The Synthesis of Complex Arithmetic Computation on Stochastic Bit Streams Using Sequential Logic

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

***Abstract*—The paradigm of logical computation on stochastic bit streams has several key advantages compared to deterministic computation based on binary radix, including error-tolerance and low hardware area cost. Prior research has shown that sequential logic operating on stochastic bit streams can compute non-polynomial functions, such as the tanh function, with less energy than conventional implementations. However, the functions that can be computed in this way are quite limited. For example, high order polynomials and non-polynomial functions cannot be computed using prior approaches. This paper proposes a new finite-state machine (FSM) topology for complex arithmetic computation on stochastic bit streams. It describes a general methodology for synthesizing such FSMs. Experimental results show that these FSM-based implementations are more tolerant of soft errors and less costly in terms of the area-time product that conventional implementations.**

I. Introduction

Circuit reliability has become increasingly important in recent years [1], [2], [3]. The paradigm of logical computation on stochastic bit stream has become an attractive solution for many applications. It uses conventional digital logic to perform computation on random bit streams, where the signal value is encoded as the probability of a one versus a zero in the stream. This approach can gracefully tolerate very large errors at lower cost than conventional techniques, such as over-design, while maintaining equivalent performance [4]. The images in Fig. 1, for example, illustrated that the fault tolerance capability of stochastic computing for image segmentation that is based on the frame difference algorithm as opposed to a conventional implementation. As the soft error injection rate increases, a conventional implementation of the image segmentation algorithm rapidly degrades until the output image is no longer recognizable. However, the second row shows that an implementation using stochastic bit streams to encode the data is able to produce the correct output at even very high error rates.

In addition, computations on stochastic bit streams can be performed with very simple logic. For example, multiplication can be implemented with an AND gate [6], [7]. Assuming that the two input stochastic bit streams A and B are independent, the number represented by the output stochastic bit stream C is $c = P(C = 1) = P(A = 1 \text{ and } B = 1) = a \cdot b$. So the AND gate multiplies the two values represented by the stochastic bit streams. Scaled addition can be implemented with a multiplexer (MUX) [6], [7], [8]. 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 $c = P(C = 1) = P(S = 1 \text{ and } A = 1) + P(S = 0 \text{ and } B = 1) = s \cdot a + (1 - s) \cdot b$. Thus the computation performed by a MUX 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.

Despite its great potential in terms of high fault-tolerance and low hardware cost, stochastic computing suffers from encoding inefficiency. Assume that a numerical value is represented by M bits using binary radix, we need 2^M bits to represent the same value stochastically. For small values of M such as $M = 8$ (e.g., used in most image processing algorithms and artificial neural networks), the benefits of stochastic computing overwhelm its high encoding overhead. In terms of performance, although computation on bit streams needs more clock cycles to finish, the circuit can be operated under a much faster clock frequency. This is because the circuit is extremely simple, and has a much shorter critical path. In addition, computation on bit streams can be implemented in parallel by trading off silicon area with the number of clock cycles [8].

In terms of energy consumption, complex computations on stochastic bit streams can be performed using quite simple sequential logic. Brown and Card [7] showed that, if a single input linear finite-state machine (FSM) is used to perform the exponentiation and tanh functions on stochastic bit streams, when $M \leq 10$ it will consume less energy than deterministic implementations using adders and multipliers based on binary radix.

However, one shortcoming of the work in [7] is that the functions that can be implemented using the proposed FSM are very limited. For example, high order polynomials and non-polynomials such as functions used in low-density parity-check coding cannot be implemented using the FSM introduced in the previous work [7]. As a result, most applications in artificial neural networks (ANNs), communication systems, and digital signal processing, cannot benefit from this technique.

To solve this problem, we find that the FSM used in stochastic computing can be analyzed using Markov chains [7], [9]. By redesigning the topology of the FSM, we increase the degree of design freedom, so that more sophisticated functions can be synthesized stochastically. The remainder of this paper is organized as follows. In Section II, we briefly review the previous work in this area. In Section III, we demonstrate the new FSM topology. In Section IV, we introduce the synthesis approach. In Section V, we present the results of synthesis trials, comparing cost, fault-tolerance, and energy consumption of the proposed designs with the previous ones. Section VI concludes and discusses future directions.

II. Related Work

Logical computation on stochastic bit streams was first introduced by Gaines [6]. He proposed two coding formats: a unipolar format and a bipolar format. Both formats can coexist in a single system. In the unipolar coding format, a real number x in the unit interval (i.e.,

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA
Copyright©2012 ACM 978-1-4503-1573-9/12/11...$15.00

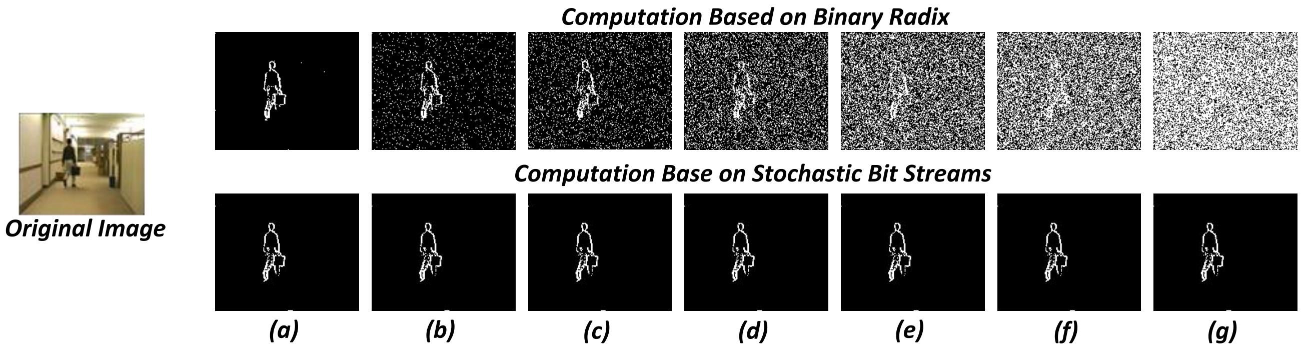
Computation Based on Binary Radix
Computation Base on Stochastic Bit Streams
Original Image
(a)
(b)
(c)
(d)
(e)
(f)
(g)

Fig. 1. A comparison of the fault tolerance capabilities of different hardware implementations for the frame difference based image segmentation algorithm. The images in the first row are generated by a conventional implementation. The images in the second row are generated using a stochastic implementation. Soft errors are injected at a rate of (a) 0%; (b) 1%; (c) 2%; (d) 5%; (e) 10%; (f) 15%; (g) 30% [5].

$0 \le x \le 1$) 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 $-1 \le x \le 1$. The probability that each bit in the stream is one is $P(X=1)=\frac{x+1}{2}$. 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 $2\frac{k}{N}-1$.

Beginning with the work by Gaines, prior research has described the implementations of specific arithmetic functions based on the stochastic representation [6]. These include constructs for multiplication and addition, discussed in the previous section, as well as constructs for the tanh and the exponentiation functions proposed by Brown and Card [7].

The tanh and the exponentiation functions are implemented with sequential logic, in the form of a single input linear finite state machine [7]. Its state transition diagram is shown in Fig. 2. The machine has a single input X and a set of states $S_0, S_1, \ldots, S_{N-1}$ arranged as a linear sequence. Given the current state S_t $(0 < t < N-1)$, the next state will be S_{t-1} if $X=0$ and will be S_{t+1} if $X=1$.

With a stochastic encoding, the input X takes the form of a stochastic bit stream. As a result the state transition process is a special type of a Markov chain [7], [9]. The output Y of this state machine, not shown in Fig. 2, is only determined by the current state: for some states the output Y is one and for the others it is zero. Thus, the output Y is also a stochastic bit stream. Based on different choices of the set of states that let the output Y be one, this linear FSM can be used to implement different functions.

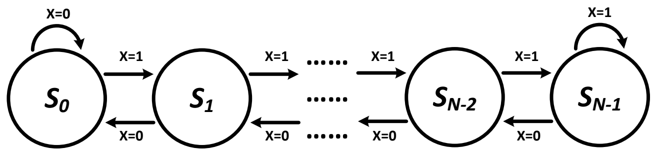

Fig. 2. A generic linear state transition diagram.

For example, consider an FSM whose output Y is one when the current state is in the set $\{S_{N/2}, \ldots, S_{N-1}\}$ and is zero when the current state is in the set $\{S_0, \ldots, S_{N/2-1}\}$, as shown in the state transition diagram in Fig. 3. If we let x be the bipolar coding of the input bit stream X and y be the bipolar coding of the output bit stream Y, then the functional relation between x and y is approximately a tanh function [7],

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

Now consider another FSM whose output Y is one when the current state is in the set $\{S_0, \ldots, S_{N-G-1}\}$ and is zero when the current state is in the set $\{S_{N-G}, \ldots, S_{N-1}\}$ (where $G \ll N$), as shown in the state transition diagram in Fig. 4. If we let x be the bipolar coding of the input bit stream X and y be the unipolar coding of the output bit stream Y, then the functional relation between x and y is approximately an exponentiation function [7],

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

However, besides these two functions, how to configure other functions based on the linear FSM has not been studied. In 2008, Qian et al. proposed a general approach for synthesizing combinational logic to implement polynomials on stochastic bit streams [8]. The procedure begins by transforming a power-form polynomial into a Bernstein polynomial [10]. For example, The polynomial

$$f(x) = \frac{1}{4} + \frac{9}{8}x - \frac{15}{8}x^2 + \frac{5}{4}x^3,$$

can be converted into a Bernstein polynomial of degree 3:

$$f(x) = \frac{2}{8}B_{0,3}(x) + \frac{5}{8}B_{1,3}(x) + \frac{3}{8}B_{2,3}(x) + \frac{6}{8}B_{3,3}(x),$$

where each $B_{i,3}(x)$ $(0 \le i \le 3)$ is a Bernstein basis polynomial of the form

$$B_{i,3}(x) = \binom{3}{i} x^i (1-x)^{3-i}. \quad (3)$$

A Bernstein polynomial,

$$y = B(x) = \sum_{i=0}^{n} b_i B_{i,n}(x),$$

with all coefficients b_i in the unit interval can be implemented stochastically by a generalized multiplexing circuit, shown in Fig. 5. The circuit consists of an adder block and a multiplexer block. The inputs to the adder are $x_1, \ldots, x_n$. The data inputs to the multiplexer

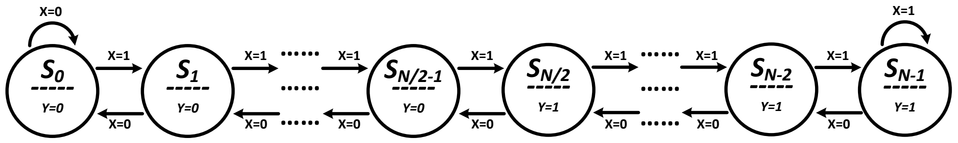

Fig. 3. The state transition diagram of an FSM implementing a tanh function stochastically. In the figure, the number below each state S_i represents the output Y of the FSM when the current state is S_i $(0 \le i \le N-1)$ [7].

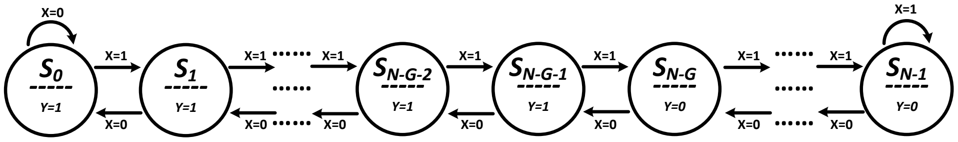

Fig. 4. The state transition diagram of an FSM implementing an exponentiation function stochastically. In the figure, the number below each state S_i represents the output Y of the FSM when the current state is S_i $(0 \le i \le N-1)$. Note that the number $G \ll N$ [7].

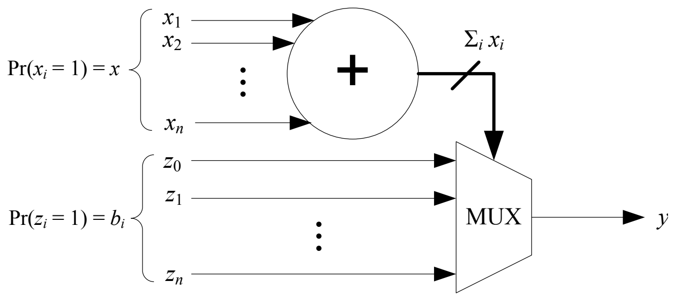

Fig. 5. A generalized multiplexing circuit implementing the Bernstein polynomial $y = B(x) = \sum_{i=0}^{n} b_i B_{i,n}(x)$ with $0 \le b_i \le 1$, for $i = 0, 1, \ldots, n$ [8].

are $z_0, \ldots, z_n$. The outputs of the adder are the selecting inputs to the multiplexer block.

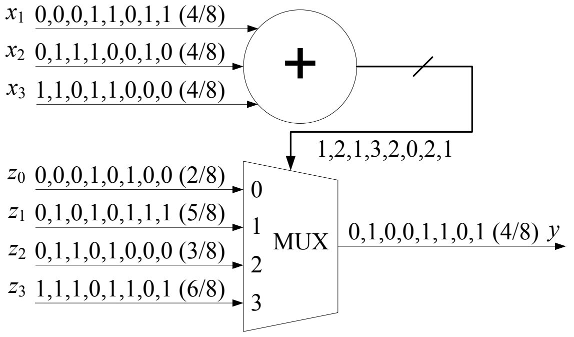
x_1 0,0,0,1,1,0,1,1 (4/8)
x_2 0,1,1,1,0,0,1,0 (4/8)
x_3 1,1,0,1,1,0,0,0 (4/8)
+
1,2,1,3,2,0,2,1
z_0 0,0,0,1,0,1,0,0 (2/8)
z_1 0,1,0,1,0,1,1,1 (5/8)
z_2 0,1,1,0,1,0,0,0 (3/8)
z_3 1,1,1,0,1,1,0,1 (6/8)
0
1
2
3
MUX
0,1,0,0,1,1,0,1 (4/8) y

Fig. 6. Logical computation on stochastic bit streams implementing the Bernstein polynomial $f(x) = \frac{2}{8}B_{0,3}(x) + \frac{5}{8}B_{1,3}(x) + \frac{3}{8}B_{2,3}(x) + \frac{6}{8}B_{3,3}(x)$ at $x = 0.5$. Stochastic bit streams x_1, x_2 and x_3 encode the value $x = 0.5$. Stochastic bit streams z_0, z_1, z_2 and z_3 encode the corresponding Bernstein coefficients [8].

When the number of ones in the input set $\{x_1, \ldots, x_n\}$ is i, then the adder will output a binary number equal to i and the output y of the multiplexer will be set to z_i. The inputs $x_1, \ldots, x_n$ are independent stochastic bit streams $X_1, \ldots, X_n$ representing the probabilities $P(X_i = 1) = x \in [0, 1]$, for $1 \le i \le n$. The inputs $z_0, \ldots, z_n$ are independent stochastic bit streams $Z_0, \ldots, Z_n$ representing the probabilities $P(Z_i = 1) = b_i \in [0, 1]$, for $0 \le i \le n$, where the b_i's are the Bernstein coefficients. The output of the circuit is a stochastic bit stream Y in which the probability of a bit being one equals the Bernstein polynomial $B(x) = \sum_{i=0}^{n} b_i B_{i,n}(x)$. A circuit implementation of the above example is shown in Fig. 6 [8].

The tanh and the exponentiation functions, as shown in equations (1) and (2), can also be implemented by this Bernstein polynomial-based approach. However, it requires more hardware than the FSM-based ones. To leverage the low-cost and low-power properties of the FSM-based stochastic computation, this paper demonstrates how to synthesize sophisticated functions based on a new FSM topology. Based on this technique, more sophisticated functions in ANNs, communication systems, and digital signal processing could be computed on the stochastic bit streams with lower hardware cost, lower energy consumption, and higher fault-tolerance.

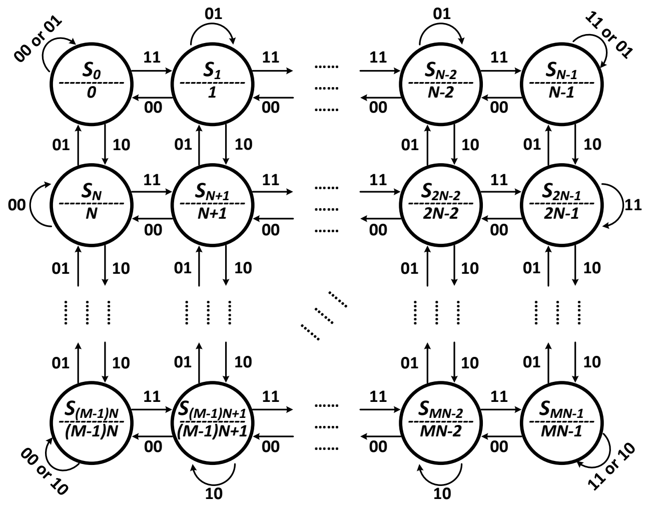

Fig. 7. The FSM has two inputs X and K. The numbers on each arrow represent the transition condition, with the first corresponding to the input X and the second corresponding to the input K. The FSM has $log_2\lceil MN \rceil$ outputs, encoding a value in binary radix. In the figure, the number below each state S_t $(0 \le t \le MN-1)$ represents the value encoded by the outputs of the FSM when the current state is S_t.

III. The New FSM Topology

The state transition diagram of the proposed FSM is shown in Fig. 7. It has two inputs (we call them X and K) and in total $M \times N$ states, arranged as an $M \times N$ two-dimensional array. We normally set $M \times N = 2^R$, where R is a positive integer, because we can implement an FSM with 2^R states by R D flip-flops (DFFs). In addition, we set $M = 2^{\lfloor \frac{R}{2} \rfloor}$, and $N = 2^{\lceil \frac{R}{2} \rceil}$. The FSM shown in Fig. 7 is a Moore-style machine, which has $log_2\lceil MN \rceil$ outputs encoding an integer in the range [0, $MN-1$] using binary radix. If the current state is S_t ($0 \leq t \leq MN-1$), the value encoded by its output is just the current state number t. In short, we say that the output of the FSM is t. It can be seen that the output configuration of the FSM looks like an up/down counter. However, the state transitions of this FSM are quite different from a conventional up/down counter. For example, if we assume that the current state is S_{N+1} in Fig. 7, its next state and the corresponding output will be: S_{N+2} and $N+2$, if $(X,K) = (1,1)$; S_N and N, if $(X,K) = (0,0)$; S_{2N+1} and $2N+1$, if $(X,K) = (1,0)$; S_2 and 2, if $(X,K) = (0,1)$. An example of such an FSM with 8 states is shown in Fig. 8.

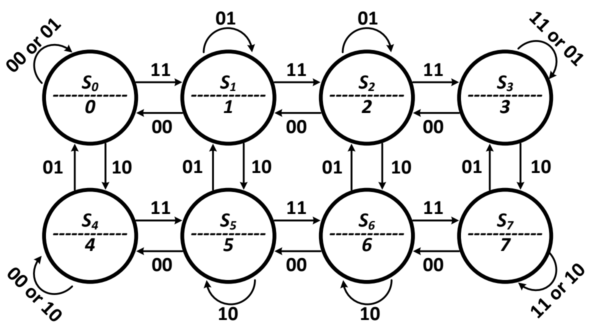

Fig. 8. An example of the state transition digram of the proposed FSM with 8 states. Here, $M = 2$ and $N = 4$.

The inputs X and K of this FSM consist of stochastic bit streams. We define the probability that each bit in the input stream X is one to be P_X, and the probability that each bit in the input stream K is one to be P_K. Because both inputs X and K are stochastic bit streams with fixed probabilities, the state transition process of the FSM can be modeled as a time-homogeneous Markov chain. It can be shown that this Markov chain is irreducible and aperiodic. Then, based on the theory of Markov chain, the FSM has an equilibrium state distribution [9]. We define the probability that the current state is S_t ($0 \leq t \leq MN-1$) in the equilibrium (or the probability that the current output is t) to be P_t. Intuitively, P_t is a function of both P_X and P_K. In the following, we derive a closed form expression of P_t in terms of P_X and P_K. This expression is used to synthesize a given target function $T(P_X)$ on P_X. The synthesis details will be discussed in the next section.

In the following discussion, we define $i = \lfloor \frac{t}{N} \rfloor$ and $j = t$ modulo N, i.e., i and j are the quotient and the remainder of t divided by N, respectively (or $t = i \times N + j$). Based on the theory of Markov chains [9], at the equilibrium stage, the probability of transitioning from the state $S_{i\times N+j}$ to its horizontal adjacent state $S_{i\times N+j-1}$, equals the probability of transitioning from the state $S_{i\times N+j-1}$ to the state $S_{i\times N+j}$. Thus, we have

$$P_{i\times N+j} \cdot (1-P_X) \cdot (1-P_K) = P_{i\times N+j-1} \cdot P_X \cdot P_K. \quad (4)$$

In addition, the probability of transitioning from the state $S_{i\times N+j}$ to its vertical adjacent state, $S_{(i-1)\times N+j}$, equals the probability of transitioning from the state $S_{(i-1)\times N+j}$ to the state $S_{i\times N+j}$:

$$P_{i\times N+j} \cdot (1-P_X) \cdot P_K = P_{(i-1)\times N+j} \cdot P_X \cdot (1-P_K). \quad (5)$$

Because all the individual state probabilities $P_{i\times N+j}$ (or P_t) must sum to unity, we have

$$\sum_{t=0}^{MN-1} P_t = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} P_{i\times N+j} = 1. \quad (6)$$

Based on equation (4), (5), and (6), we obtain

$$P_t = P_{i\times N+j} = \frac{t_x^i \cdot t_y^j}{\sum_{u=0}^{M-1} \sum_{v=0}^{N-1} t_x^u \cdot t_y^v}, \quad (7)$$

where t_x and t_y are,

$$t_x = \frac{P_X}{1-P_X} \cdot \frac{P_K}{1-P_K},$$
$$t_y = \frac{P_X}{1-P_X} \cdot \frac{1-P_K}{P_K}.$$

It can be seen that equation (7) is a closed form expression of P_t in terms of P_X and P_K. In the next section, we discuss how P_t is used to synthesize a given target function.

IV. The FSM-based Synthesis Approach

In this section, we introduce how to synthesize a target function $T(P_X)$ based on the proposed FSM. More specifically, we use the circuit shown in Fig. 9 to synthesize $T(P_X)$.

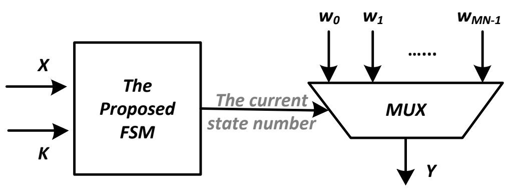

Fig. 9. The circuit for synthesizing target functions.

In Fig. 9, as we introduced in the previous section, the inputs of the proposed FSM are X and K, its output is the current state number, which is connected to the selection inputs of the multiplexer "*MUX*", which has $M \times N$ data inputs (w_0, w_1, $\cdots$, w_{MN-1}). Note that if the current state of the FSM is S_t ($0 \leq t \leq MN-1$), then the channel that connects w_t to Y will be selected in the "*MUX*."

We let X, K, and w_t be stochastic bit streams, and define P_X to be the probability of ones in X, P_K to be the probability of ones in K, P_{w_t} to be the probability of ones in w_t, and P_Y to be the probability of ones in Y. Based on the circuit shown in Fig. 9, it can be seen that the probability that the "*MUX*" input w_t is selected as its output is P_t, because this probability is the same as the probability that the current state of the FSM is S_t. Thus, we can obtain P_Y as

$$\begin{aligned} P_Y &= P(Y=1) \\ &= \sum_{t=0}^{MN-1} P(Y=1 \mid w_t \textit{ is selected}) \cdot P(w_t \textit{ is selected}) \\ &= \sum_{t=0}^{MN-1} P(w_t=1) \cdot P(w_t \textit{ is selected}) = \sum_{t=0}^{MN-1} P_{w_t} \cdot P_t. \end{aligned} \tag{8}$$

Note that P_Y is a function of P_X, P_K, and P_{w_t}, because P_t is a function of P_X and P_K (refer to (7)). It can be seen that equation (8) is a closed form expression of P_Y. This expression is used to synthesize the given function $T(P_X)$. We define the approximation error ϵ as

$$\epsilon = \int_0^1 (T(P_X) - P_Y)^2 \cdot d(P_X). \tag{9}$$

The synthesis goal is to compute P_{w_t} and P_K to minimize ϵ. In the following, we discuss how to obtain these parameters.

A. How to Compute P_{w_t}

By expanding (9), we can rewrite ϵ as

$$\begin{aligned} \epsilon = &\int_0^1 T(P_X)^2 \cdot d(P_X) - 2\int_0^1 T(P_X) \cdot P_Y \cdot d(P_X) \\ &+ \int_0^1 P_Y^2 \cdot d(P_X). \end{aligned}$$

The first term $\int_0^1 T(P_X)^2 \cdot d(P_X)$ is a constant because $T(P_X)$ is given. Thus minimizing ϵ is equivalent to minimizing the following objective function φ:

$$\varphi = \int_0^1 P_Y^2 \cdot d(P_X) - 2\int_0^1 T(P_X) \cdot P_Y \cdot d(P_X). \tag{10}$$

We define a vector $\mathbf{b}$, a vector $\mathbf{c}$, and a matrix $\mathbf{H}$ as follows,

$$\mathbf{b} = [P_{w_0}, P_{w_1}, \cdots, P_{w_{MN-1}}]^T,$$

$$\mathbf{c} = \begin{bmatrix} -\int_0^1 T(P_X) \cdot P_0 \cdot d(P_X) \\ -\int_0^1 T(P_X) \cdot P_1 \cdot d(P_X) \\ \vdots \\ -\int_0^1 T(P_X) \cdot P_{MN-1} \cdot d(P_X) \end{bmatrix},$$

$$\mathbf{H} = [\mathbf{H_0}, \mathbf{H_1}, \cdots, \mathbf{H_{MN-1}}]^T,$$

where P_t $(0 \le t \le MN-1)$ in vector $\mathbf{c}$ are defined by equation (7) and H_t $(0 \le t \le MN-1)$ in matrix $\mathbf{H}$ is a row vector defined as follows,

$$\mathbf{H_t} = \begin{bmatrix} \int_0^1 P_t \cdot P_0 \cdot d(P_X) \\ \int_0^1 P_t \cdot P_1 \cdot d(P_X) \\ \vdots \\ \int_0^1 P_t \cdot P_{MN-1} \cdot d(P_X) \end{bmatrix}^T.$$

Note that (refer to the expression of P_Y in (8)),

$$\mathbf{b}^T\mathbf{H}\mathbf{b} = \int_0^1 P_Y^2 \cdot d(P_X),$$

$$\mathbf{c}^T\mathbf{b} = -\int_0^1 T(P_X) \cdot P_Y \cdot d(P_X),$$

Thus, the objective function φ in (10) can be rewrite as

$$\varphi = \mathbf{b}^T\mathbf{H}\mathbf{b} + 2\mathbf{c}^T\mathbf{b}. \tag{11}$$

We notice that, computing P_{w_t} (i.e., the vector $\mathbf{b}$) to minimize φ in the form of equation (11) is a typical constrained quadratic programming problem, if P_K is a constant. This is because P_t is a function of both P_X and P_K (please refer to (7)). When we set P_K to a constant, the integral of P_t on P_X is also a constant, so are the vector $\mathbf{c}$ and the matrix $\mathbf{H}$. Based on (11), the solution of P_{w_t} (i.e., the vector $\mathbf{b}$) can be obtained using standard techniques [11]. Then P_K can be solved using a numerical approach, which will be discussed in the next section.

B. How to Compute P_K

As we introduced in the previous section, P_K is first set to a constant. Then we compute P_{w_t} using standard techniques [11] to minimize ϵ (or the equivalent φ). Note that all the values between 0 and 1 (with a step 0.001) will be used to set P_K in the synthesis process. More specifically, we first set P_K to 0.001, and compute the corresponding P_{w_t} and ϵ. Next, we set P_K to 0.002, and compute the corresponding P_{w_t} and ϵ. So on and so forth. Finally, we set P_K to 1, and compute the corresponding P_{w_t} and ϵ. Among these 1000 results of ϵ, we select the minimum one, and the corresponding P_K and P_{w_t}.

C. Summary of the Proposed Synthesis Approach

After we get the optimal values of P_K and P_{w_t}, the stochastic bit streams K and w_t in Fig. 9 will be generated to implement the target function $T(P_X)$ stochastically. Note that the synthesis approach discussed in this section also works for other FSM topologies. For example, we can use the same approach to synthesize the exponentiation and tanh functions proposed by Brown and Card [7] based on the FSM topology shown in Fig. 2 [12], [13], [14]. However, that FSM topology cannot be used to synthesize more sophisticated functions, such as high order polynomials and other non-polynomials. We also tried other different FSM topologies in both single input and two inputs configurations. Only the one proposed in this paper works. Intuitively, the additional input K and the balanced state transitions give the proposed FSM more degree of design freedom to synthesize more sophisticated functions.

In the next section, we use three examples to show the synthesis effect of the proposed approach. Then we compare this work to the previous ones in terms of hardware cost, energy consumption, and fault-tolerance.

V. Experimental Results

In our experiments, we first present three examples to show the synthesis results of the proposed approach. Then we compare the proposed approach to the Bernstein polynomial-based approach and the binary radix representation-based approach in terms of hardware area, performance, energy consumption, and fault-tolerance. We use an 8-state FSM (i.e., the one shown in Fig. 8) for all the experiments.

A. Synthesis Examples

Example 1: Synthesizing Gaussian distribution function:

$$T(x) = \frac{1}{\delta\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\delta^2}}, \quad (-1 \le x \le 1).$$

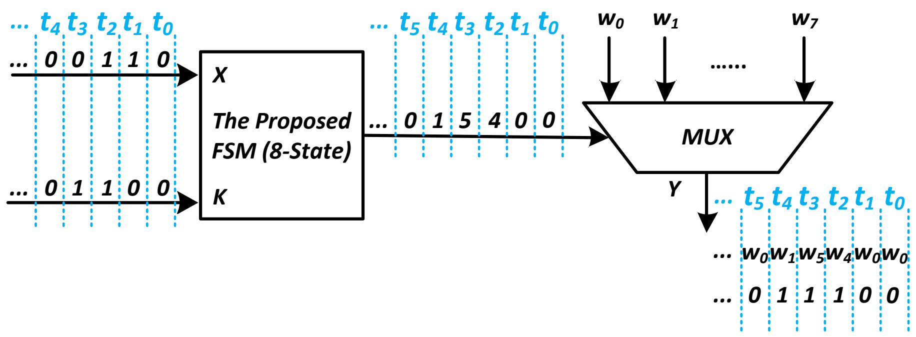

Fig. 10. An example about how the circuit shown in Fig. 9 works (the FSM is implemented with the state transition diagram shown in Fig. 8).

Because x could be negative values, we need to convert x to P_X using bipolar coding (please refer to Section II), i.e., we set $P_X = 0.5(x+1)$, and rewrite the target function as

$$T(P_X) = \frac{1}{\delta\sqrt{2\pi}} e^{-\frac{(2P_X-1-\mu)^2}{2\delta^2}}, \quad (0 \le P_X \le 1). \tag{12}$$

Note that in (12), δ could be any value as long as $\delta \ge \frac{1}{\sqrt{2\pi}}$, because in computation on stochastic bit streams, the output is a probability value that should be greater than or equal to 0 and less than or equal to 1 and when $\delta \ge \frac{1}{\sqrt{2\pi}}$, we guarantee that the maximal value of the function $T(P_X)$ is less than or equal to 1. We normally set $\mu = 0$ for simplicity, because using different value of μ will only shift the curve along on the x-axis instead of changing the shape of the curve. If we set $\delta = 2$ and $\mu = 0$, for example, we compute P_K and P_{w_t} using the synthesis approach discussed in the previous section. The results are listed in Table I. The approximation error ϵ (defined in (9)) is 7.0×10^{-4}. Fig. 11 shows the simulation result.

TABLE I
P_K AND P_{w_t} $(0 \le t \le 7)$ FOR SYNTHESIZING THE TARGET FUNCTION IN (12) WITH $\delta = 2$ AND $\mu = 0$.

$P_K = 0.5$			
$P_{w_0} = 0$	$P_{w_1} = 1$	$P_{w_2} = 1$	$P_{w_3} = 1$
$P_{w_4} = 1$	$P_{w_5} = 1$	$P_{w_6} = 1$	$P_{w_7} = 0$

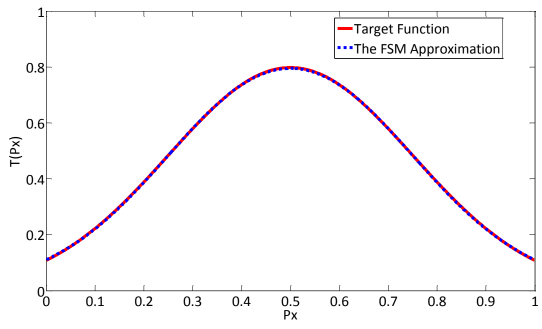

Fig. 11. Synthesis result of the target function in (12) with $\delta = 2$ and $\mu = 0$.

This means that, using the circuit shown in Fig. 9, if the probability of ones in the input bit stream K equals 0.5 and the probabilities of ones in the input bit streams w_t equal P_{w_t} shown in Table I, the probability of ones in the output bit stream Y will be

$$\frac{1}{2\sqrt{2\pi}} e^{-\frac{(2P_X-1)^2}{8}}, \quad (0 \le P_X \le 1),$$

where P_X is the probability of ones in the input X. Note that if P_{w_t} equal 0 or 1, then w_t can be set to a constant '0' or '1' correspondingly, and the hardware implementation can be further simplified.

We give an example in Fig. 10 to show how this circuit works. Assume that the circuit starts working at clock cycle t_0 and the initial state of the FSM is S_0 (please note that the initial state has no influence on the final results, it could be any one of the 8 states). The output of the FSM at t_0 is 0 (because its initial state is S_0) and the output of the multiplexer "*MUX*" at t_0 is w_0 (because its selection input equals 0 at t_0), and $w_0 = 0$ based on Table I.

Because at t_0, $X = K = 0$ and the initial state is S_0, in the next clock cycle t_1, the current state of the FSM is still S_0 (and the output of the FSM is still 0) based on the state transition diagram shown in Fig. 8. The output of the "*MUX*" at t_1 is still w_0, which equals 0 based on Table I.

In the next clock cycle t_2, based on the state transition diagram shown in Fig. 8, the current state becomes S_4 because $X = 1$ $K = 0$ at the previous clock cycle t_1, and the output of the FSM is 4. Thus, the output of the "*MUX*" at t_2 becomes w_4, which equals 1 based on Table I.

So on and so forth. Assume that we use 1024 bits to represent a value stochastically. After 1024 clock cycles, if the probability of ones in X equals P_X, and the probability of ones in K equals 0.5, then the probability of ones in Y will be $\frac{1}{2\sqrt{2\pi}} e^{-\frac{(2P_X-1)^2}{8}}$. ■

Example 2: Synthesizing the function $\phi(x)$ used in low-density parity-check decoding [15]:

$$\phi(x) = log\frac{e^x+1}{e^x-1}, \quad (x > 0).$$

For this example, we use unipolar coding because we do not deal with negative values, and set $P_X = x/\alpha$, where α is a scaling factor to map the range of x to unitary. We rewrite the target function in terms of P_X as

$$T(P_X) = log\frac{e^{\alpha P_X}+1}{e^{\alpha P_X}-1}, \quad (0 < P_X \leq 1). \tag{13}$$

If we set $\alpha = 20$, for example, we compute P_K and P_{w_t} using the proposed synthesis approach and show the results in Table II. The approximation error ϵ (defined in (9)) is 2.0×10^{-4}. Fig. 12 shows the simulation result. ■

TABLE II
P_K AND P_{w_t} FOR SYNTHESIZING THE TARGET FUNCTION IN (13) WITH $\alpha = 20$.

$P_K = 0.9375$			
$P_{w_0} = 1$	$P_{w_1} = 0$	$P_{w_2} = 0$	$P_{w_3} = 0$
$P_{w_4} = 1$	$P_{w_5} = 0$	$P_{w_6} = 0$	$P_{w_7} = 0$

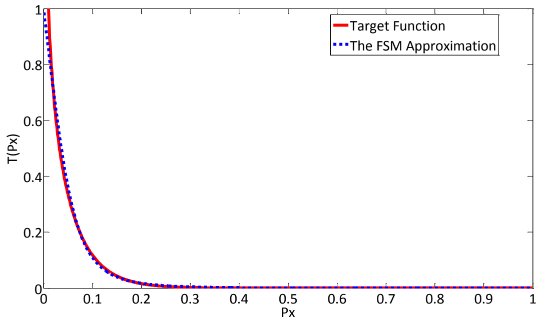

Fig. 12. Synthesis result of the target function in (13) with $\alpha = 20$.

Example 3: Synthesizing the following high order polynomial $\lambda(x)$ used in low-density parity-check coding [16]:

$$\lambda(x) = 0.1575x + 0.3429x^2 + 0.0363x^5 + 0.059x^6 \\ + 0.279x^8 + 0.1253x^9, \quad (0 \leq x \leq 1).$$

For this example, we use unipolar coding because we do not deal with negative values and x is in the unitary range. We rewrite the target function in terms of P_X ($P_X = x$) as

$$T(P_X) = 0.1575P_X + 0.3429P_X^2 + 0.0363P_X^5 + 0.059P_X^6 \\ + 0.279P_X^8 + 0.1253P_X^9, \quad (0 \leq P_X \leq 1). \tag{14}$$

We compute P_K and P_{w_t} using the proposed synthesis approach and show the results in Table III. The approximation error ϵ (defined in (9)) is 4.0×10^{-6}. Fig. 13 shows the simulation result. Note that the bit streams w_1 and w_5 can be generated with extremely low cost using the technique proposed by Qian et al. [17]. ■

TABLE III
P_K AND P_{w_t} FOR SYNTHESIZING THE TARGET FUNCTION IN (14).

$P_K = 0.1875$			
$P_{w_0} = 0$	$P_{w_1} = 0.86$	$P_{w_2} = 0$	$P_{w_3} = 0$
$P_{w_4} = 0$	$P_{w_5} = 0.89$	$P_{w_6} = 0$	$P_{w_7} = 1$

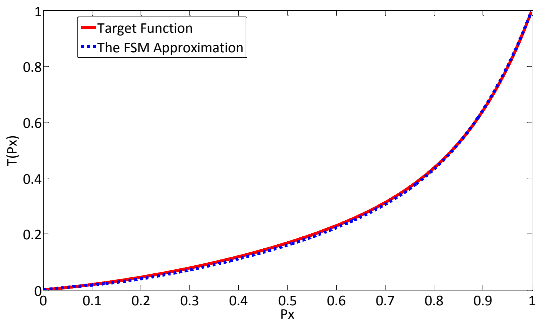

Fig. 13. Synthesis result of the target function in (14).

B. Comparison with the Bernstein Polynomial-Based Approach

As we introduced in Section II, the Bernstein polynomial-based approach uses combinational logic (an adder and a multiplexer, as shown in Fig. 6) to perform computation on stochastic bit streams. Hardware area required by this approach depends on the degree of polynomial. Table IV lists its area in terms of the number of fan-in two logic gates [8].

TABLE IV
THE NUMBER OF THE FAN-IN TWO LOGIC GATES FOR COMPUTING BERNSTEIN POLYNOMIALS OF DEGREE 3, 4, 5, AND 6 [8].

Degree n	3	4	5	6
Number of Gates	22	40	49	58

Note that by using the 8-state FSM shown in Fig. 8, we can synthesize a polynomial of degree up to 9 (refer to **Example 3**). In addition, for those non-polynomials, such as the target functions introduced in **Example 1** and **Example 2**, the Bernstein polynomial-based approach normally takes at least degree 6 to obtain the same level of approximation error. The proposed FSM with 8 states can be implemented using 3 D-flip-flops (DFFs) as follows,

$$D_2 = XKQ_0 + XQ_1 + KQ_1 + Q_1Q_0,$$
$$D_1 = X\bar{K} + XQ_2 + \bar{K}Q_2,$$
$$D_0 = \bar{X}\bar{K}Q_1\bar{Q}_0 + \bar{X}KQ_0 + XKQ_1 + X\bar{K}Q_0 + XKQ_0,$$

where D_0, D_1, and D_2 are the inputs of the three DFFs, and Q_0, Q_1, and Q_2 are the corresponding outputs. Because it is a Moore FSM, we assign $Q_2Q_1Q_0 = 000$ for state S_0, $Q_2Q_1Q_0 = 001$ for state S_1, $\cdots$, and $Q_2Q_1Q_0 = 111$ for state S_7. Based on the report of the logic synthesis tool Synonsys Design Compiler, the entire circuit (including the multiplexer in Fig. 9) can be implemented using 45 fan-in two logic gates. Please note that the evaluation is based on a generalized version of the circuit shown in Fig. 9, if w_t is set to a constant '0' or '1', the circuit can be further simplified and the number of logic gates can be further reduced. It can be seen that, to synthesize non-polynomials and polynomials of degree greater than 4, the proposed FSM takes less hardware.

In terms of performance, because both techniques compute on stochastic bit streams, they have equivalent processing time. In terms of energy consumption, we assume that given a CMOS technology, a digital circuit consumes a constant power dissipation per unit area. We use the product of area and processing time as a metric of the energy consumption [7]. Because these two techniques have equivalent processing time, the FSM-based approach consumes less energy when computing non-polynomials and high order polynomials with degree larger than 4.

In terms of fault-tolerance, we compare the two techniques when the input data is corrupted with noise. We evaluate the fault-tolerant performance on circuits implementing the target functions introduced in the last section and other functions such as trigonometric functions ($sin(x)$, $cos(x)$, and $tan(x)$) and logarithmic functions ($y = log_2(x)$, $y = log_{10}(x)$, and $y = ln(x)$). The length of the stochastic bit streams which are used to represent a value is set to 1024. We define the error ratio γ as the percentage of random bit flips that occur in the computation. We choose the error ratio γ to be 0%, 0.5%, 1%, 5%, and 10%. For example, under 10% error ratio, 102 of 1024 bits will be flipped in the computation. To measure the impact of the noise, we evaluated each target function at 13 distinct input data points: 0.2, 0.25, 0.3, $\cdots$, 0.8. For each error ratio γ, each target function, and each evaluation point, we simulated both the FSM-based implementation and the Bernstein polynomial-based implementation 1000 times. We averaged the relative errors over all simulations. Finally, for each error ratio γ, we averaged the relative errors over all target functions and all evaluation points. Table V shows the average relative error of the two different implementations versus different γ values. It can be seen that these two techniques has almost equivalent fault-tolernace (the difference is less than 0.5%), because both techniques perform computation on stochastic bit streams.

TABLE V
RELATIVE ERROR FOR THE FSM-BASED IMPLEMENTATION AND THE BERNSTEIN POLYNOMIAL-BASED IMPLEMENTATION OF TARGET FUNCTION COMPUTATION VERSUS THE ERROR RATIO γ IN THE INPUT DATA.

Error Ratio γ (%)	0	0.5	1	5	10
Relative Error of the FSM (%)	2.26	2.78	3.16	6.75	11.2
Relative Error of Bernstein (%)	2.21	2.72	3.36	6.25	11.7

C. Comparison with the Binary Radix-Based Approach

Assume that M is the number of bits used to represent a numerical value in 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. Both Qian et al. [8] and Brown et al. [7] showed that, when $M \leq 10$, computation on stochastic bit streams has better performance than the ones based on binary radix using adders and multipliers in terms of hardware area and energy consumption. In fact, in most applications of the stochastic computation, M is between 8 to 10 [7], [8]. As we discussed in the Section V-B, the proposed approach using FSM has better performance than the one proposed by Qian et al. [8] for computing non-polynomials and high order polynomials. Thus, when $M \leq 10$, the proposed approach also has better performance than the ones using binary radix for those functions. 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. The main issue of this computing technique is the long latency. However, it can be solved by using a faster clock frequency, because its logic is simple and has shorter critical path. Parallel computing can also be used to solve this issue. For example, in digital image processing applications, we can process multiple pixels in an image in parallel. In ANN applications, we can process the computation on multiple neurons in parallel.

VI. CONCLUSION

This paper proposed a new FSM topology to synthesize computation on stochastic bit streams for complex and useful functions. Compared to other implementations, the resulting circuits are less costly in terms of hardware area and energy consumption. In future work, we will study synthesis techniques for general purpose computation using these techniques. Our eventual goal is a fully stochastic design of a microprocessor.

ACKNOWLEDGMENT

This work is supported in part by National Science Foundation 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. This work is also supported in part by the Minnesota Supercomputing Institute and a donation from NVIDIA. The authors would like to thank the reviewers for their helpful feedback.

REFERENCES

[1] N. Iqbal, M. Siddique, and J. Henkel, "Seal: soft error aware low power scheduling by monte carlo state space under the influence of stochastic spatial and temporal dependencies," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pp. 134–139, IEEE, 2011.

[2] X. Shih, H. Lee, K. Ho, and Y. Chang, "High variation-tolerant obstacle-avoiding clock mesh synthesis with symmetrical driving trees," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 452–457, IEEE Press, 2010.

[3] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel, "Reliable software for unreliable hardware: embedded code generation aiming at reliability," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 237–246, ACM, 2011.

[4] 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.

[5] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *IEEE International Conference on Computer Design, ICCD'11*, 2011.

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

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

[8] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *45th ACM/IEEE Design Automation Conference, DAC'08*, pp. 648–653, 2008.

[9] A. A. Markov, "Extension of the limit theorems of probability theory to a sum of variables connected in a chain," *reprinted in Appendix B of: R. Howard. Dynamic Probabilistic Systems, volume 1: Markov Chains. John Wiley and Sons*, 1971.

[10] G. Lorentz, *Bernstein Polynomials*. University of Toronto Press, 1953.

[11] G. Golub and C. Van Loan, *Matrix computations*, vol. 3. Johns Hopkins Univ Pr, 1996.

[12] 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, IEEE, 2012.

[13] 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.

[14] P. Li, W. Qian, and D. J. Lilja, "A stochastic reconfigurable architecture for fault-tolerant computation with sequential logic," in *IEEE International Conference on Computer Design, ICCD'12*, 2012.

[15] W. Ryan, "An introduction to ldpc codess," 2003.

[16] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 619–637, 2001.

[17] W. Qian, M. Riedel, K. Bazargan, and D. Lilja, "The synthesis of combinational logic to generate probabilities," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, pp. 367–374, ACM, 2009.