Bit-Stream Processing with No Bit-Stream: Efficient Software Simulation of Stochastic Vision Machines

Sercan Aygun
University of Louisiana at Lafayette
Lafayette, LA, USA
sercan.aygun@louisiana.edu

M. Hassan Najafi
University of Louisiana at Lafayette
Lafayette, LA, USA
najafi@louisiana.edu

Mohsen Imani
University of California Irvine
Irvine, CA, USA
m.imani@uci.edu

Ece Olcay Gunes
Istanbul Technical University
Istanbul, Turkey
gunesec@itu.edu.tr

ABSTRACT

Stochastic computing (SC) is an emerging paradigm that has come to the fore in computer vision applications in the last decade. Complex arithmetic circuitry is reduced to simple logic gates, fed with uniform random bit-streams. Due to the requirement of long bit-streams, the computer-aided simulation of SC systems is facing run-time and memory-use challenges. This work presents an efficient approach for emulating SC-based systems. The proposed simulation technique does not utilize actual bit-streams but produces similar results as if the traditional stochastic bit-streams were processed. The data are processed with the aid of a correlation-controlled contingency table (CT) construct. Our technique emulates three state-of-the-art stochastic bit-streams, namely, bit-streams with binomial distribution, pseudo-random, and low-discrepancy bit-streams. We validate the proposed technique by emulating three new SC image processing designs. We propose novel SC designs for (i) template matching, (ii) image compositing, and (iii) bilinear interpolation. Our experimental results show that our simulation technique provides comparable accuracy to processing actual bit-streams, but at a significantly lower run-time and memory usage.

CCS CONCEPTS

• **Hardware** → **Emerging simulation**; • **Computing methodologies** → Computer vision.

KEYWORDS

computer vision, random sources, simulation, stochastic computing

ACM Reference Format:
Sercan Aygun, M. Hassan Najafi, Mohsen Imani, and Ece Olcay Gunes. 2023. Bit-Stream Processing with No Bit-Stream: Efficient Software Simulation of Stochastic Vision Machines . In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3583781.3590217

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA

ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590217

1 INTRODUCTION

Stochastic computing (SC) [5, 22] is re-emerging as an alternative method of computing, replacing conventional binary computing. SC offers hardware-friendly solutions for various applications, from vision to learning machines [4, 17–19, 23]. Low-implementation cost and high tolerance to noise are the main advantages of computation in the stochastic domain. Arithmetic operations are performed by simple bit-wise operations on uniform (random) bit-streams. For example, multiplication is realized by bit-wise `AND` operation on the bit-streams [5]. Both approximate and accurate computations are feasible with SC by structuring bit-streams and controlling their length [22]. More than 50× to 100× reduction in the hardware cost is common compared to the cost of binary counterparts [5]. Tolerating high rates of noise (e.g., 30%-50%) is another appealing property, as all digits of an SC bit-stream have the same weight.

An important step in designing SC systems is evaluating their performance and verifying their functionality by simulating their bit-level operations with software programs. Scsynth [2] and BitSAD [12] are examples of such programs. In SC, the accuracy of computations increases by increasing the bit-stream length. To represent a data value with a binary resolution $\frac{1}{2^n}$, a bit-stream with a length of at least $N = 2^n$ bits is needed [9]. This means that the length of a stochastic bit-stream increases *exponentially* with the resolution. Depending on the needed accuracy, SC systems process bit-streams with different lengths, from short lengths of 2^3 to longer lengths of 10^3-10^4 bits. Computer simulation of SC systems with long bit-streams often takes a long latency and a high amount of memory. Even for simulation of basic SC operations such as multiplication of two data by bit-wise `AND`ing two operand bit-streams, long latency is inevitable when very long bit-streams are processed. Aygun and Gunes [9] recently proposed a *contingency table* (CT) approach to perform stochastic logic operations without using bit-by-bit processing. This work extends the CT-based technique of simulating SC systems by modeling three state-of-the-art stochastic bit-streams, namely bit-streams with binomial distribution, linear-feedback shift register (LFSR)-based pseudo-random bit-streams, and Sobol-based low-discrepancy (LD) bit-streams. In summary, the main contributions of this work are as follows:

- Fast and efficient CT-based emulation of state-of-the-art stochastic bit-streams with binomial, LFSR-based pseudo-random, and Sobol-based LD distribution.

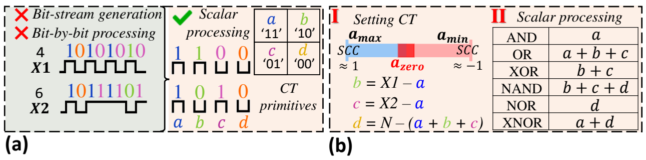


Figure 1: (a) Bit-by-bit processing, and solution with scalar processing. (b) CT setting and logic operations with scalars.

- Proposing novel SC image processing designs for template matching, image compositing, and bilinear interpolation.
- Evaluating the CT-based SC simulation at application level for the three new image processing case studies.
- Significant run-time and memory usage reduction.

2 BACKGROUND

2.1 Stochastic Bit-Stream Processing

This section briefly reviews SC's basics, including data encoding and correlation/independence requirements. Unipolar encoding (UPE) and bipolar encoding (BPE) are the two most common methods for encoding data in SC. Both UPE and BPE can encode a positive scalar value X where $0 \leq X \leq N$ (or a probability value $\frac{X}{N}$ where $0 \leq \frac{X}{N} \leq 1$). However, BPE supports negative values $-N \leq X \leq N$ (or $-1 \leq \frac{X}{N} \leq 1$). The trade-off is that BPE needs twice bit-stream length for the same accuracy. The number of logic-1s in a bit-stream determines the bit-stream value. Assume that $\boldsymbol{X}$ represents an encoded bit-stream and the i^{th} bit is accessed by $\boldsymbol{X}(i)$. In UPE, the total number of logic-1s, $\sum_{i=1}^{N} \boldsymbol{X}(i)$, divided by N determines the bit-stream value, so $P = \frac{\sum_{i=1}^{N} \boldsymbol{X}(i)}{N}$. On the other hand, in BPE, the bit-stream value is determined by $P = \frac{\sum_{i=1}^{N} \boldsymbol{X}(i)}{N} = (\frac{X}{N} + 1)/2$.

Correlation between stochastic bit-streams is another important concept in SC. Some operations, such as multiplication using logical AND, require uncorrelated or independent operand bit-streams for accurate operation. Some other operations, such as absolute value subtraction using XOR, minimum using AND, and maximum using OR, require maximally correlated operands for correct performance. Stochastic cross-correlation (*SCC*) has been frequently used in the literature to quantify the correlation between two bit-streams [3]. The piecewise *SCC* function shown in eq. (1) returns a correlation value within the $[-1, 1]$ interval. In the formula, the values denoted by a, b, c, and d are logic pairs 11, 10, 01, and 00, respectively, from the same bit positions of the two bit-streams.

$$SCC = \begin{cases} \frac{ad-bc}{N\times min(a+b,a+c)-(a+b)\times(a+c)} & , \; if \; ad > bc \\ \frac{ad-bc}{(a+b)\times(a+c)-N\times max(a-d,0)} & , \; else \end{cases} \tag{1}$$

2.2 Contingency Table (CT)

The CT approach has emerged as a promising solution for run-time-efficient and memory-aware simulation of SC systems [9]. Instead of generating and processing actual stochastic bit-streams, the CT method calculates the desired logic operation over the scalar input operands that make up the bit-streams. For any two operands (bit-streams), a CT is built. *Correlation* is the key parameter during the setup of a CT. CT is a 2×2 table with four basic primitives: a, b, c, and d. Each primitive holds the cumulative sum of the four possible states from all corresponding bit positions in the two bit-streams (a: $\boldsymbol{X1}(i) = 1$, $\boldsymbol{X2}(i) = 1 \rightarrow$ '11', b: $\boldsymbol{X1}(i) = 1$, $\boldsymbol{X2}(i) = 0 \rightarrow$ '10', c: $\boldsymbol{X1}(i) = 0$, $\boldsymbol{X2}(i) = 1 \rightarrow$ '01', and d: $\boldsymbol{X1}(i) = 0$, $\boldsymbol{X2}(i) = 0 \rightarrow$ '00'). Fig. 1 (a) compares traditional bit-stream processing and CT approaches. The CT approach processes scalar values directly rather than converting them to bit-stream format; thereby: (i) there is no bit-stream generation, and (ii) latency-prone bit-by-bit processing is avoided.

Fig. 1 (b) shows how a CT is set. The inputs are: 1) N, 2) the input scalars ($X1$, $X2$), and 3) the target correlation. a is the first CT primitive we find. It is called the *prior primitive*. Since a represents the number of overlapping 1s between two bit-streams, it denotes the number of 1s that occur at the output of logical AND operation. The determination of a is an antecedent for other primitives and depends on the correlation. According to the *SCC* metric, the three critical points in the correlation spectrum between -1 and 1 can be expressed with: a_{max}, a_{min}, and a_{zero}. Respectively, these are the maximum correlation where *SCC* converges to 1, the minimum correlation where *SCC* converges to -1, and near-zero correlation where *SCC* is around zero. The near-zero correlation is particularly critical for operations such as multiplication (AND in UPE, XNOR in BPE) where uncorrelated bit-streams are needed. As shown in Fig. 1 (b), the three critical correlation points, a_{max}, a_{zero}, and a_{min}, are determined during CT setup and initiate the corresponding $CTMAX$, $CT0$, and $CTMIN$ tables, respectively. The maximum value of a, a_{max}, is determined by $min(X1, X2)$. The minimum value of a, a_{min}, is determined by $max(0, X1 + X2 - N)$. $CT0$ is obtained by optimizing the *SCC* formula to zero, which yields $a_{zero} = \lfloor \frac{X1 \times X2}{N} \rceil$ [9]. After finding the *prior primitive* (a), the other primitives (b, c, and d) are determined by the formulas shown in Fig. 1 (b). For example, b is found by $X1 - a$. When CT is set, only the CT primitives and their linear combinations with summation are sufficient to obtain the output of logical operations. The *scalar processing* table in Fig. 1 (b) can be used to find the total count of 1s in the output of the primary logic operations.

3 CT-BASED RANDOM SOURCE SIMULATION

In this section, we propose CT-based methodology for simulating the three state-of-the-art random sources of SC.

Binomial Distribution. Bit-streams have binomial distribution when each bit is a Bernoulli random variable (RV). Considering the *Independent and Identically Distributed* RV, a stochastic bit-stream has a binomial distribution with a variance $\sigma^2 = \frac{P(1-P)}{N}$, where P is the success probability of the Bernoulli distribution. The expected result from the SC operation is called the *exact* value or P_Y. However, the produced value at the output of the SC operation can differ from the expected value due to random fluctuations. The produced value is called the *estimated* value or $\hat{P}_Y$. The difference between the exact and estimated values is evaluated with the mean squared error (MSE). Alaghi et al. [1] indicate that the random fluctuation errors are measured using MSE, $error = \mathbb{E}[(P_Y - \hat{P}_Y)^2]$, which yields $error = P_Y(1 - P_Y)/N$ in Bernoulli RV case. The random fluctuations error is defined such that the error decreases as N increases. Ting and Hayes underline that the MSE results in the variance σ^2 [28], with regards to output probability. Hence, the

CT simulation approach can directly take $\sigma = \sqrt{P_Y(1-P_Y)/N}$ into account. While setting $CT0$ for uncorrelated bit-stream emulation via a primitive, the error deviation is included in a with $\frac{a}{N} + \sqrt{P_Y(1-P_Y)/N}$, where $\frac{a}{N}$ is the output probability of an AND gate in the no error case.

LFSR. We model LFSRs in CT approach ($CTLFSR$) using the hypergeometric distribution proposed by Baker and Hayes [10]. They show that LFSR-based bit-streams fit better to hypergeometric RV bit-stream generation than the binomial distribution. They define the output deviation of the bit-wise AND operation on LFSR-based bit-streams as $\sigma = \sqrt{\frac{P_{X1}\times P_{X2}\times(1-P_{X1})\times(1-P_{X2})}{N-1}}$. Since AND output is related to the "a" primitive, we use this output deviation after setting up the CT via "a". For the near-zero CT ($CT0$), a must be updated; So the output probability model for bit-wise ANDing LFSR-based bit-streams becomes $\frac{a}{N} + \sqrt{\frac{P_{X1}\times P_{X2}\times(1-P_{X1})\times(1-P_{X2})}{N-1}}$.

Sobol. Sobol-based bit-streams achieve deterministic-like arithmetic accuracy if long enough bit-streams are processed [21]. For example, accurate result from multiplying two n-bit precision data can be achieved by processing $N \times N$-bit Sobol bit-streams where $N = 2^n$. Since $a = \lfloor \frac{X1\times X2}{N} \rceil$ is obtained from $SCC = 0$ optimization that guarantees high accuracy in AND multiplication, Sobol-based and $CT0$-based results are expected to be similar.

4 CT-BASED SC IMAGE PROCESSING

SC has been previously used for the simple execution of various image processing tasks such as median filtering, contrast stretching, image segmentation, and edge detection [4, 18]. This work extends the SC-based image processing domain with three new applications: template matching, image compositing, and bilinear interpolation. We propose three new SC architectures for these algorithms and employ the CT approach to speedup their execution.

4.1 Template Matching

In template matching, a template K is searched throughout an image I, using a sliding window. Let $I_{(r\times c)}$ be a grayscale image with $r \times c$ size. The template, as a kernel $K_{(r_K\times c_K)}$, is a subset of I like a 1-Dimensional (1D) or 2D window, where $r_K < r$ and $c_K < c$. K is applied on I by processing each intersecting pixel of I and K. The movement of K is similar to the convolution operation. The main objective of template matching is to catch the highest similarity and get the exact template match. The conventional operation is defined using $T = \sum f(I, K)$, where $f(I, K)$ can be any of the following functions: (i) $f = |I - K|$, (ii) $f = (I - K)^2$, or (iii) $f = I \times K$. The first function is an absolute difference operation and the temporary matching image T is set as the sum of the absolute differences (SAD). The second function is an MSE-related formula based on the sum of the squared differences (SSD), and the third one is a cross-correlation-based function [16]. Depending on the selected method, T is processed to find its minimum-valued (if (i) or (ii) is utilized) or maximum-valued (if (iii) is applied) positions that indicate the template coordinates. In (iii), the correlation function is similar to convolution without spatially flipping the template. Nevertheless, brighter pixels may cause a wrong template assignment. "Normalization" is proposed as a solution for this issue [14, 16]. If

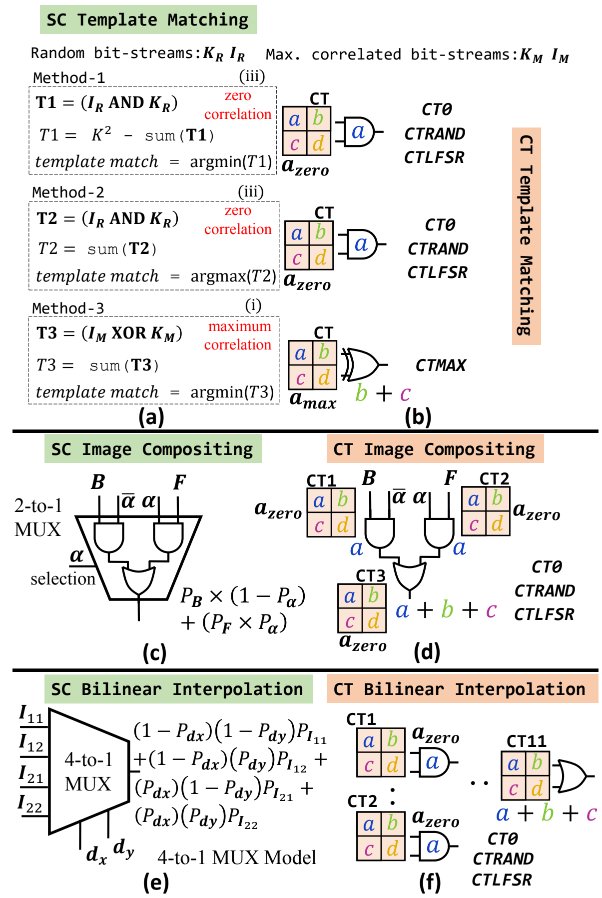


Figure 2: Proposed SC image processing methods and their CT simulation: (a) three methods for template matching and (b) their corresponding CT-based emulation. (c) Image compositing using a 2-to-1 MUX, and (d) the cascaded CT model of MUX. (e) Bilinear interpolation with a 4-to-1 MUX, and (f) corresponding multi-level CT model with *AND* & *OR*.

pixel operations are kept in the [0, 255] interval, the brighter pixel values may be assigned as the template, and [0, 1] normalized range is recommended. At this point, SC can be used by processing data in the [0, 1] interval. Lande et al. [15] discuss the correlation property of SC based on AND operation. Therefore, the cross-correlation-based approach (iii) fits SC using the AND multiplier. Besides, in SC-based template matching with (iii), performing AND operation on probabilities brings normalization. Stefano et al. [26] perform normalization by dividing $\sum(I \times K)$ by the product of the main image and the template L_2 norms, $|I|_2 \times |K|_2$. This term is approximated as $N \times N$ coming from the denominator of $P_I \times P_K$, when SC-based AND is applied.

Fig. 2 (a) presents our SC-based template matching method. Method-1 and Method-2 use the multiplication property of (iii) via AND operation, while Method-3 uses the absolute value subtraction of (i) with bit-wise XOR. In Method-1, bit-streams from I ($\boldsymbol{I}$) and bit-streams from K ($\boldsymbol{K}$) are represented in UPE and must have zero correlation for accurate multiplication. After bit-wise ANDing $\boldsymbol{I}$ and

$\boldsymbol{K}, \boldsymbol{T1}$ is decoded using a simple sum, i.e., the population count of 1s. The result is expected to be $\sum I{\times}K$. Method-1 subtracts $\sum I{\times}K$ from K^2, which produces a zero for an exact template match. Method-2 includes only an SC multiplication. It approximately targets the maximum values, i.e., K^2 on the exact matches and so is less accurate than Method-1. Finally, Method-3 uses the absolute value subtraction of (i) by performing bit-wise XOR on maximally correlated bit-streams. This method finds an exact match when decoding $\boldsymbol{T3}$ produces a zero. Therefore, minimum-valued positions represent the template coordinates. We note that, in each method, only the **bold** parts are processed in the stochastic domain. Thus, Method-2 and Method-3 have only SC operations, while Method-1 also includes binary subtraction and square operation in the conventional binary domain. For the CT approach, Method-1 and Method-2 are performed with uncorrelated CT models targeting zero correlation and random source simulation, as presented in Fig. 2 (b). We denote the binomial distribution using CT by *CTRAND* and the LFSR model with CT by *CTLFSR*. The Sobol-based model is also given by *CT0*, which is the most accurate case. Method-3 requires maximum correlation, and so uses *CTMAX*.

4.2 Image Compositing

For image compositing operation, we use a 2-input multiplexer (MUX) to combine two images: background B and foreground F. In conventional image processing, a composited image (C) is denoted by the formula $C = B(1-\alpha) + F\alpha$, where α is the foreground image opacity [27]. A MUX with $\boldsymbol{X1}$ and $\boldsymbol{X2}$ input and $\boldsymbol{S}$ select bit-stream implements $P_{\boldsymbol{X1}}(1-P_{\boldsymbol{S}}) + P_{\boldsymbol{X2}}P_{\boldsymbol{S}}$ [25]. The composited image C and the *MUX* formula are well-match by re-writing $\boldsymbol{C} = P_{\boldsymbol{B}}(1-P_{\boldsymbol{\alpha}}) + P_{\boldsymbol{F}}P_{\boldsymbol{\alpha}}$. As shown in Fig. 2 (c), connecting $\boldsymbol{B}$, $\boldsymbol{F}$, and $\boldsymbol{\alpha}$ to a MUX yields the compositing operation. Fig. 2 (d) illustrates the CT model for a MUX. Multiple CTs are constructed with near-zero correlation initially.

4.3 Bilinear Interpolation

The third SC design is bilinear interpolation used in image resizing. Bilinear interpolation is based on linear interpolations in both x (width) and y (height) directions of the xy plane. With repetitive linear interpolations for x and y, bilinear interpolation, a.k.a. bilinear filtering, is obtained [24]. Let I be a 2D matrix with x and y row-column structure. Four points in the coordinate system define the I rectangular region: $(\mathrm{x}_1,\mathrm{y}_1)$, $(\mathrm{x}_1,\mathrm{y}_2)$, $(\mathrm{x}_2,\mathrm{y}_1)$, and $(\mathrm{x}_2,\mathrm{y}_2)$. A new point lying inside this region is denoted as (x,y). Based on the vertices of I, $I(\mathrm{x},\mathrm{y})$ is to be estimated. After x- and y-related interpolations, the formula for *bilinear interpolation* is denoted as [11]:

$$I(\mathrm{x},\mathrm{y}) \approx a_{11}I(\mathrm{x}_1,\mathrm{y}_1) + a_{21}I(\mathrm{x}_2,\mathrm{y}_1) + a_{12}I(\mathrm{x}_1,\mathrm{y}_2) + a_{22}I(\mathrm{x}_2,\mathrm{y}_2)$$

where

$$a_{11} = [(\mathrm{x}_2-\mathrm{x})(\mathrm{y}_2-\mathrm{y})]/[(\mathrm{x}_2-\mathrm{x}_1)(\mathrm{y}_2-\mathrm{y}_1)]$$

$$a_{21} = [(\mathrm{x}-\mathrm{x}_1)(\mathrm{y}_2-\mathrm{y})]/[(\mathrm{x}_2-\mathrm{x}_1)(\mathrm{y}_2-\mathrm{y}_1)]$$

$$a_{12} = [(\mathrm{x}_2-\mathrm{x})(\mathrm{y}-\mathrm{y}_1)]/[(\mathrm{x}_2-\mathrm{x}_1)(\mathrm{y}_2-\mathrm{y}_1)]$$

$$a_{22} = [(\mathrm{x}-\mathrm{x}_1)(\mathrm{y}-\mathrm{y}_1)]/[(\mathrm{x}_2-\mathrm{x}_1)(\mathrm{y}_2-\mathrm{y}_1)]$$

By mapping I into the unit square via normalization of the values, the vertices are now $(\mathrm{x}_1=0,\mathrm{y}_1=0)$, $(\mathrm{x}_2=1,\mathrm{y}_1=0)$, $(\mathrm{x}_1=0,\mathrm{y}_2=1)$, and $(\mathrm{x}_2=1,\mathrm{y}_2=1)$ [13]. Thereby,

$$I(\mathrm{x},\mathrm{y}) \approx (1-\mathrm{x})(1-\mathrm{y})I(\mathrm{x}_1,\mathrm{y}_1) + (\mathrm{x})(1-\mathrm{y})I(\mathrm{x}_2,\mathrm{y}_1) + (1-\mathrm{x})(\mathrm{y})I(\mathrm{x}_1,\mathrm{y}_2) + (\mathrm{x})(\mathrm{y})I(\mathrm{x}_2,\mathrm{y}_2)$$

$I_{(r\times c)}$ represents an image with r rows and c columns, and the resized image, $\bar{I}_{(\varrho r\times\varrho c)}$, is obtained by bilinear filtering. By rewriting the equity for $I(\mathrm{x},\mathrm{y})$ we get $(1-dx)(1-dy)I_{11} + (1-dx)(dy)I_{12} + (dx)(1-dy)I_{21} + (dx)(dy)I_{22}$ where $I_{11}, I_{12}, I_{21}, I_{22}$ are neighbouring pixel values, and dx, dy are relative positions.

For SC design, the probabilities of the neighboring pixels and the relative positions are denoted as $P_{\boldsymbol{I_{11}}}$, $P_{\boldsymbol{I_{12}}}$, $P_{\boldsymbol{I_{21}}}$, $P_{\boldsymbol{I_{22}}}$, $P_{\boldsymbol{dx}}$, and $P_{\boldsymbol{dy}}$. By using the equation of $I(\mathrm{x},\mathrm{y})$, we get

$$P_{\boldsymbol{I}(\mathrm{x},\mathrm{y})} = (1-P_{\boldsymbol{dx}})(1-P_{\boldsymbol{dy}})P_{\boldsymbol{I_{11}}} + (1-P_{\boldsymbol{dx}})(P_{\boldsymbol{dy}})P_{\boldsymbol{I_{12}}} + (P_{\boldsymbol{dx}})(1-P_{\boldsymbol{dy}})P_{\boldsymbol{I_{21}}} + (P_{\boldsymbol{dx}})(P_{\boldsymbol{dy}})P_{\boldsymbol{I_{22}}}$$

As shown in Fig. 2 (e), this can be implemented with a 4-to-1 MUX, where dx and dy are connected to the select ports of the MUX [6, 8]. Fig. 2 (f) shows the CTs for modeling this MUX.

Algorithm-1: Dynamic Approach

included in run-time

```
Get initial binary seed;
Get X to be encoded in a bit-stream;
for i = 1:1:bitstream_length
    %Imitate Comparator Operation
    if X > BTD(seed)
        bitstream(i) = 1;
    else
        bitstream(i) = 0;
    Perform XOR operation(s) for taps;
    Feedback & shift binary seed;
end
```

BTD()
b2d()
bi2de()
bit2int()

Algorithm-2: Table-based Approach

included in run-time

```
Get initial binary seed;
for i = 1:1:bitstream_length
    LFSR_TABLE(i) = BTD(seed)
    Perform XOR operation(s) for taps;
    Feedback & shift binary seed;
end
for i = 1:1:bitstream_length
    %Imitate Comparator Operation
    if X > LFSR_TABLE(i)
        bitstream(i) = 1;
    else
        bitstream(i) = 0;
end
```

BTD()
b2d()
bi2de()
bit2int()

5 EXPERIMENTAL RESULTS

We evaluate the performance of the proposed techniques compared to the conventional simulation of SC. First, we evaluate different approaches for performing basic two-input SC multiplication. Then, we extend our evaluation to the three discussed image processing techniques. We set up two separate environments for all tests: 1) conventional approach of processing bit-streams and 2) CT-based approach. The first case generates and processes actual bit-streams, while the second one operates only on CTs' scalar values. All our simulations are carried out with the MATLAB tool.

For the conventional case of processing bit-streams, we simulate the three random sources. We generate bit-streams with binomial

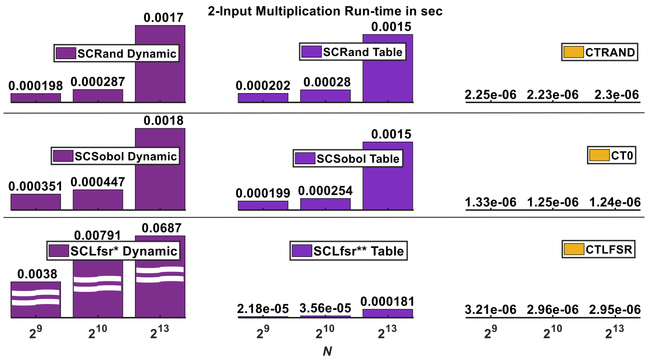


Figure 3: Run-time (in seconds) comparison of different methods for two-input multiplication.

distribution (*SCRandom*), Sobol-based LD bit-streams (*SCSobol*), and LFSR-based pseudo-random bit-streams (*SCLfsr*). For LFSR method, we implement a *dynamic* and a *table-based* approach for generating bit-streams. The *dynamic* approach algorithmically generates random numbers every time running the simulation. In contrast, in the *table-based* approach, the random numbers are generated only once, stored in a table, and will be loaded at the run time to compare with the input data and generate corresponding bit-streams. Algorithm 1 and Algorithm 2 illustrate the pseudo-code for the dynamic and table-based simulation of the LFSR-based approach, respectively. We explore three functions (`b2d`, `bi2de`, and `bit2int`) in MATLAB for the needed binary to decimal (scalar) conversion. Our simulations showed that `b2d()` [30] is faster than `bi2de()` and `bit2int()` functions and provides the best run-time. So for the best performance of the LFSR-based method, we use this function in our simulations. We use the MATLAB built-in Sobol sequence generator for the Sobol-based approach, and for the binomial approach, we use the MATLAB `binornd()` function to generate random numbers. For each of the three methods, we measure the run-time 1000 times and report the average.

Two-Input Multiplication. Fig. 3 compares the run-time of different methods for basic two-input multiplication. The run-time is reported for different bit-stream sizes (N) to evaluate the scalability of each method. As it can be seen, for most N sizes of all random sources, the *table-based* approach provides a lower run-time than the *dynamic* approach. In particular, the LFSR-based method gets the most benefit from loading random numbers from a table as its *dynamic* simulation involves time-consuming base conversion. Evidently, the run-time increases by increasing the bit-stream length with conventional bit-stream processing. On the other hand, the CT approach shows a constant run-time independent of the bit-stream size. As it can be seen, the CT approach provides significantly shorter run-times compared to the conventional bit-stream processing for all three random source methods. *CT0* is the fastest, as fast as binary computing, then *CTRAND* and *CTLFSR* come in turn. The slower performance of *CTLFSR* is due to using hypergeometric distribution that includes several multiplications.

Image Processing Case Studies. Next, we present the simulation results for the proposed image processing methods.

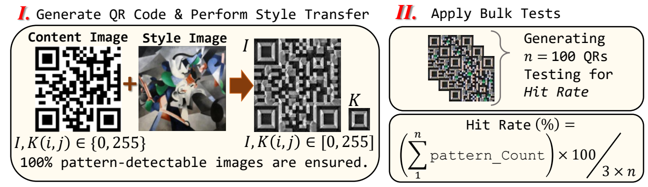


Figure 4: Test environment using style transferred QRs.

We use SC-based template matching for a visual -Quick Response (QR)- code. QR codes are widely used in daily life. Textual information is embedded in QR codes using a 2D structure. Tkachenko et al. elaborate the main steps of the QR code recognition in [29]. After finder (position) pattern localization, finder pattern (FP) coordinates are obtained. This step can be performed using several methods, such as the Hough transform, edge detection, and overlap of the centroids of continuous regions. In this work, we use template matching to search K pattern in a QR code I. Due to the increasing interest in QR beautification with background images, logos, and shapes [31], we generate and use an aesthetic QR code dataset containing grayscale images instead of full black-white standard QRs. We use the neural network-based technique proposed in [31] for visual QR code generation. Each generated QR code has 100% detectable patterns by readily-available QR cam scanners.

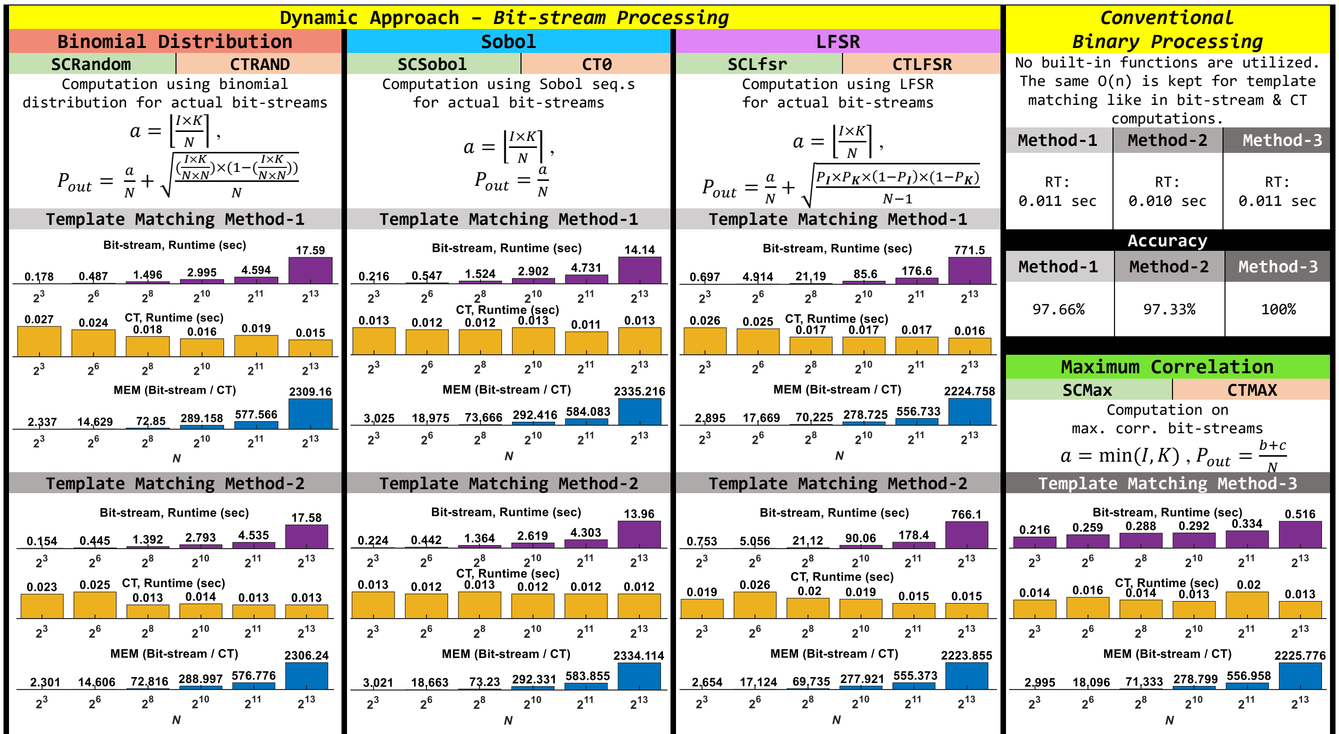
Fig. 4 shows the flow of our experiment for template matching in QR codes. After generating grayscale stylish QR codes, bulk tests are applied using the template matching methods of Section 4. The *hit rate* (HR) indicates the matching percentage in the dataset, considering that each QR code has three FPs. The total `pattern_Count` indicates cumulative successful pattern matches.

Table 1 presents our results for the template matching task. We compare the performance of the three proposed SC methods for template matching (Method-1, Method-2, Method-3) for processing conventional bit-streams (*SCRandom*, *SCSobol*, *SCLfsr*, *SCMax*), and the CT approach (*CTRAND*, *CT0*, *CTLFSR*, *CTMAX*). The run-time (purple bar plots: bit-stream processing, yellow bar plots: CT method) and memory usage (MEM - blue bar plots) are compared for different bit-stream lengths.

We record the HR accuracy of the first N that gives the accuracy of the conventional binary computing (*CONVN*). For instance, *SCRandom* with Method-1 reaches 97.66% accuracy with $N = 256$. This accuracy is validated with *CONVN* for Method-1. The accuracy results in the bottom row of Table 1 underscore that the conventional bit-stream processing and the proposed CT-based approach have the same accuracy. The Sobol-based approach achieves the *CONVN* accuracy with $N = 32$ due to its fast convergence property [20]. Method-1 and Method-2 have comparable accuracy, while Method-3 provides the best accuracy (100%). Compared to conventional bit-stream processing, memory is occupied efficiently with the CT approach, especially when emulating larger bit-stream sizes. Bit-stream processing of Method-3, which uses maximum correlated bit-streams, is relatively faster than Method-1 and Method-2, though it still suffers from high memory usage.

Table 1 shows that conventional bit-stream processing with the dynamic LFSR approach takes longer run-time compared to the binomial distribution- and Sobol-based approaches.

Table 1: Performance Comparison of the Implemented Template Matching Methods



HR Accuracy: N value is reported where the first best Hit Rate is obtained													
Template Matching Method-1		Template Matching Method-2		Template Matching Method-1		Template Matching Method-2		Template Matching Method-1		Template Matching Method-2		Template Matching Method-3	
SCRandom	CTRAND	SCRandom	CTRAND	SCSobol	CT0	SCSobol	CT0	SCLfsr	CTLFSR	SCLfsr	CTLFSR	SCMax	CTMAX
N=256	N=256	N=512	N=512	N=128	N=128	N=32	N=32	N=256	N=256	N=512	N=512	N=64	N=64
97.66%	97.66%	97.33%	97.33%	97.66%	97.66%	97.33%	97.33%	97.66%	97.66%	97.33%	97.33%	100%	100%

Tests are performed on a computer with Intel(R) Core(TM) i7-7700HQ CPU @2.80GHz, 16GB RAM, MATLAB 2017b. For memory monitoring, MATLAB whos command is utilized. Results are based on the average of 1000 different rounds. Runtime: RT (sec), Memory: MEM, and Hit Rate: HR (%).

Next, we evaluate the performance of the CT approach for the SC image compositing proposed in Section 4. Fig. 5 shows two pairs of Background (B) and Foreground (F) images as two different examples. Run-time (RT) in seconds and the output quality in PSNR (peak signal-to-noise ratio) are reported as the performance metrics. Bit-stream size is N=256. As it can be seen from the PSNR values, the CT approach successfully emulates different bit-stream generation methods and provides comparable PSNRs to the bit-stream processing. Comparing different random sources, the Sobol-based method achieves the best PSNRs, then the LFSR method, and finally, the binomial method. The *SCRandom* and *SCSobol* run-time results in Fig. 5 are obtained with the dynamic approach (the table-based approach did not significantly reduce the run-time for the binomial- and Sobol-based bit-stream processing). However, the $SCLfsr^{**}$ run-time is obtained using the table-based approach, which is faster than the dynamic $SCLfsr^{*}$ with run-times of 2278.5 sec for the Deer, and 950.5 sec for the Elephant example. In both examples, the CT approach significantly reduces the run-time compared to conventional bit-stream processing.

Finally, the SC bilinear interpolation proposed in Section 4 is tested for ϱ=4 scaling value. As with other tests, actual bit-stream processing and the CT method are evaluated. The binomial distribution and Sobol-based bit-stream processing are performed via the dynamic method. The table-based approach is used for the LFSR-based bit-stream processing. The results present PSNR values, run-time in seconds, and memory usage, while the bit-stream size is N=256. The results presented in Table 2 are the average of 1000 independent trials. According to the PSNR results obtained with reference to the image produced by the conventional binary calculation ($CONVN$), all bit-stream processing results are emulated very closely by the CT method. As expected, the Sobol method achieves the best performance, followed by the LFSR and binomial methods. CT-based simulation provides approximately 23 times more efficient memory usage than the conventional bit-streams.

6 CONCLUSIONS

In this work, we proposed three new SC-based image processing designs for template matching, image compositing, and bilinear interpolation tasks. The performance of each design was validated with a new SC emulation technique based on the CT. The CT method addresses the long latency and the high memory usage bottlenecks in the traditional approach of simulating SC systems. CT-based

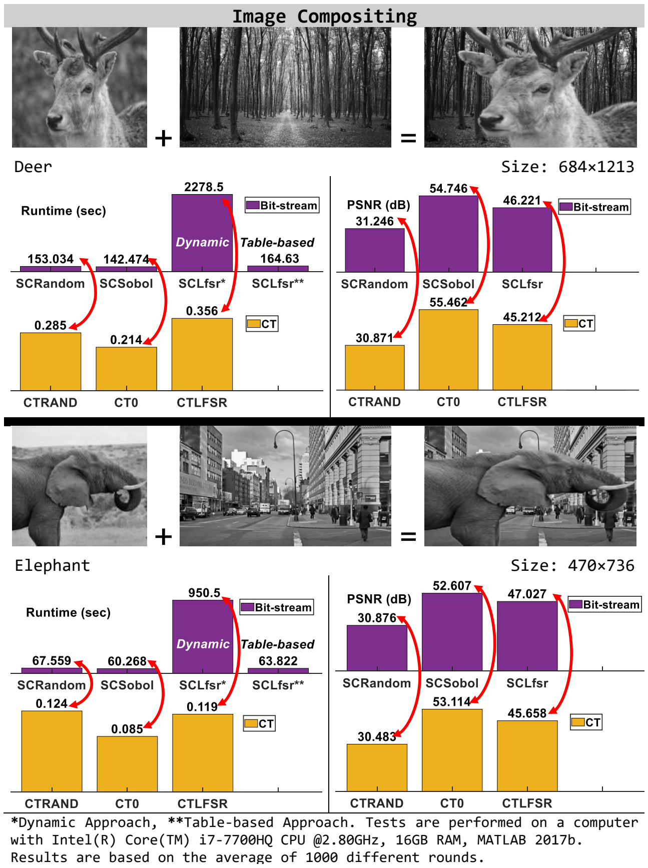
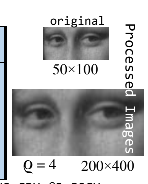


Figure 5: Examples of image compositing with different sizes.

Table 2: Bilinear Interpolation Results

SCRand	CTRAND	SCSobol	CT0	SCLfsr	CTLFSR	CONVN
RT: 34.417	RT: 0.074	RT: 37.778	RT: 0.045	RT: 29.205	RT: 0.141	RT: 0.032
PSNR: 30.450	PSNR: 29.782	PSNR: 41.533	PSNR: 41.540	PSNR: 33.165	PSNR: 33.321	PSNR: ref.
MEM: ×23.506		MEM: ×23.520		MEM: ×23.508		



Tests are performed on a computer with Intel(R) Core(TM) i7-7700HQ CPU @2.80GHz, 16GB RAM, MATLAB 2017b. Results are based on the average of 1000 different rounds.

simulation especially fits well for image processing applications. We modeled the state-of-the-art stochastic bit-stream generation approaches based on the binomial distribution, hypergeometric or pseudo-random, and low-discrepancy random sources. Our experimental results for simulation of the proposed SC image processing methods show that the CT method performs more efficiently than the conventional bit-stream processing in both run-time and memory usage. In terms of accuracy (PSNR and HR), CT-based simulation produces results with the same accuracy level as the results from traditional bit-stream processing. The proposed technique can be used for fast and efficient emulation of SC systems in other applications, such as SC-based neural network systems. Our open-source code of the CT framework can be accessed on GitHub [7].

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation (NSF) grants #2127780 and #2019511, Semiconductor Research Corporation (SRC), Office of Naval Research, grant #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research under award #FA9550-22-1-0253, the Louisiana Board of Regents Support Fund #LEQSF(2020-23)-RD-A-26, and generous gifts from Cisco, Xilinx, and Nvidia.

REFERENCES

[1] Armin Alaghi. 2015. *The logic of random pulses: Stochastic computing*. Ph. D. Dissertation. University of Michigan, Ann Arbor, USA.
[2] Armin Alaghi and Eamon Gaffney. 2019. scsynth. https://github.com/arminalaghi/scsynth. (Date accessed: Sept. 2022).
[3] Armin Alaghi and John P. Hayes. 2013. Exploiting correlation in stochastic circuit design. In *ICCD*. Asheville, NC, USA, 39–46.
[4] Armin Alaghi, Cheng Li, and John P. Hayes. 2013. Stochastic circuits for real-time image-processing applications. In *2013 DAC*. Austin, TX, USA, 1–6.
[5] Armin Alaghi, Weikang Qian, and John P. Hayes. 2018. The promise and challenge of stochastic computing. *IEEE TCAD* 37, 8 (2018).
[6] Sercan Aygun. 2022. *Stochastic bitstream-based vision and learning machines*. Ph. D. Dissertation. Istanbul Technical University, Istanbul, Turkey.
[7] Sercan Aygun et al. 2023. Contingency tables for stochastic computing. https://github.com/serco425/CT. (Date accessed: April 2023).
[8] Sercan Aygun and Ece Olcay Gunes. 2021. On the limit of multiplexers in stochastic computing. *Int. Journal of Multi. Stu. Innov. Tech.* 5, 1 (2021), 94 – 97.
[9] Sercan Aygun and Ece Olcay Gunes. 2022. Utilization of contingency tables in stochastic computing. *IEEE Trans. on Circ. and Syst. II: Expr. Br.* 69, 6 (2022).
[10] Timothy Baker and John Hayes. 2020. The hypergeometric distribution as a more accurate model for stochastic computing. In *DATE'20*.
[11] Changqing Cao et al. 2019. An improved faster R-CNN for small object detection. *IEEE Access* 7 (2019).
[12] Kyle Daruwalla et al. 2019. BitSAD v2: Compiler optimization and analysis for bitstream computing. *ACM Trans. Archit. Code Optim.* 16, 4, Article 43 (nov 2019).
[13] K.T. Gribbon and D.G. Bailey. 2004. A novel approach to real-time bilinear interpolation. In *IEEE DELTA*. 126–131.
[14] M.B. Hisham et al. 2015. Template matching using sum of squared difference and normalized cross correlation. In *IEEE SCOReD*.
[15] T.S. Lande et al. 2007. Running cross-correlation using bitstream processing. *Electronics Letters* 43 (October 2007), 1181–1183(2). Issue 22.
[16] J.P. Lewis. 1995. Fast template matching. In *Vision Interface 95*.
[17] Bingzhe Li et al. 2019. Low-cost stochastic hybrid multiplier for quantized neural networks. *J. Emerg. Technol. Comput. Syst.* 15, 2, Article 18 (March 2019).
[18] Peng Li et al. 2014. Computation on stochastic bit streams digital image processing case studies. *IEEE Trans. on VLSI Sys.* 22, 3 (2014), 449–462.
[19] Zhe Li et al. 2019. HEIF: Highly efficient stochastic computing-based inference framework for deep neural networks. *IEEE TCAD* 38, 8 (2019), 1543–1556.
[20] Siting Liu and Jie Han. 2017. Energy efficient stochastic computing with Sobol sequences. In *2017 DATE*. 650–653. https://doi.org/10.23919/DATE.2017.7927069
[21] M. Hassan Najafi et al. 2018. Deterministic methods for stochastic computing using low-discrepancy sequences. In *IEEE/ACM ICCAD* (San Diego, CA, USA).
[22] M. Hassan Najafi et al. 2019. Performing stochastic computation deterministically. *IEEE Tran. on VLSI Sys.* 27, 12 (2019), 2925–2938.
[23] M. Hassan Najafi and Mostafa E. Salehi. 2016. A Fast Fault-Tolerant Architecture for Sauvola Local Image Thresholding Algorithm Using Stochastic Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 2 (2016), 808–812. https://doi.org/10.1109/TVLSI.2015.2415932
[24] William Press et al. 1995. *Numerical recipes in C: the art of scientific computing*. Camb. Univ. Press, New York, USA.
[25] Weikang Qian et al. 2011. An architecture for fault-tolerant computation with stochastic logic. *IEEE Trans. Comput.* 60, 1 (2011).
[26] L. Stefano et al. 2003. An efficient algorithm for exhaustive template matching based on normalized cross correlation. In *12th Int. Conf. on Im. Analysis and Proc.*
[27] Richard Szeliski. 2011. Computer vision algorithms and applications. http://dx.doi.org/10.1007/978-1-84882-935-0 (Date accessed: Sept. 2022).
[28] Paishun Ting and John P. Hayes. 2019. Exploiting randomness in stochastic computing. In *2019 IEEE/ACM ICCAD*. Westminster, CO, USA, 1–6.
[29] Iuliia Tkachenko et al. 2016. Centrality bias measure for high density QR code module recognition. *Signal Proc.: Image Comm.* 41 (2016), 46–60.
[30] Zacharias Voulgaris. 2010. Efficient convertors between binary and decimal numbers. https://www.mathworks.com/matlabcentral/fileexchange/26447-efficient-convertors-between-binary-and-decimal-numbers (Date accessed: March 2022).
[31] Mingliang Xu et al. 2019. Stylized aesthetic QR code. *IEEE Transactions on Multimedia* 21, 8 (2019), 1960–1970.