# Agile Simulation of Stochastic Computing Image Processing With Contingency Tables

Sercan Aygun, *Member, IEEE*, M. Hassan Najafi, *Member, IEEE*, Mohsen Imani, *Member, IEEE*, and Ece Olcay Gunes

***Abstract*—The rapid computerized simulation of stochastic computing (SC) systems is a challenging problem. A method for agile simulation of SC image processing is proposed in this work. The input operands are processed with the aid of a correlation-controlled contingency table (CT) construct without using actual stochastic bit-streams. The proposed approach underlines the validity of CT simulation with 1) image compositing; 2) pattern detection; and 3) bilinear interpolation case studies. Using the corresponding error models, we emulate the state-of-the-art pseudo-random and quasi-random bit-streams. Experimental results show that the proposed approach achieves similar computation accuracy to the traditional SC simulation while performing runtime- and memory-efficient computations. The execution time reduces more than 200× for the image compositing task when emulating random bit-streams with CT. Pattern detection and bilinear interpolation further showed 76× and 22× lower memory usage, respectively, when employing CT.**

***Index Terms*—Computer-aided simulation, contingency table (CT), image processing, stochastic computing (SC).**

## I. Introduction

The simulation of stochastic computing (SC) systems [1] faces time and memory complexity challenges due to conducting very long bit-by-bit processing. Especially, simulating SC-based processing of high-density data like image matrices in row–column format is very time consuming. In SC, the precision of data and the quality of the results are affected by the length of stochastic bit-streams [1]. The larger the bit-stream size ($N$), the higher the accuracy of the computations [2]. Often very long bit-streams, in orders of $10^3$ to $10^4$ bits, must be processed for high-quality results. To have an output bit-stream with a resolution of 8 bits, bit-streams of at least $2^{16}$ bits need to be processed [3].

SC has been popular for the simple execution of complex image processing tasks [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. Considering the large number of image data that need to be processed, fast simulation of SC-based systems is challenging even with short bit-stream sizes of 100 bits. Sometimes large design space explorations are done by varying the bit-stream size from hundreds to tens of thousands bits to study the performance of an SC system.

Manuscript received 12 July 2022; revised 15 October 2022 and 22 December 2022; accepted 22 January 2023. Date of publication 7 February 2023; date of current version 20 September 2023. This work was supported in part by the National Science Foundation (NSF) under Grant 2127780 and Grant 2019511; by the SRC Global Research Collaboration, AIHW and HW Security; by the Department of the Navy, Office of Naval Research under Grant N00014-21-1-2225 and Grant N00014-22-1-2067; by the Air Force Office of Scientific Research under Grant 22RT0060; and by the Louisiana Board of Regents Support Fund under Grant LEQSF(2020-23)-RD-A-26, and generous gifts from Cisco, Xilinx, and Nvidia. This article was recommended by Associate Editor L. Amaru. *(Corresponding author: Sercan Aygun.)*

Sercan Aygun is with the ECE Department, Istanbul Technical University, 34469 Istanbul, Turkey, and also with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70503 USA (e-mail: ayguns@itu.edu.tr).

M. Hassan Najafi is with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70503 USA (e-mail: najafi@louisiana.edu).

Mohsen Imani is with the Department of Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: m.imani@uci.edu).

Ece Olcay Gunes is with the ECE Department, Istanbul Technical University, 34469 Istanbul, Turkey (e-mail: gunesec@itu.edu.tr).

Digital Object Identifier 10.1109/TCAD.2023.3243136

Instead of explicitly generating and processing stochastic bit-streams, this work performs SC in a radically different way. Simple arithmetic operations on scalar values replace traditional bit-wise operations on stochastic bit-streams. The input operands are processed with the aid of a contingency table (CT) construct [4] without explicitly processing bit-streams. This allows latency-free and memory-aware emulation of SC systems without impacting the results. We show that similar accuracy to traditional stochastic bit-stream processing is achieved with this approach. CT's usage has not been explored before at the application level. This work studies the effect of CT in the simulation of SC image processing case studies. We utilize some simple SC circuits (2-to-1 MUX, XOR, and 4-to-1 MUX) for *image compositing*, *pattern detection*, and *bilinear interpolation* tasks for the first time. In summary, the main contributions of this work are as follows.

1) Developing simple SC designs for three image processing tasks: 2-to-1 MUX (image compositing), XOR (pattern detection), and 4-to-1 MUX (bilinear interpolation).
2) Evaluating CT-based SC simulation for image processing case studies.
3) Emulating state-of-the-art Sobol-based low-discrepancy (LD), binomial distributed, and linear-feedback shift register-based (LFSR) bit-streams with CT.
4) Runtime comparison of SC bit-stream processing and CT-based simulation on CPU and GPU.

## II. Stochastic Computing and Contingency Tables

SC is a re-emerging paradigm for the cost-efficient and fault-tolerant design of digital systems. Data values are represented with bit-streams with equal bit significance. Arithmetic operations are implemented by performing simple bit-wise operations on the bit-streams. For example, multiplication is realized by bit-wise AND (XNOR) on unipolar (bipolar) bit-streams [6]. To accurately multiply two $n$-bit precision data, bit-streams of at least $2^{2n}$ bits must be processed [3]. Generating and processing such long bit-streams is very time and memory consuming, especially when $n$ increases.

The power of SC stems from the probabilistic behavior of random bit-streams in which 1s and 0s occur randomly in no specific order. The state-of-the-art distributions for stochastic bit-streams are binomial distribution, Sobol-based LD, and LFSR-based pseudo-random. During bit-stream generation, a *stochastic bit-stream generator* is coupled with a *comparator* that compares a randomly generated binary number with the to-be-encoded input scalar value. If the scalar value is greater than the random number, the corresponding bit of the bit-stream is 1; otherwise, it is 0. LFSR-based randomization is the most popular bit-stream generation method in the literature [15]. LFSR provides pseudo-randomness with binary numbers that are generated circularly. Some recent works use Sobol sequences to generate quasi-random numbers. Sobol sequences are used to generate LD bit-streams for highly accurate SC arithmetic [16], [17]. The third approach uses binomial distribution and has been used in the literature in quantifying the random fluctuation error of SC operations. The probability $P$, represented by a stochastic bit-stream, can be considered based on a set of samples from a random variable (RV) having a Bernoulli distribution with a success probability of $P$ [18]. Bernoulli distribution is employed to produce bit-streams with uniformly distributed bits. This distribution is obtained by using $N$ different Bernoulli trials.

As a bit-stream emulation construct, CT, is a $2 \times 2$ table of four scalar values. As if $i$-bit bit-streams are generated, the binary interaction

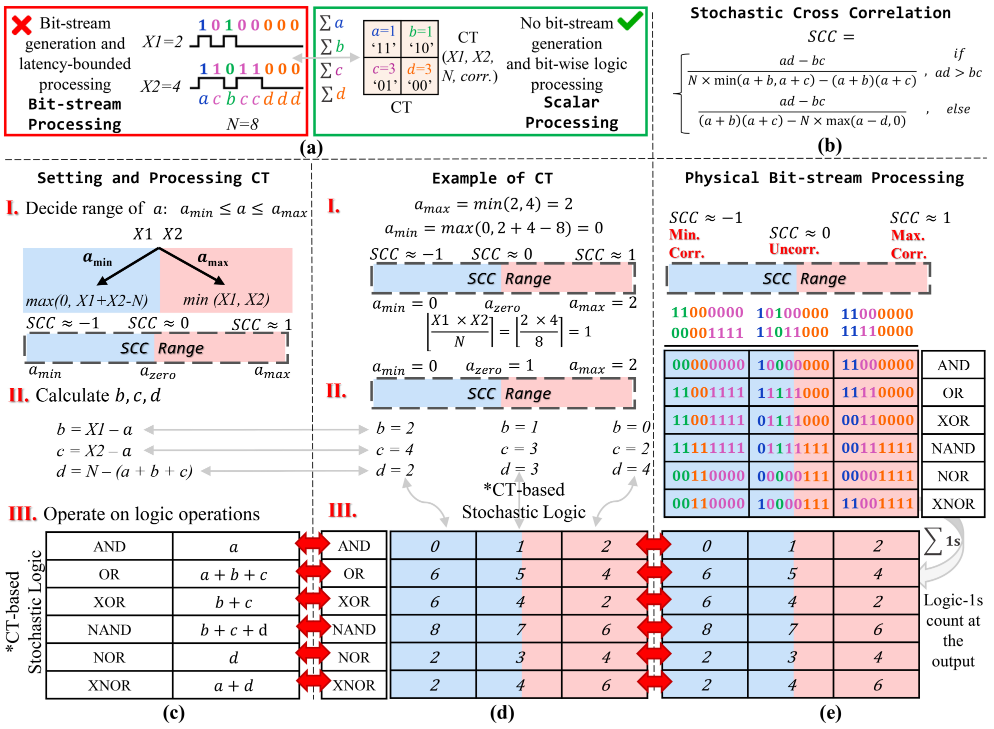

Bit-stream generation and latency-bounded processing
Bit-stream Processing
X1=2 1 0 1 0 0 0 0 0
X2=4 1 1 0 1 1 0 0 0
a c b c c d d d
N=8
Σa Σb Σc Σd
a=1 '11' b=1 '10' c=3 '01' d=3 '00'
CT (X1, X2, N, corr.)
No bit-stream generation and bit-wise logic processing
Scalar Processing
(a)
Stochastic Cross Correlation
SCC = (ad − bc) / (N × min(a+b, a+c) − (a+b)(a+c)), if ad > bc
(ad − bc) / ((a+b)(a+c) − N × max(a−d, 0)), else
(b)
Setting and Processing CT
I. Decide range of a: a_min ≤ a ≤ a_max
a_min max(0, X1+X2−N)
a_max min (X1, X2)
SCC ≈ −1 SCC ≈ 0 SCC ≈ 1
SCC Range
a_min a_zero a_max
II. Calculate b, c, d
b = X1 − a
c = X2 − a
d = N − (a + b + c)
III. Operate on logic operations
AND a
OR a + b + c
XOR b + c
NAND b + c + d
NOR d
XNOR a + d
*CT-based Stochastic Logic
(c)
Example of CT
a_max = min(2,4) = 2
a_min = max(0, 2 + 4 − 8) = 0
a_zero = ⌊X1 × X2 / N⌉ = ⌊2 × 4 / 8⌉ = 1
b = 2 c = 4 d = 2
b = 1 c = 3 d = 3
b = 0 c = 2 d = 4
*CT-based Stochastic Logic
AND 0 1 2
OR 6 5 4
XOR 6 4 2
NAND 8 7 6
NOR 2 3 4
XNOR 2 4 6
(d)
Physical Bit-stream Processing
SCC ≈ −1 Min. Corr.
SCC ≈ 0 Uncorr.
SCC ≈ 1 Max. Corr.
11000000 10100000 11000000
00001111 11011000 11110000
AND 00000000 10000000 11000000
OR 11001111 11111000 11110000
XOR 11001111 01111000 00110000
NAND 11111111 01111111 00111111
NOR 00110000 00000111 00001111
XNOR 00110000 10000111 11001111
Σ 1s
0 1 2
6 5 4
6 4 2
8 7 6
2 3 4
2 4 6
Logic-1s count at the output
(e)

Fig. 1. Summary of CT and bit-by-bit simulation, including the theory behind CT and examples. (a) From bit-streams to scalar-only processing. (b) SCC formula for quantifying correlation. (c) Steps for setting up CT, and the relation between CT primitives ($a$, $b$, $c$, $d$) and logic operations. (d) Numerical example with $X1 = 2$ and $X2 = 4$ from bit-streams to scalar-only processing. (e) Same example with actual bit-streams.

between input operands (i.e., bit-streams) are recorded for 11, 10, 01, and 00 logic pair overlaps in any $i$ position of the bit-streams. Fig. 1(a) shows an example of generating a CT. Logic pairs at any bit position $i$ are cumulatively denoted using $a$, $b$, $c$, and $d$. These are called *CT primitives*. Logic operations are defined using these primitives. For example, AND operation corresponds to 11 (i.e., $a$) bit pairs between the two bit-streams, $\boldsymbol{X1}$ and $\boldsymbol{X2}$. Without explicitly generating bit-streams, the CT is directly generated from the scalar (binary equivalent) values of $X1$ and $X2$, where $0 \leq X1, X2 \leq N$, and $N$ is the bit-stream length. Now the question is how to find $a$, $b$, $c$, and $d$ values directly from scalar values. The answer depends on "correlation." For correct functionality, some SC designs need input bit-streams with maximum correlation. Some others need inputs with minimum correlation, and some need uncorrelated inputs [1].

To emulate maximally correlated bit-streams, $a$ is set maximally, while for minimum correlation, $a$ is minimum. $a$ is the first CT primitive we find. The formulas for the maximally ($a_{\max}$) and minimally ($a_{\min}$) "11" occurrences are presented in Fig. 1(c)-I. After finding $a$, by using the scalar values of $X1$, $X2$, and $N$, the other primitives ($b$, $c$, $d$) are found. The formulas are shown in Fig. 1(c)-II. Finally, logic operations are converted to basic arithmetic on the primitives. Fig. 1(c)-III shows how different logic operations can be obtained from CT primitives.

Besides the cases with maximum and minimum correlation, emulating *uncorrelated* or independent bit-streams is important for modeling SC systems. Stochastic cross correlation (SCC) [Fig. 1(b)] [19] with range of $[-1, 1]$ is suggested as a metric to quantify the correlation level between two bit-streams. Two bit-streams are uncorrelated if SCC $\approx 0$. When the SCC formula is solved for SCC $= 0$, the following equation is obtained to set up CT for zero correlation: $a_{\text{zero}} = \lfloor([X1 \times X2]/N)\rceil$. Fig. 1(d) provides an example of setting up CT for $X1 = 2$ and $X2 = 4$. Steps I–III depict the CT formation for three correlation points: 1) minimum; 2) zero; and 3) maximum. The numeric table in step III shows the number of 1s in the output bit-stream resulting from different logic operations. Fig. 1(e) shows example bit-streams for $X1$ and $X2$ scalars for different correlation points and the corresponding logic results using traditional bit-by-bit processing. As it can be seen, the logic results shown in Fig. 1(d) and (e) are the same, proving the correctness of the CT method.

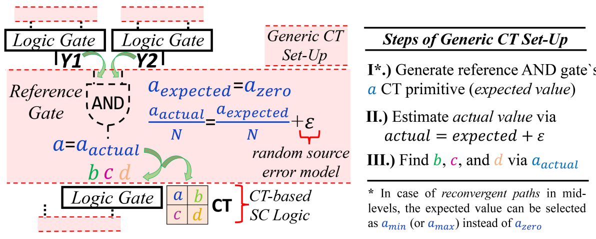

Logic Gate
Logic Gate
Y1
Y2
Generic CT Set-Up
Reference Gate
AND
a_expected = a_zero
a_actual / N = a_expected / N + ε
a = a_actual
random source error model
b c d
Logic Gate
a b c d
CT
CT-based SC Logic
Steps of Generic CT Set-Up
I*.) Generate reference AND gate's a CT primitive (expected value)
II.) Estimate actual value via actual = expected + ε
III.) Find b, c, and d via a_actual
* In case of reconvergent paths in mid-levels, the expected value can be selected as a_min (or a_max) instead of a_zero

Fig. 2. Generic CT set-up.

## III. Proposed Approach

### A. Generic CT Set-Up

We first define a generic CT set-up for simulating multilevel cascaded SC circuits. Fig. 2 depicts the encapsulated CT setting followed from the circuits' inputs to the output. Considering $Y1$ and $Y2$ connected to any gate type at the mid-level, the AND reference gate is temporarily evaluated using near-zero correlation, $a_{\text{zero}}$ (step I). The deviation from the *expected* value (i.e., error-free multiplication value) is determined by random fluctuation error, $\epsilon$, considering the model of random sources that generate input bit-streams (step II). The *actual* value is estimated through $\epsilon$. Finally, other CT primitives are calculated based on actual $a$ (step III). For more complex circuits with reconvergent paths (especially when processing longer bit-streams, e.g., 512 and 1024), the designer may benefit from the reconvergence involving correlation. At the output of the reconvergent paths, signal correlation can be used to define the expected value via $a_{\min}$ or $a_{\max}$. We observe that reconvergence awareness in CT simulations may be beneficial to reduce error especially for complex circuits (levels $> 3$) when processing long bit-streams (please see the GitHub repository [20] for some examples.) Nevertheless, we recommend the expected value assignment in Fig. 2 for the image processing circuits (those having $\epsilon \neq 0$) in Section III-B.

### B. SC Image Processing With CT-Based Random Source Emulation

This work extends the SC-based image processing techniques with three new case studies: 1) image compositing; 2) pattern detection;

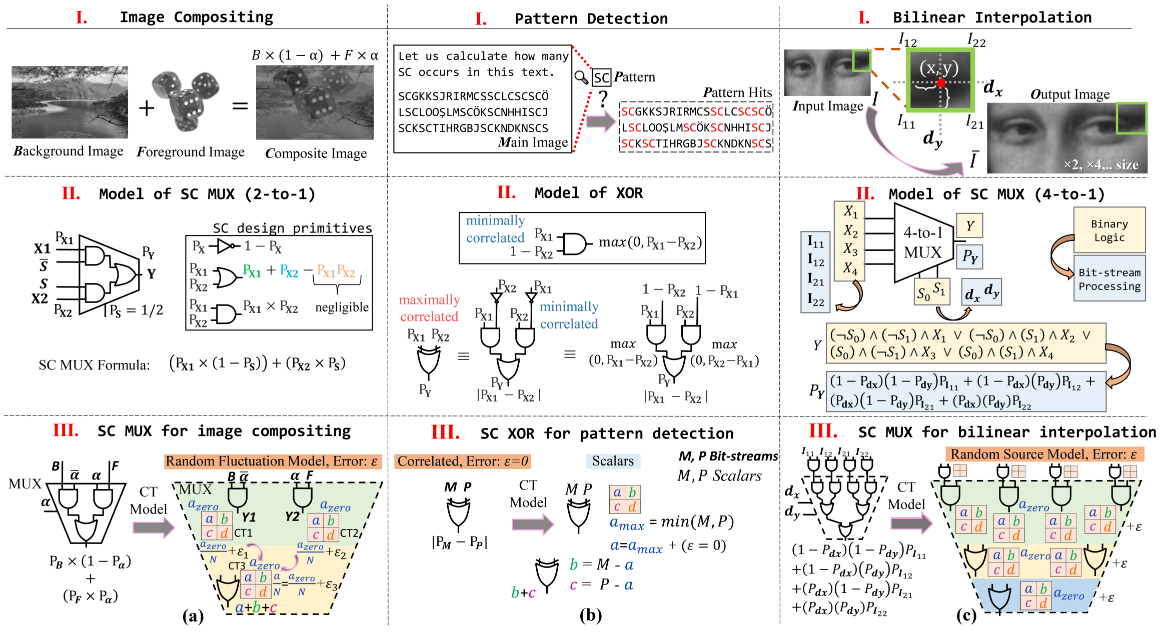


Fig. 3. Proposed CT-based (a) image compositing, (b) pattern detection, and (c) bilinear interpolation.

and 3) bilinear interpolation. The first case study is image compositing. Szeliski [21] elaborated image compositing by giving the compositing ($C$) formula $C = B(1-\alpha) + F\alpha$, where $B$ is the original background image, $\alpha$ is the foreground image opacity, and $F$ is the foreground image. Fig. 3(a)-I shows an example of image compositing. The image compositing formula reminds the equation of an SC scaled adder implemented using a multiplexer (MUX): $\text{MUX} = P_{X1}(1-P_S) + P_{X2}P_S$, where $P_S$ is the probability of the select input [6]. The SC MUX formula is elaborated in Fig. 3(a)-II. As the image compositing and MUX formulas coincide, the image compositing can be realized by simply using a MUX unit. The main inputs are the background and foreground image pixel bit-streams, and the select input is the foreground image opacity, $\alpha$. The CT model for a MUX (built from two AND and one OR gates) is obtained using multiple tables as depicted in Fig. 3(a)-III. Using the scalar values of $B$, $F$, and $\alpha$, first, the near-zero-correlation CTs are created. Since the AND operation result is the same as the "$a$" primitive, CT1 is set by considering near-zero $a$ ($a_{\text{zero}}$): $\text{CT1}_{a_{\text{zero}}} = \lfloor([B \times (N-\alpha)]/N)\rceil$. Likewise, CT2 is set by $\text{CT2}_{a_{\text{zero}}} = \lfloor([F \times \alpha]/N)\rceil$. After calculating the zero-correlated $a$ value, the random fluctuation error ($\epsilon$) from the input bit-stream generators should also be taken into account by carrying out the steps of the generic CT set-up shown in Fig. 2. In this study, we emulate three different bit-stream generation models: 1) binomial distribution; 2) Sobol LD; and 3) LFSR.

For the binomial distribution case, a stochastic bit-stream having probability $P$ is considered as a set of samples from RV with a Bernoulli distribution having a success probability, $P$. Considering the *independent and identically distributed* RV [22], a stochastic number is represented yielding binomial distribution with a variance $\sigma^2 = P(1-P)/N$. As a reference, the AND operation (corresponding to CT primitive "$a$") has an expected output probability $P_Y = \mathbb{E}[Y]$, yielding $P_{X1} \times P_{X2}$ for independent variables. Nevertheless, the obtained probability, $\hat{P}_Y$, may differ due to *random fluctuations*. Using squared error, the random fluctuations error for the case of Bernoulli RVs is obtained as $\text{err}_Y = \mathbb{E}[(P_Y - \hat{P}_Y)^2] = P_Y(1-P_Y)/N$ [1], [23]. From the squared error to the square-rooted format, the error is $\epsilon = \sqrt{\text{err}_Y}$. Considering AND as the reference and initial operation, we include $\pm\epsilon$ in the obtained CT output probability. Fig. 3(a)-III has AND gates in the first level of MUX. After obtaining $a_{\text{zero}}$, the obtained output probabilities of CT1 and CT2, $(a/N)$, are determined via $(a_{\text{zero}}/N) \pm \epsilon$. (Our error calculations show that $(a/N) + \epsilon$ has better error performance than $(a/N) - \epsilon$; therefore, we use $+\epsilon$ for the model.) For the LFSR case, we model error by assuming hypergeometric distribution as recommended by Baker and Hayes [24]. They define the output deviation of performing bit-wise AND operation on LFSR-based bit-streams as $\epsilon = \sqrt{([P_{X1} \times P_{X2} \times (1-P_{X1}) \times (1-P_{X2})]/[N-1])}$, where $X1$ and $X2$ are the inputs. Finally, unlike the binomial and LFSR cases, the Sobol-based LD model almost equals the near-zero correlation CT, i.e., $a_{\text{zero}}$, where $\epsilon = 0$.

In Fig. 3(a)-III, after including the relevant error in CT1 and CT2, a new CT is set for the next logic operation, which is an OR operation. Similar to the first level of the circuit, the generic CT set-up is followed for this level of logic. The expected output probability and the input probabilities are important for calculating $\epsilon$ in the binomial and LFSR-based random source error, respectively. The AND gate model as a reference is initially calculated for the random source error; thereby, the CT prior primitive $a$ is first fine-tuned. Returning to Fig. 3(a)-III, for modeling MUX, CT3's near-zero correlated output probability, $(a_{\text{zero}}/N)$, is updated with $\epsilon$. After updating the CT prior primitive, the other primitives ($b$, $c$, $d$) are calculated for the final logic operation (e.g., OR gate: $a + b + c$, recall Fig. 1).

The second case study is pattern detection, shown in Fig. 3(b)-I. This task is based on *maximum correlation*. The two inputs from the main image and the pattern image are maximally correlated to utilize XOR gate as an SC subtractor. Fig. 3(b)-II elaborates on the formula and functionality of the XOR gate as an SC primitive. As shown, when input bit-streams are positively correlated (i.e., have maximum overlap between the position of 1's), bit-wise XOR acts as an absolute subtractor, $|P_{X1} - P_{X2}|$ [25]. This can be used for pattern search in an image, where $\boldsymbol{X1}$ relates to the main image ($M$), and $\boldsymbol{X2}$ holds the pattern ($P$) to be searched in $X1$. The difference gives zero value for the exact pattern match. The CT for the pattern detection task is elaborated in Fig. 3(b)-III. The model of positively correlated input operands is established in CT using $a_{\text{max}}$. For this case study, the generic CT setting assigns $a_{\text{max}}$ to the expected value, which is also equal to the actual value since $\epsilon = 0$. First $a_{\text{max}}$ is found using $\min(X1, X2)$ formula, and then the $b$ and $c$ primitives are calculated. The XOR result is obtained from CT via $b + c$, without explicit bit-stream computing. Thus, the difference between $M$ and $P$ images is obtained. This operation can be considered a convolution-like operation; $P$ is shifted as a sliding window over $M$.

The third case study is bilinear interpolation used for image resizing. This task is based on linear interpolations in both x (width) and y (height) directions. By repeating linear interpolations for x and y, bilinear interpolation is performed. Assume an image $I$ defined by a rectangular region with four points: $(x_1, y_1)$, $(x_1, y_2)$, $(x_2, y_1)$, and $(x_2, y_2)$. A new pixel point in this region is denoted as $(x, y)$, and is to

TABLE I
COMPARING BIT-WISE BIT-STREAM PROCESSING AND CT APPROACH IN DIFFERENT COMPUTING PLATFORMS*: CPU AND GPU

| **Conventional Bit-stream Processing** | | | | | | | | | **Contingency Tables** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $2^3$ | $2^4$ | $2^5$ | $2^6$ | $2^7$ | $2^8$ | $2^9$ | $2^{10}$ | **N** | $2^3$ | $2^4$ | $2^5$ | $2^6$ | $2^7$ | $2^8$ | $2^9$ | $2^{10}$ |
| *NO DATA DEPENDENCY* | | | | | | | | | *NO DATA DEPENDENCY* | | | | | | | |
| 72.63 | 151.01 | 316.65 | 636.81 | 1287.5 | 2539.4 | 5437.1 | 11472 | **CPU** | 1.522 | 1.480 | 1.504 | 1.494 | 1.499 | 1.661 | 1.518 | 1.560 |
| 1.854 | 2.744 | 3.842 | 6.055 | 9.871 | 17.701 | 33.397 | 61.108 | **GPU** | 0.430 | 0.433 | 0.470 | 0.445 | 0.438 | 0.442 | 0.432 | 0.428 |
| *DATA DEPENDENCY* | | | | | | | | | *DATA DEPENDENCY* | | | | | | | |
| 200.8 | 388.5 | 806.8 | 1637.8 | 3277.7 | 6621.1 | 13636 | 27960 | **CPU** | 2.792 | 2.673 | 2.626 | 2.891 | 2.885 | 2.777 | 2.823 | 2.811 |
| 4.008 | 5.348 | 8.028 | 13.438 | 22.877 | 41.099 | 76.869 | 147.31 | **GPU** | 0.740 | 0.633 | 0.644 | 0.648 | 0.638 | 0.640 | 0.637 | 0.641 |

*The runtime (in sec) analysis is performed on a system with Intel(R) Core i7-7700HQ CPU @2.80GHz, 16GB RAM, and with NVIDIA GTX 1070 GPU, using Python.

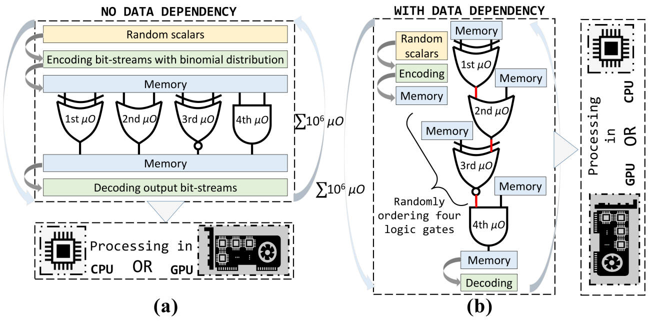
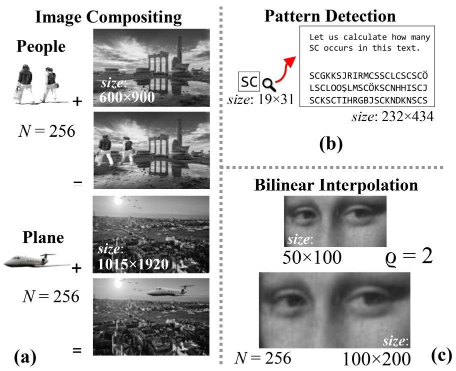


Fig. 4. Test procedure for simulating SC operations (a) with no data dependency and (b) with data dependency.

be estimated. Considering the unit square for the rectangular region, the expression for $I(\mathrm{x, y})$ is $(1-dx)(1-dy)I_{11}+(1-dx)(dy)I_{12}+(dx)(1-dy)I_{21}+(dx)(dy)I_{22}$ [26], where $I_{11}$, $I_{12}$, $I_{21}$, $I_{22}$ are neighboring pixel values, and $dx$ and $dy$ are new pixel's relative positions that define resizing coefficient, $\rho$. An example of bilinear interpolation is shown in Fig. 3(c)-I. We use a 4-to-1 MUX to realize this task in the stochastic domain. Fig. 3(c)-II shows the binary logic of a 4-to-1 MUX and the corresponding transformation to bit-stream processing using probabilities. $dx$ and $dy$ are connected to the select ports of the MUX. Fig. 3(c)-III depicts the CT model that follows the same procedures in Figs. 2 and 3(a)-III explained above.

## IV. Tests and Results

In this section, we evaluate the performance of the proposed technique compared to the conventional approach of simulating SC. Before evaluating the performance of the SC image processing designs, we conduct a preliminary analysis of computer-aided simulation of SC operations on CPU and GPU. We use Python for the tests and the `Cuda` library for GPU simulations. The operations are guaranteed to run on a single core for the CPU and in parallel for the GPU tests. For simulating the conventional bit-stream processing, the bit-streams are first generated and stored in memory and then used for logic operations. We consider two cases of "no data dependency" [Fig. 4(a)] and "with data dependency" [Fig. 4(b)] in the SC operations. For the "no data dependency" case, simulations of single and independent logic operations (i.e., 2-input `AND`, `OR`, `XOR`, and `XNOR`) are evaluated. In this case, the logic operation does not depend on the output of other operations. We denote the total number of logic operations in each test by micro-operation ($\mu O$). $N$ is selected from $\{2^3, 2^4, \ldots, 2^{10}\}$ during the tests. The cases "with data dependency" include cascaded logic operations. In each test, four-level cascaded logic operations are performed. Each operation waits for the result of another operation (except the first operation), thereby having data dependency. The processing times are reported in Table I for an average of 1000 independent tests. As it can be seen, the conventional bit-stream simulations are significantly slower than the CT-based simulations. In all cases, the GPU-based simulation is faster than CPU-based run. In particular, running on GPU reduces the simulation time of the conventional bit-stream processing up to 189× compared to running on CPU. We observe that the CT-based simulation is 142× faster than the conventional approach for simple logic operations with no data dependency, and delivers 229× better runtime for the case with data dependency when running on GPU.



Let us calculate how many SC occurs in this text.

SCGKKSJRIRMCSSCLCSCSCÖ
LSCLOOŞLMSCÖKSCNHHISCJ
SCKSCTIHRGBJSCKNDKNSCS

Fig. 5. Details of the image processing tasks under test and sample inputs. (a) Image compositing, (b) pattern detection, and (c) bilinear interpolation.

We also conducted an accuracy evaluation for the circuit structure shown in Fig. 4(b). We first fed this four-level circuit by binomially distributed bit-streams, and then constructed the corresponding CT-based model by considering $\epsilon$ at each level. We compared the output probabilities from the two models and measured the mean absolute error (MAE) for 1000 times simulating the circuit with different random input values. For $N = 1024$, the MAE between two simulation models was 0.024.

Next, we evaluate the runtime, memory usage, and accuracy of the three discussed image processing tasks, as shown in Fig. 5. Tests are developed in the MATLAB tool. For the conventional simulation method, the `binornd()` function is utilized for generating the binomially distributed bit-streams. The MATLAB built-in Sobol generator is used for generating LD sequences. For the LFSR-based approach, random numbers are generated using the maximal-period LFSRs described in [27].

First, different foreground images are embedded in different-sized background images for the image compositing design with $N = 256$ which can accurately represent 8-bit pixel values of grayscale images. We evaluate five different cases: 1) SC using binomial random bit-streams (SCRandom); 2) SC using state-of-the-art Sobol bit-streams (SCSobol) [16]; 3) CT-based SC with near-zero correlation (CT0); and 4) Conventional binary image processing (CONVN). We also evaluate a case where the random fluctuations error of random bit-streams is included in the CT method. Built-in fluctuations based on Bernoulli distribution are considered during $a$ primitive calculation. This is given as 5) CT-based SC with random fluctuations (CTRAND).

We note that the emulation of random fluctuations is fairly controlled as both SCRandom and CTRAND methods show similar PSNR (peak signal-to-noise ratio) values in the image compositing task. Besides, CT0 competently emulates Sobol-based bit-stream processing (SCSobol) as their PSNR values also match. Therefore, this study also provides a fast and efficient way to simulate LD Sobol-based SC [17]. As it can be seen in Table II, for both "People" and "Plane" test images of the image compositing task, the CT-based approaches (CT0 and CTRAND) are far better in runtime compared to SCSobol and SCRandom methods. Memory performance of the image compositing task is enhanced by ∼1048.58 Bytes per MUX-based operation compared to the conventional bit-stream processing. Since PSNR is an aggregate measure over all image pixels, we also evaluate the per-pixel performance between the composite images produced by the SCRandom and CTRAND methods using

TABLE II
SIMULATION RESULTS OF THE IMAGE PROCESSING TASKS

| Image Compositing | | | | |
|---|---|---|---|---|
| | People | | Plane | |
| **Method** | **RT (sec)** | **PSNR (dB)** | **RT (sec)** | **PSNR (dB)** |
| **SCRandom** | 46.34 | 31.17 | 166.5 | 31.80 |
| **CTRAND** | 0.194 | 30.66 | 1.101 | 31.25 |
| **SCSobol** | 604.2 | 50.42 | 2662.2 | 52.88 |
| **CT0** | 0.153 | 50.38 | 0.961 | 52.86 |
| **CONVN** | 0.072 | ref. | 0.618 | ref. |

| Pattern Detection* | | | | |
|---|---|---|---|---|
| **N** | **CONVN** | **SCMax (sec)** | **CTMAX (sec)** | **Memory** |
| 8 | 7.653 sec | 91.43 | 7.59 | $\times$ 5.075 |
| 16 | | 128.69 | 7.62 | $\times$ 9.819 |
| 32 | | 211.60 | 7.60 | $\times$ 19.305 |
| 64 | | 384.88 | 7.61 | $\times$ 38.277 |
| 128 | | 755.64 | 7.62 | $\times$ 76.223 |

*Accuracy: 100% Pattern hits are obtained in all tests.

| Bilinear Interpolation | | | |
|---|---|---|---|
| **Method** | **RT* (sec)** | **PSNR (dB)** | **Memory** |
| **SCRandom** | 8.186 | 30.496 | $\times$ 22.792 |
| **CTRAND** | 0.024 | 29.857 | |
| **SCSobol** | 9.325 | 45.778 | $\times$ 22.803 |
| **CT0** | 0.015 | 44.661 | |
| **SCLFSR** | 7.707 | 35.927 | $\times$ 22.779 |
| **CTLFSR** | 0.025 | 34.665 | |
| **CONVN** | 0.009 | ref. | |

*RT: RunTime. For memory monitoring, MATLAB *whos* command is utilized. Tests are performed on a computer with Intel(R) Core(TM) i7-7700HQ CPU @2.80GHz.

MAE. First, excluding pixels that bring an absolute error of zero ($|\text{CTRAND} - \text{SCRandom}| = 0$), we only considered the pixels that have an error ($|\text{CTRAND} - \text{SCRandom}| \geq 1$). The MAEs were 5.897 and 5.352 for the people and plane test images, respectively. MAEs, when including all pixels, were 2.762 and 2.496, respectively. We also measured the PSNR values for the case of having all pixels. The PSNR for the people example was 34.416 dB, and that for the plane example was 34.950 dB.

We evaluate the pattern detection case study in the following scenarios: 1) SC with maximally correlated random bit-streams (SCMax); 2) CT with maximum correlation (CTMAX); and 3) CONVN. The simulation results are shown in Table II.

For pattern detection, the proposed CTMAX method is as fast as the conventional binary execution (CONVN). While the SCMax's runtime increases by increasing the bit-stream length ($N$), CTMAX's runtime remains constant. The CTMAX's runtime was the same until $N = 2^{32}$, after which MATLAB was unresponsive due to the array size limit. A constant runtime independent of $N$ is another important advantage of the proposed technique. As Table II shows, the memory efficiency of CT simulation improves compared to the bit-stream processing as $N$ increases. The MATLAB built-in absolute value function (`abs()`) makes CONVN slightly slower in pattern detection application. However, we observed that the MATLAB built-in functions speed up for a large number of iterations ($> 10^5$) when the size of testing images increases. Our simulation results show that the CT-based approach can address the long latency issue of simulating SC image processing designs, completing the computations as fast as conventional binary processing.

Finally, we evaluate the simulation of the bilinear interpolation design. In addition to the previous computing scenarios, we include conventional SC with LFSR-based bit-stream processing (SCLFSR) and CT with LFSR-model fluctuations (CTLFSR) in our simulations. We validated the emulation of the three random source methods. Particularly, we considered hypergeometric distribution for emulating LFSR-based bit-streams in the CT model. As expected, the runtimes were much shorter with CT, and the PSNR values were closely followed. Memory efficiency was also 22× better with CT-based simulations compared to conventional SC simulation.

## V. CONCLUSION

This work proposes a fast method for the simulation of SC image processing systems. The data are processed with the aid of CTs without actual bit-stream processing. The methodology is evaluated on three SC image processing case studies: 1) image compositing; 2) pattern detection; and 3) bilinear interpolation. Experimental results show that the proposed approach can simulate SC nearly as fast as conventional binary processing with a substantial reduction in memory usage. In future work, we will employ the CT-based SC simulation technique for fast and efficient simulation of SC-based deep learning systems.

## REFERENCES

[1] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 8, pp. 1515–1531, Aug. 2018.

[2] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Proc. DAC*, Austin, TX, USA, 2013, pp. 1–6.

[3] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 12, pp. 2925–2938, Dec. 2019.

[4] S. Aygun and E. O. Gunes, "Utilization of contingency tables in stochastic computing," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 6, pp. 2942–2946, Jun. 2022.

[5] M. H. Najafi and D. J. Lilja, "High-speed stochastic circuits using synchronous analog pulses," in *Proc. 22nd Asia–South Pacific Design Autom. Conf. (ASP-DAC)*, 2017, pp. 481–487.

[6] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 93–105, Jan. 2011.

[7] M. Ranjbar, M. E. Salehi, and M. H. Najafi, "Using stochastic architectures for edge detection algorithms," in *Proc. 23rd Iran. Conf. Electr. Eng.*, 2015, pp. 723–728.

[8] R. Wang, J. Han, B. F. Cockburn, and D. G. Elliott, "Stochastic circuit design and performance evaluation of vector quantization for different error measures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 10, pp. 3169–3183, Oct. 2016.

[9] S. Aygün, M. Altun, and E. O. Güneş, "Sobel filter operation in image processing via stochastic arithmetic-logic unit design," in *Proc. IEEE SIU*, 2017, pp. 1–4.

[10] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 449–462, Mar. 2014.

[11] N. Onizawa, D. Katagiri, K. Matsumiya, W. J. Gross, and T. Hanyu, "Gabor filter based on stochastic computation," *IEEE Signal Process. Lett.*, vol. 22, no. 9, pp. 1224–1228, Sep. 2015.

[12] N. Onizawa, D. Katagiri, K. Matsumiya, W. J. Gross, and T. Hanyu, "An accuracy/energy-flexible configurable Gabor-filter chip based on stochastic computation with dynamic voltage–frequency–length scaling," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 8, no. 3, pp. 444–453, Sep. 2018.

[13] K. Boga, N. Onizawa, F. Leduc-Primeau, K. Matsumiya, T. Hanyu, and W. J. Gross, "Stochastic implementation of the disparity energy model for depth perception," in *Proc. SiPS*, 2015, pp. 1–6.

[14] H. Abdellatef, M. Khalil-Hani, and N. Shaikh-Husin, "Accurate and compact stochastic computations by exploiting correlation," *Turkish J. Electr. Eng. Comput. Sci.*, vol. 27, no. 1, pp. 547–564, 2019.

[15] J. H. Anderson, Y. H. Azumi, and S. Yamashita, "Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy," in *Proc. DATE*, Dresden, Germany, 2016, pp. 1550–1555.

[16] S. Liu and J. Han, "Toward energy-efficient stochastic circuits using parallel Sobol sequences," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1326–1339, Jul. 2018.

[17] M. H. Najafi, D. J. Lilja, and M. Riedel, "Deterministic methods for stochastic computing using low-discrepancy sequences," in *Proc. ICCAD*, 2018, pp. 1–8.

[18] A. Alaghi, "The logic of random pulses: Stochastic computing," Ph.D. dissertation, Dept. Comput. Sci. Eng., Univ. Michigan, Ann Arbor, MI, USA, 2015.

[19] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *Proc. IEEE 31st ICCD*, 2013, pp. 39–46.

[20] S. Aygun, M. H. Najafi, M. Imani, and E. O. Gunes. "Github." 2022. [Online]. Available: https://github.com/serco425/Agile-SC-Simulation

[21] R. Szeliski, *Computer Vision: Algorithms and Applications*. London, U.K.: Springer, 2011. [Online]. Available: https://link.springer.com/book/10.1007/978-1-84882-935-0

[22] R. Manohar, "Comparing stochastic and deterministic computing," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 119–122, Jul.–Dec. 2015.

[23] B. Moons and M. Verhelst, "Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 4, no. 4, pp. 475–486, Dec. 2014.

[24] T. J. Baker and J. P. Hayes, "The hypergeometric distribution as a more accurate model for stochastic computing," in *Proc. DATE*, 2020, pp. 592–597.

[25] V. T. Lee, A. Alaghi, and L. Ceze, "Correlation manipulating circuits for stochastic computing," in *Proc. DATE*, 2018, pp. 1417–1422.

[26] K. Kim, P.-S. Shim, and S. Shin, "An alternative bilinear interpolation method between spherical grids," *Atmosphere*, vol. 10, no. 3, p. 123, 2019. [Online]. Available: https://www.mdpi.com/2073-4433/10/3/123

[27] P. Koopman. "Maximal length LFSR feedback terms." Accessed: May 10, 2022. [Online]. Available: https://users.ece.cmu.edu/~koopman/lfsr/