

Accuracy and Correlation in Stochastic Computing



Armin Alaghi, Paishun Ting, Vincent T. Lee, and John P. Hayes

Abstract This chapter begins by reviewing the sources of inaccuracy in stochastic computing, focusing on correlation, that is, dependencies among stochastic bit-streams. The measurement of correlation is considered, and the *SCC* metric is defined. The properties of correlation are then explored including some that have only been discovered recently. Correlation can be seen in two ways: either as corrupting a function f , or as changing f to a different, but potentially useful one. Therefore, to ensure that a stochastic circuit works as expected it is important to manage correlation appropriately. This can be done with correlation-controlling units, which must be used carefully to avoid unexpected functional changes and excessive hardware area or latency overhead. There are also cases where correlation has no effect at all (correlation insensitivity). Identifying such immunity to correlation can aid the design of stochastic circuits. Finally, design of stochastic number generators to provide specified levels of correlation is discussed.

Keywords Cross correlation · Autocorrelation · Random number sources · Correlation metrics · Correlation mitigation

Error Sources in Stochastic Circuits

Stochastic computing (SC) can be summarized concisely as computing with probabilities represented by pseudo-random bit-streams [7]. As discussed in the preceding chapters, its advantages include small size, low power, error tolerance, the use of standard digital components, and bio-compatibility. Its primary disadvantages are low accuracy, long computing times, and costly randomness sources. Consequently,

P. Ting · J. P. Hayes (✉)
University of Michigan, Ann Arbor, MI, USA
e-mail: paishun@umich.edu; jhayes@umich.edu

A. Alaghi · V. T. Lee
University of Washington, Seattle, WA, USA
e-mail: armin.alaghi@cs.washington.edu; vlee2@cs.washington.edu

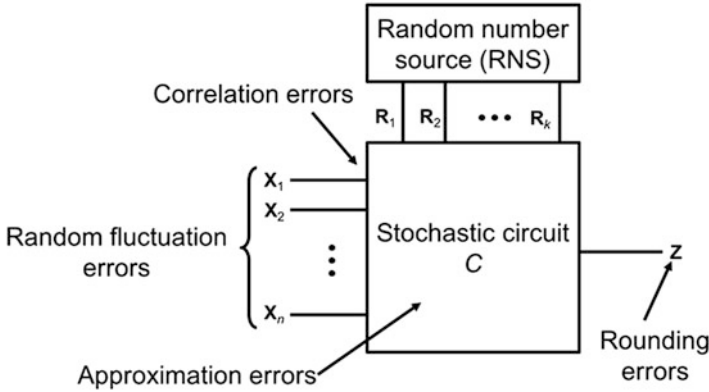


Fig. 1 Structure of a generic stochastic circuit annotated with known sources of inaccuracy

the design of a stochastic circuit involves complex trade-offs among accuracy, computing time and hardware cost. These trade-offs are vividly seen in the phenomenon of correlation, which is the focus of this chapter.

Figure 1 illustrates the general structure of a stochastic circuit. At its core is a combinational or sequential logic circuit C , which is supplied by a user with n stochastic numbers (SNs) $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$ of length N . \mathbf{X}_i is composed of N bits, where the t -th bit $\mathbf{X}(t)$ is assumed to be randomly generated in clock cycle t . These bits are, at least approximately, independent of one another, hence the name *stochastic*. \mathbf{X}_i has a numerical value denoted X_i which lies in the unit interval $[0,1]$. In SC's unipolar format, X_i is the probability p_{X_i} of a 1 appearing anywhere within \mathbf{X}_i . C also has a set of k ancillary inputs $\mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_k$, which are typically N -bit SNs of constant value $R_i = 0.5$. These constants are usually not user-supplied; rather they are an integral part of a stochastic circuit's design and enable the circuit to approximate a desired function to some desired level of accuracy. If the \mathbf{X}_i 's are derived from binary (base-2) numbers, additional randomness sources (not shown) are needed to perform binary-to-stochastic number conversion. In general, if C is combinational, it computes an arithmetic function of the form $Z(X_1, X_2, \dots, X_n, R_1, R_2, \dots, R_k)$. For example, when used as a stochastic adder, a two-way multiplexer computes the function

$$Z(X_1, X_2, R_1) = 0.5(X_1 + X_2) \quad (1)$$

where R_1 supplies the scaling factor 0.5.

An assumption made almost universally in determining the stochastic function computed by a circuit C is that its input bit-streams $X_1, X_2, \dots, X_n, R_1, R_2, \dots, R_k$ are independent. Roughly speaking, this means that the bit-streams should be as uncorrelated as possible. This requirement is assumed to be satisfied when all $n + k$ inputs are derived from $n + k$ separate Bernoulli randomness sources. *Correlation* therefore refers to stochastic signals that are related in some way, and it can take

many forms. Cross correlation, or simply correlation, occurs between two or more non-independent SNs. For example, the SN $\mathbf{X} = 10111010$ is highly correlated in a negative sense with $\mathbf{Y}_1 = 01000101$, since their 1s and 0s never overlap. The SN $\mathbf{Y}_2 = 10011000$ is also correlated with \mathbf{X} because its 1s always overlap the 1s of \mathbf{X} . The SN $\mathbf{Y}_3 = 01011101$, which is generated by rotating or shifting \mathbf{X} to the right by one bit, is not significantly cross correlated with \mathbf{X} , but the one-cycle-delayed version of \mathbf{Y}_3 is. Cross correlation may change the functionality of both combinational and sequential stochastic circuits by favoring certain input patterns. On the other hand, *temporal* correlation, or *autocorrelation*, refers to correlation between a bit-stream or part of a bit-stream and a delayed version of itself. For instance, $\mathbf{Y}_4 = 011001110$ contains some autocorrelation due to the fact that 01 is always followed by 1. Autocorrelation can severely affect the functionality of a sequential stochastic circuit by biasing it towards certain state-transition behavior.

Defining and measuring correlation is surprisingly difficult. A survey made in 2010 by Choi et al. [11] catalogs 76 different correlation metrics developed in different fields over many years, none of which is well suited to SC! Relatively easy to define is the independence or no-correlation assumption, which allows a stochastic circuit C 's SN inputs to be treated as Bernoulli processes, and the function of C to be expressed and analyzed using basic probability theory. For example, if two independent SNs \mathbf{X}_1 and \mathbf{X}_2 of value X_1 and X_2 , respectively, are applied to an AND gate, the output value Z is the arithmetic product X_1X_2 . This reflects the fact that the probability of the AND gate outputting a 1 is the probability of a 1 at the first input multiplied by the probability of a 1 at the second input, provided the inputs are not cross correlated. If \mathbf{X}_1 and \mathbf{X}_2 are correlated, Z can deviate from X_1X_2 in complex ways, as we will see shortly.

Random number sources (RNSs) play a central role in the design and operation of stochastic circuits. They provide the stochasticity needed by stochastic number generators (SNGs) to produce SNs with a sufficient level of independence, but they are a big contributor to overall hardware cost [25]. SC designers generally rely on linear feedback shift registers (LFSRs) as RNSs because of their relatively small size and low cost. An LFSR is a deterministic finite-state machine (FSM) whose behavior is pseudo-random, meaning that it only approximates a true random source [14]. An SC designer must usually optimize the use of RNSs in a way that provides sufficient randomness while meeting a cost budget.

SC is a type of approximate computing and trades off computational errors for other benefits. It has several error sources, as shown in Fig. 1. These error sources are peculiar to SC and do not include *physical* errors due to unreliable hardware or *soft* errors caused by environmental effects like cosmic radiation [8]. The errors in question are briefly summarized next.

Rounding Errors Errors caused by rounding or quantization reflect the fact that with N bits, a bit-stream can only represent exactly the $N + 1$ numbers in the set $S_N = \{0, 1/N, 2/N, \dots, (N-1)/N, 1\}$. If a desired number X is not in this set, then it must be rounded off to the nearest member of S_N . For instance, with $N = 16$ and

$X = 0.1555$, we can round X down to $2/16 = 0.1250$ or, slightly more accurately, round it up to $3/16 = 0.1785$. Rounding errors can be mitigated by increasing N to expand S_N . Note, however, that N must be doubled just to add 1 bit of precision to the numbers in S_N .

Approximation Errors These errors result from the fact that most arithmetic functions of interest cannot be implemented exactly by a stochastic circuit. As a result, they must be approximated by stochastic functions that are implementable. All stochastic function values must be scaled to lie in the unit interval $[0,1]$. Without constant \mathbf{R}_i 's as inputs, the only single-variable stochastic functions that can be combinational realized exactly are the trivial cases X and $1-X$. Hence, common arithmetic functions like X^2 , \sqrt{X} and $\sin(X)$ must be approximated by some synthesizable stochastic function of the form $Z(X, R_1, R_2, \dots, R_k)$. Only a few general methods for finding such functions are known; all are relatively complex and have particular design styles [4, 26]. For example, the ReSC synthesis method employs Bernstein polynomials with constant coefficients in the unit interval to approximate $Z(X)$ [25].

Random Fluctuations The (pseudo) random nature of the bits forming an N -bit SN \mathbf{X} as it emerges from an SNG is also a major error source. Fluctuations in \mathbf{X} 's bit-pattern cause its estimated or measured value \hat{X} to deviate from the target or exact value X . Since \mathbf{X} can have any of 2^N different bit-patterns, X and \hat{X} can differ significantly, especially when N is small. Figure 2 shows how three SNG-generated SNs fluctuate around their target value 0.5 as N changes. Such random fluctuation errors can be quantified by the mean square error (MSE) $E_{\mathbf{X}} = \mathbb{E}[(\hat{X} - X)^2]$. Like

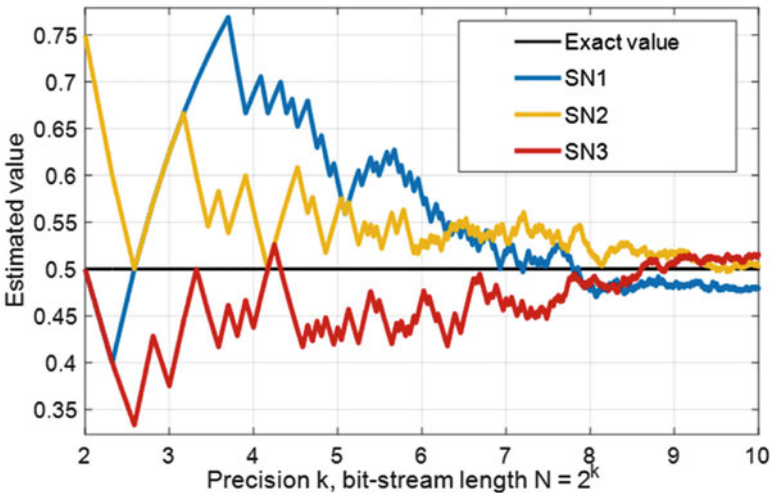


Fig. 2 Random fluctuations in three SNs with the exact value 0.5 as bit-stream length N increases [31]

rounding errors, random fluctuation errors tend to diminish with increasing N . Note, however, that when N is odd, \bar{X} must differ from $X = 0.5$ by at least one bit. Hence as N increases toward infinity, the graphs plotted in Fig. 2 continue to oscillate around 0.5 with a steadily decreasing MSE that approaches, but never reaches, zero.

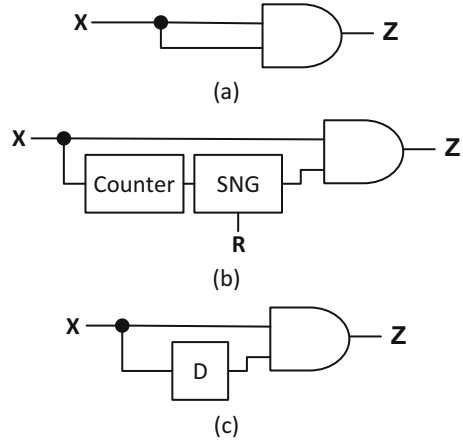
Constant-Induced Errors It was recently observed that the ancillary SNs $\mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_k$ (see Fig. 1) found in most SC designs are an unexpected and significant error source [31]. This is because their influence on the output value Z is subject to time-dependent random variations. Interestingly, constant-induced errors can be eliminated completely by removing the \mathbf{R}_i 's and transferring their function to sequential subcircuits inside C that track the behavior of the \mathbf{R}_i 's. A systematic algorithm called CEASE has been devised for efficiently removing constants and the errors they produce [31].

Correlation To maintain accuracy, it is often desirable that the bit-streams applied to a stochastic circuit retain their independence as they are being processed. This independence is reduced by correlation from several sources including: interactions among bit-streams during normal computation that introduce dependencies and similarities, poor randomness properties of individual RNSs that cause successive bits to be related, sharing of RNSs either directly or indirectly across the SNGs to reduce overall hardware costs, and temporal dependencies injected by sequential circuits. As a result, correlation errors tend to increase with circuit size and the number of layers of processing. They cannot be eliminated merely by increasing bit-stream length N .

At this point, we see that the accuracy of a stochastic circuit is impacted by many loosely related factors that are addressed by many different methods and are by no means fully understood. Correlation is amongst the most intractable of these factors. Figure 3 illustrates an example of how cross correlation can introduce errors and how to appropriately fix such errors. The problem here is to design a stochastic squarer to compute X^2 using the standard AND-gate-based multiplier described previously. To use it for squaring requires two independent, and therefore different, bit-streams with the same value X . This may be achieved by generating the bit-streams from two independent RNSs. However, the design of Fig. 3a uses a single input bit-stream \mathbf{X} that fans out into two identical, and therefore highly correlated copies that have a shared RNS and re-converge at the AND gate. Consequently, $Z = X$ instead of X^2 . This illustrates correlation due to RNS sharing and reconvergent fanout.

Figure 3b, c shows two ways to mitigate the correlation problem. The circuit in Fig. 3b converts one copy of \mathbf{X} from stochastic to binary and then back to stochastic again using a new RNS; this process is known as *regeneration*. As a result, the AND gate sees two independent SNs of value X and so computes a good approximation to X^2 . The design of Fig. 3c employs a D flip-flop called an isolator [13] to delay one copy of \mathbf{X} by a clock cycle. Instead of seeing the same bit $\mathbf{X}(t)$ twice in clock cycle t , the AND gate sees $\mathbf{X}(t)$ and $\mathbf{X}(t-1)$, which are independent by the Bernoulli

Fig. 3 Three designs for a squarer: (a) Incorrect design that ignores correlation. Decorrelated designs using (b) regeneration, and (c) isolation [29]



property. This method of decorrelation is termed *isolation* and is usually much less expensive than regeneration [29].

Some stochastic operations, notably the scaled addition of Eq. (1) implemented by a multiplexer, do not require their inputs to be independent. Such circuits are said to be *correlation insensitive* (CI) [5]. The CI property allows the two input SNs \mathbf{X}_1 and \mathbf{X}_2 of the adder to share a common RNS without producing correlation-based errors of the type illustrated by Fig. 3a. This can be explained by the fact that the adder's output bit $\mathbf{Z}(t)$ at clock cycle t is either $\mathbf{X}_1(t)$ or $\mathbf{X}_2(t)$, so there is no interaction between the two data inputs.

While correlation usually reduces the accuracy of stochastic circuits, in some cases its deliberate use can change a circuit's function to a new one that is advantageous in some way [2]. For example, an XOR (exclusive-OR) gate supplied with uncorrelated inputs \mathbf{X}_1 and \mathbf{X}_2 realizes the not-so-useful function $X_1 + X_2 - X_1X_2$. If the inputs are positively correlated by enforcing maximum overlap of 1s, the XOR realizes the absolute difference function $|X_1 - X_2|$. This has been used to design an edge-detector for image processing that contains orders of magnitude fewer gates than a comparable non-stochastic circuit [2]. Correlation is similarly used in the design a stochastic division circuit CORDIV that has accuracy advantages [10].

The design and optimization of RNSs for correlation management are also an important issue in SC [2, 23]. The problems fall into two categories: (1) designing RNSs and SNGs to generate bit-streams with desirable cross correlation and autocorrelation properties, and (2) strategically reducing the use of RNSs to decrease hardware cost while maintaining moderate independence requirements for SNs. The latter problem usually requires inexpensive re-randomization techniques and can take advantage of any CI properties for RNS sharing. Making effective use of correlation in SC is by no means well understood and is a subject of on-going research.

The rest of the chapter is organized as follows. Section “Measuring Correlation” reviews the SC correlation metric for correlation measurement and describes how

it can be used to analyze and design stochastic circuits. Section “Correlation-Controlling Units” introduces correlation-controlling circuits intended to eliminate or insert correlation. Correlation insensitivity and its role in avoiding correlation problems are discussed in section “Correlation Insensitive Stochastic Circuits”. Finally, section “Design and Optimization of RNSs” explains how careful RNS design can be exploited to engineer correlation and reduce hardware cost.

Measuring Correlation

Being able to measure and quantify correlation rigorously in the SC context is of crucial importance in the analysis and the design of stochastic circuits. We begin by examining stochastic cross correlation SCC , a correlation measure specifically designed for SC. We then discuss how the function of a stochastic circuit changes with different values of SCC .

Quantifying Correlation with SCC

An early effort to quantify correlation for SC was made by Jeavons et al. [16]. Instead of directly providing a correlation measure for SC, they define two SNs \mathbf{X} and \mathbf{Y} as independent or uncorrelated if the value of the SN \mathbf{Z} obtained from ANDing \mathbf{X} and \mathbf{Y} is XY . This definition effectively says that two SNs are independent if a stochastic multiplier can compute their product accurately. Obviously, this definition of independence assumes the computation to be otherwise error-free, i.e., it has no random fluctuation errors, rounding errors, etc. However, it is rarely the case that \mathbf{Z} 's value is exactly XY , even when \mathbf{X} and \mathbf{Y} are generated using independent RNSs. With only this definition of independence, it remains challenging to quantify the behavior of stochastic circuits under different levels of correlation.

Table 1 shows how the function of an AND-based multiplier changes under the influence of correlation. The multiplier performs as expected when the inputs \mathbf{X} and \mathbf{Y} are independent. However, it computes $Z = \min(X, Y)$ when \mathbf{X} and \mathbf{Y} are

Table 1 SC functions implemented by a two-input AND gate with different levels of input SN correlations

	X	Y	$X \wedge Y$	Function
Uncorrelated	01010101(0.5)	11110011(0.75)	01010001(0.375)	$X \times Y$
Positively correlated	11110000(0.5)	11111100(0.75)	11110000(0.5)	$\min(X, Y)$
Negatively correlated	11110000(0.5)	00111111(0.75)	00110000(0.25)	$\max(0, X + Y - 1)$

maximally correlated in the positive sense, i.e., when the 1s in \mathbf{X} and \mathbf{Y} overlap as much as possible. On the other hand, it computes $Z = \max(0, X + Y - 1)$ when the 1s in \mathbf{X} and in \mathbf{Y} overlap as little as possible. Instead of using vague terms like maximally correlated or negatively correlated, it is desirable to be able to rigorously quantify correlation for SC. Unfortunately, none of the 76 correlation measures summarized in [11] perfectly fits the needs of SC, including the Pearson correlation measure ρ which is widely used in statistical analysis. Pearson correlation presents a problem for SC, because its value depends on the actual value of the bit-streams being compared. For example, the maximum Pearson correlation value $\rho = +1$ implies that the bit-streams are identical. This means that bit-streams having different values, even if their 1s maximally overlap, fail to attain the maximum value of ρ .

A suitable correlation metric for SNs would yield a value $+1$ for maximum overlapping of 1s and 0s, a value -1 for minimum overlapping of 1s and 0s, and a value 0 for independent SNs. The metric should not be impacted by the actual value of the SN, and should also provide intuitive functional interpolation for correlation value other than $+1$, -1 or 0 .

The correlation measure called the *SC correlation coefficient* or *stochastic cross correlation (SCC)* has been proposed to fit SC's needs [2]. For a pair of SNs \mathbf{X} and \mathbf{Y} , SCC is defined as follows

$$SCC(\mathbf{X}, \mathbf{Y}) = \begin{cases} \frac{p_{\mathbf{X}\wedge\mathbf{Y}} - p_{\mathbf{X}}p_{\mathbf{Y}}}{\min(p_{\mathbf{X}}, p_{\mathbf{Y}}) - p_{\mathbf{X}}p_{\mathbf{Y}}} & \text{if } p_{\mathbf{X}\wedge\mathbf{Y}} > p_{\mathbf{X}}p_{\mathbf{Y}} \\ \frac{p_{\mathbf{X}\wedge\mathbf{Y}} - p_{\mathbf{X}}p_{\mathbf{Y}}}{p_{\mathbf{X}}p_{\mathbf{Y}} - \max(p_{\mathbf{X}} + p_{\mathbf{Y}} - 1, 0)} & \text{otherwise} \end{cases} \quad (2)$$

where $p_{\mathbf{X}} = X$, $p_{\mathbf{Y}} = Y$, and $p_{\mathbf{X}\wedge\mathbf{Y}} = p(\mathbf{X}(t) = 1, \mathbf{Y}(t) = 1)$ for all t . To measure SCC for SNs \mathbf{X} and \mathbf{Y} , $p_{\mathbf{X}\wedge\mathbf{Y}}$ is first computed which is the probability of both \mathbf{X} and \mathbf{Y} being 1. This value is then centralized by shifting by the estimate for $-p_{\mathbf{X}}p_{\mathbf{Y}}$ that corresponds to the uncorrelated value. Centralization ensures uncorrelated bit-streams yield $SCC = 0$, which is consistent with both Pearson correlation and the definition of independence in [16]. Finally, the centralized value is normalized by dividing it by the maximum attainable value. Normalization guarantees that for two maximally similar or different SNs \mathbf{X} and \mathbf{Y} , $SCC(\mathbf{X}, \mathbf{Y})$ has value $+1$ or -1 , respectively. Unlike the Pearson correlation measure $\rho(\mathbf{X}, \mathbf{Y})$, $SCC(\mathbf{X}, \mathbf{Y})$ does not vary with the value of the SNs.

Equivalently, and perhaps more intuitively, SCC can also be defined in terms of how often 1s and 0s occur in the two SNs. For two N -bit SNs \mathbf{X} and \mathbf{Y} , let $N_{i,j}$ denote the number of occurrences of the 2-bit pattern $\mathbf{X}(t)\mathbf{Y}(t) = ij$. Obviously, $N_{00} + N_{01} + N_{10} + N_{11} = N$. We can then compute SCC by replacing the estimates of $p_{\mathbf{X}}$, $p_{\mathbf{Y}}$, and $p_{\mathbf{X}\wedge\mathbf{Y}}$ in Eq. (2) by $(N_{11} + N_{10})/N$, $(N_{11} + N_{01})/N$ and N_{11}/N , respectively, to obtain

$$SCC(\mathbf{X}, \mathbf{Y}) = \begin{cases} \frac{N_{11}N_{00} - N_{10}N_{01}}{N \times \min(N_{11} + N_{10}, N_{11} + N_{01}) - (N_{11} + N_{10})(N_{11} + N_{01})} & \text{if } N_{11}N_{00} > N_{10}N_{01} \\ \frac{N_{11}N_{00} - N_{10}N_{01}}{(N_{11} + N_{10})(N_{11} + N_{01}) - N \times \max(N_{11} - N_{00}, 0)} & \text{otherwise} \end{cases}$$

where the numerator $N_{11}N_{00} - N_{10}N_{01}$ is common to many correlation measures including Pearson correlation:

$$\rho(\mathbf{X}, \mathbf{Y}) = \frac{N_{11}N_{00} - N_{10}N_{01}}{\sqrt{(N_{11} + N_{10})(N_{11} + N_{01})(N_{00} + N_{01})(N_{00} + N_{01})}}$$

The major difference between SCC and ρ lies in the denominator. SCC normalizes the measure in a way that maximally positively/negatively correlated SNs would produce a $+1/-1$ correlation value. Pearson correlation, on the other hand, is normalized by the variance of the bit-streams, which does depend on the value of the bit-streams.

Table 2 shows some examples of SN pairs and compares their ρ and SCC values. Note that ρ and SCC are the same for independent SNs and for SNs with equal values. When the SNs have different values, SCC consistently gives the value $+1$ and -1 when the maximal overlap and minimal overlap of 1s and 0s occur, respectively.

The SCC metric of correlation provides a precise way to define a circuit’s stochastic behavior under the influence of various (cross) correlation levels. It further allows us to explore new SC designs enabled by intentionally introducing non-zero correlations. Figure 4 shows a pair of SNs \mathbf{X} and \mathbf{Y} having $SCC(\mathbf{X}, \mathbf{Y}) = +1$ applied to an XOR gate, which computes $X + Y - 2XY$ if \mathbf{X} and \mathbf{Y} are independent. The correlation between the inputs changes the circuit’s functionality to the potentially more useful absolute difference function, which leads to a highly efficient way of implementing edge detection in SC-based vision chips [6]. This illustrates the usefulness of deliberately injected correlation in designing stochastic circuits.

So far, we have only discussed cross correlation between SNs. Autocorrelation in stochastic circuits is much less well understood. Except the standard autocorrelation metric used in signal processing, an autocorrelation measure that is suitable for SC appears to be lacking. Almost all existing SC designs therefore assume the

Table 2 Example SNs with their SCC and Pearson correlation values [2]

SN pattern	SCC(X, Y)	$\rho(\mathbf{X}, \mathbf{Y})$
$\mathbf{X} = 11110000 \quad \mathbf{Y} = 11001100$	0	0
$\mathbf{X} = 11110000 \quad \mathbf{Y} = 00001111$	-1	-1
$\mathbf{X} = 11111100 \quad \mathbf{Y} = 11110000$	1	0.58
$\mathbf{X} = 11111100 \quad \mathbf{Y} = 00001111$	-1	-0.58
$\mathbf{X} = 11111100 \quad \mathbf{Y} = 11100001$	0	0
$\mathbf{X} = 11000000 \quad \mathbf{Y} = 11111100$	1	0.33



Fig. 4 XOR gate with maximal positively correlated inputs which implements the absolute-difference subtraction function $|X - Y|$ [2]

inputs to be free of autocorrelation. It is well recognized that the presence of autocorrelation, if not carefully handled, presents accuracy problems in sequential SC designs, including applications such as Bayesian inference [12], LDPC decoding [24, 27], and neural networks [9]. There are, fortunately, several ways to mitigate autocorrelation in SC, which we discuss in section “Correlation-Controlling Units”.

Stochastic Functions with Different SCC Levels

Table 1 shows the functionality of the AND gate at SCC 0, +1, and -1. To derive the stochastic function of the AND gate at any other SCC level, we need to calculate the linear combination of the function at $SCC = 0$ and the function at $SCC = +1$ or -1 , depending on the direction of the correlation [2]. For instance, the AND gate with $SCC = 0.5$ implements the function $Z = 0.5(\min(X, Y) + XY)$. In the general case, if we have a circuit implementing a two-input Boolean function $z = f(x, y)$ with input SNs \mathbf{X} and \mathbf{Y} having arbitrary correlation level SCC , the value of SN \mathbf{Z} at the output of the circuit will be

$$Z = \begin{cases} (1 + SCC) \cdot F_0 + SCC \cdot F_{-1} & \text{if } SCC(\mathbf{X}, \mathbf{Y}) < 0 \\ (1 - SCC) \cdot F_0 + SCC \cdot F_{+1} & \text{otherwise} \end{cases} \quad (3)$$

Here F_0 , F_{+1} and F_{-1} denote the stochastic function implemented by the same circuit at SCC levels 0, -1 and +1, respectively. Using probabilistic transfer matrices (PTMs), Alaghi and Hayes [2] show that for any two-input combinational circuit, we can derive F_0 , F_{-1} , and F_{+1} via the following matrix multiplication

$$[i_0 \ i_1 \ i_2 \ i_3] \cdot [t_0 \ t_1 \ t_2 \ t_3]^T$$

in which the t_k 's denote the truth table of the corresponding Boolean function and the i_k 's are obtained from Table 3. As an example, suppose we want to derive the stochastic function implemented by an XOR gate at SCC levels 0 and +1. The truth table PTM of the XOR gate is $[0 \ 1 \ 1 \ 0]^T$, so we will have $F_0 = (1 - X) \cdot Y + (1 - Y) \cdot X$ and $F_{+1} = \max(Y - X, 0) + \max(X - Y, 0) = |X - Y|$. To find the stochastic function of the XOR gate with $SCC = 0.25$, we simply calculate the linear combination $F_{0.25} = 0.75F_0 + 0.25F_{+1}$.

Table 3 PTM elements used to derive the stochastic function of a two-input combinational circuit at SCC levels 0, -1 and +1 [2]

	$F_0, SCC = 0$	$F_{-1}, SCC = -1$	$F_{+1}, SCC = +1$
i_0	$(1 - X) \cdot (1 - Y)$	$\max(1 - X - Y, 0)$	$\min(1 - X, 1 - Y)$
i_1	$(1 - X) \cdot Y$	$\min(1 - X, Y)$	$\max(Y - X, 0)$
i_2	$(1 - Y) \cdot X$	$\min(1 - Y, X)$	$\max(X - Y, 0)$
i_3	$X \cdot Y$	$\max(X + Y - 1, 0)$	$\min(X, Y)$

Deriving the stochastic function of circuits or Boolean function with more than two inputs is not trivial, because SCC does not extend easily to multiple inputs. The most convenient method of quantifying correlation between more than two inputs is done by using PTMs, which enumerate the probability distribution of any combination of 0s and 1s among the signals. However, a systematic method of handling multi-input functions with arbitrary SCC levels is not known, except in a few special cases. One such case is when all a function's inputs are independent [1]. When all the inputs are maximally positively correlated with $SCC = +1$, we may also be able to derive the circuit's stochastic function. For instance, a k -input AND gate with maximally correlated inputs $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_k$ implements the function $\min(X_1, X_2, \dots, X_k)$.

Correlation-Controlling Units

As noted previously, managing correlation is among the more challenging design tasks in SC, as correlation plays a key role in defining a stochastic circuit's accuracy and functionality. It involves complex trade-offs between hardware cost, latency, and the desired level of accuracy. This section reviews some of the approaches proposed for managing correlation in SC using correlation-controlling units, a class of sub-circuits inserted into stochastic arithmetic units to increase or reduce correlation levels. The major design consideration in deploying correlation controllers is to achieve an appropriate level of correlation while meeting hardware and latency constraints. Although correlation can be viewed as a resource to facilitate the design of certain stochastic functions, most stochastic circuits are intended to work with independent SNs. Therefore, much prior work in correlation control was devoted to decorrelation, the process of eliminating undesired correlation completely (i.e., making SCC as close to 0 as possible). There have also been efforts in designing circuits to increase or decrease SCC , such that the SCC is pushed towards $+1$ or -1 , respectively.

Regeneration-Based Decorrelation

Perhaps the most direct way to eliminate correlation is through *regeneration*, where SNs are first converted to binary form using stochastic-to-binary converters, and then are converted back to SNs by SNGs with suitably independent RNSs. A regenerated SN has a value which is the same as, or very close to, its original value. However, the positions of its 1s are expected to be different.

An example of regeneration-based decorrelation is shown in Fig. 3b, where the goal is to produce one of the two copies of \mathbf{X} using an RNS that is independent of the original \mathbf{X} . In this example, it is sufficient to regenerate \mathbf{X} such that the two inputs of the multiplier are not cross correlated, as the multiplier is a combinational

stochastic circuit. However, regeneration-based decorrelation can also be applied to remove autocorrelation, which can be a major factor impacting the accuracy of sequential stochastic circuits. This is because the process of regeneration provides an opportunity to completely reorder the bits in the SN being regenerated. To eliminate autocorrelation in the target SNs, it is sufficient to use RNSs that have no autocorrelation.

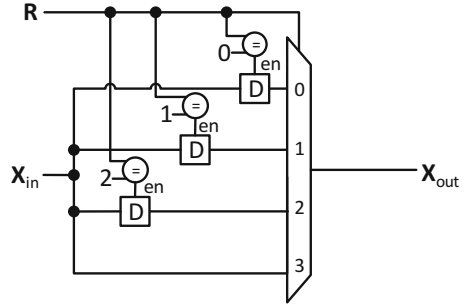
In principle, regeneration-based decorrelation can solve most types of correlation problems. However, it is also subject to several design constraints that must be taken into consideration when deploying it. First, regeneration incurs significant latency overhead and stalls computation. This is because it requires a stochastic-to-binary format conversion that is implemented using a standard binary counter, which simply counts the number of 1s in the incoming SN. This process can take as many clock cycles as the length N of the SN. The arithmetic units after the regeneration-based decorrelator cannot start computation until conversion completes. Given that many SC applications have N s in the thousands, regeneration can easily incur a latency of many thousands of clock cycles. Furthermore, the hardware overhead for a regeneration-based decorrelator is very high. In the squarer example shown in Fig. 3b, the overhead of regenerating a single SN is a counter, a stochastic number generator (SNG) which is essentially a comparator, and a new independent RNS, which is usually a precious resource in SC systems. The actual hardware cost depends on and grows with the length of the SN to be decorrelated. When compared to the main arithmetic component, which is a single AND gate in this case, the hardware overhead is extremely high. A study of a typical SC image-processing system, reported that more than 80% of its area came from circuits for SN generation [25]. Excessive use of regeneration-based decorrelators can consequently degrade the performance and increase the cost of a stochastic circuit drastically. It is therefore good practice to consider other types of decorrelation when feasible.

Lastly, it is worth noting that some variants of the regeneration-based method actively trade higher accuracy for less hardware area and latency by regenerating each bit of the SN based on a small portion of the bit-stream [28]. Since each regenerated bit is based on partial information from the original SN, the regeneration process can incur more errors, but in return has shorter latency and less hardware. For example, the single-ended counter hysteresis (SCH) decorrelator proposed in [28] keeps updating the current estimate of the input's binary form at each clock cycle through a dynamically balanced system that reaches steady state quickly. This is enabled by using a small counter for stochastic-to-binary conversion that is usually much smaller than the full counter needed to count the entire input SN.

Shuffle-Based Decorrelation

Similar to regeneration, shuffling performs decorrelation by re-positioning 1s in the target SN [19]. The difference is that the shuffling method limits the likelihood of

Fig. 5 Shuffle-based decorrelator of depth 3, where \mathbf{R} is a random number uniformly distributed among 0, 1, 2 and 3 [19]



moving a 1 to a faraway location. In other words, a 1 appearing in the t -th bit of the original SN is likely to be placed somewhere near the t -th bit of the new SN. Shuffling works by first observing and storing a subset of bits from the initial SN, and then using a new RNS to select a stored bit per cycle for outputting. Figure 5 shows a shuffler design with depth $D = 3$, meaning that it can store 3 bits for future output. The RNS here is a random number R taking a value in $\{0, 1, 2, 3\}$ uniformly at each clock cycle. Each newly received bit from the input is stored in the shuffler. At the same time, one of the stored bits (including the newly received one) is selected for output. The output SN therefore contains the same number of 1s as the input SN, since the shuffler only outputs what it received. However, the position of 1s can be very different, hence achieving the effect of decorrelation.

The advantages of the shuffle-based decorrelation include relatively low latency and low hardware cost compared to the regeneration-based method. Typical numbers for the depth D are relatively small, like 4, 6 or 8. This not only allows the received bits from the input bit-stream to be stored directly in stochastic form without inducing excessive hardware overhead, but also reduces the number of bits in the random source serving as the selection signal. The latency for the shuffling method, i.e., the number of clock cycles required before generating the first output bit, is roughly equal to the depth D , which is low compared to regeneration, especially when the input SN is very long.

Shuffling re-randomizes the position of 1s in an SN, and can thus cope with both cross correlation and autocorrelation. However, it may not completely eliminate correlation, but rather only reduce it. This is because the new SN generated by shuffling is not completely independent of the original SN. Consider an SN of value 0.5, with all 1s in the first half of the bit-stream and all 0s in the second half. In this case, the shuffler would receive and store only 1s in the beginning, and so has no choice but to output 1s initially. After the 0s start coming into the shuffler, the 1s remaining in the shuffler will quickly be released to the output, and the likelihood that a 1 is released into the output at a very late clock cycle is low. One can expect that the output SN will tend to have more 1s in the first half and more 0s in the second half. This is, however, very unlikely for a Bernoulli bit-stream. Shuffling and its variants have been successfully deployed in applications like Bayesian inference [12] and LDPC decoding [27, 28] to reduce autocorrelation.

Isolation-Based Decorrelation

Unlike the aforementioned decorrelation methods, isolation does not alter the positions of 0s and 1s in the SN. It was proposed in the 1960s [13] mainly to cope with cross correlation by adding appropriate delays to SNs. The added delays shift SNs temporally so that correlated bits from different SNs are staggered. An example of isolation-based decorrelation appears in Fig. 3c, where the isolator (a delay element implemented by a D flip-flop) is inserted into one of the two inputs of the squarer. By delaying one copy of \mathbf{X} by one clock cycle, the output $\mathbf{Z}(t) = p(\mathbf{X}(t) = 1, \mathbf{X}(t - 1) = 1) = p(\mathbf{X}(t) = 1)p(\mathbf{X}(t - 1) = 1)$, so $Z = X^2$ as expected, provided that $\mathbf{X}(t)$ and $\mathbf{X}(t - 1)$ are statistically independent for all t , as asserted by the Bernoulli property.

The major advantages of isolation are very low hardware cost and low latency, compared to regeneration. However, the application of isolators tends to be difficult. Carelessly placing isolators in a stochastic circuit can lead to several problems, such as failure to decorrelate correctly and unexpectedly changing the circuit’s function. These problems occur when the placement fails to track and delay correlated signals properly for some signal lines since isolators can inject undesired autocorrelation into the circuit and some isolators can turn out to be unnecessary. Figure 6a shows a stochastic circuit that is intended to compute X^4 by naïvely cascading two squarer circuits of the kind in Fig. 3c. While this construction appears to make sense at the first sight, the resulting circuit does not compute X^4 as expected; instead, it computes $Z = X^3$, a huge functional error! To see this, observe that at time t , the output of the first AND gate is $\mathbf{X}(t) \wedge \mathbf{X}(t - 1)$, and therefore the inputs to the second AND gate are $\mathbf{Y}_1(t) = \mathbf{X}(t) \wedge \mathbf{X}(t - 1)$ and $\mathbf{Y}_2(t) = \mathbf{X}(t - 1) \wedge \mathbf{X}(t - 2)$. By ANDing these two bit-streams, we get the final output as $\mathbf{Z}(t) = \mathbf{Y}_1(t) \wedge \mathbf{Y}_2(t) = \mathbf{X}(t) \wedge \mathbf{X}(t - 1) \wedge \mathbf{X}(t - 2)$, implying that $Z = XXX = X^3$. The cause of this error is unanticipated autocorrelation. Note that the squarer is implemented by an AND gate and an isolator, which effectively makes the circuit sequential. The adjacent bits of

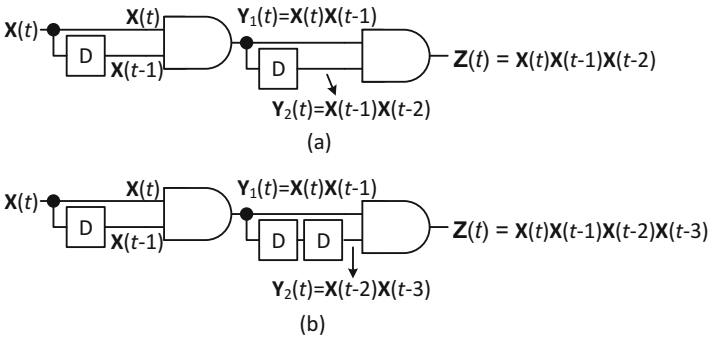


Fig. 6 Stochastic implementation of X^4 using isolator flip-flops; (a) an incorrect implementation with insufficient isolators; (b) a correct implementation with three isolators

the squarer's output bit-stream are correlated. Therefore, delaying this bit-stream by only one clock cycle yields a cross-correlated SN. A correct implementation of X^4 is given in Fig. 6b, where the second squarer has two isolators inserted in the bottom input line.

Generally speaking, isolators must be inserted in a way that all undesired correlations between interacting SNs are eliminated. Finding a correct isolator placement while minimizing the isolator usage is a challenging problem. An isolator insertion algorithm called VAIL has been proposed for combinational stochastic circuits [29]. It formulates isolator insertion as a linear integer program, where the objective is to minimize the isolator count. A set of constraints are enforced on the number of isolators that can be placed on each line of the circuit to be decorrelated. These constraints, when satisfied, ensure that undesired correlation between interacting SNs is removed without affecting other SN interactions.

Correlation Injection by Synchronization

While almost all stochastic circuits are designed to work with uncorrelated inputs, there exist circuits implementing useful functions enabled by positively or negatively correlated inputs. For example, if the XOR gate in Fig. 4 is used to compute absolute difference, it requires its two inputs to be maximally correlated. To generate inputs with predetermined correlation for such circuits, one can resort to special types of SNGs that are capable of controlling the amount of correlation. However, regenerating SNs with specific correlation levels in the middle of an SC system is expensive, both in hardware cost and in system latency.

In error-tolerant SC applications such as many machine-learning and image-processing tasks, another way to inject correlation is to use a sequential unit called a *synchronizer*, which attempts to maximize the correlation level between a pair of SNs [19]. This approach, while providing no guarantee of attaining the desired correlation, is usually far less expensive than regeneration in terms of hardware and latency cost. Figure 7a shows the state-transition graph of a three-state synchronizer, whose key idea is to align the bits with the same value from inputs \mathbf{X} and \mathbf{Y} as much as possible. For example, when the synchronizer receives the pattern $\mathbf{X}(t)\mathbf{Y}(t) = 01$, it will output 00 and then go from state S_0 to S_2 , which remembers the 1 received from \mathbf{Y} for later release. If $\mathbf{X}(t)\mathbf{Y}(t) = 10$ is received, then the synchronizer will return to S_0 and output 11. This effectively transforms $\mathbf{X}(t)\mathbf{Y}(t) = (01, 10)$ to $(00, 11)$, which has obviously become more correlated.

Observe that the synchronizer in Fig. 7a does not guarantee that its outputs will have exactly the same value as \mathbf{X} and \mathbf{Y} . This synchronizer-induced error occurs when the computation ends at any state other than S_0 , and hence there are some remembered bits yet to be released into the outputs. Also, the synchronizer only increases the correlation level; it does not guarantee that the output will be maximally correlated. In fact, it does not provide any promises on the final correlation level of the outputs. This is because this synchronizer can only remember

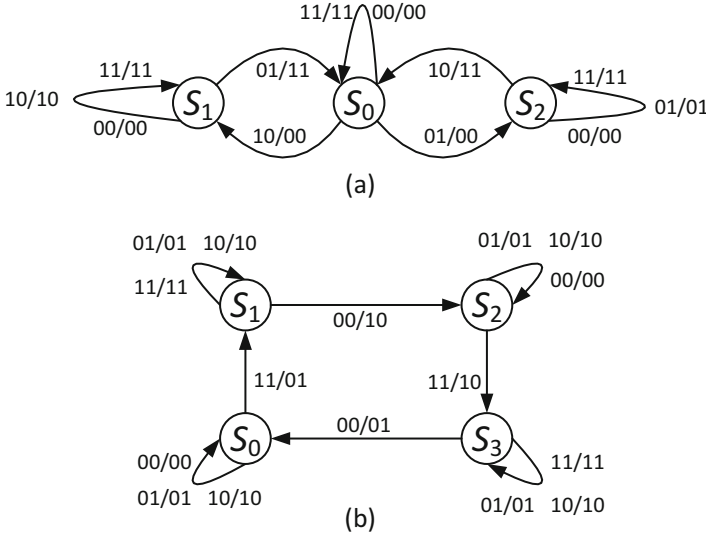


Fig. 7 State-transition graphs for correlation-controlling units that inject correlation between a pair of SN: (a) synchronizer that increases SCC; (b) desynchronizer that reduces SCC [19]

one unreleased bit from either \mathbf{X} or \mathbf{Y} . Thus, at state S_0 , if two consecutive bit patterns $\mathbf{XY} = (01, 01)$ are received, the synchronizer will have no choice but to release a 1 from \mathbf{Y} without matching it with another 1 from \mathbf{X} . In that case, the output will be $(00, 01)$, and the synchronizer will end at state S_2 . In general, increasing the number of states allows the synchronizer to remember more yet-to-be-aligned bits, and hence can produce outputs that are more correlated. But this comes at the cost of more synchronizer-induced error, because the probability of ending at a state other than the initial state is higher.

Based on the synchronizer concept, we can push the *SCC* of two SNs towards -1 using a desynchronizer. The state-transition graph of a four-state desynchronizer is depicted in Fig. 7b. Like the synchronizer, the desynchronizer takes two input SNs \mathbf{X} and \mathbf{Y} , and generates two output SNs with the same value but with stronger negative correlation or an *SCC* closer to -1 . The key idea in the desynchronizer design is to intentionally misalign bits of the same value while still preserving the encoded SN value. To do this, the desynchronizer selectively absorbs and releases bits to maximize the occurrence of the patterns $\mathbf{XY} = (10)$ and (01) , and minimize the occurrence of the patterns $\mathbf{XY} = (11)$ and (00) . If the desynchronizer receives the pattern $\mathbf{XY} = (11)$, it will pass one of the bits and save the other bit to emit later. In the desynchronizer design shown in Fig. 7b, the FSM alternates between storing \mathbf{X} and \mathbf{Y} when it receives $\mathbf{XY} = (11)$ but alternative variants are possible. When the desynchronizer receives the pattern $\mathbf{XY} = (00)$ it will emit the stored bit in the FSM to misalign the bits. If the desynchronizer receives the pattern $\mathbf{XY} = (01)$ or (10) , it will simply pass the inputs to the outputs since the bits at that SN offset

are already different. This effectively yields more negatively correlated SNs. For instance, the input pattern $\mathbf{XY} = (11, 00)$ becomes $\mathbf{XY} = (01, 10)$ after passing through the desynchronizer.

The desynchronizer has similar tradeoffs to the synchronizer. Bits that get saved in the desynchronizer may not be emitted before the end of execution which can yield a slight negative bias. Notice also that the desynchronizer FSM can only save one bit at a time. As a result, there are cases where it may be forced to pass the pattern $\mathbf{XY} = (11)$ or (00) . For instance, if the desynchronizer receives the pattern $\mathbf{XY} = (11, 11)$ it will output $(01, 11)$. In this case, the desynchronizer absorbs a bit from the first occurrence of $\mathbf{XY} = (11)$ but not from the second $\mathbf{XY} = (11)$. This forces the desynchronizer to simply pass $\mathbf{XY} = (11)$ to the output on the second occurrence. This limitation can be addressed by augmenting the desynchronizer to allow it to absorb more bits to improve its efficacy. Again, this increases the potential error due to bits that get saved in the FSM but are not released before the end of execution.

To illustrate the strengths and weaknesses of each correlation manipulation technique, consider an image processing pipeline which consists of a 3×3 Gaussian blur followed by a Roberts Cross edge detector. The Gaussian blur kernel requires input SNs for each multiply in the kernel to be uncorrelated, while the Roberts Cross edge detector requires inputs to the subtractor to be positively correlated. Figure 8 shows the resulting image along with energy efficiency and average absolute error for three different configurations: (1) no correlation correction between kernels, (2) regeneration before the edge detector, and (3) synchronizers before the edge detector. Absolute error is measured as the deviation from a floating-point baseline implementation. The resulting image without any correlation correction at all clearly suffers from significant accuracy losses. Using correlation controlling circuits like regeneration or the synchronizer, on the other hand, leads to much more accurate results. The synchronizer is more energy efficient and yields comparable accuracy to regeneration.





	Floating point	No correction	Regeneration	Synchronizer
Image result				
Energy	-	1383 nJ / frame	1971 nJ / frame	1505 nJ / frame
Absolute error	0	0.076	0.019	0.020

Fig. 8 Image processing case study results for Gaussian blur kernel followed by Roberts Cross edge detector [19]

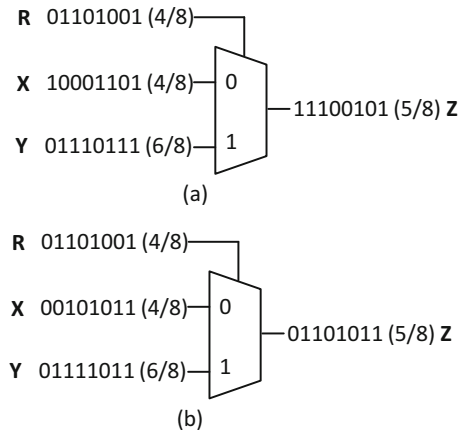
Correlation Insensitive Stochastic Circuits

Next we address a desirable SC property called correlation insensitivity, which is found in some stochastic circuits. *Correlation insensitive* (CI) circuits are immune to the effects of cross correlation or autocorrelation. In other words, a designer does not have to worry about the correlation among certain CI inputs, and can treat them as if they are uncorrelated. Our discussion will focus on cross-correlation insensitivity (cross-CI), including the detection and the characteristics of cross-CI. We will also briefly review autocorrelation insensitivity. Finally, we discuss some special classes of stochastic circuits that have distinctive autocorrelation features.

Cross-Correlation Insensitivity

Cross-correlation insensitivity (cross-CI) implies that a stochastic circuit C 's function is not impacted by cross correlation between some of its primary input SNs. Roughly speaking, cross-CI occurs if two SNs cannot simultaneously impact the circuit's outputs. This is a very useful property for stochastic circuits, especially for their input lines to which user-supplied and potentially correlated SNs are applied. For instance, Fig. 9a shows a standard multiplexer-based stochastic adder that computes $Z = 0.5(X + Y)$ with independent input SNs X and Y , while Fig. 9b shows the same adder but with highly correlated X and Y . Here R is an SN with a fixed probability 0.5, and must be independent of both of the user-supplied inputs X and Y . On the other hand, X and Y can be cross correlated in any way without affecting the adder's functionality. This is because at each clock cycle, either a bit from X or a bit from Y is selected by R to send to the output Z , i.e., Z can never depend on both X and Y at the same time.

Fig. 9 (a) Stochastic addition showing accurate results with (a) uncorrelated and (b) correlated inputs X and Y



A formal definition of cross-CI for combinational stochastic circuits is given in [5], which states that for a stochastic circuit C that implements the Boolean function $z = f(x_1, x_2, \dots, x_n)$, x_i and x_j form a cross-CI pair for C , if and only if the following holds:

$$(dz/dx_i) \wedge (dz/dx_j) = 0 \quad (4)$$

where 0 denotes the zero Boolean function and dz/dx_i denotes the Boolean difference of z with respect to x_i , i.e.,

$$\begin{aligned} dz/dx_i &= f(x_1, x_2, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) \oplus \\ & f(x_1, x_2, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \end{aligned}$$

A proof of Eq. (4) can be found in [5]; here we provide a brief intuitive explanation. The Boolean difference dz/dx_i is a Boolean function of (x_1, x_2, \dots, x_n) whose minterms correspond to the input assignments such that a change of x_i 's value will lead to a change of z 's value. Therefore, Eq. (4) simply says that if there is no input assignment such that x_i 's value change and x_j 's value change can each change z 's value, then x_i and x_j form a CI pair.

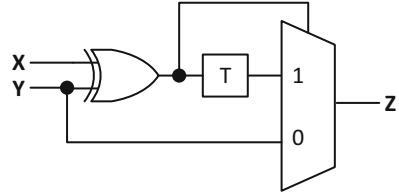
The preceding definition is useful for identifying CI pairs in a given stochastic circuit. For example, recall that the multiplexer in Fig. 9 implements the function $Z = 0.5(X + Y)$ in the stochastic domain, and the function $z = x \wedge r' \vee y \wedge r$ in the Boolean domain. Here x and y form a CI pair, because

$$\begin{aligned} dz/dx &= (y \wedge r) \oplus (r' \vee y) = r' \\ dz/dy &= (y \wedge r') \oplus (r \vee x) = r \end{aligned}$$

so $(dz/dx) \wedge (dz/dy) = r \wedge r' = 0$, which confirms that x and y are indeed a CI pair. On the other hand, x and r are not a CI pair because $dz/dr = x \oplus y$ and $(dz/dx) \wedge (dz/dr) \neq 0$. From this, we see that in the multiplexer-based adder, \mathbf{X} and \mathbf{Y} can be correlated without incurring any correlation error. However, \mathbf{R} must be independent of both \mathbf{X} and \mathbf{Y} for the adder to work accurately.

The ancillary input \mathbf{R} in the adder case is introduced to provide the scaling operation needed by this type of adder. However, it imposes a constraint of independence that must be satisfied by using an RNS that is uncorrelated to both \mathbf{X} and \mathbf{Y} . This not only incurs a large hardware overhead, it also injects unnecessary random fluctuation into the circuit, causing a constant-induced error [31]. It turns out that it is always possible to remove the need for independent RNSs and the constant-induced error by transferring the role of all ancillary inputs to sequential elements. This was first shown possible in [20] via an ancillary-input-free adder constructed in ad hoc fashion; see Fig. 10. This design therefore accurately computes the scaled sum $Z = 0.5(X + Y)$, regardless of correlation between \mathbf{X} and \mathbf{Y} . More recently, a systematic method CEASE for removing ancillary inputs by role transfer was

Fig. 10 A sequential adder computing $Z = 0.5(X + Y)$ whose inputs are CI [20]



proposed that is applicable to all combinational stochastic circuits with ancillary inputs [31]. CEASE thus removes the correlation associated with ancillary inputs.

The CI property enables inputs to take SNs generated from a shared RNS without compromising the circuit's accuracy. This is useful in statistical analysis of stochastic circuits by greatly improving the simulation quality, and in SC designs by allowing CI inputs to accept SNs generated from a common RNS.

Circuits with Autocorrelation Features

Combinational stochastic circuits are inherently immune to autocorrelation, as the change of input ordering only changes the output ordering, but not the output value. On the other hand, sequential stochastic circuits operate in response to their input sequences, and thus are generally impacted by autocorrelation. The autocorrelation among input patterns at different time steps can steer the state transition in a biased way. Further, sequential components can also inject new autocorrelation into the SNs they process. Careless use of sequential components in SC systems can thus lead to significant autocorrelation-induced errors [9]. Managing autocorrelation in SC is an open research area, since many sequential SC designs can only operate in autocorrelation-free environments, including those synthesized using the “linear finite-state machine” architecture [21]. Some classes of sequential circuits with special autocorrelation properties are known and are discussed below.

Modulo-Counting Circuits The circuits generated by CEASE have a highly desirable property: they are immune to autocorrelation-induced errors. CEASE, as mentioned previously, is an algorithm to remove ancillary inputs of a combinational circuit by introducing sequential components [31]. The resulting sequential circuit is equivalent to a modulo counter with multiple moduli that keeps accumulating (weighted) input bits for later release, which occurs when the counter overflows. For instance, the ancillary-input-free adder in Fig. 10, while constructed in ad hoc fashion, is also effectively a modulo counter as can be seen from its state-transition graph in Fig. 11. When the adder receives the input pattern 11 or 00, it directly outputs a 1 or a 0, respectively. However, when the adder receives 10 or 01, it will directly output a 0, but it accumulates a value 0.5 into its memory by going from state S_0 to state S_1 . The next time the adder receives 10 or 01, it will overflow, output a 1, and return to state S_0 . One can easily see that this modulo counting

Fig. 11 State transition graph for the sequential adder in Fig. 9 [31]

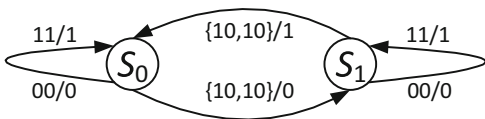
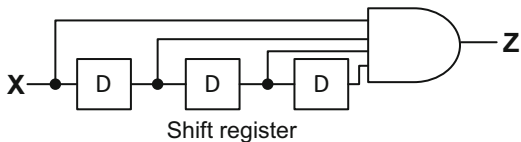


Fig. 12 Implementation of X^4 in canonical SRB form [30]



process guarantees that there will be a single 1 released into the output, whenever the adder receives two 1s from X or Y , thereby computing $0.5(X + Y)$. In general, CEASE-generated circuits not only avoid the potential correlation problems induced by ancillary inputs, but also are insensitive to autocorrelation. This is because the number of 1s in the output is completely determined by the number of times the modulo counter overflows, which is obviously independent of the ordering of the input pattern.

Shift-Register-Based Circuits In general, sequential stochastic circuits have strict correlation specifications on their inputs, which usually requires them to be autocorrelation-free. However, sequential stochastic circuits also inject autocorrelation into the SNs they process. This makes it difficult to cascade sequential designs, since autocorrelation introduced by an upstream circuit will degrade the accuracy of a downstream circuit. For example, sequential circuits employing the linear FSM architecture [21] require their inputs to be autocorrelation-free, but at the same time they produce output SNs with a high level of autocorrelation. It is therefore difficult to connect multiple linear circuits without sacrificing accuracy. Autocorrelation injected by a linear FSM has a diminishing but continuing effect over time. A current output bit can be correlated with all previous output bits, although the correlation level is lower with bits that are further away in time. This implies that when its input value changes, the output of a linear FSM may take a very long time to respond to the change, so the change can have an extended accuracy-reducing impact.

Thus, it is sometimes desirable to use alternative designs that have less severe autocorrelation problems. There is a class of sequential stochastic circuits called *shift-register-based (SRB)* which have a highly desirable property: their output autocorrelation is bounded in time [30]. SRB circuits realize a type of FSM termed a *definite machine* that has finite input memory [18]. They also have a canonical implementation consisting of a feed-forward shift register built around a combinational component. Many SC designs, including those generated by the STRAUSS synthesizer [4], belong to the SRB class. For example, Fig. 12 shows the canonical SRB implementation of X^4 , which has a 3-tap shift register that produces three delayed copies of the input SN X . SRB circuits have their output autocorrelation bounded in time, because each output bit is completely determined by the m most recent input bits, where $m - 1$ is the number of taps of the shift

register. Therefore, output bits that are separated by m clock cycles are determined by different and independent sets of input bits, and hence must be uncorrelated. In the X^4 example, $\mathbf{Z}(4) = \mathbf{X}(4)\mathbf{X}(3)\mathbf{X}(2)\mathbf{X}(1)$, while $\mathbf{Z}(8) = \mathbf{X}(8)\mathbf{X}(7)\mathbf{X}(6)\mathbf{X}(5)$, so $\mathbf{Z}(8)$ and $\mathbf{Z}(4)$, which are four clock cycles apart, are uncorrelated. The definiteness of the SRB circuits guarantees that m clock cycles after an input value change, the output value will have fully responded to the change. Furthermore, it is possible to sample the output SN every m cycles to get a completely autocorrelation-free bit-stream, which facilitates the use of such SNs as inputs to circuits that must avoid autocorrelation.

Design and Optimization of RNSs

Random number sources provide the randomness to drive the dynamics of the stochastic signals. RNSs with insufficient randomness can result in significant accuracy loss for stochastic circuits that require independent inputs. This can occur, for example, if a shared RNS is used to drive multiple SNGs for SN generation. The quality of RNSs also plays an important role in the accuracy of SC. It has been shown that, instead of using an RNS that has good randomness property like an LFSR, using carefully designed deterministic number sequences can sometimes result in significantly improved accuracy. For specialized circuits that work with correlated inputs, SNs with any SCC level can be generated by interpolating independent SNs and maximally correlated SNs with multiple independent RNSs. The CI property also is important in SNG design, as it allows a single RNS to be shared by multiple SNGs without compromising accuracy.

Generating SNs with Predetermined Correlation

In most existing stochastic circuits, it is desirable to have SNGs that can generate high quality uncorrelated SNs, i.e., SNs that have $\text{SCC} = 0$. In SNG design, arguably the most common RNSs are obtained by tapping an LFSR of maximum period, and are essentially pseudo-random. However, it has been shown that deterministic number sources such as plain binary counters can also be used as in SN generation without compromising accuracy [17]. In fact, circuits that use such deterministic number sources are usually more accurate than the ones using LFSRs, because random fluctuation errors are eliminated and correlation control is easier.

To achieve fast convergence rates during a stochastic computation, researchers have also looked into using quasi-Monte Carlo methods and low-discrepancy sequences [3, 22]. While these methods provide good convergence when generating a few uncorrelated SNs, they are affected by the curse of dimensionality and are no better than counter-based SNGs. In many cases, the convergence rate of the SNs is not relevant, and only the accuracy at the end of computation matters. In such cases,

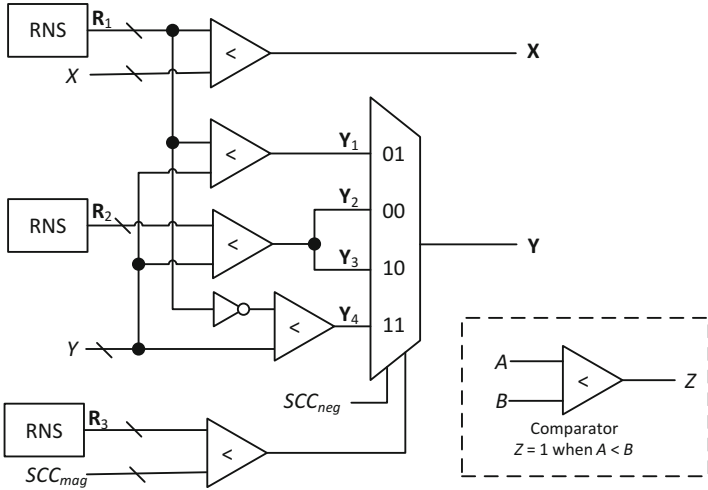


Fig. 13 SNG that generates a pair of SNs with a user-specified SCC level [2]

low-discrepancy sequences have no significant benefit over other deterministic number sources. It is worth noting that low-discrepancy number sources (and similar deterministic number sources) usually create autocorrelation within SNs, which may not be desirable.

As noted earlier, while most stochastic circuits require independent inputs, circuits exist that need a specific SCC level at their inputs. Figure 13 shows an SNG design [2] that generates a pair of SNs with a predetermined SCC level using three independent RNSs. This SNG is essentially a direct reflection of Eq. (3). Depending on the sign of SCC, the SNG selects a maximally correlated \mathbf{XY} pair and linearly combines it with the independent \mathbf{XY} pair, weighted by the magnitude of SCC. For example, to generate the bit-streams with $SCC(\mathbf{X}, \mathbf{Y}) = 0.5$, the inputs SCC_{neg} and SCC_{mag} of Fig. 13 are set to 0.0 and 0.5, respectively, so that the selection signal of the multiplexer will have 0.5 probability of choosing 01 and another 0.5 probability of choosing 00. The SN \mathbf{Y}_1 that is maximally correlated with \mathbf{X} , is applied to data input 01 of the multiplexer, while \mathbf{Y}_2 , which is completely independent with \mathbf{X} is applied to multiplexer input 00. Combining \mathbf{Y}_1 and \mathbf{Y}_2 with a 0.5 probability of outputting each result in an SN \mathbf{Y} that has correlation $SCC = 0.5$ with \mathbf{X} .

Optimizing Random Number Sources

RNSs provide the randomness required by stochastic circuits, and are a key design resource. While there are stochastic systems that use non-LFSR-based RNSs, LFSRs remain the popular choice due to their compatibility with digital logic and their relatively small hardware area, on top of the fact that they have been

intensively studied for many years. The design considerations around deploying RNSs include: providing adequate randomness, minimizing hardware overhead, and reducing unnecessary use of the RNSs.

As discussed earlier, an SN \mathbf{X} can be derived from its binary counterpart B by an SNG containing an RNS and a comparator that compares the B with the random number \mathbf{R} from the RNS at each clock cycle. The SNG outputs a 1 whenever $B > R$; otherwise the SNG outputs a 0. A common approach is to treat k taps from the LFSR as the k -bit random number \mathbf{R} . Sharing exactly the same random number \mathbf{R} with other SNGs can reduce overall hardware cost, but will result in a maximally correlated SN which is usually undesirable. Previous work attempts to squeeze out more randomness from a single LFSR by adding a re-wiring layer that shuffles the order of the bits in \mathbf{R} . In [15], the authors show that circularly shifting \mathbf{R} is a low-cost and effective way to reduce the SCC of two SNs sharing the same LFSR. Specifically, they experimentally demonstrate that by circularly shifting a k -bit random number by approximately $k/2$ bits, the SCC level can be reduced by around 75%, compared to random shuffling which achieves only 40% reduction in SCC on average. They further show that taking advantage of the CI property can reduce the need for RNSs.

Conclusions

Correlation is a pervasive phenomenon in stochastic circuits and a major source of computational inaccuracy and high hardware cost. Although it has been intensively studied in recent years, it still remains quite poorly understood. In general, correlation is difficult to quantify and expensive to mitigate. The *SCC* metric has proven useful as a measure of the cross correlation between two bit-streams but it is not readily extended to more than two bit-streams. No comparable measure of autocorrelation has emerged, even though autocorrelation has a significant impact on the accuracy of sequential stochastic circuits. The main methods of reducing or eliminating undesired correlation are regeneration and isolation, with isolation being by far the less expensive of the two.

Correlation affecting the inputs of a circuit with output function f can be viewed in two contrasting ways: as corrupting f 's output by introducing errors, or as changing f to a different function which may be useful. While a few examples of such "good" correlation-based designs are known, no general and practical way of taking advantage of correlation in SC design is known at present. In some cases, interacting bit-streams are unaffected by correlation, but again there is no easy way to identify and exploit correlation insensitivity. Among the more promising recent discoveries in this direction are several classes of highly accurate sequential machines that are insensitive to correlation on their inputs, and the development of general-purpose correlation-controlling units.

Abbreviations

CI	Correlation insensitive
FSM	Finite-state machine
LDPC	Low density parity check code
LFSR	Linear feedback shift register
MSE	Mean square error
PTM	Probabilistic transfer matrix
RNS	Random number source
SC	Stochastic computing
SCC	Stochastic correlation coefficient
SCH	Single-ended counter hysteresis
SN	Stochastic number
SNG	Stochastic number generator
SRB	Shift register based

References

1. A. Alaghi and J.P. Hayes, "A Spectral Transform Approach to Stochastic Circuits," *Proc. Intl. Conf. Computer Design (ICCD)*, pp. 315–312, 2012.
2. A. Alaghi and J.P. Hayes, "Exploiting Correlation in Stochastic Circuit Design," *Proc. Intl. Conf. on Computer Design (ICCD)*, pp. 39–46, Oct. 2013.
3. A. Alaghi and J.P. Hayes, "Fast and Accurate Computation Using Stochastic Circuits," *Proc. Design, Automation, and Test in Europe Conf. (DATE)*, pp. 1–4, 2014.
4. A. Alaghi and J.P. Hayes, "STRAUSS: Spectral Transform Use in Stochastic Circuit Synthesis," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 34, pp. 1770–1783, 2015.
5. A. Alaghi and J.P. Hayes, "Dimension Reduction in Statistical Simulation of Digital Circuits," *Proc. Symp. on Theory of Modeling & Simulation (TMS-DEVS)*, pp. 1–8, 2015.
6. A. Alaghi, C. Li and J.P. Hayes, "Stochastic Circuits for Real-time Image-Processing Applications," *Proc. Design Autom. Conf. (DAC)*, article 136, 6p, 2013.
7. A. Alaghi, W.-K. Qian and J.P. Hayes, "The Promise and Challenge of Stochastic Computing," *IEEE Trans. CAD*, vol. 37, pp.1515–1531, Aug. 2018.
8. R. Baumann, "Soft Errors in Advanced Computer Systems," *IEEE Design & Test of Computers*, vol.22, pp. 258–266, 2005.
9. B.D. Brown and H.C. Card, "Stochastic Neural Computation I: Computational Elements," *IEEE Trans. Comp.*, vol. 50, pp. 891–905, 2001.
10. T.-H. Chen and J.P. Hayes, "Design of Division Circuits for Stochastic Computing," *Proc. IEEE Symp. on VLSI (ISVLSI)*, pp. 116–121, 2016.
11. S.S. Choi, S.H. Cha and C. Tappert, "A Survey of Binary Similarity and Distance Measures," *Jour. Systemics, Cybernetics and Informatics*, vol. 8, pp. 43–48, 2010.
12. J. Friedman et al. "Approximation Enhancement for Stochastic Bayesian Inference," *Elsevier Int. Jour. of Approximate Reasoning*, 85, pp.139–158, 2017.
13. B.R. Gaines, "Stochastic Computing Systems," *Advances in Information Systems Science*, vol. 2, J.T. Tou (ed.), Springer, pp. 37–172, 1969.
14. S.W. Golomb, *Shift Register Sequences*. Revised ed., Aegean Park Press, Laguna Hills, CA, 1982.

15. H. Ichihara et al., "Compact and Accurate Digital Filters Based on Stochastic Computing," *IEEE Trans. Emerging Topics in Comp.*, 2018 (early access).
16. P. Jeavons, D.A. Cohen and J. Shawe-Taylor., "Generating Binary Sequences for Stochastic Computing," *IEEE Trans. Info. Theory*, vol. 40, pp. 716–720, 1994.
17. D. Jenson and M. Riedel, "A Deterministic Approach to Stochastic Computation," *Proc. Intl. Conf. Computer-Aided Design (ICCAD)*, pp. 1–8, 2016.
18. Z. Kohavi and N.K. Jha, *Switching and Finite Automata Theory*, 3rd ed. Cambridge Univ. Press, 2010.
19. V.T. Lee, A. Alaghi and L. Ceze, "Correlation Manipulating Circuits for Stochastic Computing," *Proc. 2018 Design, Automation & Test in Europe (DATE) Conf.*, pp. 1417–1422, 2018.
20. V.T. Lee et al., "Energy-Efficient Hybrid Stochastic-Binary Neural Networks for Near-Sensor Computing," *Proc. Design, Automation and Test in Europe Conf. (DATE)*, pp. 13–18, 2017.
21. P. Li et al., "Logical Computation on Stochastic Bit Streams with Linear Finite-State Machines," *IEEE Trans. Computers*, vol. 63, pp. 1474–1486, 2014.
22. S. Liu and J. Han, "Energy Efficient Stochastic Computing with Sobol Sequences," *Proc. Design, Automation, and Test in Europe Conf. (DATE)*, pp. 650–653, 2017.
23. Y. Liu et al., "Synthesis of Correlated Bit Streams for Stochastic Computing," *Proc. Asilomar Conf. on Signals, Systems and Computers*, pp. 167–174, 2016.
24. A. Naderi et al., "Delayed Stochastic Decoding of LDPC Codes," *IEEE Trans. Signal Processing*, vol. 59, pp. 5617–5626, 2011.
25. W. Qian et al., "An Architecture for Fault-Tolerant Computation with Stochastic Logic," *IEEE Trans. Comp.*, vol. 60, pp. 93–105, 2011.
26. W. Qian and M. D. Riedel, "The Synthesis of Robust Polynomial Arithmetic with Stochastic Logic," *Proc. Design Autom. Conf. (DAC)*, pp. 648–653, 2008.
27. S.S. Tehrani, W. J. Gross, and S. Mannor, "Stochastic Decoding of LDPC Codes," *IEEE Comm. Letters*, vol. 10, pp. 716–718, 2006.
28. S. Tehrani et al., "Relaxation Dynamics in Stochastic Iterative Decoders," *IEEE Trans. Signal Processing*, vol. 58, pp. 5955–5961, 2010.
29. P.S. Ting and J.P. Hayes, "Isolation-Based Decorrelation of Stochastic Circuits," *Proc. Intl. Conf. Computer Design (ICCD)*, pp. 88–95, 2016.
30. P.S. Ting and J.P. Hayes, "On the Role of Sequential Circuits in Stochastic Computing," *Proc. Great Lakes VLSI Symp. (GLSVLSI)*, pp. 475–478, 2017.
31. P. S. Ting and J.P. Hayes, "Eliminating a Hidden Error Source in Stochastic Circuits," *Proc. Symp. Defect & Fault Tolerance in VLSI and Nano. Systems (DFT)*, pp.44–49, Oct. 2017.