An Ultra-Fast Parallel Architecture Using Sequential Circuits Computing on Random Bits

Daran Cai, Ang Wang, Ge Song, and Weikang Qian
University of Michigan-Shanghai Jiao Tong University Joint Institute
Shanghai Jiao Tong University, Shanghai, China
Email:{darancai, wang0813sjtu, happy.songge}@gmail.com, qianwk@sjtu.edu.cn

***Abstract*—Digital computation on stochastic bit streams is a non-conventional type of computation, which uses digital circuits to process *probabilistic* inputs. In this paradigm, digital circuits compute on the probability values. This leads to extremely simple digital implementations for complex arithmetic operations. The combinational logic-based designs can be implemented in parallel to realize fast computation by trading off silicon area with delay. However, it is impossible to implement the current sequential logic-based designs in parallel to reduce delay, since they need to compute for a large number of clock cycles to obtain the result. In this work, we propose a novel design methodology for synthesizing sequential circuits that compute on stochastic bit streams, which does not require a long computation delay. We further demonstrate a parallel implementation based on our design, which provides ultra-fast arithmetic computation.**

I. Introduction and Background

Emerging nanoscale devices such as carbon nanotubes, molecular switches, and nanowire crossbars hold the promise for ultra-dense integration beyond CMOS, since they offer vast numbers of switches and bits at a small scale [1]–[3]. However, they suffer from high defect rates and exhibit randomness in their interconnections. Most researches on how to design circuits with these emerging nanoscale devices target at overcoming the randomness [1], [4].

A promising but different design strategy is to apply the paradigm of logical computation on random bits [5]. In this paradigm, ordinary digital circuits are employed but they operate on sequences of *random* bits instead of deterministic values. Indeed, a real value x in the unit interval $[0, 1]$ is represented by a sequence of random bits, each of which has probability x of being 1 and probability $1 - x$ of being 0. These bits can either be serial streaming on a single wire or in parallel on a bundle of wires. When serially streaming, the signals are probabilistic in *time*, as illustrated in Fig. 1(a); when in parallel, they are probabilistic in *space*, as illustrated in Fig. 2(a) [6]. With this type of computation, we can deliberately exploit the randomness provided by the nanoscale devices instead of overcoming it.

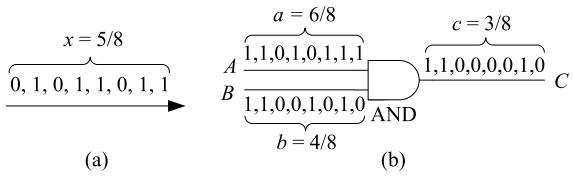

Fig. 1: Stochastic bit stream and computation on stochastic bit streams: (a) A stochastic bit stream encoding the value $x = 5/8$; (b) An AND gate multiplying two values encoded by two input stochastic bit streams.

When computing on sequences of random bits, a digital circuit essentially transforms input probability values into output probability values. This type of computation enables complex arithmetic operations to be implemented with simple circuits. It also has strong tolerance to bit-flip errors [7]. The major disadvantage of digital computation on stochastic bit streams is that it is subject to error due to stochastic variance. Thus, this type of computation is suitable for applications that do not demand high accuracy, such as image processing and computer vision.

Both combinational and sequential circuits can be applied to implement such computation. Basic arithmetic operations such as

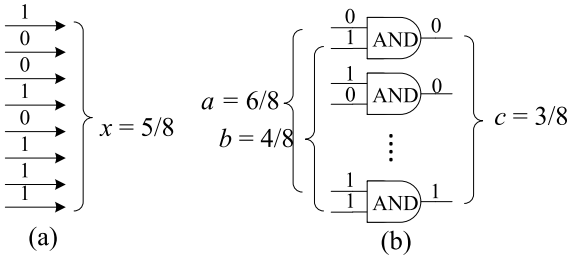

Fig. 2: Stochastic bit bundle and computation on stochastic bit bundles: (a) A stochastic bit bundle encoding the value $x = 5/8$; (b) A multiplier consisting of multiple AND gates which multiplies two values encoded by two stochastic bit bundles.

multiplication and addition can be implemented using combinational circuits [6], [8]. For example, with the serial stochastic encoding, multiplication can be implemented by a single AND gate, since the probability of obtaining a one in the output bit stream equals the product of the two input probabilities. This is shown in Fig. 1(b). In this specific example, the inputs and the output are represented by stochastic bit streams of length 8. Thus, it takes 8 clock cycles for the circuit to obtain the multiplication result.

Since the outputs of any combinational circuit only depend on its current inputs, we can transform the serial processing into parallel processing by copying the combinational circuit multiple times. Fig. 2(b) shows how we put AND gates in parallel to implement a multiplier on stochastic bit bundles. In this specific example, the multiplier takes only 1 clock cycle to get the result, but, as a trade-off, it needs 8 AND gates. Thus, by choosing between the serial implementation and the parallel implementation, we are able to trade off circuit area with computational delay or vice versa.

Besides combinational circuits, sequential circuits have also been proposed to compute on stochastic bit streams. A sequential circuit taking stochastic bit streams as inputs can be modeled as a time-homogeneous Markov chain. We can implement division and square root function using a counter-based sequential circuit [8], [9]. Recently, Brown and Card proposed a linear finite state machine (FSM)-based design to implement the sigmoid function and exponentiation function, which are widely used in artificial neural networks [8]. The FSM is shown in Fig. 3, which has N states arranged in a line and is controlled by a single input X. Li *et al.* generalized the work of Brown and Card. They proposed a method which can synthesize a sequential circuit based on the linear FSM to implement an arbitrary arithmetic function [10].

However, all the sequential logic-based designs proposed so far are all serial implementations; no parallel implementations have been proposed yet. The reason is that all the known design techniques are based on the "steady-state" probability distribution of the underlying Markov chain, which can only be reached by running the sequential circuit for a sufficiently large number of clock cycles. No methods have been proposed to implement a target function using a sequential design with a significantly reduced number of clock cycles. This

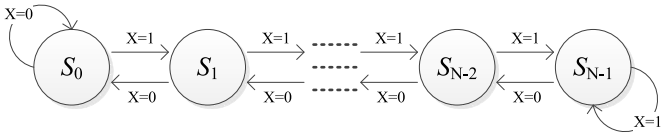

Fig. 3: The state transition diagram of a linear finite state machine proposed in [8].

limits the application of sequential circuits in implementing computation on random bits, since they can only be used when a long computational delay is allowed.

In this work, we propose a method to implement target computation using a sequential circuit that computes on random bits with a limited number of clock cycles. With the computational delay significantly reduced, we can implement the sequential circuit in parallel to trade off area for delay. Based on this, we propose a parallel architecture with the sequential circuit to implement arbitrary arithmetic functions. We compare our parallel stochastic implementation with the conventional implementations, showing that our system is significantly faster than the conventional implementations. Although subject to approximation error and stochastic variance, our system can implement an arbitrary target function very closely. It is also highly tolerant to bit flip errors.

II. Sequential Circuits Computing on Random Bits

Our sequential design is based on the FSM shown in Fig. 3, which consists of a set of N states $S_0, S_1, \ldots, S_{N-1}$ arranged in a linear form. We assume that this FSM is built with K flip-flops. Thus, we have $N = 2^K$. The FSM has a single input X, which takes a stream of random bits as input, with each bit having probability x of being 1. The state transition of the FSM depends only on the present state and the present input bit; it does not depend on the past states. Therefore, it can be modeled as a time-homogeneous Markov chain. Let $P_i(x,t)$ be the probability that the FSM is at the state S_i after t clock cycles. Based on the property of Markov chain, $P_i(x,t)$ can be recursively calculated as

$$P_i(x,t) = \sum_{i=0}^{N-1} P_j(x,t-1)q_{ji} \tag{1}$$

where q_{ji} denotes the conditional probability that the next state is i given the present state being j. With i as a row index and j as a column index, all the q_{ij}'s form an $N \times N$ *transition matrix* $\boldsymbol{Q}$. Based on the FSM shown in Fig. 3, the transition matrix is

$$\boldsymbol{Q} = \begin{bmatrix} 1-x & x & & & & \mathbf{0} \\ 1-x & 0 & x & & & \\ 0 & 1-x & 0 & x & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1-x & 0 & x \\ \mathbf{0} & & & & 1-x & x \end{bmatrix} \tag{2}$$

Let $\boldsymbol{P}(x,t) = [P_0(x,t), \ldots, P_{N-1}(x,t)]$ be the row vector denoting the probability distribution of all the states after t clock cycles. Given an initial uniform distribution over all the states, i.e., $\boldsymbol{P}(x,0) = [\frac{1}{N}, \ldots, \frac{1}{N}]$, we can rewrite the recursive relation (1) in matrix form as

$$\boldsymbol{P}(x,t) = \boldsymbol{P}(x,t-1) \cdot \boldsymbol{Q} = \boldsymbol{P}(x,0) \cdot \boldsymbol{Q}^t. \tag{3}$$

Our new design is based on the probability distribution over all the states of the FSM after a short number of clock cycles, say T clock cycles. From Eqn. (3), we can see that the probability $P_i(x,T)$ is a polynomial on x of degree T. Increasing N increases the number of terms in the distribution vector $\boldsymbol{P}(x,T)$, while increasing T increases the degree of each polynomial $P_i(x,T)$.

Example 1

Consider an FSM shown in Fig. 3 with $N = 4$ states. The transition matrix is

$$\boldsymbol{Q} = \begin{bmatrix} 1-x & x & 0 & 0 \\ 1-x & 0 & x & 0 \\ 0 & 1-x & 0 & x \\ 0 & 0 & 1-x & x \end{bmatrix}$$

With a uniform initial state distribution, if we choose $T = 3$, the probability distribution over all the states after T clock cycles is $\boldsymbol{P}(x,3) = [P_0(x,3), \ldots, P_3(x,3)]$, with

$$P_0(x,3) = x^2 - 2x + 1, \quad P_1(x,3) = x^3 - \frac{5}{2}x^2 + \frac{3}{2}x,$$
$$P_2(x,3) = -x^3 + \frac{1}{2}x^2 + \frac{1}{2}x, \quad P_3(x,3) = x^2.$$

III. The Parallel Architecture

Our design, shown in Fig 4, is a parallel implementation of the sequential construct proposed by Li *et al.*, which consists of an up/down counter and a multiplexer [10]. However, the parallel version faces one issue of how to make multiple copies of the input stochastic bit bundles. We introduce a circular shift register to solve this problem. We also empirically study the optimal design choice between the parameters T and N.

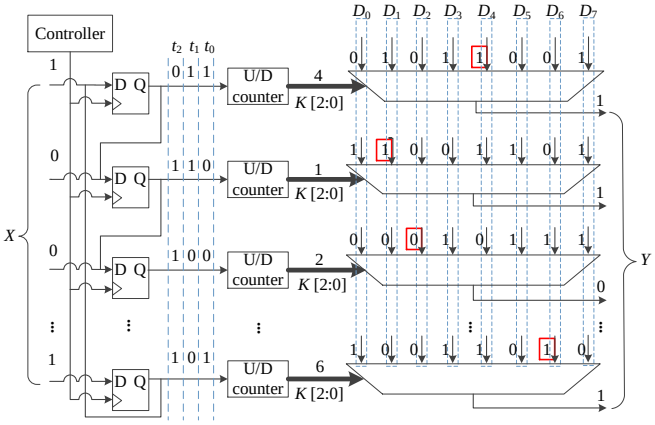

Fig. 4: The parallel architecture with sequential circuits. Each up/down counter implements the FSM shown in Fig. 3 with 8 states.

The Up/Down Counters: Each up/down counter in Fig. 4 is used to implement the state transition shown in Fig. 3. A stochastic bit stream X of length T is input into each counter. A bit 0 will decrease the counter by one and a bit 1 will increase it by one; this corresponds to the forward and backward state transition shown in Fig. 3. Also note that if the current state is the first one, a bit 0 will not lead to any transition. Similarly, a bit 1 will have no influence on the FSM if the current state is the last one. The output of each up/down counter is a K-bit binary number, encoding the current state number. The probability that a counter is at the i-th state after T clock cycles is given by $P_i(x,T)$.

The Multiplexers: The output of each up/down counter is connected to the selection input of a multiplexer (MUX). The MUX has N data inputs $D_0, \ldots, D_{N-1}$. Note that the i-th data input of each multiplexer has probability d_i to be one. If the output of a counter is i, then the MUX will choose its i-th data input as its output. Given that both the data inputs D_i to the multiplexer and the state S of an up/down counter after T clock cycles are random, the probability of each bit in the output bundle Y to be one can be calculated as

$$y = P(Y=1) = \sum_{i=0}^{N-1} P(Y=1|S=S_i)P(S=S_i).$$

By our design, $P(S = S_i) = P_i(x, T)$. Further, $P(Y = 1|S = S_i) = P(D_i = 1) = d_i$. Therefore, the probability of each bit in Y to be one can be expressed as

$$y = P(Y = 1) = \sum_{i=0}^{N-1} d_i P_i(x, T), \tag{4}$$

which is a linear combination on the polynomials $P_0(x, T), \ldots, P_{N-1}(x, T)$.

By choosing different sets of probabilities d_i, we can realize different polynomials on x. Given a target function $y = f(x)$, we can realize it by finding a set of d_i's so that the linear combination (4) gives the best approximation to $f(x)$. We use a technique proposed in [10] to find the optimal choice of d_i.

The Circular Shift Register: The parallel design takes a bundle of L random bits as inputs. We also require its output to be a bundle of the same width. Thus, the circuit contains L copies of the sequential construct, each generating a single output bit. However, since each counter is driven by a stream of T random bits, we need LT random bits in total. Thus, we need to reuse each input random bit in the bundle X for T times. We design a circular shift register shown in Fig. 4 for this purpose. The shift register consists of L D-flip-flops (DFFs), each holding one bit of the bundle X. A controller loads X into the shift register at the beginning and shifts the bits from top to bottom in the next T clock cycles. In each clock cycle, each bit stored in a DFF drives the corresponding counter. As a result, the k-th to the $(k + T - 1)$-th bits of the input bundle X drive the k-th counter.

Fig. 4 also illustrates how our proposed circuit works. Assume that the circuit starts at clock cycle t_0 and the parallel input X is $1, 0, 0, \ldots, 1$ as shown in the figure. In the next clock cycle t_1, the sequence stored in the shift register becomes $1, 1, 0, \ldots, 0$. The controller controls the shift for 3 clock cycles. Thus, there will be 3 bits sent into each counter. Also note that the initial state of each FSM is chosen randomly to generate a uniform initial state distribution. After 3 clock cycles, the output of each counter, which is equal to its final state number, is input into a multiplexer. Each bit of the output Y is selected from the corresponding bit of D_i. As shown in the figure, the up/down counters from top to bottom output $4, 1, 2, \ldots, 6$ in parallel. Given this, the first bit of D_4 is selected as the first bit of Y, the second bit of D_1 is selected as the second bit of Y, and so on. Therefore, the output Y is $1, 1, 0, \ldots, 1$.

The Optimal Choice between the Parameters T and N: T, the stop time, and N, the number of states of the FSM, are two parameters in our design. Since our design approximates a target function $f(x)$ through a linear combination of the probability functions $P_0(x, T), \ldots, P_{N-1}(x, T)$ (as given in Eqn. (4)), the choice of T and N may affect the approximation error. We empirically study the optimal choices between T and N by performing approximation on five randomly selected functions. We set $N = 8$ and vary T from 4 to 16. We quantify the approximation error using an objective function given in [10]:

$$\epsilon = \int_0^1 \left(f(x) - \sum_{i=0}^{N-1} d_i P_i(x, T) \right)^2 \mathrm{d}x \tag{5}$$

The plot of their approximation errors versus T is shown in Fig 5. We find that when $T = N - 1$ the approximation error reaches its minimum.

IV. Experimental Results

In our experiments, we set the width of the bit bundles as 1024. The FSM has $N = 8$ states and the number of clock cycles is $T = 7$.

A. Accuracy of the Computation

We first study the computation accuracy of our stochastic design, which is subject to two types of errors: 1) the approximation error

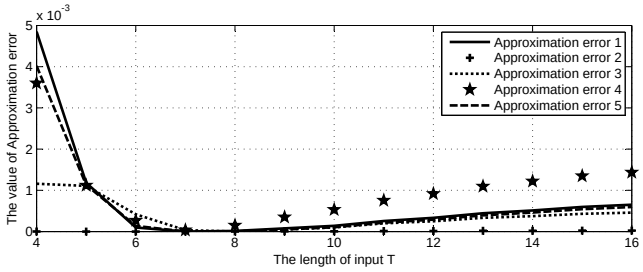

Fig. 5: The approximation errors versus different choices of T for five functions: $y_1 = x^{2.2}$; $y_2 = \frac{1}{4} + \frac{9}{8}x - \frac{18}{5}x^2 + \frac{5}{4}x^3$; $y_3 = \sin x \cdot (1 + 0.3\cos(5x))$; $y_4 = e^{-3x}$; and $y_5 = \tanh(2x)$. Here, the number of states N is 8.

through the linear combination shown in Eqn. (4), and 2) the error due to stochastic variance in the random sequences [7]. In stochastic encoding, although each bit in the sequence has probability p of being one, the actual number of ones in an observation of N bits is not guaranteed to be Np. It could be any integer in the range $[0, N]$. This kind of error is due to the stochastic variance in generating a sequence of random bits.

We randomly generate 50 polynomials which map the unit interval into the unit interval as our target functions. We restrict the degree of these polynomials to 7, since for approximating polynomials with higher degrees, the FSM with more than 8 states should be used. For each polynomial, we choose ten evaluation points $x = 0, 0.1, \ldots, 0.9$ and compare the ideal result, the approximation by the linear combination (4), and the stochastic simulation results. The stochastic simulation simulates the real operation condition of the hardware and thus contains the error due to stochastic variance.

As a result, the average absolute error between the approximation and the ideal value is 2.6×10^{-9}. The average absolute error between the stochastic simulation result and the ideal value is 1.25×10^{-2}. We can see that the average error due to the approximation by the linear combination is very small. However, there is a large difference between the stochastic simulation result and the ideal value, which is mainly caused by the error due to stochastic variance. We can decrease this error by making more copies of the basic circuit to increase the width of the bit bundles. However, for many applications that do not demand high accuracy, such as image processing and computer vision, this scale of error is still acceptable.

B. Delay Comparison

We compare the delay of our stochastic design to that of the conventional design based on binary radix. We consider computation of polynomials. We apply the approach used in [11] to evaluate the conventional design. Suppose that the polynomial is of degree n and the precision of the computation corresponds to M binary digits. Then, the delay of the conventional design is equal to $(12M - 11)n$ basic gate delays. In contrast, to compute a polynomial of degree n, the delay of our parallel stochastic implementation is equal to $(6n + 2)$ gate delays. For precision $M = 10$, a comparison of the delay of the conventional design with the delay of our proposed stochastic design is shown in Table I. From the table, we can see that our parallel implementation achieves a huge speedup over the conventional design.

One thing to mention is that in order to achieve the same precision, our stochastic design requires $2^M = 1024$ copies of the basic sequential construct. However, the design of the basic circuit is much smaller than the conventional designs using binary adders and multipliers. Given these facts, we estimate that the area increase of our design is less than two orders of magnitude than the conventional design. Hence, our proposed method can be viewed as trading off circuit area for speed.

TABLE I: Delay comparison of the conventional implementation and the proposed parallel stochastic implementation of polynomial arithmetic. n refers to the degree of a polynomial. The precision of the computation corresponds to 10 binary digits.

n	conv. delay	stoch. delay	stoch. delay/conv. delay
5	545	32	0.0587
6	654	38	0.0581
7	763	44	0.0577
8	872	50	0.0573
9	981	56	0.0571

C. Comparison of Circuit Performance on Noisy Input Data

We compare the performance of the conventional implementation and the FSM-based stochastic implementation on polynomial calculations when the input data are corrupted with noise. The width of the bit bundles of our stochastic implementation is 1024. To achieve the same precision, the conventional implementation operates on binary numbers of 10 bits.

We experiment on the same set of 50 polynomials used in Section IV-A. We set the error ratio of the input data to be $0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.15$ and 0.2, as measured by the fraction of random bit flips that occur. We obtain the absolute output errors of each implementation for all the polynomials and then average them. Fig. 6 compares the average absolute output errors for the two implementations.

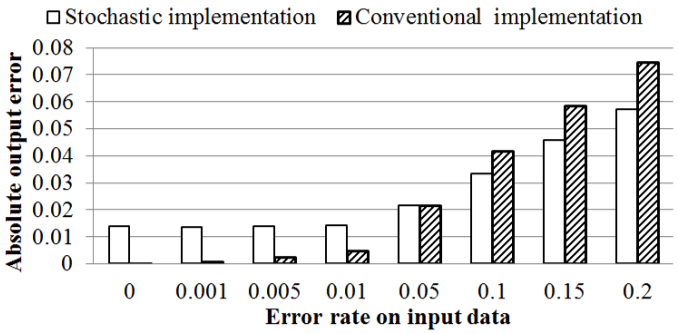

Fig. 6: Absolute output errors of the conventional implementation and the stochastic implementation for different error rates on the input data.

From Fig. 6, we can see that when the error rate is small, the stochastic implementation has larger output errors than the conventional implementation. However, the stochastic implementation is more tolerant to larger input errors than the conventional implementation.

We further analyze the distributions on the absolute output errors for the two implementations when the input error rate is 0.05, which leads to almost the same average output errors for the two different implementations. We plot the distribution of the absolute output errors for the two implementations in Fig. 7. From the figure, we can see that the stochastic implementation never generates output errors larger than 0.1, while the conventional implementation could generate error as large as 0.5. The reason for this is because the stochastic encoding is a uniform encoding. Thus, a single bit flip occurred any where does not have a significant influence on the encoded value. In contrast, conventional implementations operate on binary radix encoded values. Thus, if a bit flip occurs at the most significant bit, it will cause a large change to the value.

V. Conclusion and Future Work

In this paper, we propose a new method to design sequential circuits computing on stochastic bit streams to implement arithmetic computation. It synthesizes a target function based on the probability

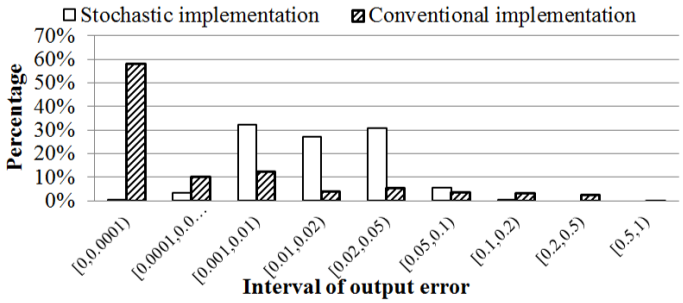

Fig. 7: The distributions of the absolute output errors for the conventional implementation and the stochastic implementation under input error rate of 0.05.

distribution over the FSM states after a limited number of clock cycles. With this, the delay of a sequential circuit computing on stochastic bit streams is greatly reduced. Therefore, it can be implemented in parallel to trade off the circuit area with the computation delay. We further demonstrate a parallel implementation of a linear FSM. Compared with conventional designs using binary radix encoding, our design is much faster. Also, our design is more fault tolerant to bit flip errors than conventional designs. In this work, we focus on a special linear FSM. However, the synthesis method and the architecture can also be applied to other FSMs. In our future work, we will study some other complex FSMs to find good choices that are more powerful in realizing different target functions.

Acknowledgement

This work is supported by a grant from the National Natural Science Foundation of China (NSFC), Project No. 61204042, and a grant from the Shanghai Jiao Tong University Undergraduate Innovation Program.

References

[1] N. Patil, J. Deng, A. Lin, H.-S. P. Wong, and S. Mitra, "Design methods for misaligned and mispositioned carbon-nanotube immune circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1725–1736, 2008.

[2] D. Goldhaber-Gordon, M. Montemerlo, J. Love, G. Opiteck, and J. Ellenbogen, "Overview of nanoelectronic devices," *Proceedings of the IEEE*, vol. 85, no. 4, pp. 521–540, 1997.

[3] A. DeHon, "Nanowire-based programmable architectures," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 1, no. 2, pp. 109–162, 2005.

[4] J. Han and P. Jonker, "A system architecture solution for unreliable nanoelectronic devices," *IEEE Transactions on Nanotechnology*, vol. 1, no. 4, pp. 201–208, 2002.

[5] W. Qian, J. Backes, and M. D. Riedel, "The synthesis of stochastic circuits for nanoscale computation," *International Journal of Nanotechnology and Molecular Computation*, vol. 1, no. 4, pp. 39–57, 2010.

[6] B. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*. Plenum, 1969, vol. 2, ch. 2, pp. 37–172.

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

[8] B. Brown and H. Card, "Stochastic neural computation I: Computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.

[9] S. Toral, J. Quero, and L. Franquelo, "Stochastic pulse coded arithmetic," in *International Symposium on Circuits and Systems*, vol. 1, 2000, pp. 599–602.

[10] P. Li, W. Qian, M. Riedel, K. Bazargan, and D. Lilja, "The synthesis of linear finite state machine-based stochastic computational elements," in *Asia and South Pacific Design Automation Conference*, 2012, pp. 757–762.

[11] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Design Automation Conference*, 2008, pp. 648–653.