Logical Computation on Stochastic Bit Streams with Linear Finite State Machines

Peng Li, *Student Member, IEEE,* David J. Lilja, *Fellow, IEEE,* Weikang Qian, *Member, IEEE,* Marc D. Riedel, *Senior Member, IEEE,* and Kia Bazargan, *Senior Member, IEEE*

Abstract—Most digital systems operate on a positional representation of data, such as binary radix. An alternative is to operate on random bit streams where the signal value is encoded by the probability of obtaining a one versus a zero. This representation is much less compact than binary radix. However, complex operations can be performed with very simple logic. Furthermore, since the representation is uniform, with all bits weighted equally, it is highly tolerant of soft errors (i.e., bit flips). Both combinational and sequential constructs have been proposed for operating on stochastic bit streams. Prior work has shown that combinational logic can implement multiplication and scaled addition effectively while linear finite-state machines (FSMs) can implement complex functions such as exponentiation and tanh effectively.

Prior work on stochastic computation has largely been validated empirically.This paper provides a rigorous mathematical treatment of stochastic implementation of complex functions such as exponentiation and tanh implemented using linear finite state machines. It presents two new functions, an absolute value function and exponentiation based on an absolute value, motivated by specific applications. Experimental results show that the linear FSM-based constructs for these functions have smaller area-delay products that corresponding deterministic constructs. They also are much more tolerant of soft errors.

Index Terms—stochastic computing, finite state machine, stochastic bit streams.

1 Introduction

In a paradigm first advocated by Gaines [1], logical computation is performed on stochastic bit streams: each real-valued number x $(0 \leq x \leq 1)$ is represented by a sequence of random bits, each of which has probability x of being one and probability $1 - x$ of being zero. Compared to a binary radix representation, a stochastic representation is not very compact. However, it leads to remarkably simple hardware for complex functions; it also provides very high tolerance to soft errors.

In the decades since Gaines' original work, there have been numerous papers discussing the paradigm. For example, Keane and Atlas [2] proposed a stochastic implementation of a finite impulse response filter for digital signal processing. Gaudet and Rapley proposed a stochastic implementation of an iterative decoding algorithm [3]. Gross et al. proposed a stochastic implementation of a low-density parity-check decoder [4]. McNeill and Card proposed a refractory system for counting pulses in neural computation [5]. Li et al. proposed a stochastic implementation of a neural network controller for small wind turbine systems [6]. Hori et al. proposed a stochastic implementation of a blind source separation system [7]. Onomi et al. proposed a stochastic implementation of a high-speed single flux-quantum up/down counter for neural computation [8]. Qian et al. presented a general synthesis method for logical computation on stochastic bit streams [9], [10], [11]. They showed that combinational logic can be synthesized to implement arbitrary polynomial functions, provided that such polynomials map the unit interval onto the unit interval. Their method is based on novel mathematics for manipulating polynomials in a form called Bernstein polynomials. In [10] Qian et al. showed how to convert a general power-form polynomial into a Bernstein polynomial with coefficients in the unit interval. In [9] they showed how to realize such a polynomial with a form of "generalized multiplexing." In [11], they demonstrated a reconfigurable architecture for computation on stochastic bit streams. They analyzed cost as well as the sources of error: approximation, quantization, and random fluctuations. They also studied the effectiveness of the architecture on a collection of benchmarks for image processing. More recently, Li and Lilja demonstrated a stochastic implementation of a kernel density estimation-based image segmentation algorithm [12] and other digital image processing algorithms [13].

Among them, most notable has been the work by Brown and Card [14], [15]. They demonstrated efficient constructs for a wide variety of basic functions, including multiplication, squaring, addition, subtraction, and division. Further, they provided elegant constructs for complex functions such as tanh, linear gain, and exponentiation.[1] They used combinational logic to implement simple functions such

- *Peng Li, David J. Lilja, Marc D. Riedel, and Kia Bazargan are with the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, MN, USA, 55455. E-mail: see http://www.ece.umn.edu*
- *Weikang Qian is with University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai, China, 200241. E-mail: qianwk@sjtu.edu.cn.*

1. Such functions were of interest to the artificial neural networks community. The tanh function, in particular, performs a non-linear, sigmoidal mapping; this is used to model the activation function of a neuron.

as multiplication and scaled addition. They further used sequential logic in the form of linear finite-state machines (FSMs) to implement complex functions such as tanh. It was the first time that FSMs were used to construct sophisticated functions stochastically. Their contributions are significant: systems implemented using the FSM-based stochastic computational elements (SCEs) normally have better performance in terms of energy consumption, hardware cost, and fault-tolerance. However, many questions still need to be answered regarding FSM-based SCEs. For example, why does the FSM perform a tanh function stochastically based on the state-transition diagram shown in Fig. 1? Can we implement more functions stochastically using the FSM, and how can we design the corresponding state transition diagrams? Since Brown and Card's work had an empirical focus and complex constructs were validated by simulation only, we cannot obtain such answers directly from their prior work. As a result, the algorithms that can be implemented using the FSM-based SCEs are very limited. Most algorithms in digital image processing and artificial neural networks cannot benefit from this technique without additional research.

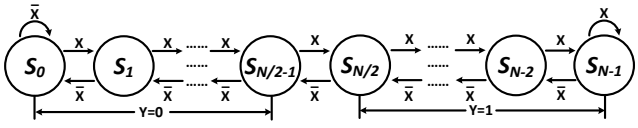


Fig. 1. State transition diagram of the FSM-based stochastic tanh function.

The first goal of this paper is to provide a rigorous mathematical treatment of the most intriguing class of functions, namely complex functions such as tanh and exponentiation implemented with linear FSMs. We present three fundamental properties of linear FSMs operating on stochastic bit streams. Based on these properties, we analyze and validate theoretically the constructs for tanh and exponentiation (Exp) proposed by Brown and Card [14]. Next we present stochastic constructs for two new functions, exponentiation based on an absolute value (ExpAbs) and absolute value (Abs), motivated by specific applications. We also provide detail analysis of the area, performance and error tolerance of all these constructs. The mathematical foundation presented in this paper make it possible to develop a general synthesis approach to construct arbitrary functions using the FSMs so that more algorithms can benefit from this technique [16], [17], [18], [19].

The remainder of this paper is organized as follows. Section 2 provides background information including the details of stochastic encoding and decoding, combinational logic for simple functions, and several FSM-based constructs proposed by Brown and Card [14]. Section 3 defines and proves three fundamental properties of linear FSMs operating on stochastic bit streams. Section 4 presents and analyzes four linear FSM-based constructs: tanh, Exp, ExpAbs, and Abs. Section 5 analyzes the sources of error in these four constructs. Section 6 presents experimental results for the cost, the performance, and the error-tolerance of both stochastic and deterministic implementations of the four constructs. Section 7 discusses future directions.

2 Background

2.1 Conversion Approach

In the stochastic paradigm, logical operations are performed on stochastic bit streams. Signal values are encoded by probabilities. To convert a deterministic value x_d ($x_d \in [a, b]$) into a stochastic bit stream X, we can generate a random number and compare it to x_d. The pseudocode for this operation is shown as follows,

$$
\begin{array}{l}
\hline
for\ (t = 0;\ t < L;\ t++)\{ \\
\quad if\ rand() < \frac{x_d - a}{b - a} \\
\quad\quad X(t) = 1; \\
\quad else \\
\quad\quad X(t) = 0;\} \\
\hline
\end{array}
$$

where L is the length of the stochastic bit stream X. The function $rand()$ is used to generate a random number in the range $[0, 1]$ based on a uniform distribution. The stochastic bit stream X generated by this code has the probability

$$P(X = 1) = \frac{x_d - a}{b - a}.$$

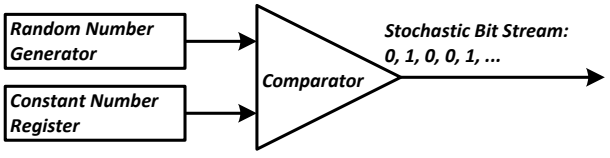


Fig. 2. The Randomizer Unit [11].

This conversion can be implemented with the circuit shown in Fig. 2, which is called a *Randomizer Unit* [11]. The *Random Number Generator* is implemented with a linear feedback shift register (LFSR). By setting the constant value of the *Constant Number Register*, we can generate a stochastic bit stream with a desired probability.

By counting the number of ones in a stochastic bit stream, we can convert the stochastic bit stream back to the corresponding deterministic value as

$$x'_d = a + \frac{sum(X)}{L} \cdot (b - a) \approx x_d.$$

The error between x'_d and x_d stems from quantization effects and random fluctuations [11], which are discussed further in Section 5 of this paper.

In fact there are two possible coding formats: a unipolar coding format and a bipolar coding format [1]. These two coding formats are the same in essence, and can coexist in a single system. In the unipolar coding format, a real number x in the unit interval (i.e., $0 \le x \le 1$) corresponds to a bit stream $X(t)$ of length L, where $t = 1, 2, ..., L$. The probability that each bit in the stream is one is $P(X = 1) = x$. For example, the value $x = 0.3$ would be represented by a random stream of bits such as 0100010100, where

approximately 30% of the bits are "1" and the remainder are "0." In the bipolar coding format, the range of a real number x is extended to $-1 \leq x \leq 1$. However, the probability that each bit in the stream is one is $P(X = 1) = \frac{x+1}{2}$. The trade-off between these two coding formats is that the bipolar format can deal with negative numbers directly while, given the same bit stream length, L, the precision of the unipolar format is twice that of the bipolar format.

2.2 Combinational Logic-based SCEs

Three basic arithmetic operations – scaled addition, scaled subtraction, and multiplication – can be implemented very simply with combinational logic in the stochastic paradigm [1], [14]. Scaled addition can be implemented with a multiplexer (MUX) for both the bipolar and the unipolar coding formats. Scaled subtraction can be implemented with a MUX and a NOT gate using the bipolar coding format. Multiplication can be implemented with a two-input AND gate using the unipolar coding format, and with a two-input XNOR gate using the bipolar coding format. We will briefly explain each of these constructs in this section.

2.2.1 Scaled Addition

This operation can be implemented with a multiplexer (MUX) as shown in Fig. 3 for both the bipolar and the unipolar coding formats. The function of the MUX is,

$$C = S \wedge A \vee \bar{S} \wedge B,$$

where $\wedge$ represents logical AND, $\vee$ represents logical OR, and $\bar{S}$ represents the negation of S.

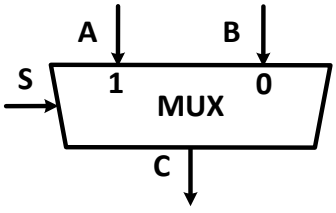


Fig. 3. The scaled addition implemented with a MUX.

If we assume A, B, C, and S are stochastic bit streams, and P_A, P_B, P_C, and P_S are their corresponding probabilities, we obtain:

$$P_C = P_S \cdot P_A + (1 - P_S) \cdot P_B. \quad (1)$$

For the unipolar coding format, the values represented by the stochastic bit streams A, B, and C are $a = P_A$, $b = P_B$, and $c = P_C$, respectively. For the bipolar coding format, the values represented by the stochastic bit streams A, B, and C are $a = 2P_A - 1$, $b = 2P_B - 1$, and $c = 2P_C - 1$, respectively. Based on (1), for both of the two coding formats, we have

$$c = P_S \cdot a + (1 - P_S) \cdot b. \quad (2)$$

In (2), we normally set $P_S = 0.5$ to perform unbiased scaled addition.

2.2.2 Scaled subtraction

This operation can be implemented with a MUX and a NOT gate as shown in Fig. 4. It works only for the bipolar coding format. The function of the circuit is

$$C = S \wedge A \vee \bar{S} \wedge \bar{B}.$$

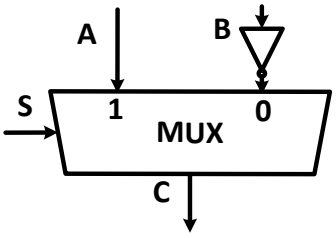


Fig. 4. The scaled subtraction implemented with a MUX and a NOT gate for the bipolar coding format.

If we assume A, B, C, and S are stochastic bit streams, and P_A, P_B, P_C, and P_S are their corresponding probabilities, we obtain:

$$P_C = P_S \cdot P_A + (1 - P_S) \cdot (1 - P_B). \quad (3)$$

We define a, b, and c as the corresponding values encoded by the stochastic bit streams A, B, and C using the bipolar coding format, respectively, i.e., $a = 2P_A - 1$, $b = 2P_B - 1$, and $c = 2P_C - 1$. Then equation (3) can be rewritten as

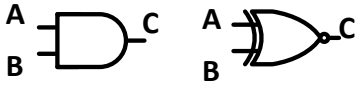
$$c = P_S \cdot a - (1 - P_S) \cdot b. \quad (4)$$

If we set $P_S = 0.5$, in equation (4) we have $c = 0.5 \cdot (a - b)$.

2.2.3 Multiplication

This operation can be implemented with a two-input AND gate as shown in Fig. 5(a) for the unipolar coding format, and with a two-input XNOR gate as shown in Fig. 5(b) for the bipolar coding format. The multiplication based on the AND gate for unipolar coding format is straightforward. We will explain the one based on the XNOR gate for the bipolar coding format as follows. In Fig. 5(b), we have

$$C = A \wedge B \vee \bar{A} \wedge \bar{B}. \quad (5)$$



(a) AND Gate (b) XNOR Gate

Fig. 5. Multiplication in stochastic computing.

If we define P_A, P_B, and P_C as the probabilities of the streams A, B, and C, respectively, then based on the Boolean function of the XNOR gate, we have

$$P_C = P_A \cdot P_B + (1 - P_A) \cdot (1 - P_B). \quad (6)$$

If we define a, b, and c as the corresponding values encoded by the streams A, B, and C using the bipolar coding format, respectively (i.e., $a = 2P_A - 1$, $b = 2P_B - 1$, and $c = 2P_C - 1$), we can rewrite (6) as

$$\frac{c+1}{2} = \frac{a+1}{2} \cdot \frac{b+1}{2} + (1 - \frac{a+1}{2}) \cdot (1 - \frac{b+1}{2}). \quad (7)$$

Simplifying equation (7), we have $c = a \cdot b$.

2.3 FSM-based SCEs

Combinational logic can only implement polynomial functions of a specific form – namely those that map the unit interval to the unit interval [10]. Non-polynomial functions can be approximated by combinational logic, for instance with MacLaurin expansions [9]. However, highly nonlinear functions such as exponentiation and tanh cannot be approximated effectively with this approach. As discussed in Section 2.2.1, the limitation stems from the fact combinational logic can only implement scaled addition in the stochastic paradigm. The implementation of polynomials with coefficients not in the unit interval is sometimes not possible and is generally not straightforward [11].

Gaines [1] described the use of an ADaptive DIgital Element (ADDIE) for generation of arbitrary functions. The ADDIE is based on a saturating counter, that is, a counter which will not increment beyond its maximum state value or decrement below its minimum state value. In the ADDIE, the state of the counter is controlled in a closed loop fashion. The problem is that ADDIE requires that the output of the counter to be converted into a stochastic bit stream in order to implement the closed loop feedback [1]. This is potentially inefficient and hardware intensive.

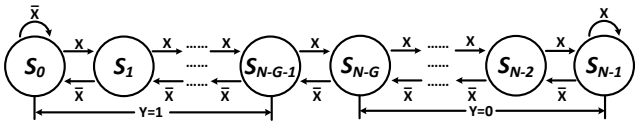


Fig. 6. State transition diagram of the FSM-based stochastic exponentiation function.

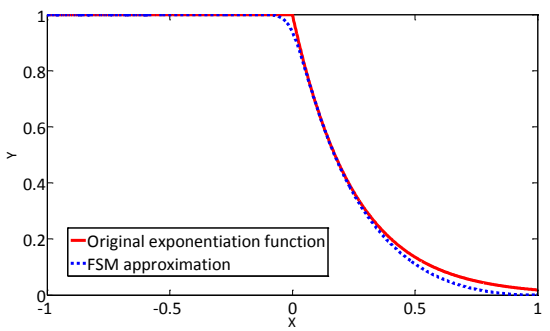


Fig. 7. Simulation result of the FSM-based stochastic exponentiation function.

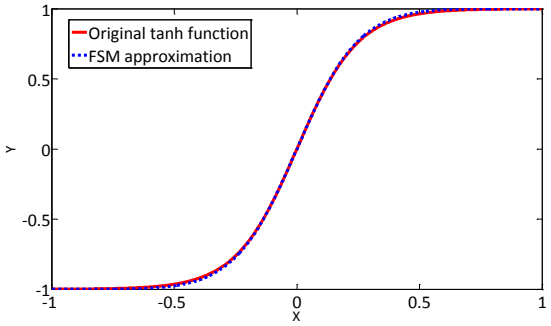


Fig. 8. Simulation result of the FSM-based stochastic tanh function.

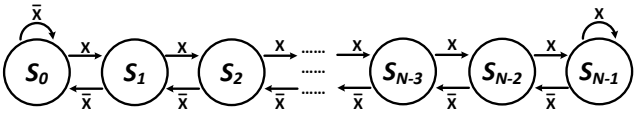


Fig. 9. A generic linear state transition diagram.

In 2001, Brown and Card [14] presented two FSM-based constructs. The first one is called the stochastic exponentiation (SExp) function, with the state transition diagram shown in Fig. 6. This configuration approximates an exponentiation function stochastically as follows,

$$y \approx \begin{cases} e^{-2Gx}, & 0 \le x \le 1, \\ 1, & -1 \le x < 0, \end{cases} \quad (8)$$

where x is the bipolar encoding of the input bit stream X and y is the unipolar encoding of the output bit stream Y. The simulation result based on $N = 16, G = 2$ is shown in Fig. 7.

The second one is called the stochastic tanh (STanh) function. We show the state transition diagram in Fig. 1. This configuration approximates a tanh function stochastically as follow,

$$y \approx \frac{e^{\frac{N}{2}x} - e^{-\frac{N}{2}x}}{e^{\frac{N}{2}x} + e^{-\frac{N}{2}x}}, \quad (9)$$

where x is the bipolar encoding of the input bit stream X and y is also the bipolar encoding of the output bit stream Y. The simulation result based on $N = 8$ is shown in Fig. 8.

Note that both of these FSM-based constructs use the linear state transition pattern shown in Fig. 9. This linear FSM is similar to Gaines' ADDIE. The difference is that this linear FSM does not use a closed loop [1], [14]; accordingly this construct is much more efficient. In the next section, we will introduce three fundamental properties of linear FSM-based stochastic computing elements.

3 Properties of the Linear FSM

The state machine shown in Fig. 9 contains a set of states, $S_0, S_1, \cdots, S_{N-1}$, arranged in a linear form (i.e., a saturating counter) [14]. It has a total of N states, where

N is a positive integer. We usually set $N = 2^K$, where K is also a positive integer, since the maximum number of states can be implemented by K D-Flip-Flops (DFFs) is 2^K. *X* is the input of this state machine. The output *Y* (not shown in Fig. 9) of this state machine is only determined by the current state. We assume that the input *X* is a Bernoulli sequence (i.e., a stochastic bit stream).

The system can be modeled as a time-homogeneous irreducible and aperiodic Markov chain and will have one single stable hyperstate [1]. We define the probability that each bit in the input stream *X* is one to be P_X, the probability that each bit in the corresponding output stream *Y* is one to be P_Y, and in the steady state the probability that the current state is $S_i (0 \leq i \leq N-1)$ under the input probability P_X to be $P_{i(P_X)}$. The individual state probability $P_{i(P_X)}$ in the hyperstate must sum to unity, and the probability of transitioning from state S_{i-1} to state S_i must equal the probability of transitioning from state S_i to state S_{i-1}. Thus, we have

$$P_{i(P_X)} \cdot (1 - P_X) = P_{i-1(P_X)} \cdot P_X, \tag{10}$$

$$\sum_{i=0}^{N-1} P_{i(P_X)} = 1, \tag{11}$$

$$P_Y = \sum_{i=0}^{N-1} s_i \cdot P_{i(P_X)}, \tag{12}$$

where s_i in (12) only has two choices of values, 0 or 1, and specifies the output Y of the system when the current state is S_i, i.e., if the current state is S_i, then the output $Y = s_i$. Based on (10) and (11), $P_{i(P_X)}$ can be computed as follows,

$$P_{i(P_X)} = \frac{(\frac{P_X}{1-P_X})^i}{\sum_{j=0}^{N-1} (\frac{P_X}{1-P_X})^j}. \tag{13}$$

Furthermore, based on different intervals of P_X, $P_{i(P_X)}$ can also be computed as follows,

$$P_{i(P_X)} = \begin{cases} \frac{(\frac{P_X}{1-P_X})^i \cdot (\frac{1-2P_X}{1-P_X})}{1-(\frac{P_X}{1-P_X})^N}, & 0 \leq P_X < 0.5, \\ \frac{1}{N}, & P_X = 0.5, \\ \frac{(\frac{1-P_X}{P_X})^{N-1-i} \cdot (\frac{2P_X-1}{P_X})}{1-(\frac{1-P_X}{P_X})^N}, & 0.5 < P_X \leq 1. \end{cases} \tag{14}$$

If we define

$$t_{(P_X)} = \frac{0.5 - |P_X - 0.5|}{0.5 + |P_X - 0.5|}, \tag{15}$$

equation (14) can be rewritten as follows,

$$P_{i(P_X)} = \begin{cases} \frac{t^i_{(P_X)} \cdot (1 - t_{(P_X)})}{1 - t^N_{(P_X)}}, & 0 \leq P_X < 0.5, \\ \frac{1}{N}, & P_X = 0.5, \\ \frac{t^{N-1-i}_{(P_X)} \cdot (1 - t_{(P_X)})}{1 - t^N_{(P_X)}}, & 0.5 < P_X \leq 1. \end{cases} \tag{16}$$

In this section, we introduce three fundamental properties of the linear FSMs used in stochastic computing. These properties can be proved using the above equation.

3.1 Property 1

$P_{i(P_X)}$ and $P_{N-1-i(P_X)}$ are symmetric about $P_X = 0.5$. In other words, $P_{i(P_X)} = P_{N-1-i(1-P_X)}$.

3.2 Property 2

As $N \to \infty$ (i.e., N is "large enough"),

- P_Y will be mainly determined by the configuration of the states from S_0 to $S_{N/2-1}$ when $P_X \in [0, 0.5)$;
- P_Y will be mainly determined by the configuration of the states from $S_{N/2}$ to S_{N-1} when $P_X \in (0.5, 1]$.

In other words, we can rewrite (12) as follows,

$$P_Y \begin{cases} \approx \sum_{i=0}^{N/2-1} s_i \cdot P_{i(P_X)}, & 0 \leq P_X < 0.5, \\ = \sum_{i=0}^{N-1} \frac{s_i}{N}, & P_X = 0.5, \\ \approx \sum_{i=N/2}^{N-1} s_i \cdot P_{i(P_X)}, & 0.5 < P_X \leq 1. \end{cases}$$

3.3 Property 3

For the configuration

$$P_Y = \sum_{i=0}^{N-1} s_i \cdot P_{i(P_X)},$$

- if we set $s_i = s_{N-1-i}$, for i = 0, 1, $\cdots$, $\frac{N}{2} - 1$, P_Y will be symmetric about the line $P_X = 0.5$;
- if we set $s_i = 1 - s_{N-1-i}$, for i = 0, 1, $\cdots$, $\frac{N}{2} - 1$, P_Y will be symmetric about the point $(P_X, P_Y) = (0.5, 0.5)$.

In other words,

- if $s_i = s_{N-1-i}$, for $i = 0, 1, \cdots, \frac{N}{2}-1$, $P_{Y(P_X)} = P_{Y(1-P_X)}$;
- if $s_i = 1 - s_{N-1-i}$, for $i = 0, 1, \cdots, \frac{N}{2}-1$, $P_{Y(P_X)} = 1 - P_{Y(1-P_X)}$.

Based on these properties, we present four linear FSM-based constructs for computing complex functions in the next section.

4 The Linear FSM-based SCEs

In this section, we present and analyze FSM-based stochastic computation elements (SCEs) for the STanh function, the SExp function, the stochastic ExpAbs (SExpAbs) function, and the stochastic Abs (SAbs) function. The constructs for the STanh and SExp functions were presented by Brown and Card [14]; they provided no proof of correctness, only empirical validation. The constructs for the SExpAbs and SAbs functions are new. We prove the correctness of all four constructs.

4.1 Stochastic Tanh Function (STanh)

Brown and Card [14] proposed the STanh function to implement sigmoid nonlinear mapping with stochastic computing. They set s_i in (12) as follows,

$$s_i = \begin{cases} 0, & 0 \le i \le \frac{N}{2}-1, \\ 1, & \frac{N}{2} \le i \le N-1. \end{cases} \tag{17}$$

This configuration is shown in Fig. 1(a), and equation (9) is the corresponding approximate transfer function. We will prove (9) based on the configuration in (17) as follows.

Proof: Based on the configuration in (17), we have

$$P_Y = \sum_{i=N/2}^{N-1} P_{i(P_X)}. \tag{18}$$

If we substitute $P_{i(P_X)}$ in (18) with (13), we have

$$\begin{aligned} P_Y &= \frac{\sum_{i=N/2}^{N-1} (\frac{P_X}{1-P_X})^i}{\sum_{k=0}^{N-1} (\frac{P_X}{1-P_X})^k} = \frac{(\frac{P_X}{1-P_X})^{\frac{N}{2}} - (\frac{P_X}{1-P_X})^N}{1-(\frac{P_X}{1-P_X})^N} \\ &= \frac{(\frac{P_X}{1-P_X})^{\frac{N}{2}} \cdot (1-(\frac{P_X}{1-P_X})^{\frac{N}{2}})}{(1+(\frac{P_X}{1-P_X})^{\frac{N}{2}}) \cdot (1-(\frac{P_X}{1-P_X})^{\frac{N}{2}})} \\ &= \frac{(\frac{P_X}{1-P_X})^{\frac{N}{2}}}{1+(\frac{P_X}{1-P_X})^{\frac{N}{2}}}. \end{aligned} \tag{19}$$

If we substitute P_X and P_Y in (19) with their corresponding bipolar coding format x and y, we have

$$\frac{y+1}{2} = \frac{(\frac{1+x}{1-x})^{\frac{N}{2}}}{1+(\frac{1+x}{1-x})^{\frac{N}{2}}}. \tag{20}$$

Simplifying (20), we have,

$$y = \frac{(\frac{1+x}{1-x})^{\frac{N}{2}} - 1}{(\frac{1+x}{1-x})^{\frac{N}{2}} + 1}. \tag{21}$$

By using Taylor's expansion, we have

$$1 + x \approx e^x,$$
$$1 - x \approx e^{-x}.$$

Thus, we can rewrite (21) as follows,

$$y = \frac{(e^{2x})^{\frac{N}{2}} - 1}{(e^{2x})^{\frac{N}{2}} + 1} = \frac{e^{\frac{N}{2}x} - e^{-\frac{N}{2}x}}{e^{\frac{N}{2}x} + e^{-\frac{N}{2}x}}. \quad \blacksquare$$

4.2 Stochastic Exponentiation Function (SExp)

The SExp function is configured by setting the parameters s_i in (12) as follows,

$$s_i = \begin{cases} 1, & 0 \le i \le N-G-1, \\ 0, & N-G \le i \le N-1, \end{cases} \tag{22}$$

where G is a positive integer and $G \ll N$. We have shown this configuration in Fig. 6(a), and equation (8) is the corresponding approximate transfer function. We will prove (8) as follows.

Proof: Based on the configuration in (22), we have

$$P_Y = \sum_{i=0}^{N-G-1} P_{i(P_X)}. \tag{23}$$

If we substitute $P_{i(P_X)}$ in (23) with (16), we have

$$P_Y = \begin{cases} \sum_{i=0}^{N-G-1} \frac{t_{(P_X)}^i \cdot (1-t_{(P_X)})}{1-t_{(P_X)}^N}, & 0 \le P_X < 0.5, \\ \frac{N-G}{N}, & P_X = 0.5, \\ \sum_{i=0}^{N-G-1} \frac{t_{(P_X)}^{N-1-i} \cdot (1-t_{(P_X)})}{1-t_{(P_X)}^N}, & 0.5 < P_X \le 1. \end{cases} \tag{24}$$

Based on Property 2, and because $G \ll N$, we can rewrite (24) as follows,

$$P_Y \approx \begin{cases} \sum_{i=0}^{N/2-1} \frac{t_{(P_X)}^i \cdot (1-t_{(P_X)})}{1-t_{(P_X)}^N}, & 0 < P_X \le 0.5, \\ 1, & P_X = 0.5, \\ \sum_{i=N/2}^{N-G-1} \frac{t_{(P_X)}^{N-1-i} \cdot (1-t_{(P_X)})}{1-t_{(P_X)}^N}, & 0.5 < P_X \le 1. \end{cases}$$

(1). When $P_X < 0.5$, $t(P_X) = \frac{P_X}{1-P_X} < 1$. When N is large enough, we have

$$\sum_{i=0}^{N/2-1} \frac{t_{(P_X)}^i \cdot (1-t_{(P_X)})}{1-t_{(P_X)}^N} = \frac{1-t_{(P_X)}^{N/2}}{1-t_{(P_X)}^N} \approx 1.$$

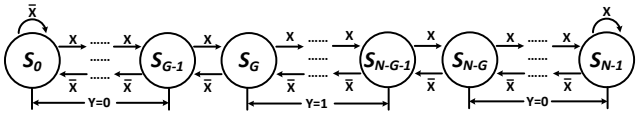


Fig. 10. State transition diagram of the SExpAbs function.

(2). When $P_X > 0.5$, $t(P_X) = \frac{1-P_X}{P_X}$. When N is large enough, we have

$$\sum_{i=N/2}^{N-G-1} \frac{t_{(P_X)}^{N-1-i} \cdot (1 - t_{(P_X)})}{1 - t_{(P_X)}^{N}} = \frac{t_{(P_X)}^{G} - t_{(P_X)}^{N/2}}{1 - t_{(P_X)}^{N}} \approx t_{(P_X)}^{G}.$$

Because $x = 2P_X - 1$ and $P_X > 0.5$, we can rewrite $t_{(P_X)}$ in (15) as follows,

$$t_{(P_X)} = \frac{1 - |x|}{1 + |x|} = \frac{1 - x}{1 + x}.$$

By using Taylor's expansion, we have

$$1 + x \approx e^{x}, \quad 1 - x \approx e^{-x},$$

Thus,

$$t_{(P_X)} = \frac{1 - x}{1 + x} \approx \frac{e^{-x}}{e^{x}} = e^{-2x},$$

and

$$t_{(P_X)}^{G} \approx e^{-2Gx}. \quad \blacksquare$$

4.3 SExp with an Absolute Value (SExpAbs)

This SCE can be implemented by setting s_i as follows,

$$s_i = \begin{cases} 1, & G \le i \le N - G - 1, \\ 0, & i < G \text{ or } i > N - G - 1. \end{cases} \tag{25}$$

We also show this configuration in Fig. 10. The approximate transfer function in this configuration is

$$P_Y = e^{-2G|x|}, \tag{26}$$

where x is the bipolar coding of P_X, i.e., $x = 2P_X - 1$.

Proof: (1). When $P_X > 0.5$ (i.e., $x > 0$), the configuration from $s_{N/2}$ to s_{N-1} is the same as in (22). Based on Property 2, the configuration in (25) has the same approximate function as in (22) when $P_X > 0.5$. Thus, we have

$$P_Y = e^{-2Gx}. \tag{27}$$

(2). When $P_X < 0.5$ (i.e., $x < 0$), because $s_i = s_{N-1-i}$, based on Property 3, P_Y is symmetric about $P_X = 0.5$, i.e., P_Y is symmetric about $x = 0$. Thus, we have

$$P_Y = e^{2Gx}. \tag{28}$$

As a result, based on (27) and (28), we obtain (26). Note that, since $G \ll N$, when $P_X = 0.5$ (i.e., $x = 0$), based on (16), $P_Y = (N - 2G + 1)/N \approx 1$. $\blacksquare$

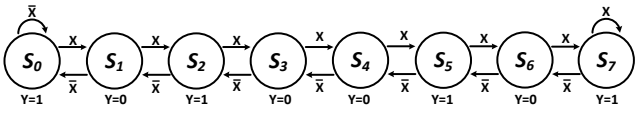


Fig. 11. State transition diagram of the SAbs function.

4.4 Stochastic Absolute Value Function (SAbs)

This SCE can be implemented by setting s_i as follows,

- When $0 \le i \le N/2 - 1$,

$$s_i = \begin{cases} 1, & i \text{ is even,} \\ 0, & i \text{ is odd;} \end{cases} \tag{29}$$

- When $N/2 \le i \le N - 1$,

$$s_i = \begin{cases} 1, & i \text{ is odd,} \\ 0, & i \text{ is even.} \end{cases} \tag{30}$$

We also show the configuration based on a 8-state FSM (i.e., $N = 8$) in Fig. 11. If we define x as the bipolar coding of P_X and y as the bipolar coding of P_Y. The approximate transfer function in this configuration is

$$y = |x|. \tag{31}$$

Proof: Based on the configuration in (29) and (30), we have

$$P_Y = \sum_{i=0}^{N/4-1} P_{2i(P_X)} + \sum_{i=N/4}^{N/2-1} P_{(2i+1)(P_X)}. \tag{32}$$

(1). When $P_X < 0.5$, based on Property 2, we have

$$P_Y \approx \sum_{i=0}^{N/4-1} P_{2i(P_X)}. \tag{33}$$

If we substitute $P_{2i(P_X)}$ in (33) with (16), we have

$$\begin{aligned} \sum_{i=0}^{N/4-1} P_{2i(P_X)} &= \sum_{i=0}^{N/4-1} \frac{t_{(P_X)}^{2i} \cdot (1 - t_{(P_X)})}{1 - t_{(P_X)}^{N}} \\ &= \frac{1 - t_{(P_X)}^{N/2}}{1 - t_{(P_X)}^{N}} \cdot \frac{1 - t_{(P_X)}}{1 - t_{(P_X)}^{2}} \\ &\approx \frac{1 - t_{(P_X)}}{1 - t_{(P_X)}^{2}} = \frac{1}{1 + t_{(P_X)}}. \end{aligned} \tag{34}$$

If we substitute $t_{(P_X)}$ in (34) with (15), we have

$$\frac{1}{1 + t_{(P_X)}} = 1 - P_X.$$

i.e., $P_Y \approx 1 - P_X$. Thus,

$$y = 2P_Y - 1 = 2(1 - P_X) - 1 = -x.$$

(2). When $P_X > 0.5$, because $s_i = s_{N-1-i}$, based on Property 3, P_Y is symmetric about $P_X = 0.5$, i.e., $P_Y \approx 1 - (1 - P_X) = P_X$. Thus,

$$y = 2P_Y - 1 = 2P_X - 1 = x.$$

(3). When $P_X = 0.5$, based on (14), $P_{i(P_X)} = \frac{1}{N}$. If we substitute $P_{i(P_X)}$ in (32) with $\frac{1}{N}$, we have

$$P_Y = \sum_{i=0}^{N/4-1} \frac{1}{N} + \sum_{i=N/4}^{N/2-1} \frac{1}{N} = 0.5,$$

i.e., $y = 0$.

As a result, we have

$$y = \begin{cases} -x, & -1 \le x < 0, \\ 0, & x = 0, \\ x, & 0 < x \le 1, \end{cases}$$

i.e., $y = |x|$. ■

5 Error Analysis

By its nature, the paradigm of computing on stochastic bit streams introduces errors and uncertainty. As discussed in [11], there are three primary sources of errors:

1) the error due to the approximation of the desired function (e_a),
2) the quantization error (e_q), and
3) the error due to random fluctuations (e_r).

We analyze each of these three errors for the FSM-based SCEs in this section.

5.1 Error Due to the Function Approximation

Based on the analysis from Section 4, it can be seen that the approximation error of the linear FSM-based SCEs compared to the desired functions depends on the number of states. The more states we use to implement the FSM, the less approximation error we have. Brown and Card [14] showed simulation results with 8, 16, and 32-state in linear FSMs for the STanh function and the SExp function. Here we show the simulation results using different numbers of states for the two new FSM-based SCEs, namely SExpAbs and SAbs.

The approximation results of the SExpAbs function based on 8, 16, 32, and 64-state linear FSMs are shown in Fig. 12. It can be seen that the main distortion is located at $x = 0$ (i.e., $P_X = 0.5$). By using more states, this distortion can be reduced.

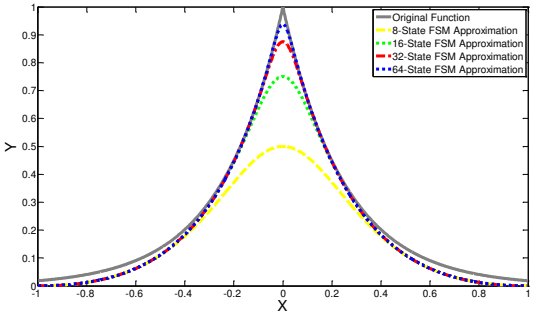


Fig. 12. Simulation results of the SExpAbs function ($G = 2$ in (26)).

The approximation results of the SAbs function based on 8, 16, 32, and 64-state linear FSMs are shown in Fig. 13. It can be seen that the main distortion is again located at $x = 0$ (i.e., $P_X = 0.5$). By using more states, this distortion again can be reduced.

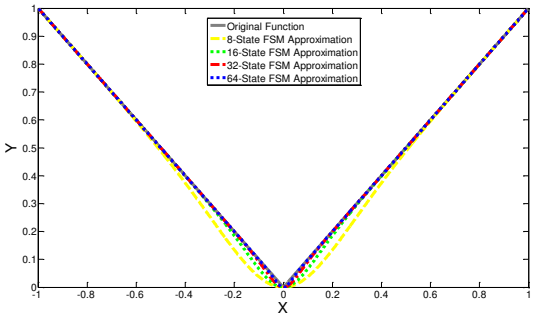


Fig. 13. Simulation results of the SAbs function.

We list the approximation errors for each of the FSM-based SCEs versus different numbers of states in Table 1. The approximation error is computed using a standard least squares (2-norm) distance:

$$e_a = \int_0^1 (T(P_X) - \sum_{i=0}^{N-1} s_i \cdot P_{i(P_X)})^2 \cdot dP_X,$$

where $T(P_X)$ is the original function, and $\sum_{i=0}^{N-1} s_i \cdot P_{i(P_X)}$ is the corresponding FSM-based approximation.

TABLE 1
Approximation errors (e_a) of the four FSM-based SCEs versus different numbers of states.

	Number of States			
The FSM-based SCEs	**8**	**16**	**32**	**64**
STanh	0.00%	0.00%	0.00%	0.00%
SExp	0.38%	0.07%	0.03%	0.02%
SExpAbs	1.48%	0.22%	0.07%	0.05%
SAbs	0.03%	0.00%	0.00%	0.00%

For many fault-tolerant applications, 10^{-3} would be considered an acceptable error rate [11]. As a result, in our subsequent experiments we choose the 8-state FSM to approximate the STanh function, the 16-state FSM to approximate the SExp function, the 32-state FSM to approximate the SExpAbs function, and the 8-state FSM to approximate the SAbs function.

5.2 Quantization Error

To analyze the quantization error, we follow the approach of Qian et al. [11]. In stochastic computing, we round an arbitrary value p $(0 \le p \le 1)$ to the closest number p' in the set of discrete probabilities we can generate $S = \{0, \frac{1}{M}, \cdots, 1\}$, where M is the maximal integer value that can be generated by a random number generator shown in

Fig. 2. Let L be the length of the stochastic bit stream. In order to supply good randomness, we need to choose $M \leq L$ [11]. Thus, the quantization error, e_q, for p is

$$e_q = |p - p'| \leq \frac{1}{2M} \leq \frac{1}{2L}.$$

Due to the effect of quantization, we actually compute

$$\sum_{i=0}^{N-1} s_i \cdot P_{i(P'_X)},$$

instead of the FSM-based approximation

$$\sum_{i=0}^{N-1} s_i \cdot P_{i(P_X)},$$

where P'_X is the closest number to P_X in set S. Thus, the quantization error is

$$e_q = |\sum_{i=0}^{N-1} s_i \cdot P_{i(P'_X)} - \sum_{i=0}^{N-1} s_i \cdot P_{i(P_X)}|.$$

We define $\Delta P_X = P'_X - P_X$, and using a first order approximation, the error due to quantization is

$$e_q \approx |\sum_{i=0}^{N-1} s_i \cdot \frac{dP_{i(P_X)}}{dP_X} \Delta P_X|. \quad (35)$$

Computing (35) is complex, and the result depends on s_i, i.e., the configuration of the FSM. Because $\sum_{i=0}^{N-1} s_i \cdot P_{i(P_X)}$ approximates the desired function $T(P_X)$ closely, we can approximate e_q by using $T(P_X)$:

$$e_q \approx |\frac{dT(P_X)}{dP_X} \Delta P_X|.$$

By applying the upper bound for both $|\frac{dT(P_X)}{dP_X}|$ and ΔP_X, we can derive the maximum quantization error for each of the four FSM-based SCES, which is listed in Table 2.

TABLE 2
The maximum quantization errors (e_q) of the four FSM-based SCEs.

The FSM-based SCEs	Quantization error e_q
STanh	$N/2L$
SExp	$4G/L$
SExpAbs	$4G/L$
SAbs	$1/L$

From Table 2, it can be seen that increasing L will reduce the quantization error, which is consistent with our intuition.

5.3 Error Due to Random Fluctuations

The output bit stream $Y(\gamma)$ ($\gamma = 1, 2, \cdots, L$, where L is the length of the stochastic bit streams) of the FSM-based SCEs has probability

$$p' = \sum_{i=0}^{N-1} s_i \cdot P_{i(P'_X)},$$

that each bit is one. By using a counter, we can translate this stochastic bit stream $Y(\gamma)$ to a deterministic value y, where

$$y = \frac{1}{L} \sum_{\gamma=1}^{L} Y(\gamma).$$

It is easily seen that the expected value of y is $E[y] = p'$. However, the realization of y is not, in general, exactly equal to p' due to random fluctuations in the bit streams. The error can be measured by the variance as

$$Var[y] = Var\left[\frac{1}{L}\sum_{\gamma=1}^{L} Y(\gamma)\right] = \frac{1}{L^2}\sum_{\gamma=1}^{L} Var[Y(\gamma)] = \frac{p'(1-p')}{L}.$$

Since $Var[y] = E[(y - E[y])^2] = E[(y - p')^2]$, the error due to random fluctuation is

$$e_r = |y - p'| \approx \sqrt{\frac{p'(1-p')}{L}}.$$

Thus, the error due to random fluctuations is inversely proportional to $\sqrt{L}$, and increasing the length of the stochastic bit stream will reduce this error.

5.4 Summary of Error Analysis

The overall error, e, is bounded by the sum of the aforementioned three error components, i.e.,

$$e = e_a + e_q + e_r.$$

We evaluate each of the four FSM-based SCEs for different lengths of stochastic bit streams on 16 points: 1/16, 2/16, 3/16, $\cdots$, 15/16, 1. For each length, we repeat the simulation 1000 times and average the errors over all simulations. The final result is shown in Table 3. Note that for each SCE, we use the minimum number of states required to make the approximation error e_a less than 10^{-3}. Compared to the results from Table 1, we conclude that the overall error is mainly due to quantization and random fluctuations.

6 Experimental Evaluation

In our experiments, we first consider an image edge detection algorithm as a case study demonstrating how one might use the proposed SAbs function in practical applications (a similar implementation can be found in Li and Lilja [13]). Other SCEs, such as the STanh and SExp functions, had been introduced by Brown and Card [15] for implementing

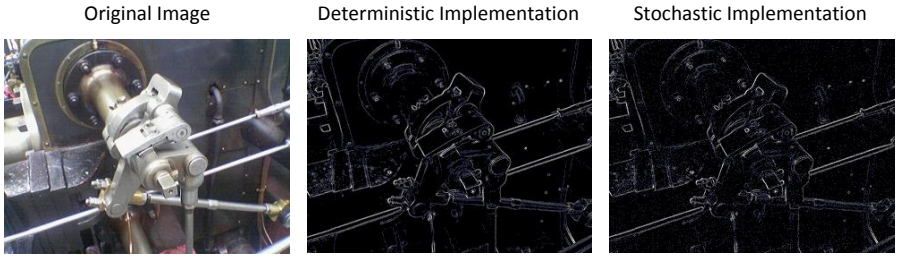


Fig. 14. Comparison of the deterministic and stochastic implementations of the image edge detection algorithm. The leftmost image is the original image, i.e., the input for both the deterministic and stochastic implementations of the algorithm. The middle one is the output of the deterministic implementation. The rightmost one is the output of the stochastic implementation. It can be seen that the difference between the two outputs is trivial.

TABLE 3
The average overall errors (e) of the four FSM-based SCEs versus different lengths of stochastic bit streams.

	Length of Stochastic Bit Streams (L)			
The FSM-based SCEs	**256**	**512**	**1024**	**2048**
STanh (8-state)	3.71%	2.54%	1.73%	1.26%
SExp (16-state)	3.97%	2.39%	1.95%	1.88%
SExpAbs (32-state)	11.10%	6.44%	4.02%	3.22%
SAbs (8-state)	2.65%	2.00%	1.63%	1.42%

a soft competitive learning algorithm in artificial neural networks. The SExpAbs function had been introduced by Li and Lilja [12] for implementing a kernel density estimation-based image segmentation algorithm.

In addition, we compare the stochastic implementations to deterministic implementations of the four functions in terms of hardware area and fault tolerance. Lastly, we discuss real-time processing and latency issues of stochastic computing for image processing applications.

6.1 Image Edge Detection Case Study

Image edge detection is a fundamental tool in digital image processing and computer vision, which aims at identifying points in a digital image at which the image brightness has discontinuities [20]. The simplest approach is to use central differences:

$$N(x,y) = \frac{1}{2}\left(|M(x+1,y) - M(x-1,y)|\right) + \frac{1}{2}\left(|M(x,y+1) - M(x,y-1)|\right), \quad (36)$$

where $M(x,y)$ is the pixel value at location (x,y) of the original image and $N(x,y)$ is the pixel value at location (x,y) of the processed image. To implement Equation (36) stochastically, we need to use one scaled addition (refer to Section 2.2.1), two scaled subtractions (refer to Section 2.2.2), and two SAbs functions (refer to Section 4.4). The circuit of the stochastic implementation of Equation (36) is shown in Fig. 15, in which $P_{M_{x+1,y}}$ is the probability

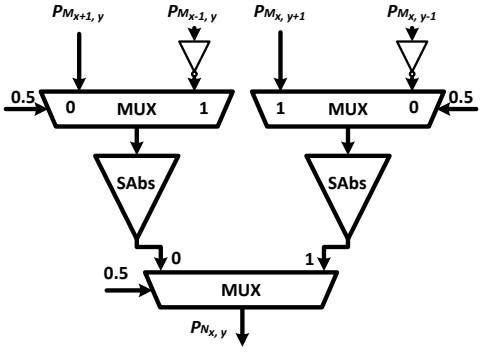


Fig. 15. The stochastic implementation of an image edge detection algorithm based on Equation (36).

of ones in the stochastic bit stream which is converted from the pixel value $M(x+1,y)$, i.e., $P_{M_{x+1,y}} = \frac{M(x+1,y)}{256}$. The probabilities $P_{M_{x-1,y}}$, $P_{M_{x,y-1}}$, and $P_{M_{x,y+1}}$ are converted similarly. Based on this circuit, we have

$$P_{N_{x,y}} = \frac{1}{4}\left|P_{M_{x+1,y}} - P_{M_{x-1,y}}\right| + \frac{1}{4}\left|P_{M_{x,y+1}} - P_{M_{x,y-1}}\right| + \frac{1}{2} = \frac{N(x,y)}{512} + \frac{1}{2}.$$

Thus, by counting the number of ones in the output bit stream, we can convert it back to $N(x,y)$. The simulation results of this algorithm based on both the deterministic implementation and the stochastic implementation are shown in Fig. 14. It can be seen that the one using the stochastic implementation has almost the same result as the one using the deterministic implementation.

6.2 Comparison with Binary Radix

6.2.1 Hardware Area Comparison

For a comparison with a conventional binary radix implementation, we need to evaluate the cost of conventional adders and multipliers. Of course, there are a wide variety of conventional implementation types, tailored for many

TABLE 4
The area and delay of the stochastic implementations and the deterministic implementations of the four functions. Delay values of the stochastic implementations stand for the overall delay. Critical path delay of the stochastic implementations can be obtained by dividing 2^M.

	Stochastic Implementation			Deterministic Implementation
Functions	Area	Delay	Area-Delay Product	Area-Delay Product
Tanh	35	3×2^M	105×2^M	$(10M^2 - 4M - 9) \cdot (12M - 11) \cdot 6$
Exp	75	4×2^M	300×2^M	$(10M^2 - 4M - 9) \cdot (12M - 11) \cdot 5$
ExpAbs	126	4×2^M	504×2^M	$(10M^2 - 4M - 9) \cdot (12M - 11) \cdot 5 + 9M^2$
Abs	40	3×2^M	120×2^M	$9M^2$

TABLE 5
Relative errors of the stochastic implementations and the deterministic implementations of the four functions versus different error ratios ϵ in the input data.

	Tanh		**Exp**		**ExpAbs**		**Abs**	
	Rel. Error of		Rel. Error of		Rel. Error of		Rel. Error of	
Error Ratio ϵ	Stoch. Impl. (%)	Deter. Impl. (%)	Stoch. Impl. (%)	Deter. Impl. (%)	Stoch. Impl. (%)	Deter. Impl. (%)	Stoch. Impl. (%)	Deter. Impl. (%)
0	1.73	0.00	1.95	0.00	4.02	0.00	1.63	0.00
0.001	1.93	0.52	2.17	0.35	4.21	0.67	1.65	0.43
0.002	2.21	1.23	2.25	0.79	4.27	1.58	1.69	0.62
0.005	2.23	2.42	2.33	2.11	4.32	3.19	1.73	1.62
0.01	2.26	5.36	2.37	3.47	4.39	4.89	1.92	4.10
0.02	2.35	9.36	2.42	7.43	4.46	8.32	2.36	6.01
0.05	2.69	19.68	2.46	14.50	5.06	15.78	3.91	15.44
0.1	3.83	34.95	2.97	25.36	7.21	26.83	6.55	29.56

different purposes. Some of them require very little area but have high latency. Others require huge area, but offer better performance [21]. We evaluate the hardware cost using the area-delay product [11]. Using this metric, the performance of nearly all multiplier and adder circuits are similar. Sophisticated adders and multipliers, such as Kogge-Stone adders and Wallace Tree multipliers, were developed for additions and multiplications on wide datapaths. Such architectures are not well suited for computations on narrow datapaths. For accuracies of $M \leq 10$, where M is the number of bits used to represent a numerical value in binary radix, the overhead of such sophisticated structures outweighs their benefits.

For our studies we chose an M-bit multiplier based on the logic design of the ISCAS'85 circuit C6288, given in the benchmark as 16 bits [22]. We use this circuit because its structure is regular. The C6288 is built with *carry-save adders*. It consists of 240 full- and half-adder cells arranged in a 15×16 matrix. Each full adder is realized by 9 NOR gates. Incorporating the M-bit multiplier and optimizing it, the circuit requires $10M^2 - 4M - 9$ gates; these are inverters, fanin-2 AND gates, fanin-2 OR gates, and fanin-2 NOR gates. The critical path of the circuit passes through $12M - 11$ logic gates [9].

We use the Maclaurin polynomial [9] approximation for the deterministic implementations of the four functions. We need a polynomial of degree 6 to approximate the tanh function, and a polynomial of degree 5 to approximate the exponentiation function (and the one with absolute value) to achieve the same level of approximation errors as those of the corresponding stochastic implementations shown in Table 1. The corresponding Maclaurin polynomials are computed using adders and multipliers. We show the corresponding area-delay products of the deterministic implementations of the four functions in Table 4. To evaluate the area-delay product of the circuit which computes absolute value, we assume it has the same area-delay product of a full adder (the area is $9M$ gates, and the delay is M gates).

As we mentioned in the previous section, we implement the four FSM-based SCEs with the minimum number of states to make the approximation error less than 10^{-3}, i.e., we implement the STanh function with an 8-state FSM, the SExp function with a 16-state FSM, the SExpAbs function with a 32-state FSM, and the SAbs function with an 8-state FSM. The corresponding configurations have been introduced in Section 4. Table 4 also shows the area-delay products of the stochastic implementations of the four functions. Each circuit of the stochastic implementations is composed of the seven basic types of logic gates: inverters, fanin-2 AND gates, fanin-2 NAND gates, fanin-2 OR gates, fanin-2 NOR gates, fanin-2 XOR gates, and fanin-2 XNOR gates. The DFF is implemented with 6 fanin-2 NAND gates. When characterizing the area and delay, we assume that the operation of each fanin-2 logic gate requires unit area and unit delay. Note that if we assume M is the number of bits used to represent a numerical value in a binary radix, in order to get the same resolution for computation on stochastic bit streams, we need a 2^M-bit stream to represent

the same value.

From Table 4, we can see that, except the SAbs function, the area-delay product of the stochastic implementation is less than that of the deterministic implementation when $M \leq 10$. Although the stochastic implementation of the SAbs function has larger area-delay product than the corresponding deterministic implementation, this SCE is a necessary component for some applications, such as the stochastic implementation of an edge detection algorithm in digital image processing, and the overall hardware cost is still less than that of the deterministic implementation [13].

In fact, both Qian et al. [11] and Brown et al. [14] had shown that, when $M \leq 10$, computation on stochastic bit streams has better performance than those based on binary radix using adders and multipliers in terms of area-delay product. Furthermore, since the energy consumed by computation is generally proportional to the product of the circuit area and the computation delay, the stochastic implementation also has an advantage in energy consumption. Note that stochastic computation is usually applied to those applications that do not require a high resolution, typically with an M in the range from 8 to 10. For example, in image processing applications [11], [13], M is normally set to 8; in artificial neural network applications [14], M is normally set to 10. Thus, for most applications of stochastic computation, the stochastic implementation has an advantage both in area-delay product and energy consumption [11], [14]. In addition, computing on stochastic bit streams offers tunable precision—as the length of the stochastic bit stream increases, the precision of the value represented by it also increases. Thus, without hardware redesign, we have the flexibility to trade-off precision and computation time.

6.2.2 Fault-Tolerance Comparison

To make comparisons on fault-tolerance, we compare the performance of the deterministic implementations versus the stochastic implementations of the four functions when the input data is corrupted with noise. Suppose that the input data of a deterministic implementation is represented using $M = 10$ bits. In order to achieve the same resolution, the bit stream of a stochastic implementation contains $2^M = 1024$ bits. We choose the error ratio ϵ of the input data to be 0, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, and 0.1, as measured by the fraction of random bit flips that occur.

We evaluated each of the four functions on 16 points: 1/16, 2/16, 3/16, $\cdots$, 15/16, 1. For each error ratio ϵ, each function, and each evaluation point, we simulated both the stochastic and the deterministic implementations 1000 times. We averaged the relative errors over all simulations. Finally, for each error ratio ϵ, we averaged the relative errors over all evaluation points. Table 5 shows the average relative error of the stochastic implementations and the deterministic implementations for the four functions versus different error ratios ϵ. We average the relative errors of the four functions, and plot the results in Fig. 16 to give a clear comparison.

When $\epsilon = 0$, meaning that no noise is injected into the input data, the deterministic implementation computes without any error. However, due to the inherent variance, the stochastic implementation produces a small relative error [11]. When $\epsilon > 0$, the relative error of the deterministic implementation blows up dramatically as ϵ increases. Even for small values, the stochastic implementation performs much better.

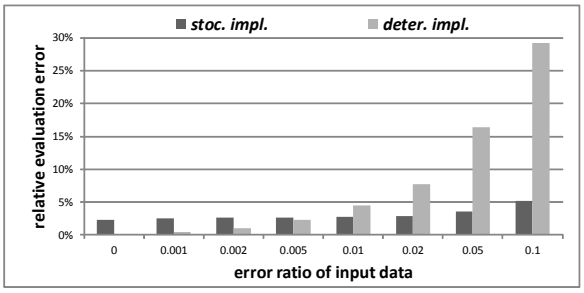


Fig. 16. The average of the relative errors of the four functions shown in Table 5.

It is not surprising that the deterministic implementation is so sensitive to errors, given that the representation used is binary radix. In a noisy environment, bit flips affect all the bits with equal probability. In the worst case, the most significant bit gets flipped, resulting in an error of $2^{M-1}/2^M = 1/2$ on the input value. In contrast, in a stochastic implementation, the data is represented as the fractional weight on a bit stream of length 2^M. Thus, a single bit flip only changes the input value by $1/2^M$, which is minuscule in comparison [11].

6.3 Real-Time Processing and Latency Analysis

A potential drawback of logical computation on stochastic bit streams is the long latency. In this section, we discuss the latency issues and strategies for mitigating them. We focus on stochastic real-time implementations of image processing functions.

For practical image processing applications with real-time constraints, the frequency of the clock is a critical parameter. In a deterministic implementation, assuming that one pixel can be processed per clock cycle, the required clock frequency to meet the real-time constraint is: $f_c \geq$ frame rate $\times$ image size. The corresponding stochastic implementation requires a frequency of: $f_s \geq$ frame rate $\times$ image size $\times$ L, where L is the number of bits used to represent a pixel value stochastically. Assuming each pixel is represented using M bits in binary radix, then we require $L = 2^M$. The latency is normally defined as the duration between the time a pixel is input into the system and the time the result is produced. Thus the latency of the deterministic implementation is $1/f_c$, and the latency of the the stochastic implementation is L/f_s.

Assuming the video sequence is grayscale QCIF format (176$\times$144 pixels), the frame rate is 30 frame per second (fps), and $L = 2^8 = 256$. Based on the above discussion, we have $f_s \approx 30 \times 176 \times 144 \times 256$ = 200MHz.

Thus, running the stochastic implementation with a clock frequency of 200MHz will meet the real-time constraint, and the latency is 256/200 = 1.28μs. These results show that for image processing applications, the real-time processing and latency issues of the stochastic implementations can be solved by running them at a higher operational frequency than a conventional deterministic implementation. This is because their circuits are extremely simple, and have shorter critical paths. More specifically, the critical path of the circuit shown in Fig. 15 is a NOT gate plus a two-input multiplexer. For those high resolution and high frame rate image processing applications, although we cannot increase the clock of the stochastic implementations without bound, the latency issue can be solved by parallel processing. In this approach, multiple pixels can be processed stochastically at the same time by duplicating the circuit of the stochastic implementations in parallel. This approach trades off silicon area with the number of clock cycles.

7 Discussions and Conclusions

The stochastic computing paradigm offers a novel view of digital computation: instead of transforming definite inputs into definite outputs, circuits transform probability values into probability values; so, conceptually, real-valued probabilities are both the inputs and the outputs. The computation has a pseudo *analog* character, reminiscent of computations performed by physical systems such as electronics on continuously variable signals such as voltage. Here the variable signal is the probability of obtaining a one versus a zero in a stochastic yet digital bit stream. The circuits can be built from ordinary digital electronics such as CMOS. And yet they computed complex, continuous-valued transfer functions.

Prior work has shown constructs for a variety of interesting functions. Most intriguing among these are the complex functions produced by linear finite-state machines: exponentiation, tanh, and absolute value. Prior work has focused on particular application domains, ranging from neural networks to signal processing. The stochastic constructs were demonstrated and validated empirically. This paper provides the mathematical foundation for the analysis and synthesis of such circuits.

Because a stochastic representation is uniform, with all bits weighted equally, it is highly tolerant of soft errors (i.e., bit flips). Computation on stochastic bit streams offers tunable precision: as the length of the stochastic bit stream increases, the precision of the value represented by it also increases. Thus, without hardware redesign, one has the flexibility to trade off precision and computation time. In contrast, with a conventional binary-radix implementation, when a higher precision is required, the underlying hardware must be redesigned.

A significant drawback of the paradigm is the long latency of the computations. The accuracy depends on the length of the bit streams; with long bit streams, each operation requires many clock cycles to complete. However, potentially the operations could be performed at a much faster clock rate, mitigating the latency issue. This is because operations can be implemented with extremely simple hardware in the stochastic paradigm. Note that the circuits that we have shown all have very short critical paths. In future work, we will perform a detailed circuit-level timing analysis of all the stochastic constructs. In addition, parallel processing can be also used to solve this issue, especially for stochastic implementations of digital image processing applications and artificial neural network applications.

The accuracy of the computation also depends on the quality of the randomness. If the stochastic bit streams are not statistically independent, the accuracy will drop. Furthermore, *correlation* is an issue in any circuit that has feedback or reconvergent paths. If the circuit has multiple outputs, these will have correlated probability values. In future work, we will study how to design circuits with multiple outputs – and so correlations in space. Also, we will study the impact of feedback – and so correlations in time. Also, in future work we will study the *dynamic* behavior of stochastic constructs. We have observed that, using bit streams of length L to represent the inputs values, the output values of FSM-based stochastic constructs are always correct and stable after L clock cycles, no matter what the initial state. We will justify this claim mathematically.

Acknowledgments

This work was supported in part by an National Science Foundation (NSF) Career Award, NO. 0845650 and NSF grant NO. CCF-1241987. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. Portions of this work were presented in the 22nd IEEE International Conference on Application – specific Systems, Architectures and Processors [12], and in the 29th IEEE International Conference on Computer Design [13].

References

[1] B. R. Gaines, "Stochastic computing systems," *Advances in Information System Science, Plenum*, vol. 2, no. 2, pp. 37–172, 1969.

[2] J. Keane and L. Atlas, "Impulses and stochastic arithmetic for signal processing," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, vol. 2, pp. 1257–1260, IEEE, 2001.

[3] V. Gaudet and A. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, vol. 39, no. 3, pp. 299–301, 2003.

[4] W. Gross, V. Gaudet, and A. Milner, "Stochastic implementation of ldpc decoders," in *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on*, pp. 713–717, IEEE, 2005.

[5] D. McNeill and H. Card, "Refractory pulse counting processes in stochastic neural computers," *Neural Networks, IEEE Transactions on*, vol. 16, no. 2, pp. 505–508, 2005.

[6] H. Li, D. Zhang, and S. Foo, "A stochastic digital implementation of a neural network controller for small wind turbine systems," *Power Electronics, IEEE Transactions on*, vol. 21, no. 5, pp. 1502–1507, 2006.

[7] M. Hori and M. Ueda, "Fpga implementation of a blind source separation system based on stochastic computing," in *Soft Computing in Industrial Applications, 2008. SMCia'08. IEEE Conference on*, pp. 182–187, IEEE, 2008.

[8] T. Onomi, T. Kondo, and K. Nakajima, "Implementation of high-speed single flux-quantum up/down counter for the neural computationusing stochastic logic," *Applied Superconductivity, IEEE Transactions on*, vol. 19, no. 3, pp. 626–629, 2009.

[9] W. Qian and M. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *45th ACM/IEEE Design Automation Conference, DAC'08*, pp. 648–653, 2008.
[10] W. Qian, M. D. Riedel, and I. Rosenberg, "Uniform approximation and Bernstein polynomials with coefficients in the unit interval," *European Journal of Combinatorics*, vol. 32, pp. 448–463, 2011.
[11] W. Qian, X. Li, M. Riedel, K. Bazargan, and D. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, pp. 93–105, January 2010.
[12] P. Li and D. J. Lilja, "A low power fault-tolerance architecture for the kernel density estimation based image segmentation algorithm," in *IEEE International Conference on Application - specific Systems, Architectures and Processors, ASAP'11*, 2011.
[13] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *29th IEEE International Conference on Computer Design, ICCD'11*, 2011.
[14] B. D. Brown and H. C. Card, "Stochastic neural computation I: Computational elements," *IEEE Transactions on Computers*, vol. 50, pp. 891–905, September 2001.
[15] B. D. Brown and H. C. Card, "Stochastic neural computation II: Soft competitive learning," *IEEE Transactions on Computers*, vol. 50, pp. 906–920, September 2001.
[16] P. Li, W. Qian, M. Riedel, K. Bazargan, and D. Lilja, "The synthesis of linear finite state machine-based stochastic computational elements," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pp. 757 –762, jan. 2012.
[17] P. Li, D. J. Lilja, W. Qian, and K. Bazargan, "Using a two-dimensional finite-state machine for stochastic computation," in *International Workshop on Logic and Synthesis, IWLS'12*, 2012.
[18] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," in *Computer-Aided Design, 2012. ICCAD 2012. IEEE/ACM International Conference on*, IEEE, 2012.
[19] P. Li, W. Qian, and D. J. Lilja, "A stochastic reconfigurable architecture for fault-tolerant computation with sequential logic," in *Computer Design (ICCD), 2011 IEEE 30th International Conference on*, IEEE, 2012.
[20] R. C. Gonzalez and R. E. Woods, "Digital image processing, 3rd edition," *Prentice Hall*, 2008.
[21] B. Parhami, *Computer arithmetic*. Oxford university press, 2000.
[22] "ISCAS'85 C6288 16×16 multiplier," in *http://www.eecs.umich.edu/~jhayes/iscas/c6288.html*.

PLACE PHOTO HERE

Peng Li received the BSci degree in Electrical Engineering from North China Electric Power University, Beijing, in 2005; the MEng degree from the Institute of Microelectronics, Tsinghua University, Beijing, in 2008. He is currently pursuing the Ph.D. degree in the Electrical and Computer Engineering Department, University of Minnesota Twin Cities, Minneapolis, MN. His main research interests are in the area of VLSI design for digital signal processing, stochastic computing, and high performance storage architecture.

PLACE PHOTO HERE

David J. Lilja received the BS degree in Computer Engineering from Iowa State University in Ames, and the MS and PhD degrees in Electrical Engineering from the University of Illinois at Urbana-Champaign. He is currently a Professor and Head of Electrical and Computer Engineering at the University of Minnesota in Minneapolis. He has worked as a Development Engineer at Tandem Computers Inc. and has chaired and served on the program committees of numerous conferences. He was elected a Fellow of the IEEE and the AAAS. His main research interests include computer architecture, computer systems performance analysis, and high-performance storage systems, with a particular interest in the interaction of computer architecture with software, compilers, and circuits.

PLACE PHOTO HERE

Weikang Qian received his Ph.D. degree in Electrical Engineering at the University of Minnesota in 2011 and his B.Eng. degree in Automation at Tsinghua University, Beijing, China in 2006. He has been an Assistant Professor at the University of Michigan-Shanghai Jiao Tong University Joint Institute since 2011. He has research interests in the fields of computer-aided design of integrated circuits, circuit design for emerging technologies, and fault-tolerant computing. He is a member of the IEEE.

PLACE PHOTO HERE

Marc D. Riedel received the BEng degree in Electrical Engineering with a minor in Mathematics from McGill University, and the MSc and PhD degrees in Electrical Engineering from Caltech. He is currently an Associate Professor of Electrical and Computer Engineering at the University of Minnesota. He is also a member of the Graduate Faculty in Biomedical Informatics and Computational Biology. From 2004 to 2005, he was a Lecturer in Computation and Neural Systems at Caltech. He has held positions at Marconi Canada, CAE Electronics, Toshiba, and Fujitsu Research Labs. His PhD dissertation titled "Cyclic Combinational Circuits" received the Charl H. Wilts Prize for the Best Doctoral Research in Electrical Engineering at Caltech. His paper "The Synthesis of Cyclic Combinational Circuits" received the Best Paper Award at the Design Automation Conference. He is a recipient of the US National Science Foundation (NSF) CAREER Award. He is a Senior Member of the IEEE and the IEEE Computer Society.

PLACE PHOTO HERE

Kia Bazargan received the BSci degree in Computer Science from Sharif University, Tehran, Iran, and the MS and PhD degrees in Electrical and Computer engineering from Northwestern University, Evanston, Illinois, in 1998 and 2000, respectively. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis. He was a guest coeditor of the ACM Transactions on Embedded Computing Systems Special Issue on Dynamically Adaptable Embedded Systems in 2003. He has served on the technical program committee of a number of IEEE/ACM-sponsored conferences (e.g., Field Programmable Gate Array (FPGA), Field Programmable Logic (FPL), Design Automation Conference (DAC), International Conference on Computer-Aided Design (ICCAD), and Asia and South Pacific DAC). He is an associate editor of the IEEE Transactions on CAD of Integrated Circuits and Systems. He was a recipient of the US National Science Foundation (NSF) Career Award in 2004. He is a Senior Member of the IEEE and the IEEE Computer Society.