Open access • Proceedings Article • DOI:10.1109/ASPDAC.2012.6165056

The synthesis of linear Finite State Machine-based Stochastic Computational Elements

Peng Li, Weikang Qian, Marc D. Riedel, Kia Bazargan ...+1 more authors

Institutions: University of Minnesota, Shanghai Jiao Tong University

Published on: 09 Mar 2012 - Asia and South Pacific Design Automation Conference

Topics: Stochastic computing, Finite-state machine, Combinational logic, Stochastic process and Exponentiation

The Synthesis of Linear Finite State Machine-Based Stochastic Computational Elements

Peng Li†, Weikang Qian‡, Marc D. Riedel†, Kia Bazargan†, and David J. Lilja†
†Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, USA
{lipeng, mriedel, kia, lilja}@umn.edu
‡University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai, China
qianwk@sjtu.edu.cn

Abstract— The Stochastic Computational Element (SCE) uses streams of random bits (stochastic bits streams) to perform computation with conventional digital logic gates. It can guarantee reliable computation using unreliable devices. In stochastic computing, the linear Finite State Machine (FSM) can be used to implement some sophisticated functions, such as the exponentiation and tanh functions, more efficiently than combinational logic. However, a general approach about how to synthesize a linear FSM-based SCE for a target function has not been available. In this paper, we will introduce three properties of the linear FSM used in stochastic computing and demonstrate a general approach to synthesize a linear FSM-based SCE for a target function. Experimental results show that our approach produces circuits that are *much more tolerant of soft errors* **than deterministic implementations, while the area-delay product of the circuits are less than that of deterministic implementations.**

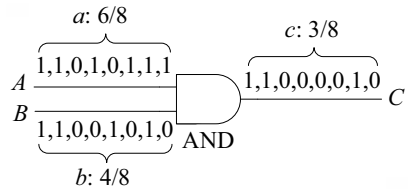
I. Introduction

Future integrated circuits are expected to be more sensitive to noise and variations [1]. Stochastic computing, which uses conventional digital logic to perform computing based on stochastic bit streams, has been known in the literature for many years [2]. It can gracefully tolerate very large numbers of errors at lower cost than conventional over-design techniques while maintaining equivalent performance.

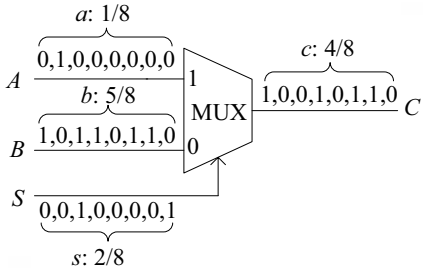
In stochastic computing, computation in the deterministic Boolean domain is transformed into probabilistic computation in the real domain. Such computation is based on a *stochastic representation* of data. Gaines [2] proposed two types of stochastic representation: a unipolar coding format and a bipolar coding format. 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., $0 \leq x \leq 1$) corresponds to a sequence of random bits, each of which has probability x of being one and probability $1 - x$ of being zero. If a stochastic bit stream of length N has k ones, then the real value represented by the bit stream is $\frac{k}{N}$. In the bipolar coding format, the range of a real number x is extended to $-1 \leq x \leq 1$. The probability that each bit in the stream is one is $P(X = 1) = (x + 1)/2$. Thus, a real number $x = -1$ is represented by a stream of all zeros and a real number $x = 0$ is represented by a stream of bits that have probability 0.5 of being one. If a stochastic bit stream of length N has k ones, then the real value represented by the bit stream is $2\frac{k}{N} - 1$.

With stochastic representation, some basic arithmetic operations can be very simply implemented by combinational logic. For example, as shown in Fig. 1(a), with the unipolar coding format, multiplication can be implemented with an AND gate. Assuming that the two input stochastic bit streams A and B are independent, the number represented by the output stochastic bit stream C is $c = a \cdot b$. So the AND gate multiplies the two values represented by the stochastic bit streams.

We can also implement scaled addition with a multiplexer, as shown in Fig. 1(b) [1]. With the assumption that the three input stochastic bit streams A, B, and S are independent, the number represented by the output stochastic bit stream C is $c = s \cdot a + (1 - s) \cdot b$. Thus, with the unipolar coding format, the computation performed by a multiplexer is the scaled addition of the two input values a and b, with a scaling factor of s for a and $1 - s$ for b.



(a) Multiplication with the unipolar coding. Here the inputs are $6/8$ and $4/8$. The output is $6/8 \times 4/8 = 3/8$, as expected.



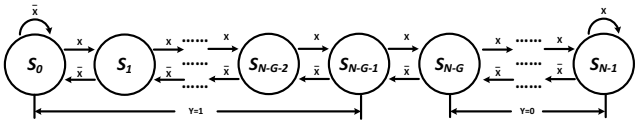
(b) Scaled addition with the unipolar coding. Here the inputs are $1/8$, $5/8$, and $2/8$. The output is $2/8 \times 1/8 + (1 - 2/8) \times 5/8 = 4/8$, as expected.

Fig. 1. Stochastic implementation of arithmetic operations.

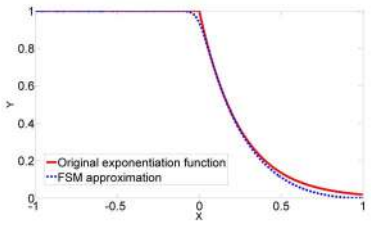
[1] It is not feasible to add two probability values directly; this could result in a value greater than one, which cannot be represented as a probability value.

978-1-4673-0772-7/12/$31.00 ©2012 IEEE

The main issue of the combinational logic-based SCEs is that they cannot be efficiently used to implement sophisticated computations, such as the exponentiation and tanh functions [3]. Gaines described the use of an ADaptive DIgital Element (ADDIE) for generation of arbitrary functions in 1967 [2]. 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, however, the state of the counter is controlled in a closed loop fashion. The problem is that ADDIE requires that the output of the counter is converted into a stochastic pulse stream in order to implement the closed loop feedback [2]. This will make the system inefficient and require substantial amounts of hardware.

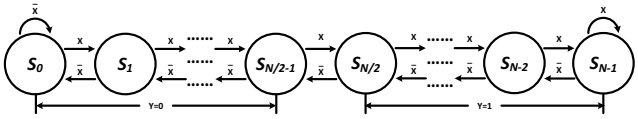


(a) State transition diagram.

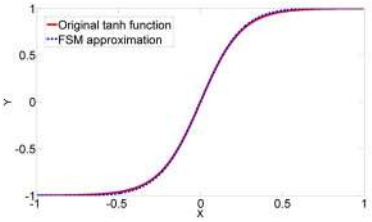


(b) Approximation result.

Fig. 2. The FSM-based stochastic exponential function.



(a) State transition diagram.



(b) Approximation result.

Fig. 3. The FSM-based stochastic tanh function.

In 2001, Brown and Card [3] presented two FSM-based SCEs. The first one is called a stochastic exponentiation function, we show the state transition diagram in Fig. 2(a). This configuration approximates an exponentiation function stochastically as follow,

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

where x is the bipolar coding of the input bit stream X and y is the unipolar coding of the output bit stream Y. The approximation result based on $N = 16, G = 2$ is shown in Fig. 2(b).

The second one is called a stochastic tanh function. We show the state transition diagram in Fig. 3(a). This configuration approximates a tanh function stochastically as follow,

$$y \approx \frac{e^{\frac{N}{2}x} - e^{-\frac{N}{2}x}}{e^{\frac{N}{2}x} + e^{-\frac{N}{2}x}}, \tag{2}$$

where x is the bipolar coding of the input bit stream X and y is also the bipolar coding of the output bit stream Y. The approximation result based on $N = 8$ is shown in Fig. 3(b).

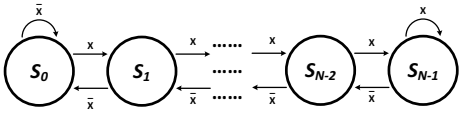


Fig. 4. A generic linear state transition diagram.

We notice that both of these two FSM-based SCEs use variations of the linear state transition pattern shown in Fig. 4. The linear FSM is similar to Gaines' ADDIE, which can be also viewed as being based on a saturating counter. The difference is that the linear FSM did not use closed loop, which makes this new FSM more efficient. Based on the aforementioned two examples, it can be seen that the linear FSM can be used to implement sophisticated computations which are difficult to implement with combinational logic. However, the stochastic exponentiation and tanh functions are developed based on an empirical approach. A general approach about how to synthesize a target function based on the linear FSM is still unknown.

To take advantage of the linear FSM in stochastic computing, it is necessary to find a general approach to synthesize a target function based on it. In this paper, we will introduce the basic properties of the linear FSM used in stochastic computing and demonstrate a general approach to synthesize the linear FSM-based SCEs. The remainder of this paper is organized as follows. Section II introduces the properties of the linear FSM. Section III proposes the general approach to synthesize the linear FSM-based SCEs. Section IV presents experimental results on both the cost and the error-tolerance of the stochastic and deterministic implementations of different target functions. Conclusions are drawn in Section V.

II. Properties of the Linear FSM Used in Stochastic Computing

The basic form of the state machine shown in Fig. 4 is a set of states $S_0 \rightarrow S_{N-1}$ arranged in a linear form (e.g. like a saturating counter) [3]. It has totally $N = 2^K$ states, where *K* is a positive integer. *X* is the input of this state machine. The output *Y* (not shown in Fig. 4) of this state machine is determined only by the current state. Assume the input *X* is a stochastic bit stream, and it is a Bernoulli sequence. The system will be ergodic and will have one single stable hyperstate [2]. 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 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. Additionally, in the steady state, 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,

$$P_{i(P_X)} \cdot (1 - P_X) = P_{i-1(P_X)} \cdot P_X, \tag{3}$$

$$\sum_{i=0}^{N-1} P_{i(P_X)} = 1, \tag{4}$$

$$P_Y = \sum_{i=0}^{N-1} s_i \cdot P_{i(P_X)}. \tag{5}$$

where s_i only has two values, 0 or 1 (this denotes the configuration of the states that control the output, for example, setting $s_i = 1$ denotes $Y = 1$ if the current state is S_1).

Based on (3) and (4), $P_{i(P_X)}$ can be computed as follows,

$$P_{i(P_X)} = \frac{(\frac{P_X}{1-P_X})^i}{\sum_{j=0}^{N-1} (\frac{P_X}{1-P_X})^j}. \tag{6}$$

Based on (6), we obtain three properties for the linear FSM shown in Fig. 4.

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

Property 2 If N is large enough, for example $N \geq 8$,

- 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 (5) as follows,

$$P_Y \begin{cases} \approx \sum_{i=0}^{N/2-1} s_i \cdot P_{i(P_X)}, & 0 \leq P_X < 0.5, \\ = \sum_{i=0}^{N-1} \frac{s_i}{N}, & P_X = 0.5, \\ \approx \sum_{i=N/2}^{N-1} s_i \cdot P_{i(P_X)}, & 0.5 < P_X \leq 1. \end{cases} \tag{7}$$

♦

Property 3 For the configuration

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

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

In other words,

- if $s_i = s_{N-1-i}$, $P_Y(P_X) = P_Y(1 - P_X)$;
- if $s_i = 1 - s_{N-1-i}$, $P_Y(P_X) = 1 - P_Y(1 - P_X)$. ♦

The three properties can be proved based on (6). Additionally, the two linear FSM-based SCEs introduced by Brown and Card [3] can be proved based on these three properties. Due to space limitations, we omit the proofs here.

III. A General Approach to Synthesize the Linear FSM-based SCEs

In this section, we will introduce a general approach to synthesize a target function based on the linear FSM.

A. Circuit Implementation of the Linear FSM

Before we introduce the synthesis approach, we first present another linear state transition diagram shown in Fig. 5. This transition diagram has the same state transition pattern as the one shown in Fig. 4. However, the parameters s_i assigned to each state in Fig. 5 are different from the one defined in (5). In (5), we only consider two deterministic values for s_i, i.e., 0 and 1. Here in Fig. 5, s_i stands for a stochastic bit stream, in which we define the probability that each bit is one to be P_{s_i}. Thus, we rewrite (5) as follows,

$$P_Y = \sum_{i=0}^{N-1} P_{s_i} \cdot P_{i(P_X)}. \tag{8}$$

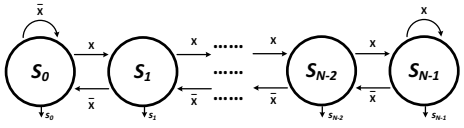


Fig. 5. A linear state transition diagram with input parameters assigned to each state.

The un-optimized circuit implementation of this generalized state transition diagram is shown in Fig. 6.

- The *Combinational Logic* block and the D-Flip-Flops are used to implement the state transitions shown in Fig. 5.
- The *K to N Decoder* is used to decode the current state from the outputs of the D-Flip-Flops. Note that at each clock cycle, only one state will be valid, i.e., only one of the outputs of the *Decoder* will be '1' at each clock cycle, and all the others will be '0'.
- The AND gate is used to perform $P_{s_i} \cdot P_{i(P_X)}$ in (8), and the OR gate performs the summation in (8) because all of its inputs are independent of each other.

Note that all the three properties introduced in Section II will still hold if we consider s_i as the stochastic bit stream. Additionally, if s_i equals '0' or '1', the *K to N Decoder*, the *AND* gates, and the *OR* gate can be substantially simplified.

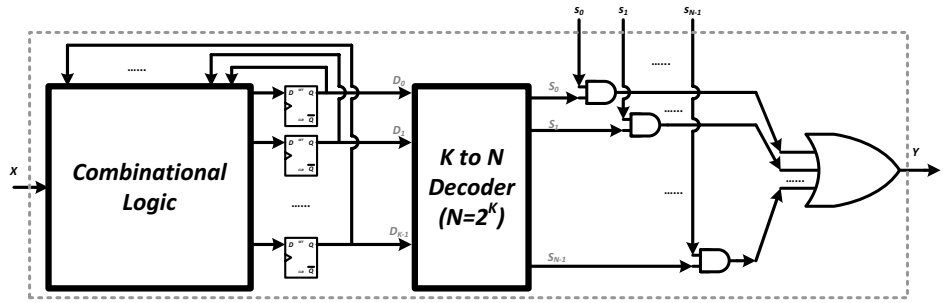


Fig. 6. The circuit implementation of the linear FSM-based stochastic computational elements.

B. The General Synthesis Approach

Now, we will introduce the general approach to synthesize the linear FSM-based SCEs. Assume the target function is $T(P_X)$, and $P_X \in [0,1]$ and $T(P_X) \in [0,1]$. Our goal is to find a set of coefficients P_{s_0}, P_{s_1}, ..., $P_{s_{N-1}}$ $(0 \leq P_{s_i} \leq 1)$ in (8) to minimize the objective function

$$\int_0^1 (T(P_X) - \sum_{i=0}^{N-1} P_{s_i} \cdot P_{i(P_X)})^2 \cdot dP_X, \tag{9}$$

where $P_{i(P_X)}$ is the linear FSM basis functions introduced in (6). By expanding (9), an equivalent objective function can be obtained:

$$T(\mathbf{b}) = \frac{1}{2}\mathbf{b}^T\mathbf{H}\mathbf{b} + \mathbf{c}^T\mathbf{b}, \tag{10}$$

where

$$\mathbf{b} = [P_{s_0}, P_{s_1}, ..., P_{s_{N-1}}]^T,$$

$$\mathbf{c} = \begin{bmatrix} -\int_0^1 T(P_X)P_{0(P_X)}dP_X \\ \vdots \\ -\int_0^1 T(P_X)P_{N-1(P_X)}dP_X \end{bmatrix},$$

$$\mathbf{H} = \begin{bmatrix} \int_0^1 P_{0(P_X)}P_{0(P_X)}dP_X & \cdots & \int_0^1 P_{0(P_X)}P_{N-1(P_X)}dP_X \\ \int_0^1 P_{1(P_X)}P_{0(P_X)}dP_X & \cdots & \int_0^1 P_{1(P_X)}P_{N-1(P_X)}dP_X \\ \vdots & \ddots & \vdots \\ \int_0^1 P_{N-1(P_X)}P_{0(P_X)}dP_X & \cdots & \int_0^1 P_{N-1(P_X)}P_{N-1(P_X)}dP_X \end{bmatrix}.$$

This optimization problem, in fact, is a typical constrained quadratic programming problem. Its solution can be obtained using standard techniques [4]. Once we obtain the coefficients P_{s_i}, we can implement the target function with the circuit shown in Fig. 6.

Example 1. Here we use the same example presented by Qian et al. [4] (See **Example 1** in [4]). The target function is a polynomial with a degree of three,

$$T(P_X) = \frac{1}{4} + \frac{9}{8}P_X - \frac{15}{8}P_X^2 + \frac{5}{4}P_X^3.$$

Based on the aforementioned synthesis approach, we find that a 4-state linear FSM can be used to synthesize this function with an approximation error less than 10^{-3}. The corresponding parameters are shown in Table I. ■

TABLE I
PARAMETERS ASSIGNED FOR EACH STATE FOR THE TARGET FUNCTION IN **Example 1**.

States	S_0	S_1	S_2	S_3
Parameters	0.274	1	0	0.726

Example 2. Synthesize the stochastic exponentiation function proposed by Brown and Card [3] based on $G = 2$.

To synthesize this function, we first need to rewrite the original target function (1) in terms of P_X. Since $G = 2$, we have

$$T(P_X) = \begin{cases} 1, & 0 \leq P_X \leq 0.5, \\ e^{-4(2P_X-1)}, & 0.5 \leq P_X \leq 1. \end{cases}$$

Based on the proposed synthesis approach, we find that a 16-state linear FSM can be used to synthesize this function with an approximation error less than 10^{-3}. The corresponding parameters are shown in Table II. It can be seen that the results are the same as the ones proposed by Brown and Card [3]. ■

TABLE II
PARAMETERS ASSIGNED FOR EACH STATE FOR THE TARGET FUNCTION IN **Example 2**.

States	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Parameters	1	1	1	1	1	1	1	1
States	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
Parameters	1	1	1	1	1	1	0	0

Example 3. Synthesize the stochastic tanh function proposed by Brown and Card [3] based on $N = 8$.

To synthesize this function, we first need to rewrite the original target function (1) in terms of P_X. Since $N = 8$, we have

$$T(P_X) = \frac{e^{8(2P_X-1)}}{e^{8(2P_X-1)} + 1}.$$

Based on our proposed synthesis approach, we find that a 8-state linear FSM can be used to synthesize this function with an approximation error less than 10^{-3}. The corresponding parameters are shown in Table III. It can be seen that the results are the same as the ones proposed by Brown and Card [3]. ■

TABLE III
PARAMETERS ASSIGNED FOR EACH STATE FOR THE TARGET FUNCTION IN **Example 3**.

States	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Parameters	0	0	0	0	1	1	1	1

IV. EXPERIMENTAL RESULTS

In our experiments, we first compare the hardware cost of deterministic digital implementations to that of stochastic implementations for the three examples we introduced in the last section. Then we compare the performance of these two implementations of the corresponding target functions on noisy input data. Finally, we compare our FSM-based synthesis approach to the Bernstein polynomial-based synthesis approach introduced by Qian et al. [4].

A. Hardware Comparison

In a deterministic implementation of polynomial arithmetic, a polynomial $T(P_X) = \sum_{i=0}^{n} a_i P_X^i$ can be factorized as $T(P_X) = a_0 + P_X(a_1 + P_X(a_2 + \cdots + P_X(a_{n-1} + P_X a_n)))$. With such a factorization, we can evaluate the polynomial in n iterations. In each iteration, a single addition and a single multiplication are needed. Hence, for such an iterative calculation, the hardware consists of an adder and a multiplier.

We build the M-bit multiplier based on the logic design of the ISCAS'85 circuit C6288, given in the benchmark as 16 bits [5]. 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 [4].

We build the stochastic implementation computing the target function based on the circuit structure shown in Fig. 6 (note that if the parameter s_i is set to 0 or 1, the circuit will be substantially simplified). Table IV shows the area (A) and delay (D) of each stochastic implementation of the three target functions presented in the last section. Each circuit 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 D flip-flop is implement 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.

TABLE IV
THE AREA AND DELAY OF THE STOCHASTIC IMPLEMENTATIONS OF THE THREE TARGET FUNCTIONS PRESENTED IN SECTION III.

Target Function	area (A)	delay (D)
Example 1	28	4
Example 2	75	4
Example 3	35	3

As stated in Section I, the result of the stochastic computation is obtained as the fractional weight of the 1's in the output bit stream. Hence, the resolution of the computation by a bit stream of N bits is $1/N$. Thus, in order to get the same resolution as the deterministic implementation, we need $N = 2^M$. Therefore, we need 2^M *cycles* to get the result.

As a measure of hardware cost, we compute the area-delay product. The area-delay product of the deterministic implementation computing a polynomial of degree n is $(10M^2 - 4M - 9)(12M - 11)n$, where n accounts for the n iterations in the implementation. The area-delay product of the stochastic implementation is $A \cdot D \cdot 2^M$, where A and D are the area and delay of the stochastic implementation for the corresponding target function. Table IV lists the area and delay of the stochastic implementations of the target functions of the three examples.

In Table V, we compare the area-delay product for the deterministic implementation and the stochastic implementation for $M = 7, 8, 9, 10, 11$ for each of the three target functions. Note that by using a Maclaurin polynomial approximation for the deterministic implementations, we need a polynomial of degree 6 to approximate the tanh function in **Example 2**, and a polynomial of degree 5 to approximate the exponentiation function in **Example 3** [4]. The last column of Table V shows the ratio of the area-delay product of the stochastic implementation to that of the deterministic implementation. We can see that the area-delay product of the stochastic implementation is less than that of the deterministic implementation and when $M \leq 10$, the area-delay product of the stochastic implementation is less than half that of the deterministic implementation.

TABLE V
THE AREA-DELAY PRODUCT COMPARISON OF THE DETERMINISTIC IMPLEMENTATION AND THE STOCHASTIC IMPLEMENTATION OF THE THREE TARGET FUNCTIONS WITH DIFFERENT RESOLUTION 2^{-M}.

Target Function	M	area-delay product		stoch. prod. / deter. prod.
		deter. impl.	stoch. impl.	
Example 1	7	99207	14336	0.145
	8	152745	28672	0.188
	9	222615	57344	0.258
	10	310977	114688	0.369
	11	419991	229376	0.546
Example 2	7	198414	38400	0.194
	8	305490	76800	0.251
	9	445230	153600	0.345
	10	621954	307200	0.494
	11	839982	614400	0.731
Example 3	7	165345	13440	0.081
	8	254575	26880	0.106
	9	371025	53760	0.145
	10	518295	107520	0.207
	11	699985	215040	0.307

B. Comparison of Circuit Performance on Noisy Input Data

We compare the performance of deterministic vs. stochastic computation on polynomial evaluations when the input data are corrupted with noise. Suppose that the input data of a deterministic implementation are $M = 10$ bits. In order to have 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 three target functions on 10 points: 0.1, 0.2, 0.3, $\cdots$, 0.9, 1. For each error ratio ϵ, each target func-

tion, 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 VI shows the average relative error of the stochastic implementations and the deterministic implementations vs. different error ratios ϵ. We average the relative errors of the three target functions, and plot the results in Fig. 7 to give a clear comparison.

TABLE VI
RELATIVE ERROR FOR THE STOCHASTIC IMPLEMENTATION AND THE DETERMINISTIC IMPLEMENTATION OF THE THREE TARGET FUNCTIONS VS. THE ERROR RATIO ϵ IN THE INPUT DATA.

	Example 1 rel. error of		**Example 2** rel. error of		**Example 3** rel. error of	
Error ratio ϵ	stoch. impl. (%)	deter. impl. (%)	stoch. impl. (%)	deter. impl. (%)	stoch. impl. (%)	deter. impl. (%)
0	2.09	0.00	2.31	0.00	2.17	0.00
0.001	2.09	0.44	2.30	0.35	2.23	0.52
0.002	2.03	0.71	2.32	0.79	2.21	1.23
0.005	2.03	2.40	2.33	2.11	2.23	2.42
0.01	1.96	4.48	2.30	3.47	2.26	5.36
0.02	1.96	8.49	2.30	7.43	2.35	9.36
0.05	2.55	15.64	2.46	14.50	2.69	19.68
0.1	4.93	27.97	2.97	25.36	3.83	34.95

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 of the stochastic bit streams, the stochastic implementation produces a small relative error [6]. With increasing errors on the input data stream, the relative error of the deterministic implementation blows up dramatically as ϵ increases. Even for small values, the stochastic implementation performs much better.

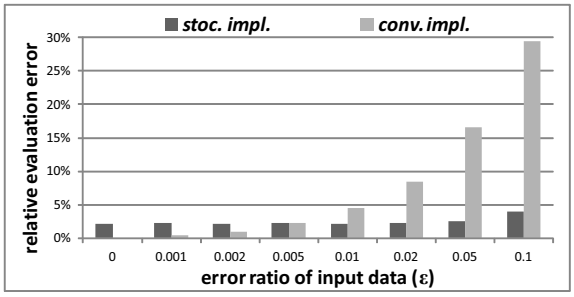


Fig. 7. A plot of the average relative error for the stochastic implementations and the deterministic implementations of the three target functions vs. the error ratio ϵ in the input data.

C. Comparison with Bernstein Polynomial-based Approach

Qian et al. introduced a Bernstein polynomial-based approach to synthesize a target function in stochastic computing [4]. Both the Bernstein polynomial-based approach and the FSM-based approach can be used to synthesize SCEs with comparable fault-tolerance performance. Table VII list the area and delay of the stochastic implementations of the three target functions by using the Bernstein polynomial-based synthesis approach. The last column of Table VII shows the ratio of the area-delay product of the stochastic implementation of the FSM-based approach to that of the Bernstein polynomial-based approach. It can be seen that the area-delay product of the FSM-based synthesis approach is less than that of the Bernstein polynomial-based synthesis approach. This is mainly because the Bernstein polynomial-based synthesis approach uses combinational logic, such as adders and multiplexers, to synthesize SCEs.

TABLE VII
THE AREA AND DELAY OF THE STOCHASTIC IMPLEMENTATIONS OF THE THREE TARGET FUNCTIONS BY USING THE BERNSTEIN POLYNOMIAL-BASED SYNTHESIS APPROACH.

Target Function	area (A)	delay (D)	FSM. / Bern.
Example 1	22	10	0.509
Example 2	58	20	0.259
Example 3	49	20	0.107

V. CONCLUSIONS

This paper proposed a general approach to synthesize the linear FSM-based SCEs. The area-delay product is less than that of deterministic implementations with adders and multipliers. Additionally, the circuits are much more error-tolerant. The precision of the results is dependent only on the statistics of the bit-streams that flow through the datapaths, and so the computation can tolerate errors gracefully.

VI. ACKNOWLEDGMENT

This work is supported by the US National Science Foundation (NSF) CAREER Award No. 0845650. The authors would like to thank the reviewers for their helpful feedback.

REFERENCES

[1] A. Kahng, S. Kang, R. Kumar, and J. Sartori, "Slack redistribution for graceful degradation under voltage overscaling," in *15th Asia and South Pacific Design Automation Conference, ASP-DAC'10*, pp. 825–831, 2010.

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

[3] B. D. Brown and H. C. Card, "Stochastic neural computation I: Computational elements," *IEEE Transactions on Computers*, vol. 50, pp. 891–905, September 2001.

[4] 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.

[5] "ISCAS'85 C6288 16×16 multiplier," in *http://www.eecs.umich.edu/∼jhayes/iscas/c6288.html*.

[6] 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.