Here we use the OMC adder C_{CA} of Fig. 1*c* with the STG given in Fig. 4*b*, followed by the squarer C_S of Fig. 6 to illustrate the impact of autocorrelation. Specifically, we feed $\mathcal{X}_V = \{\boldsymbol{X}, \boldsymbol{Y}\}$, the two inputs of C_{CA}, with independent Bernoulli SNs, each of value 0.5. Thus $\boldsymbol{Z}$, the output of C_{CA}, should have a value of 0.5. Since $\boldsymbol{Z}$ is produced by the sequential circuit C_{CA}, extra autocorrelation is injected into it. We then use $\boldsymbol{Z}$ as the input SN $\boldsymbol{X}_S$ to the squarer C_S. Obviously, the exact value for C_S's output $\boldsymbol{Z}_S$ should be $0.5^2 = 0.25$. To compute the actual value for $\boldsymbol{Z}_S$ with $\boldsymbol{X}_S$ as the input, we first construct the stochastic function for C_S using its CBF $z_S^{(t)} = x_S^{(t)} \cdot x_S^{(t-1)}$, which is

$$Z_S = p\left(\boldsymbol{Z}_S^{(t)} = 1\right) = \sum_{\boldsymbol{b}} \left[f_S(\boldsymbol{b}) p_{\{X_S^{(t)}, X_S^{(t-1)}\}}(\boldsymbol{b}) \right]$$

$$= p_{\{X_S^{(t)}, X_S^{(t-1)}\}}(1, 1) = p(\boldsymbol{X}_S^{(t)} = 1, \boldsymbol{X}_S^{(t-1)} = 1).$$

We then apply LTP by conditioning on the adder C_{CA}'s state at time $t-1$

$$Z_S = p(X_S^{(t)} = 1, X_S^{(t-1)} = 1 \,|\, S^{(t-1)} = s_0)p(S^{(t-1)} = s_0) + p(X_S^{(t)} = 1, X_S^{(t-1)} = 1 \,|\, S^{(t-1)} = s_1)p(S^{(t-1)} = s_1).$$

One very appealing feature of OMC is that when the inputs are Bernoulli, the steady-state distribution of the states is uniform.

Thus $p(S^{(t-1)} = s_0) = p(S^{t-1)} = s_1) = 0.5$. Furthermore, $p(X_S^{(t)} = 1, X_S^{(t-1)} = 1 \,|\, S^{(t-1)} = s_0) = p\left(\mathcal{X}_V^{(t)} = \{1, 1\}, \mathcal{X}_V^{(t-1)} = \{1, 1\}\right)$. This is because in state s_0, C_{CA} can only output two consecutive 1s when it receives inputs 11 and 11. On the other hand, there are five cases where C_{CA} can add two consecutive 1s in state s_1; see Fig. 8. Each of these cases occurs with a probability of 1/16, so $p(X_S^{(t)} = 1, X_S^{(t-1)} = 1 \,|\, S^{(t-1)} = s_1) = 5/16$. Summarising, $Z_S = 0.5 \times (1/16) + 0.5 \times (5/16) = (3/16) \neq 0.25$, which is a 25% error! In terms of MSE, this error is $(3/16 - 0.25)^2 = 0.004$, which is clearly the value to which Line D in Fig. 7 converges.

Example 3 shows how to analyse the autocorrelation error introduced by an OMC circuit for given input values. One can also go one step further to compute the expected error by averaging all possible input values. Fig. 9 provides a pseudo-code algorithm summarising the preceding analysis.

Finally, we consider how to mitigate autocorrelation in an OMC circuit so that it can be coupled efficiently with other sequential stochastic circuits. For this, we adopt a de-autocorrelation method based on shuffling. This is similar to the edge memory [19] and the shuffle buffer [20], which randomly permute or shuffle the incoming SN by means of an extra random source. Fig. 10*a* shows how a de-autocorrelator is inserted between the OMC circuit C_{OMC} and the downstream circuit C_S, while Fig. 10*b* depicts the implementation of a two-bit de-autocorrelator that contains two D flip-flops. At each clock cycle, the de-autocorrelator randomly selects a D flip-flop and sends the bit value stored in that flip-flop to the output. At the same time, the selected flip-flop is filled by the value from the current incoming bit. Since the de-autocorrelator only outputs what it has received, the values of its input and output SNs are the same. However, the location of 1s can potentially be very different, hence achieving the goal of de-autocorrelation.

A k-bit de-autocorrelator can be built with k D flip-flops. In each clock cycle, one of the k D flip-flops is selected to output its stored value, and at the same time store the current input. Lines B and C in Fig. 7 show the accuracy of the squarer with its OMC-circuit-supplied inputs de-autocorrelated by two- and eight-bit de-autocorrelators, respectively. Obviously, with an eight-bit de-autocorrelator, the accuracy of the squarer can be improved to a level similar to a squarer having a Bernoulli input. As a final note, with more D flip-flops, a de-autocorrelator can regenerate a more Bernoulli-like SN, but this comes with the drawback of higher hardware cost and requiring more warmup time to initialise the de-autocorrelator.

5 Applications of CEASE

This section applies CEASE to some representative circuits and assesses the corresponding accuracy improvements. It also examines the accuracy of CEASE using randomly generated stochastic circuits.

5.1 Typical application

CEASE can be applied to SN formats other than unipolar since it deals directly with probabilities rather than their interpretation or format. Suppose, e.g. that CEASE is applied to the circuit C_{ST} synthesised by STRAUSS [10] and outlined in Fig. 11*a*. C_{ST} uses the inverted bipolar (IBP) SN format to handle negative values, and realises the following stochastic function:

$$\tilde{Z} = \frac{7}{16} - \frac{1}{8}(\tilde{X}_1 + \tilde{X}_2) - \frac{9}{16}\tilde{X}_1\tilde{X}_2, \quad (10)$$

where $\tilde{X}_1$ and $\tilde{X}_2$ are independent IBP SNs with the same value. This STRAUSS design heavily relies on constant SNs, as it employs four constants R_1–R_4, each of value 0.5. Another implementation C_{RE} of the same function $\tilde{Z}$ synthesised by ReSC [11] is given in Fig. 11*b*; it relies on three constants C_1–C_3 to provide the same level of accuracy. To implement (10) using CEASE, we first derive the corresponding unipolar stochastic function from the relation $\tilde{X} = 1 - 2X$, where $X = p_X(1)$ 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 (10) and re-arranging, we obtain

	$\mathcal{X}_V^{(t)}$	$\mathcal{X}_V^{(t-1)}$	$p\left(\mathcal{X}_V^{(t)}, \mathcal{X}_V^{(t-1)}\right)$
Case 1	1, 1	1, 0	1/16
Case 2	1, 1	0, 1	1/16
Case 3	1, 0	1, 1	1/16
Case 4	0, 1	1, 1	1/16
Case 5	1, 1	1, 1	1/16

Fig. 8 *Five input cases in which the CEASE-generated adder of Fig. 4b outputs two consecutive 1s when in state s_0*

Input: An OMC circuit C_{OMC} and its input SNs. A downstream single-input SRB circuit C_D.

Output: F_D, the stochastic function implemented by C_D when used with C_{OMC}.

Step 1. Express C_D's logic function through a CBF $z_D^{(t_k)} = f_D(\boldsymbol{b}) = f_D(x_D^{(t_k)}, x_D^{(t_{k-1})}, \ldots, x_D^{(t_1)})$, where $t_k > t_{k-1} > \ldots > t_1$.

Step 2. Construct C_D's stochastic function as $F_D = \sum_{\boldsymbol{b}} \left[f_D(\boldsymbol{b}) p_{\{x_D^{(t_k)}, x_D^{(t_{k-1})}, \ldots, x_D^{(t_1)}\}}(\boldsymbol{b}) \right]$.

Step 3. For all $\boldsymbol{b}$ such that $f_D(\boldsymbol{b}) = 1$, compute the joint probability $p_{\{x_D^{(t_k)}, x_D^{(t_{k-1})}, \ldots, x_D^{(t_1)}\}}(\boldsymbol{b})$ using LTP by conditioning on the state variable $S^{(t_1)}$ of C_{OMC}. Specifically, $p_{\{x_D^{(t_k)}, x_D^{(t_{k-1})}, \ldots, x_D^{(t_1)}\}}(\boldsymbol{b}) = \frac{1}{q}\sum_s \left[p_{\{x_D^{(t_k)}, x_D^{(t_{k-1})}, \ldots, x_D^{(t_1)}\}}(\boldsymbol{b}) \,|\, S^{(t_1)} = s \right]$. Here q is the number of states in C_{OMC}.

Step 4. Output the resulting F_D.

Fig. 9 *Algorithm to compute the stochastic function with OMC-supplied (and therefore autocorrelated) inputs*

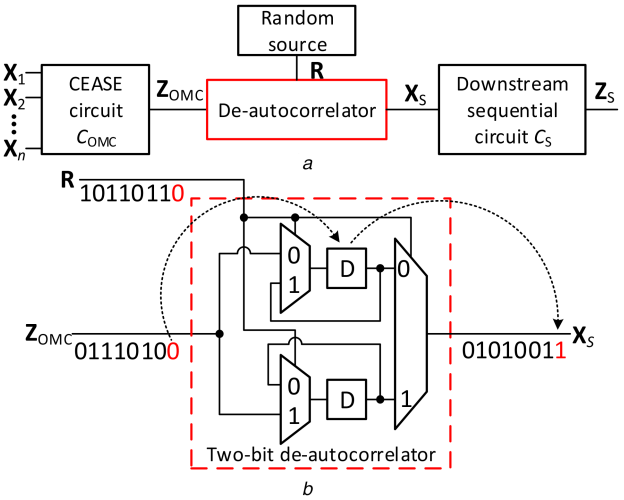

Fig. 10 *Shuffle-based reduction of autocorrelation in an SN*
(a) De-autocorrelator to mitigate OMC-induced autocorrelation in a downstream sequential circuit, ***(b)*** Two-bit shuffling de-autocorrelator

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

Since X_1 and X_2 are independent, the term X_1X_2 can be written as $p_{X_1}(1)\,p_{X_2}(1) = p_{\mathcal{X}_V}(1,1)$, where $\mathcal{X}_V = \{\boldsymbol{X}_1, \boldsymbol{X}_2\}$. Furthermore, we can 'de-marginalise' 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 (11) with these probabilities yields a unipolar stochastic function to which we can apply CEASE

$$Z = F(f, p_{\mathcal{X}_V}) = \frac{11}{16}p_{\mathcal{X}_V}(0,0) + \frac{7}{16}p_{\mathcal{X}_V}(1,1). \quad (12)$$

Equation (12) is the unipolar or probability interpretation of (10) with coefficients in [0, 1]. This fact can also be directly seen from the ReSC design C_{RE} in Fig. 11*b*, which outputs 11/16 and 7/16 when the input pattern is 00 and 11, respectively.

It is also worth mentioning that (12) can be systematically constructed from the Boolean function generated by STRAUSS to approximate the desired function. This function is $z = x_1'x_2'(r_1 + r_2(r_3 + r_4)) + x_1x_2r_1(r_2 + r_3 + r_4)$, as indicated in Fig. 11*a*. It is obvious that if $x_1x_2 = 10$ or 01, then $z = 0$. Thus, the terms $p_{\mathcal{X}_V}(0,1)$ and $p_{\mathcal{X}_V}(1,0)$ in (12) both have a coefficient of 0. When $x_1x_2 = 00$, we get $z = r_1 + r_2(r_3 + r_4)$. Here, out of the 16 equally-likely combinations of $r_1r_2r_3r_4$, there are 11 cases that make $z = 1$, and hence the coefficient for $p_{\mathcal{X}_V}(0,0)$ is 11/16. Similarly, when $x_1x_2 = 11$, $z = r_1(r_2 + r_3 + r_4)$. Seven of the 16 possible combinations of $r_1r_2r_3r_4$ make $z = 1$, so the coefficient of $p_{\mathcal{X}_V}(1,1)$ is 7/16.

A CEASE-generated OMC design C_{OMC} implementing (10) in the IBP domain and (12) in the unipolar domain is given in Fig. 11*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; 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_{OMC} requires four flip-flops for its 16-state counter. C_{ST} shown in Fig. 11*a* requires four constant SNs that are generated by a 4-tap linear feedback shift register (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 a 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. 11*b*) for B_{R1} and B_{R3}.

An MSE comparison of the above three circuits appears in Fig. 12. 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 state; this error may be removed by replacing the LFSR with a better random number source. The OMC circuit C_{OMC}, 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_{OMC} can compute in far less time, and hence with better energy efficiency, than the other designs. For example, C_{OMC} achieves an MSE of 0.002 with $N = 32$ bits, while the ReSC design C_{RE} needs ~128 bits for the same accuracy.

The OMC circuit C_{OMC} in Fig. 11*c* represents just one possible way to realise the CEASE FSM. In fact, a CEASE design can always be implemented by a circuit built around a MAJ function, as in the adder example in Fig. 1*c*. Consider the circuit C_{MAJ} shown in Fig. 13 which is the MAJ incarnation of C_{OMC}. Besides MAJ, C_{MAJ} also has a modulo counter *CNT* like that discussed previously. The difference is that *CNT* has 16 output lines designed in a way such that the number of 1s on these lines indicates *CNT*'s state. i.e. the number of 1s *n* on the output lines of *CNT* is the state

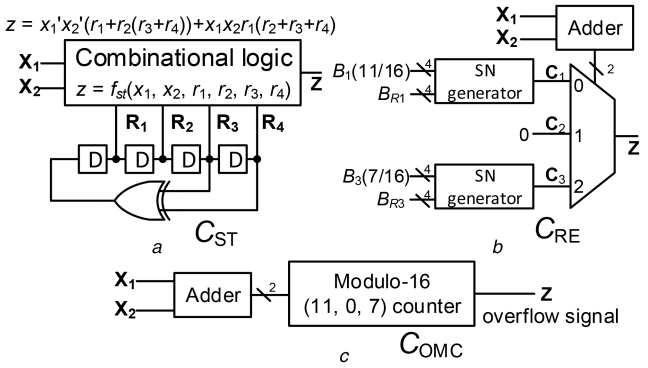

Fig. 11 *Three implementations of (10)*
(a) STRAUSS design C_{ST}, ***(b)*** ReSC design C_{RE}, ***(c)*** CEASE design C_{OMC}

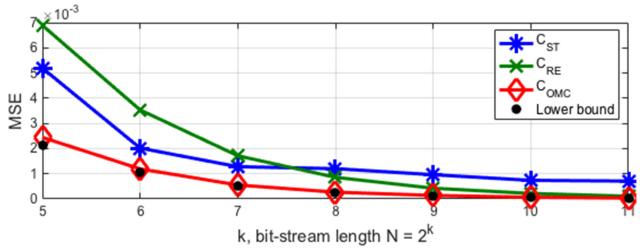

Fig. 12 *MSE comparison for the circuits in Fig. 11*

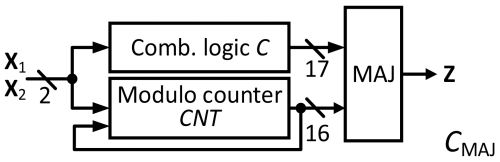

Fig. 13 *MAJ-based implementation C_{MAJ} of the OMC circuit C_{OMC} given in Fig. 11c*

s_n that *CNT* is currently at. The subcircuit *C* produces 11 1s and seven 1s in the output lines when $\boldsymbol{X}_1\boldsymbol{X}_2$ is 00 and 11, respectively. It is obvious that C_{MAJ} will output a 1 when the values of *CNT* and the values outputted by *C* is >16. This is exactly what C_{OMC} in Fig. 11*c* does. Finally, we stress the fact that while a CEASE design based on a majority function is always realisable, its hardware cost may not be optimal.

5.2 Other applications

Next, we examine the performance of CEASE by applying it to some other representative circuit types: a complex matrix multiplier, random combinational circuits, and a sequential circuit implemented by the linear FSM architecture [21].

Complex matrix multiplication: Fig. 14*a* shows a combinational stochastic circuit with 12 constant inputs implementing complex matrix multiplication [22]. It has four outputs, each of which depends on three constant inputs, all of which can be eliminated by CEASE. Here we examine the accuracy improvement after applying CEASE to the sub-circuit spanned by $\boldsymbol{Z}_i^1$, one of the circuit's four primary outputs. The resulting STG has four states, which require two flip-flops to implement. The CEASE-generated OMC circuit is similar in structure to that in Fig. 11*c*. An MSE comparison between the circuit in Fig. 14*a* and the OMC circuit appears in Fig. 14*b*, which again shows that CEASE improves accuracy and, at the same time, matches the lower bound on MSE.

Random circuits: In the absence of benchmark stochastic circuits, we use randomly generated ones to further estimate the performance of CEASE. Specifically, we first generate 100,000 random stochastic functions in the form of (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 and synthesise OMC circuits implementing these random functions.

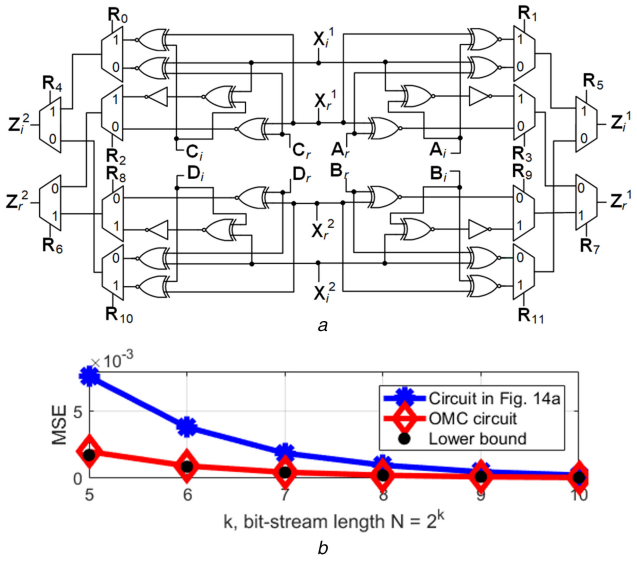

Fig. 14 *Application of CEASE to a complex matrix multiplier*
(a) Stochastic circuit implementing complex matrix multiplication [22], ***(b)*** MSE comparison with an OMC design

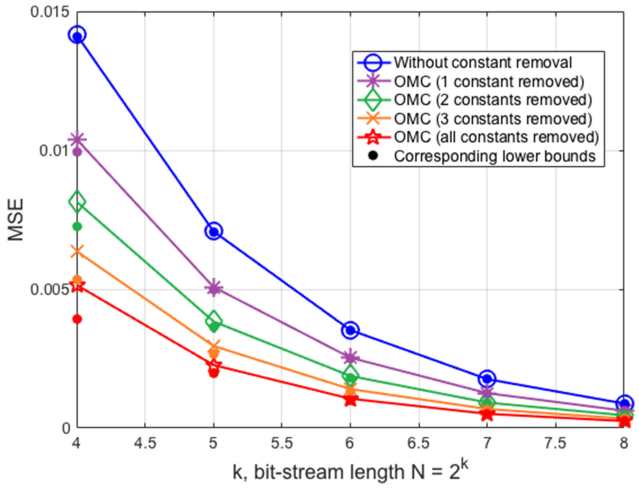

Fig. 15 *MSE comparison of random circuits with four constant- and two variable-input SNs. The lower bounds treat the unremoved constants as variables*

Fig. 15 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. 15, 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.

Sequential stochastic circuits: CEASE can be used to remove some RFEs in sequential stochastic circuits as well. We illustrate this via the circuit C_{SEQ} in Fig. 16*a*, which is a sequential realisation of the stochastic function $Z = (X - 2X^2 + 1.5X^3)/(1 - 2X + 2X^2)$ using the linear FSM architecture, one of the most common and best understood sequential SC designs [21]. C_{SEQ} is built around a four-state FSM M that realises a saturating up/down counter (Fig. 16*c*), and a four-way MUX Q that uses S, the state of M, to select one of its four input SNs to send to the output. Three of these SNs are **0** or **1**, whose bit-streams can be produced by static sources that do not induce RFEs. The remaining constant SN $\boldsymbol{R}$ has value 0.5 which is the usual RFE-inducing constant. $\boldsymbol{R}$ is fed to the combinational MUX Q but not to the sequential part M. Hence, we can incorporate CEASE into C_{SEQ} to remove $\boldsymbol{R}$ and improve accuracy. This is done by directly applying the CEASE algorithm to Q to obtain the new FSM $C_{\mathrm{SEQ–CEASE}}$ shown in Fig. 16*b*. $C_{\mathrm{SEQ–CEASE}}$ transfers the role of all constant inputs $\boldsymbol{R}$, **0** and **1** to its sequential OMC part, and eliminates RFEs due to $\boldsymbol{R}$. Fig. 16*d*

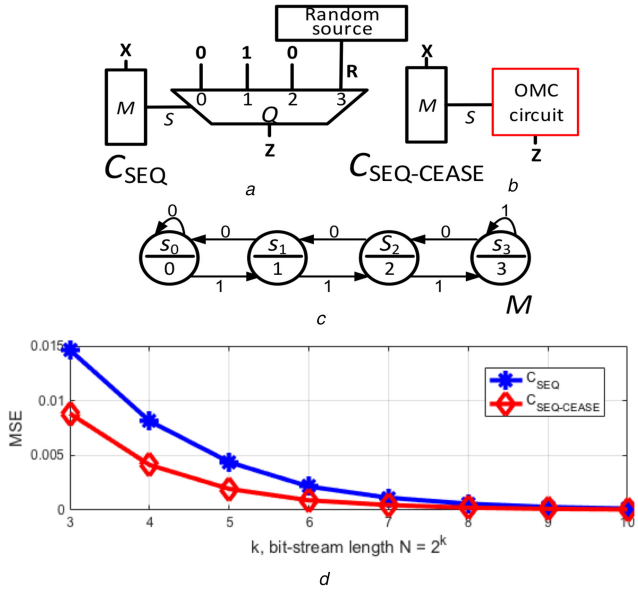

Fig. 16 *Application of CEASE to a sequential circuit synthesised by the method of [21]*
(a) Sequential circuit C_{SEQ} implementing $Z = (X - 2X^2 + 1.5X^3)/(1 - 2X + 2X^2)$ [21], ***(b)*** Sequential circuit $C_{\mathrm{SEQ–CEASE}}$ obtained by applying CEASE to the combinational part Q of C_{SEQ}, ***(c)*** STG for the sequential part M of C_{SEQ}, ***(d)*** MSE comparison between C_{SEQ} and $C_{\mathrm{SEQ–CEASE}}$

compares the MSEs of C_{SEQ} *and* $C_{\mathrm{SEQ–CEASE}}$. It shows clearly that $C_{\mathrm{SEQ–CEASE}}$ has less error than C_{SEQ} for any given bit-stream length. Note, however, that $C_{\mathrm{SEQ–CEASE}}$ does not guarantee the minimum possible error since CEASE does not consider RFEs produced by the sequential component M whose behaviour on random fluctuations is very different from that of a combinational circuit.

As we have demonstrated CEASE can achieve significantly higher accuracy than conventional SC designs with very little increase in hardware cost. In particular, a CEASE design never uses more memory elements than the STRAUSS and ReSC synthesisers. For example, all three designs in Fig. 11 require just four D flip-flops. Thus, CEASE provides an attractive design alternative with high accuracy that may enable a circuit to achieve a satisfactory level of performance with shorter bit-streams. For example, the deep neural network presented in [15] uses a set of highly accurate stochastic adders that are functionally identical to the OMC circuit in Fig. 1*c* to strike a balance between low hardware cost and high accuracy requirements.

6 Conclusion

We have clarified the role of constant SNs as inputs in stochastic circuits, and shown that, while such constant inputs are essential in practical SC design, they are an unexpected source of significant amounts of RFEs. 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. The resulting FSMs can be implemented as OMC circuits. We analysed in detail several important issues including state initialisation and autocorrelation, for OMC circuits. We further proved analytically that CEASE is optimal in terms of its ability to eliminate RFEs. 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.

7 Acknowledgments

This work was supported by a grant CCF-1318091 from the U.S. National Science Foundation.

8 References

[1] Alaghi, A., Li, C., Hayes, J.P.: 'Stochastic circuits for real-time image-processing applications'. IEEE Proc. Design Automation Conf., Austin, TX, USA, 2013, pp. 1–6

[2] Gaudet, V.C., Rapley, A.C.: 'Iterative decoding using stochastic computation', *IET Electron Lett.*, 2003, **39**, pp. 299–301

[3] Brown, B.D., Card, H.: 'Stochastic neural computation I', *IEEE Trans. Comput.*, 2001, **50**, pp. 891–905

[4] Ting, P.-S., Hayes, J.P.: 'Isolation-based decorrelation of stochastic circuits'. IEEE Proc. Int. Conf. of Stochastic Circuits, Scottsdale, AZ, USA, 2016, pp. 88–95

[5] Braendler, D., Hendtlass, T., O'Donoghue, P.: 'Deterministic bit-stream digital neurons', *IEEE Trans. Neural Netw.*, 2002, **13**, pp. 1514–1525

[6] Gupta, P.K., Kumaresan, R.: 'Binary multiplication with PN sequences', *IEEE Trans. Acoust. Speech Signal Process.*, 1988, **36**, pp. 603–606

[7] Jenson, D., Riedel, M.: 'A deterministic approach to stochastic computation'. IEEE Proc. 35th Int. Conf. on Computer-Aided Design, Austin, TX, USA, 2016, pp. 1–8

[8] Vahapoglu, E., Altun, M.: 'Accurate synthesis of arithmetic operations with stochastic logic'. 2016 IEEE Computer Society Annual Symp. on VLSI, Pittsburgh, PA, USA, 2016, pp. 415–420

[9] Alaghi, A., Hayes, J.P.: 'Fast and accurate computation using stochastic circuits'. IEEE Proc. Conf. on Design, Automation & Test in Europe, Dresden, Germany, 2014, pp. 24–28

[10] Alaghi, A., Hayes, J.P.: 'STRAUSS: spectral transform use in stochastic circuit synthesis', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2015, **34**, pp. 1770–1783

[11] Qian, W., Li, X., Riedel, M.D., *et al.*: 'An architecture for fault-tolerant computation with stochastic logic', *IEEE Trans. Comput.*, 2011, **60**, pp. 93–105

[12] Ting, P.-S., Hayes, J.P.: 'Eliminating a hidden error source in stochastic circuits'. IEEE Proc. Defect and Fault Tolerance, Cambridge, UK, 2017, pp. 44–49

[13] Alaghi, A., Hayes, J.P.: 'On the functions realized by stochastic computing circuits'. ACM Proc. 25th edition on Great Lakes Symp. on VLSI, Pittsburgh, PA, USA, 2015, pp. 331–336

[14] Chen, T.-H., Hayes, J.P.: 'Equivalence among stochastic logic circuits and its applications', IEEE Proc. Design Automation Conf., San Francisco, CA, USA, 2015, pp. 131–136

[15] Lee, V.T., Alaghi, A., Hayes, J.P., *et al.*: 'Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing'. IEEE Proc. Conf. on Design, Automation & Test in Europe, Lausanne, Switzerland, 2017, pp. 13–18

[16] Gubner, J.A.: '*Probability and random processes for electrical and computer engineers*' (Cambridge University Press, New York, NY, USA, 2006)

[17] Golomb, S.W., Gong, G.: '*Signal design for good correlation*' (Cambridge University Press, New York, NY, USA, 2005)

[18] Ting, P.-S., Hayes, J.P.: 'On the role of sequential circuits in stochastic computing'. ACM Proc. 25th edition on Great Lakes Symp. on VLSI, Banff, Alberta, Canada, 2017, pp. 475–478

[19] Tehrani, S.S., Gross, W.J., Mannor, S.: 'Stochastic decoding of LDPC codes', *IEEE Commun. Lett.*, 2006, **10**, (10), pp. 716–718

[20] Lee, V.T., Alaghi, A., Ceze, L.: 'Correlation manipulating circuits for stochastic computing'. IEEE Proc. Conf. on Design, Automation & Test in Europe, Dresden, Germany, 2018, pp. 1417–1422

[21] Li, P., Qian, W., Riedel, M.D.: 'The synthesis of linear finite state machine-based stochastic computational elements'. IEEE Proc. Asia and South Pacific Design Automation Conf., Sydney, NSW, Australia, 2012, pp. 757–762

[22] Paler, A., Kinseher, J., Polian, I., *et al.*: 'Approximate simulation of circuits with probabilistic behavior'. IEEE Proc. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, New York, NY, USA, 2013, pp. 95–100

[23] Keener, R.W.: '*Theoretical statistics: topics for a core course*' (Springer, New York, NY, USA, 2010)

9 Appendix

9.1 Proof of theorem 1

By classifying SN inputs into variable and constant parts as in $\mathcal{X} = \{\mathcal{X}_\mathrm{V}, \mathcal{X}_\mathrm{C}\}$, (3) can be re-written as

$$Z = F(f, p_{\mathcal{X}_\mathrm{V}.\mathcal{X}_\mathrm{C}}) = \sum_{\boldsymbol{b}_\mathrm{V}, \boldsymbol{b}_\mathrm{C}} \left[f(\boldsymbol{b}_\mathrm{V},\ \boldsymbol{b}_\mathrm{C}) p_{\mathcal{X}_\mathrm{V}.\mathcal{X}_\mathrm{C}}(\boldsymbol{b}_\mathrm{V},\ \boldsymbol{b}_\mathrm{C}) \right]. \tag{13}$$

Using the properties of conditional probability, we can rewrite $p_{\mathcal{X}_\mathrm{V}.\mathcal{X}_\mathrm{C}}(\boldsymbol{b}_\mathrm{V},\ \boldsymbol{b}_\mathrm{C})$ as $p_{\mathcal{X}_\mathrm{C}|\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{C}|\boldsymbol{b}_\mathrm{V}) \cdot p_{\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{V})$, where the term $p_{\mathcal{X}_\mathrm{C}|\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{C}|\boldsymbol{b}_\mathrm{V})$ is a function of $\boldsymbol{b}_\mathrm{C}$ and $\boldsymbol{b}_\mathrm{V}$. Equation (13) then becomes

$$\begin{aligned} Z = &\sum_{\boldsymbol{b}_\mathrm{V}, \boldsymbol{b}_\mathrm{C}} \left[f(\boldsymbol{b}_\mathrm{V},\ \boldsymbol{b}_\mathrm{C}) p_{\mathcal{X}_\mathrm{C}|\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{C}|\boldsymbol{b}_\mathrm{V}) \cdot p_{\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{V}) \right] \\ &\sum_{\boldsymbol{b}_\mathrm{V}} \left[p_{\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{V}) \cdot \sum_{\boldsymbol{b}_\mathrm{C}} \left[f(\boldsymbol{b}_\mathrm{V},\ \boldsymbol{b}_\mathrm{C}) \cdot p_{\mathcal{X}_\mathrm{C}|\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{C}|\boldsymbol{b}_\mathrm{V}) \right] \right]. \end{aligned} \tag{14}$$

The summation $\sum_{\boldsymbol{b}_\mathrm{C}} \left[f(\boldsymbol{b}_\mathrm{V},\ \boldsymbol{b}_\mathrm{C}) \cdot p_{\mathcal{X}_\mathrm{C}|\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{C}|\boldsymbol{b}_\mathrm{V}) \right]$ is over all combinations of $\boldsymbol{b}_\mathrm{C}$, and hence $g(\boldsymbol{b}_\mathrm{V}) = \sum_{\boldsymbol{b}_\mathrm{C}} \left[f(\boldsymbol{b}_\mathrm{V},\ \boldsymbol{b}_\mathrm{C}) \cdot p_{\mathcal{X}_\mathrm{C}|\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{C}|\boldsymbol{b}_\mathrm{V}) \right]$ does not depend on $\boldsymbol{b}_\mathrm{C}$, so we can re-write (14) as $Z = F(p_{\mathcal{X}_\mathrm{V}}) = \sum_{\boldsymbol{b}_\mathrm{V}} \left[g(\boldsymbol{b}_\mathrm{V}) \cdot p_{\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{V}) \right]$ which is linear in $p_{\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{V})$ with all coefficients $g(\boldsymbol{b}_\mathrm{V})$ in the range [0,1]. The dependency of $F(p_{\mathcal{X}_\mathrm{V}})$ on f and $p_{\mathcal{X}_\mathrm{C}}$ is implicit via $g(\boldsymbol{b}_\mathrm{V})$ only.

9.2 Proof of theorem 2

To get the number of 1s in the output $\boldsymbol{Z}$, we first compute the total number of states that M_C will advance after receiving the N-bit SNs, which is $\sum_{i=1}^{m} a_i N_i + a_0$. The number of 1s in $\boldsymbol{Z}$ is therefore

$$\left\lfloor \frac{1}{q} \left(\sum_{i=1}^{m} a_i N_i + a_0 \right) \right\rfloor = \left\lfloor \sum_{i=1}^{m} \frac{a_i}{q} N_i + \frac{a_0}{q} \right\rfloor = \sum_{i=1}^{m} \frac{a_i}{q} N_i + \frac{a_0}{q} - \epsilon,$$

where $\varepsilon \in [0, 1]$ is an offset term that takes into account the floor operation. Setting $\tilde{\epsilon} = (a_0/q) - \epsilon$, we obtain the number of 1s in $\boldsymbol{Z}$ as $\sum_{i=1}^{m} (a_i/q) N_i + \tilde{\epsilon}$, which completes the first part of the proof. The value of $\boldsymbol{Z}$ is thus the expectation of $(1/N) \sum_{i=1}^{m} (a_i/q) N_i + (\tilde{\epsilon}/N)$, which is

$$\begin{aligned} Z = &\mathbb{E}\left(\frac{1}{N} \sum_{i=1}^{m} \frac{a_i}{q} N_i + \frac{\tilde{\epsilon}}{N} \right) = \sum_{i=1}^{m} \frac{a_i}{q} \mathbb{E}\left(\frac{N_i}{N} \right) \\ &+ \frac{\mathbb{E}(\tilde{\epsilon})}{N} = \sum_{i=1}^{m} \frac{a_i}{q} p_{\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_i) \ + \frac{1}{N} \mathbb{E}(\tilde{\epsilon}), \end{aligned}$$

which completes the second part of the proof.

Here we provide another way of understanding the functions realised by M_C from the viewpoint of the FSM's steady-state distribution. Let $S(\boldsymbol{b}_\mathrm{V})$ denote all states of M_C that produce the output $z = 1$ in response to the input bit-pattern $\boldsymbol{b}_\mathrm{V}$. M_C's output probability can then be written as

$$Z = \sum_{\boldsymbol{b}_\mathrm{V}} \left(\sum_{s \in \mathrm{S}(\boldsymbol{b}_\mathrm{V})} p_\mathrm{S}(s) \right) p_{\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{V}). \tag{15}$$

Note that M_C represents a modulo counter with no redundant states. Its steady state distribution is equally distributed across all its q states, i.e. $p_S(s_i) = 1/q$, for $i = 0, 1, \ldots, q-1$, since the state transition behaviour is exactly the same for all states. Then (15) can be simplified to

$$Z = \sum_{\boldsymbol{b}_\mathrm{V}} \frac{|\mathrm{S}(\boldsymbol{b}_\mathrm{V})|}{q} p_{\mathcal{X}_\mathrm{V}}(\boldsymbol{b}_\mathrm{V}), \tag{16}$$

where $|S(\boldsymbol{b}_\mathrm{V})|$ denotes the cardinality of $S(\boldsymbol{b}_\mathrm{V})$. Since $|S(\boldsymbol{b}_\mathrm{V})|/q$ is, in general, a non-binary number in [0, 1] that does not depend on $\mathcal{X}_\mathrm{V}$, (16) is indeed in the form of (5) and (8). Finally, we note here that using the analysis based on steady-state distribution does not take into account M_C's initial state that determines the rounding policy of M_C. Indeed, when $N \to \infty$, the rounding error approaches 0, and (8) becomes equivalent to (16).

9.3 Proof outline of theorem 3

For a CEASE-generated OMC circuit C_OMC, the value of its output SN $\boldsymbol{Z}_\mathrm{C}$ estimated by the fraction of 1s is

$\hat{Z}_C = \sum_{i=1}^{m}(a_i/q)(N_i/N) + \tilde{\epsilon}/N = \hat{Z}_u + \tilde{\epsilon}/N$ (Theorem 2), where $\tilde{\epsilon}/N \in (-1/N + a_0/qN, a_0/qN)$ is a term accounting for the rounding error which, in the worst case, can only cause a 1-bit difference in $\hat{Z}_C$. Furthermore, $\hat{Z}_u = \sum_{i=1}^{m}(a_i/q)(N_i/N)$ is an unbiased estimator for $\boldsymbol{Z}_C$'s value which achieves the Cramér–Rao bound, a lower bound on MSE for an unbiased estimator [23]. Specifically, let the random vector $\boldsymbol{B} = [B(1), B(2), B(3),\ldots]$ with $B(i)$ being the bit-pattern received by C_{OMC} at clock cycle i, and let $\boldsymbol{p} = [p_1, p_2,\ldots, p_m]$ with p_i being the probability that C_{OMC} receives pattern b_i at any clock cycle. In other words, $p(B(t) = b_i) = p_i$, for all t and i. Then Cramér–Rao bound for the unbiased estimator $\hat{Z}_u$ asserts that

$$\left[(\hat{Z}_u - Z)\right]^2 \geq \nabla F(p)^T I^{-1}(p) \nabla F(p), \tag{17}$$

where ∇ is the gradient operator and the superscripts T and −1 denote matrix transpose and inversion. $I(\boldsymbol{p})$ is the Fisher information matrix of $\boldsymbol{B}$ [23], whose (i, j)th entry is $\mathbb{E}\left[\frac{\partial \log(p(\boldsymbol{B}|\boldsymbol{p}))}{\partial p_i}\frac{\partial \log(p(\boldsymbol{B}|\boldsymbol{p}))}{\partial p_j}\right]$. In other words, the minimal MSE achievable is $\nabla F(\boldsymbol{p})^{\mathrm{T}} I^{-1}(\boldsymbol{p}) \nabla F(\boldsymbol{p})$, and $\hat{Z}_u$ can be shown to attain this minimal MSE. Summarising, we conclude that C_C has the minimum MSE among all designs up to a rounding error.