The Synthesis of Stochastic Logic to Perform Multivariate Polynomial Arithmetic*

Weikang Qian and Marc D. Riedel
Department of Electrical and Computer Engineering,
University of Minnesota, Twin Cities
{qianx030, mriedel}@umn.edu

***Abstract*—As the feature size of integrated circuits scales to ever smaller regimes, maintaining the paradigm of deterministic Boolean computation is increasingly challenging. Indeed, mounting concerns over noise and uncertainty in signal values motivate a new approach: the design of stochastic logic, that is to say, digital circuitry that processes signals *probabilistically*, and so can cope with errors and uncertainty. In this paper, we present a general methodology for synthesizing stochastic logic for the computation of multivariate polynomials, a category that is important for applications such as digital signal processing. The method is based on converting polynomials into a particular mathematical form – multivariate Bernstein polynomials – and then implementing the computation with stochastic logic. The resulting logic processes serial or parallel streams that are random at the bit level. In the aggregate, the computation becomes *accurate*, since the results depend only on the precision of the statistics. Experiments show that our method produces circuits that are *highly tolerant of errors* in the input stream, while the area-delay product of the circuit is comparable to that of deterministic implementations.**

I. Introduction

The successful paradigm for integrated circuit design has been to maintain a sharp boundary in abstraction between the physical and logical layers. From the logic level up, the computation consists of a deterministic sequence of zeros and ones. The precise Boolean functionality of a circuit is prescribed; it is up to the physical layer to produce voltage values that can be interpreted as the exact logical values that are called for. This abstraction is firmly entrenched yet costly: variability, uncertainty, noise – all must be compensated for through ever more complex design and manufacturing. As technology continues to scale, with mounting concerns over noise and uncertainty in signal values, the cost of the abstraction is becoming untenable.

We are developing a framework for digital IC design based on the concept of *stochastic logic*. This paradigm has been known in the literature for many years [6]. Instead of computing with deterministic signals, operations at the logic level are performed on random serial or parallel bit streams. The streams are digital, consisting of zeros and ones; they are processed by ordinary logic gates, such as AND and OR. However, they convey values through the *statistical distribution* of the logical values. Real values in the interval [0, 1] correspond to the *probability of occurrence* of logical one versus logical zero in an observation interval. In this way, computations in the deterministic Boolean domain are transformed into probabilistic computations in the real domain.

Stochastic logic has the advantage that basic arithmetic operations can be performed with simple logic circuits [4]. It suffers from small estimation errors due to the inherent variance in the stochastic bit streams; however, this does not hinder its applications in areas like artificial neural networks and image processing where some inaccuracy can be tolerated [1], [3], [5].

Early work on the topic of stochastic logic discussed the implementation of basic arithmetic operations like multiplication, addition, division, etc., [4], [7], [11]. We have been studying the synthesis of stochastic logic more broadly. We described a procedure that, given a target Boolean function, builds a *multiplicative binary moment diagram* (*BMD) and then synthesizes the stochastic circuit from the *BMD [9]. Also, we described a general method for synthesizing stochastic logic to compute *univariate* polynomials [10].

In this work, we generalize the method in [10]: we propose a method for synthesizing stochastic logic to perform *multivariate* polynomial computation. Given a target power-form multivariate polynomial, we first convert it into a multivariate Bernstein polynomial. Then, we synthesize stochastic logic to compute that Bernstein polynomial. We present the results of synthesis trials for bivariate polynomials. The results show that our method produces circuits that are highly tolerant of errors, while the area-delay product of the circuit is comparable to that of deterministic implementations.

A. Mathematical Model of Stochastic Logic

In a conventional interpretation, a combinational circuit performs deterministic computation: the inputs are deterministic Boolean values and we expect the outputs to be also deterministic. In stochastic logic, the inputs to the circuit are random Boolean variables. Consequently, the outputs are also random Boolean variables, with probability distribution determined by the probability distribution of the inputs. (In general, stochastic logic can be implemented by sequential circuits Here, we only consider combinational circuits.) We can model stochastic logic as follows.

Assume that $y = f(x_1, x_2, \ldots, x_n)$ is an output Boolean function of a combinational logic circuit. Let $X_1, X_2, \ldots, X_n$

*This work is supported by a grant from the Semiconductor Research Corporation's Focus Center Research Program on Functional Engineered Nano-Architectonics, contract No. 2003-NT-1107.

be n *independent* random variables with Bernoulli distribution and assume that the probability of X_i being 1 is p_{X_i}. We write $P(X_i = 1) = p_{X_i}$ and $P(X_i = 0) = 1 - p_{X_i}$.

When the Boolean function has X_i's as its arguments, the result is also a random variable $Y = f(X_1, X_2, \ldots, X_n)$ with Bernoulli distribution. We assume that the probability of Y being 1 is p_Y. We write $P(Y = 1) = p_Y$ and $P(Y = 0) = 1 - p_Y$.

Evidently, p_Y is uniquely determined by the given n-tuple $(p_{X_1}, p_{X_2}, \ldots, p_{X_n})$, which we write as $p_Y = F(p_{X_1}, p_{X_2}, \ldots, p_{X_n})$. The function F is the computation performed by the stochastic logic.

In [10] we showed that F is a specific kind of multivariate polynomial on arguments $p_{X_1}, p_{X_2}, \ldots, p_{X_n}$: each product term of F has an integer coefficient and the degree of each variable in that term is less than or equal to 1. Mathematically, F is of the form

$$F(p_{X_1}, \ldots, p_{X_n}) = \sum_{i_1=0}^{1} \cdots \sum_{i_n=0}^{1} \left(\alpha_{i_1 \ldots i_n} \prod_{k=1}^{n} p_{X_k}^{i_k} \right), \quad (1)$$

where $\alpha_{i_1 \ldots i_n}$'s are integer coefficients.

As an example, we consider stochastic logic built on a multiplexer. The Boolean function of the multiplexer is

$$y = f(x_1, x_2, s) = (x_1 \wedge s) \vee (x_2 \wedge \neg s),$$

where $\wedge$ means logical AND, $\vee$ means logical OR and $\neg$ means logical negation. From the definition of p_Y, we have

$$\begin{aligned} p_Y &= F(p_{X_1}, p_{X_2}, p_S) \\ &= P(X_1 = 1, S = 1) + P(X_2 = 1, S = 0) \\ &= p_{X_1} p_S + p_{X_2}(1 - p_S) \\ &= p_{X_2} + p_{X_1} p_S - p_{X_2} p_S, \end{aligned} \quad (2)$$

which confirms that F is an integer-coefficient polynomial on the arguments p_{X_1}, p_{X_2} and p_S and the degree of each variable in each product term is less than or equal to 1.

B. Implementation of Stochastic Logic

If we want to implement stochastic logic based on a Boolean function $f(x_1, x_2, \ldots, x_n)$, we first build a combinational circuit implementing the Boolean function f. Then, we generate n independent stochastic bit streams $X_1, X_2, \ldots, X_n$, each consisting of N bits. Each bit in the stream X_i equals logical 1 with independent probability p_{X_i}. The stream is fed into the corresponding input x_i. Thus, in a statistical sense, each bit stream represents a random Boolean variable. In this way, when we measure the rate of the occurrence of 1 in the output bit stream, it gives us an estimate of p_Y. If the bit stream is sufficiently long, we can get an accurate estimate of p_Y. We assume that the input and output of the circuit are directly usable in this form. For instance, in sensor applications, analog voltage discriminating circuits might be used to transform real-valued input and output values into and out of probabilistic bit streams.

These bit streams may be *serial* or *parallel*. In serial streams, the random bits arrive sequentially in time. For parallel streams, we make N identical copies of the combinational logic circuit and feed independent random bits to each copy simultaneously. The choice between serial and parallel stochastic logic translates into a trade-off between time and area.

Figure 1 illustrates a serial implementation with two inputs and one output. The inputs and output are stochastic bit streams that are 8 bits in length. For the output, there are four 1's out of a total of 8 bits. Thus, the estimate is $p_Y = 4/8 = 0.5$.

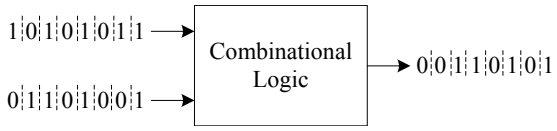

Fig. 1. A serial implementation of stochastic logic with inputs and outputs as serial bit streams.

II. Synthesis of Stochastic Logic for Multivariate Polynomial Arithmetic

Stochastic logic generally implements a specific type of multivariate polynomial F on arguments $p_{X_1}, p_{X_2}, \ldots, p_{X_n}$. If we associate some of the p_{X_i}'s in the polynomial $F(p_{X_1}, p_{X_2}, \ldots, p_{X_n})$ with real constants in the unit interval, some with a variable t_1, some with a variable t_2 and so on, then the function F becomes a real-coefficient multivariate polynomial. Specifically, if we set

$$\begin{aligned} &p_{X_1} = a_1, p_{X_2} = a_2, \cdots, p_{X_{r_1}} = a_{r_1}; \\ &p_{X_{r_1+1}} = \cdots = p_{X_{r_2}} = t_1; \\ &p_{X_{r_2+1}} = \cdots = p_{X_{r_3}} = t_2; \\ &\qquad \cdots \\ &p_{X_{r_d+1}} = \cdots = p_{X_n} = t_d, \end{aligned}$$

we obtain a real-coefficient multivariate polynomial $g(t_1, t_2, \ldots, t_d)$. For example, consider

$$\begin{aligned} y &= F(p_{X_1}, p_{X_2}, \ldots, p_{X_6}) \\ &= p_{X_1} p_{X_3} - p_{X_1} p_{X_3} p_{X_4} + p_{X_2} p_{X_5} p_{X_6} - p_{X_2} p_{X_3} p_{X_5} p_{X_6}. \end{aligned}$$

If we set $p_{X_1} = 0.4, p_{X_2} = 0.7, p_{X_3} = t_1, p_{X_4} = p_{X_5} = p_{X_6} = t_2$, then we obtain a multivariate polynomial

$$g(t_1, t_2) = 0.4t_1 - 0.4t_1 t_2 + 0.7t_2^2 - 0.7t_1 t_2^2.$$

With different choices of the original Boolean function f and different settings of the probabilities p_{X_i}'s, we obtain different polynomials $g(t_1, t_2, \ldots, t_d)$.

For applications, we need to perform the computation of specific polynomials. Given an arbitrary multivariate polynomial, how can we synthesize stochastic logic to implement it?

In what follows, we first introduce the concept of a multivariate Bernstein polynomial [2]. Next, we show that a multivariate Bernstein polynomial with coefficients in the unit interval can be computed by stochastic logic. Finally, we show how a general power-form polynomial can be converted into a Bernstein polynomial with coefficients in the unit interval.

A. Multivariate Bernstein Polynomial

Definition 1

The family of $n+1$ polynomials in the form

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \qquad i = 0, \ldots, n$$

are called **univariate Bernstein basis polynomials** *of degree n.*

Definition 2

The multiplication of univariate Bernstein basis polynomials gives a **multivariate Bernstein basis polynomial**, *i.e.,*

$$B_{i_1 \ldots i_d}^{n_1 \ldots n_d}(t_1, \ldots, t_d) = \prod_{k=1}^{d} B_{i_k}^{n_k}(t_k), \quad 0 \le i_k \le n_k, \text{for } k = 1, \ldots, d \tag{3}$$

is a multivariate Bernstein basis polynomial of degree $(n_1, \ldots, n_d)$.

Definition 3

A linear combination of multivariate Bernstein basis polynomials of degree $(n_1, \ldots, n_d)$

$$B^{n_1 \ldots n_d}(t_1, \ldots, t_d) = \sum_{i_1=0}^{n_1} \cdots \sum_{i_d=0}^{n_d} b_{i_1 \ldots i_d}^{n_1 \ldots n_d} B_{i_1 \ldots i_d}^{n_1 \ldots n_d}(t_1, \ldots, t_d) \tag{4}$$

is called a **Bernstein polynomial** *of degree $(n_1, \ldots, n_d)$. The $b_{i_1 \ldots i_d}^{n_1 \ldots n_d}$'s are the* **Bernstein coefficients**.

B. Stochastic Logic Computing Multivariate Bernstein Polynomial with Coefficients in the Unit Interval

If all the coefficients of a multivariate Bernstein polynomial are in the unit interval, i.e., $0 \le b_{i_1, \ldots, i_d}^{n_1 \ldots n_d} \le 1$, for all $0 \le i_1 \le n_1, \ldots, 0 \le i_d \le n_d$, then we can build stochastic logic to compute the multivariate Bernstein polynomial. Figure 2 shows the block diagram that implements the computation.

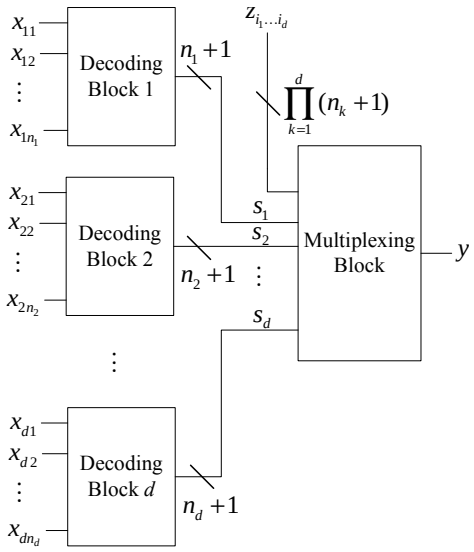

Fig. 2. Stochastic logic computing the multivariate Bernstein polynomial with coefficients in the unit interval.

The k-th decoding block in Figure 2 has n_k inputs $x_{k1}, x_{k2}, \ldots, x_{kn_k}$ and $n_k + 1$ outputs $s_{k0}, s_{k1}, \ldots, s_{kn_k}$. If i $(0 \le i \le n_k)$ out of n_k inputs of the k-th decoding block are logical 1, then s_{ki} is set to 1 and the other outputs are set to 0. Figure 3 gives the implementation of the decoding block with 8 inputs. The eight inputs are grouped into 4 pairs and each pair is fed into a 1-bit adder, which gives a 2-bit sum as the output. The 4 sets of outputs of the 1-bit adder are further grouped into 2 pairs and each pair is fed into a 2-bit adder, which gives a 3-bit sum as the output. The pair of outputs of the 2-bit adder is fed into a 3-bit adder, which gives a 4-bit sum as the output. Given an input binary number equal to k $(0 \le k \le 8)$, the decoder has its k-th output set to 1 and its other outputs set to 0. The output of the decoder gives the output signal s. A decoding block with n inputs can be implemented in a similar way.

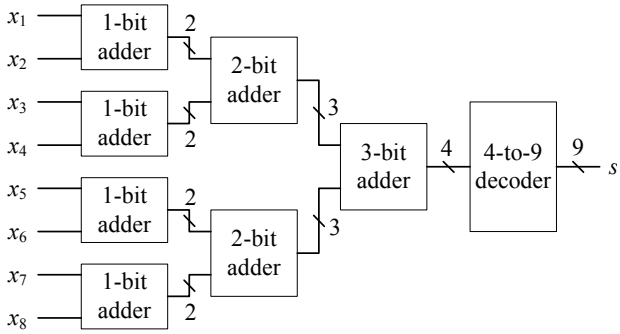

Fig. 3. The implementation of the decoding block.

The outputs of each decoding block are further fed into the multiplexing block and act as the selecting signals. The data signals of the multiplexing block consist of $\prod_{k=1}^{d}(n_k+1)$ inputs $z_{0 \ldots 0}, \ldots, z_{n_1 \ldots n_d}$. The Boolean logic of the multiplexing block is

$$y = \bigvee_{i_1=0}^{n_1} \cdots \bigvee_{i_d=0}^{n_d} \left(z_{i_1 \ldots i_d} \wedge \bigwedge_{k=1}^{d} s_{ki_k} \right). \tag{5}$$

The output of the multiplexing block y is set to be the input $z_{i_1 i_2 \ldots i_d}$ if and only if $s_{1i_1} = 1, s_{2i_2} = 1, \ldots, s_{di_d} = 1$. Since each decoding block has exactly one output signal set to 1, the multiplexing block selects one input data signal as its output.

Let $X_{11}, \ldots, X_{1n_1}, X_{21}, \ldots, X_{2n_2}, \ldots, X_{d1}, \ldots, X_{dn_d}$ be independent Boolean random variables with probability of being 1 set to

$$p_{X_{kj}} = t_k, \text{ for } 1 \le k \le d, 1 \le j \le n_k,$$

and feed these signals to the corresponding inputs of the decoding blocks. Thus, all the inputs to the k-th decoding block are independent Boolean random variables with the same probability t_k of being 1. Let S_{ki_k} denote the corresponding output random variable of the k-th decoding block. Since s_{ki_k} is set to 1 if and only if i_k out of n_k inputs of the k-th decoding block are 1, the probability of S_{ki_k} being 1 is

$$P(S_{ki_k} = 1) = \binom{n_k}{i_k} t_k^{i_k} (1-t_k)^{n_k - i_k} = B_{i_k}^{n_k}(t_k). \tag{6}$$

Let $Z_{0\ldots0}, \ldots, Z_{n_1\ldots n_d}$ be independent Boolean random variables with probability of being 1 set to

$$p_{Z_{i_1\cdots i_d}} = b^{n_1\ldots n_d}_{i_1\ldots i_d},\ 0 \le i_1 \le n_1, \ldots, 0 \le i_d \le n_d,$$

and feed these signals to the corresponding data inputs of the multiplexing block. Notice that we can set the probability value to be $b^{n_1\ldots n_d}_{i_1\ldots i_d}$ because we assume that $0 \le b^{n_1\ldots n_d}_{i_1\ldots i_d} \le 1$. Let Y denote the output random Boolean variable. Then, the probability of Y being 1 is

$$\begin{aligned} p_Y &= P(Y=1) \\ &= \sum_{i_1=0}^{n_1} \cdots \sum_{i_d=0}^{n_d} (P(Y=1|S_{1i_1}=1,\ldots,S_{di_d}=1) \\ &\cdot P(S_{1i_1}=1,\ldots,S_{di_d}=1)). \end{aligned} \tag{7}$$

Since when $S_{1i_1}=1, S_{2i_2}=1, \ldots, S_{di_d}=1$, Y equals $Z_{i_1\ldots i_d}$, we have

$$\begin{aligned} P(Y=1|S_{1i_1}=1,\ldots,S_{di_d}=1) &= P(Z_{i_1\ldots i_d}=1) \\ &= b^{n_1\ldots n_d}_{i_1\ldots i_d}. \end{aligned} \tag{8}$$

Based on the independence of the input random variables, as well as Equations (3) and (6), we have

$$\begin{aligned} P(S_{1i_1}=1,\ldots,S_{di_d}=1) &= \prod_{k=1}^{d} P(S_{ki_k}=1) \\ &= \prod_{k=1}^{d} B^{n_k}_{i_k}(t_k) = B^{n_1\ldots n_d}_{i_1\ldots i_d}(t_1,\ldots,t_d). \end{aligned} \tag{9}$$

Thus, from Equation (4), (7), (8) and (9), we have

$$\begin{aligned} p_Y &= \sum_{i_1=0}^{n_1} \cdots \sum_{i_d=0}^{n_d} b^{n_1\ldots n_d}_{i_1\ldots i_d} B^{n_1\ldots n_d}_{i_1\ldots i_d}(t_1,\ldots,t_d) \\ &= B^{n_1\ldots n_d}(t_1,\ldots,t_d), \end{aligned} \tag{10}$$

which means that the stochastic logic shown in Figure 2 implements the given multivariate Bernstein polynomial with coefficients in the unit interval. Thus, we have

Theorem 1

If all the coefficients of a multivariate Bernstein polynomial are in the unit interval, i.e., $0 \le b^{n_1\ldots n_d}_{i_1,\ldots,i_d} \le 1$, for all $0 \le i_1 \le n_1, \ldots, 0 \le i_d \le n_d$, then we can build stochastic logic to compute the multivariate Bernstein polynomial. □

C. Converting Multivariate Power-Form Polynomials into Multivariate Bernstein Polynomials

Polynomials that we are interested in are usually represented in the power form. If a multivariate power-form polynomial can be converted into a multivariate Bernstein polynomial whose coefficients are in the unit interval, then the power-form polynomial computation can be implemented by stochastic logic which computes the corresponding Bernstein polynomial. In this section, we show how to do this.

We will call a multivariate Bernstein polynomial converted from the original power-form polynomial an "equivalent Bernstein polynomial". We will say a multivariate power-form polynomial $g(t_1, t_2, \ldots, t_d)$ is of degree $(n_1, n_2, \ldots, n_d)$, if the degree of t_k in g is n_k, for all $1 \le k \le d$. We will say that degree $(n'_1, \ldots, n'_d)$ is *greater than* degree $(n_1, \ldots, n_d)$, if $n'_k \ge n_k$, for all $1 \le k \le d$ and, there is at least one $1 \le k \le d$ such that $n'_k > n_k$.

As shown in [2], a multivariate power-form polynomial

$$g(t_1,\ldots,t_d) = \sum_{i_1=0}^{n_1} \cdots \sum_{i_d=0}^{n_d} \left(a^{n_1\ldots n_d}_{i_1\ldots i_d} \prod_{k=1}^{d} t_k^{i_k} \right)$$

can be uniquely converted into a multivariate Bernstein polynomial of degree $(n_1, \ldots, n_d)$. The relationship between $b^{n_1\ldots n_d}_{i_1\ldots i_d}$ and $a^{n_1\ldots n_d}_{i_1\ldots i_d}$ is

$$b^{n_1\ldots n_d}_{i_1\ldots i_d} = \sum_{j_1=0}^{i_1} \cdots \sum_{j_d=0}^{i_d} \left(a^{n_1\ldots n_d}_{j_1\ldots j_d} \prod_{k=1}^{d} \frac{\binom{i_k}{j_k}}{\binom{n_k}{j_k}} \right), \tag{11}$$

for all $0 \le i_1 \le n_1, \ldots, 0 \le i_d \le n_d$.

An equivalent formula to Equation (11) is

$$b^{n_1\ldots n_d}_{i_1\ldots i_d} = \frac{1}{\prod_{k=1}^{d} \binom{n_k}{i_k}} \sum_{j_1=0}^{i_1} \cdots \sum_{j_d=0}^{i_d} \left(a^{n_1\ldots n_d}_{j_1\ldots j_d} \prod_{k=1}^{d} \binom{n_k - j_k}{i_k - j_k} \right). \tag{12}$$

A given multivariate power-form polynomial can also be uniquely converted into a multivariate Bernstein polynomial of degree greater than $(n_1, n_2, \ldots, n_d)$. One way to obtain the Bernstein coefficients of a higher degree Bernstein polynomial is to raise the degree of the original power-form polynomial by setting the coefficients of higher degree terms to be 0, and then applying Equation (12). Another way is to derive the coefficients of the higher degree Bernstein polynomial from those of the lower degree Bernstein polynomial, as shown by the following theorem.

Theorem 2

Suppose $g(t_1, \ldots, t_d)$ is a multivariate power-form polynomial of degree $(m_1, \ldots, m_d)$. Then, the coefficients of the equivalent Bernstein polynomials of degree $(n_1, \ldots, n_k+1, \ldots, n_d)$ $(m_1 \le n_1, \ldots, m_d \le n_d)$ can be derived from the coefficients of the equivalent Bernstein polynomials of degree $(n_1, \ldots, n_k, \ldots, n_d)$ based on the following recursive relation:

$$b^{n_1\ldots(n_k+1)\ldots n_d}_{i_1\ldots i_k\ldots i_d} = \begin{cases} b^{n_1\ldots n_k\ldots n_d}_{i_1\ldots 0\ldots i_d} & i_k = 0, \\ (1 - \dfrac{i_k}{n_k+1}) b^{n_1\ldots n_k\ldots n_d}_{i_1\ldots i_k\ldots i_d} & \\ \quad + \dfrac{i_k}{n_k+1} b^{n_1\ldots n_k\ldots n_d}_{i_1\ldots(i_k-1)\ldots i_d} & 1 \le i_k \le n_k, \\ b^{n_1\ldots n_k\ldots n_d}_{i_1\ldots n_k\ldots i_d} & i_k = n_k+1, \end{cases} \tag{13}$$

where

$$\begin{aligned} &0 \le i_1 \le n_1, \ldots 0 \le i_{k-1} \le n_{k-1}, \\ &0 \le i_{k+1} \le n_{k+1}, \ldots, 0 \le i_d \le n_d. \end{aligned}$$ □

Due to space limitations, we omit the proof.

If we apply Equation 13 l_1 times for $k=1$, l_2 times for $k=2$, ..., l_d times for $k=d$, we obtain all coefficients for the equivalent Bernstein polynomial of degree

$((n_1+l_1),\ldots,(n_d+l_d))$ from the coefficients of the equivalent Bernstein polynomial of degree $(n_1,\ldots,n_d)$.

Conversely, if we know the coefficients of the equivalent Bernstein polynomial of degree $(n_1,\ldots,n_k+1,\ldots,n_d)$, we can derive the coefficients of the equivalent Bernstein polynomial of degree $(n_1,\ldots,n_k,\ldots,n_d)$ in increasing order of i_k as

$$b_{i_1\ldots i_k\ldots i_d}^{n_1\ldots n_k\ldots n_d} = \begin{cases} b_{i_1\ldots 0\ldots i_d}^{n_1\ldots(n_k+1)\ldots n_d} & i_k = 0, \\ \dfrac{n+1}{n+1-i_k} b_{i_1\ldots i_k\ldots i_d}^{n_1\ldots(n_k+1)\ldots n_d} & \\ \quad -\dfrac{i_k}{n+1-i_k} b_{i_1\ldots(i_k-1)\ldots i_d}^{n_1\ldots n_k\ldots n_d} & 1 \le i_k \le n_k-1, \\ b_{i_1\ldots(n_k+1)\ldots i_d}^{n_1\ldots(n_k+1)\ldots n_d} & i_k = n_k. \end{cases} \tag{14}$$

Similarly, if we apply Equation 14 l_1 times for $k=1$, l_2 times for $k=2$, …, l_d times for $k=d$, we obtain all coefficients for the equivalent Bernstein polynomial of degree $(n_1,\ldots,n_d)$ from the coefficients of the equivalent Bernstein polynomial of degree $((n_1+l_1),\ldots,(n_d+l_d))$.

D. Synthesis of Stochastic Logic to Compute Multivariate Power-Form Polynomials

Since the input arguments $t_1,\ldots,t_d$ and the polynomial evaluation $g(t_1,\ldots,t_d)$ are represented by probability values in stochastic logic, we require that $0 \le g(t_1,\ldots,t_d) \le 1$, when $(t_1,\ldots,t_d) \in [0,1]^d$. However, even if the power-form polynomial satisfies $0 \le g(t_1,\ldots,t_d) \le 1$, when $(t_1,\ldots,t_d) \in [0,1]^d$, it is still not guaranteed that the polynomial computation can be implemented by stochastic logic. Nevertheless, if the polynomial evaluation is strictly greater than 0 and less than 1, i.e., $0 < g(t_1,\ldots,t_d) < 1$, for $(t_1,\ldots,t_d) \in [0,1]^d$, then we can implement the polynomial computation by stochastic logic. This is due to the following theorem.

Theorem 3
Suppose

$$g(t_1,\ldots,t_d) = \sum_{i_1=0}^{m_1}\cdots\sum_{i_d=0}^{m_d}\left(a_{i_1\ldots i_d}^{m_1\ldots m_d}\prod_{k=1}^{d} t_k^{i_k}\right)$$

is a multivariate power-form polynomial of degree $(m_1,\ldots,m_d)$. If $0 < g(t_1,\ldots,t_d) < 1$, for any $(t_1,\ldots,t_d) \in [0,1]^d$, then there exist $n_1 \ge m_1,\ldots,n_d \ge m_d$, such that the multivariate Bernstein polynomial of degree $(n_1,\ldots,n_d)$ equivalent to $g(t_1,\ldots,t_d)$ has all its coefficients in the unit interval. In other words, we can represent $g(t_1,\ldots,t_d)$ as

$$g(t_1,\ldots,t_d) = \sum_{i_1=0}^{n_1}\cdots\sum_{i_d=0}^{n_d} b_{i_1\ldots i_d}^{n_1\ldots n_d} B_{i_1\ldots i_d}^{n_1\ldots n_d}(t_1,\ldots,t_d),$$

with $0 \le b_{i_1\ldots i_d}^{n_1\ldots n_d} \le 1$, for all $0 \le i_1 \le n_1,\ldots,0 \le i_d \le n_d$. □

Due to the space limitations, we omit the proof.

Here, we should notice that the equivalent Bernstein polynomial with coefficients in the unit interval may have degree greater than that of the original power-form polynomial. For example, consider

$$g(t_1,t_2) = 0.9 - 1.6t_2 + 1.6t_2^2 - 0.8t_1 + 4t_1t_2 - 4t_1t_2^2,$$

a multivariate power-form polynomial of degree $(1,2)$. It can be verified that $0 < g(t_1,t_2) < 1$, for any $(t_1,t_2) \in [0,1]^2$.

If $g(t_1,t_2)$ is converted into a multivariate Bernstein polynomial of degree $(1,2)$, we have

$$\begin{aligned} g(t_1,t_2) &= 0.9B_{00}^{12}(t_1,t_2) + 0.1B_{01}^{12}(t_1,t_2) + 0.9B_{02}^{12}(t_1,t_2) \\ &\quad + 0.1B_{10}^{12}(t_1,t_2) + 1.3B_{11}^{12}(t_1,t_2) + 0.1B_{12}^{12}(t_1,t_2). \end{aligned}$$

Notice that the coefficient of $B_{11}^{12}(t_1,t_2)$ is $1.3 > 1$.

However, if $g(t_1,t_2)$ is converted into a multivariate Bernstein polynomial of degree $(1,3)$, we have

$$\begin{aligned} g(t_1,t_2) &= 0.9B_{00}^{13}(t_1,t_2) + \frac{11}{30}B_{01}^{13}(t_1,t_2) + \frac{11}{30}B_{02}^{13}(t_1,t_2) \\ &\quad + 0.9B_{03}^{13}(t_1,t_2) + 0.1B_{10}^{13}(t_1,t_2) + 0.9B_{11}^{13}(t_1,t_2) \\ &\quad + 0.9B_{12}^{13}(t_1,t_2) + 0.1B_{13}^{13}(t_1,t_2). \end{aligned}$$

All the Bernstein coefficients in the above equation are in the unit interval. Thus, an equivalent Bernstein polynomial to $g(t_1,t_2)$ that has all coefficients in the unit interval has degree greater than that of the power-form polynomial $g(t_1,t_2)$.

Based on Theorem 1 and 3, we have the following corollary, which gives a sufficient condition for the implementability of a multivariate power-form polynomial.

Corollary 1
If a multivariate power-form polynomial $g(t_1,\ldots,t_d)$ satisfies $0 < g(t_1,\ldots,t_d) < 1$, for any $(t_1,\ldots,t_d) \in [0,1]^d$, then the computation of that polynomial can be implemented by stochastic logic. □

If we are given a multivariate power-form polynomial

$$g(t_1,\ldots,t_d) = \sum_{i_1=0}^{m_1}\cdots\sum_{i_d=0}^{m_d}\left(a_{i_1\ldots i_d}^{m_1\ldots m_d}\prod_{k=1}^{d} t_k^{i_k}\right)$$

which satisfies $0 < g(t_1,\ldots,t_d) < 1$ for any $(t_1,\ldots,t_d) \in [0,1]^d$, then we can synthesize stochastic logic to compute the polynomial in the following steps:

1) Let $s=0$, $n_k = m_k$, for all $1 \le k \le d$. Get $b_{i_1\ldots i_d}^{n_1\ldots n_d}$'s from $a_{i_1\ldots i_d}^{m_1\ldots m_d}$'s based on Equation (11) or (12). Then, check to see if $0 \le b_{i_1\ldots i_d}^{n_1\ldots n_d} \le 1$, for all $0 \le i_1 \le n_1,\ldots,0 \le i_d \le n_d$. If so, go to step 5. Otherwise, let $p_k = n_k$, for all $1 \le k \le d$.
2) Let $s = s+1$.
3) While there is still a solution $(l_1,l_2,\ldots,l_d)$ to the equation $\sum_{k=1}^{d} l_k = s$ that we have not checked, go to step 4. Otherwise, go to step 2.
4) Let $n_k = m_k + l_k$, for all $1 \le k \le d$. Let $q_k = \min\{p_k, n_k\}$, for all $1 \le k \le d$. Since we know $b_{i_1,\ldots,i_d}^{p_1,\ldots,p_d}$, the coefficients of the equivalent multivariate Bernstein polynomial of degree $(p_1,\ldots,p_d)$, from the previous computation, we can obtain $b_{i_1,\ldots,i_d}^{q_1,\ldots,q_d}$ from $b_{i_1,\ldots,i_d}^{p_1,\ldots,p_d}$ by recursively applying Equation (14). Then, we can obtain $b_{i_1\ldots i_d}^{n_1\ldots n_d}$, the coefficients of the equivalent Bernstein polynomial of degree $(n_1,\ldots,n_d)$,

from $b_{i_1\ldots i_d}^{q_1\cdots q_d}$ by recursively applying Equation (13). Then, we check to see if $0 \le b_{i_1\ldots i_d}^{n_1\ldots n_d} \le 1$, for all $0 \le i_1 \le n_1, \ldots, 0 \le i_d \le n_d$. If so, go to step 5. Otherwise, Let $p_k = n_k$, for all $1 \le k \le d$ and go to step 3.

5) Build stochastic logic to compute the equivalent Bernstein polynomial of degree $(n_1, n_2, \ldots, n_d)$ as shown in Section II-B.

In the case that the polynomial $g(t_1, \ldots, t_d)$ is defined in the interval $[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$, where some a_k's may be less than 0 and some b_k's may be greater than 1, we can do a pre-processing on $t_1, t_2, \ldots, t_d$ by letting

$$t'_k = \frac{t_k - a_k}{b_k - a_k}, \text{ for } 1 \le k \le d. \tag{15}$$

Then, the objective polynomial is changed to

$$g'(t'_1, \ldots, t'_d) = g((b_1 - a_1)t'_1 + a_1, \ldots, (b_d - a_d)t'_d + a_d), \tag{16}$$

with $(t'_1, \ldots, t'_d)$ defined in interval $[0, 1]^d$.

If $g'(t'_1, \ldots, t'_d)$ evaluates in the interval $[u, v]$, where $u \le 0$ or $v \ge 1$, we need to further change our objective polynomial into

$$h(t'_1, \ldots, t'_d) = \frac{g'(t'_1, \ldots, t'_d) - u + \epsilon}{v - u + 2\epsilon}, \tag{17}$$

where ϵ is a small positive number. With the above linear transformation (17), $h(t'_1, \ldots, t'_d)$ evaluates in the interval $(0, 1)$, for any $(t'_1, \ldots, t'_d) \in [0, 1]^d$. Thus, we can build stochastic logic to compute $h(t'_1, \ldots, t'_d)$, by Corollary 1.

Corresponding to the linear transformation (17), we need to do a post-processing on the output of stochastic logic by letting

$$g = (v - u + 2\epsilon)h + u - \epsilon. \tag{18}$$

The final result is given by g.

III. Experimental Results

In our experiments, we use bivariate polynomials

$$g(t_1, t_2) = \sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} a_{i_1 i_2}^{n_1 n_2} t_1^{i_1} t_2^{i_2}$$

as our synthesis objective function. We first compare the hardware cost of deterministic digital implementations to that of stochastic implementations of the computation of bivariate polynomials. Then, we compare the performance of these two implementations on noisy input data.

A. Hardware Comparison

In a deterministic implementation of polynomial arithmetic, the data is generally encoded as a binary radix. We assume that the data consists of M bits, so the resolution of the computation is 2^{-M}.

We can rewrite a bivariate polynomial as

$$g(t_1, t_2) = \sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} a_{i_1 i_2}^{n_1 n_2} t_1^{i_1} t_2^{i_2} = \sum_{i_1=0}^{n_1} \left(\sum_{i_2=0}^{n_2} a_{i_1 i_2}^{n_1 n_2} t_2^{i_2} \right) t_1^{i_1}.$$

Define $a'_{i_1}(t_2) = \sum_{i_2=0}^{n_2} a_{i_1 i_2}^{n_1 n_2} t_2^{i_2}$. Then, $g(t_1, t_2)$ is a univariate polynomial on variable t_1 with coefficients $a'_{i_1}(t_2)$, and $a'_{i_1}(t_2)$ $(0 \le i_1 \le n_1)$ is a univariate polynomial on variable t_2. Since a univariate polynomial $g(t) = \sum_{i=0}^{n} a_i^n t^i$ can be factorized as $g(t) = a_0 + t(a_1 + t(a_2 + \cdots + t(a_{n-1} + ta_n)))$, we can evaluate the univariate polynomial in n iterations. In each iteration, a single addition and a single multiplication are needed. Therefore, we can evaluate the polynomial $g(t_1, t_2)$ in $(n_1 + 1)n_2 + n_1$ iterations, each of which consists of a single addition and a single multiplication. Hence, for such an iterative calculation, the hardware consists of an adder and a multiplier.

We adopt the same deterministic implementation as in [10], which contains $10M^2 - 4M - 9$ 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.

We build the implementation computing the bivariate Bernstein polynomial of degree (n_1, n_2) based on the circuit structure shown in Figure 2. Table I shows the area $A_d(n)$ and delay $D_d(n)$ of the decoding block for different number of inputs $n = 2, 3, 4, 5$ and 6. (When characterizing the area and delay, we assumed that the operation of each logic gate requires unit area and unit delay.) Each circuit is composed of the same four types of gates that we used in the deterministic implementation. For each specific value of n, we also properly designed the adder tree. For example, for $n = 3$, we used a full adder to construct the adder tree, since a full adder takes 3 inputs and gives a 2-bit sum.

TABLE I
The area and delay of the decoding block in Figure 2 for number of inputs $n = 2, 3, 4, 5$ and 6.

number of inputs n	area $A_d(n)$	delay $D_d(n)$
2	6	4
3	15	7
4	31	13
5	38	16
6	45	16

The multiplexing block could be implemented by $3(n_1 + 1)(n_2 + 1) - 1$ logic gates with a delay of $\lceil \log_2(n_1 + 1)(n_2 + 1) \rceil + 2$. Thus, stochastic logic implementing the bivariate Bernstein polynomial of degree (n_1, n_2) has area

$$A(n_1, n_2) = A_d(n_1) + A_d(n_2) + 3(n_1 + 1)(n_2 + 1) - 1,$$

and delay

$$D(n_1, n_2) = \max\{D_d(n_1), D_d(n_2)\} + \lceil \log_2(n_1 + 1)(n_2 + 1) \rceil + 2.$$

As stated in Section I-B, 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 when using a serial implementation; alternatively, we need 2^M *copies* when using a parallel implementation.

As a measure of hardware cost, we compute the area-delay product. Note that a serial implementation of stochastic logic takes less area and more delay; a parallel implementation takes

more area and less delay. In both cases, the area-delay product is the same.

The area-delay product of the deterministic implementation computing the bivariate polynomial of degree (n_1, n_2) is $(10M^2 - 4M - 9)(12M - 11)(n_1n_2 + n_1 + n_2)$, where $n_1n_2 + n_1 + n_2$ accounts for the number of iterations in the computation. The area-delay product of the stochastic implementation computing the Bernstein polynomial of degree (n_1, n_2) is $A(n_1, n_2)D(n_1, n_2)2^M$ – no matter whether implemented serially or in parallel

In Table II we compare the area-delay product for the deterministic implementation and the stochastic implementation for $(n_1, n_2) \in \{2, 3, \ldots, 6\}^2$ and $M = 8, 9, 10, 11$. Each cell in the main part of the table shows the ratio of the area-delay product of the stochastic implementation to that of the deterministic implementation for specific values of n_1, n_2, and M. We can see that when $M \leq 9$, the area-delay product of the stochastic implementation is less than that of the deterministic implementation and when $M \leq 11$, the area-delay product of the stochastic implementation is less than twice of that of the deterministic implementation.

TABLE II
THE RATIO OF THE AREA-DELAY PRODUCT OF THE STOCHASTIC IMPLEMENTATION TO THAT OF THE DETERMINISTIC IMPLEMENTATION OF POLYNOMIALS WITH DEGREE (n_1, n_2) AND RESOLUTION 2^{-M}.

M	n_1	n_2				
		2	3	4	5	6
8	2	0.239	0.333	0.553	0.660	0.653
	3	0.333	0.336	0.556	0.624	0.613
	4	0.553	0.556	0.570	0.630	0.639
	5	0.660	0.624	0.630	0.631	0.612
	6	0.653	0.613	0.639	0.612	0.593
9	2	0.328	0.457	0.759	0.906	0.897
	3	0.457	0.460	0.763	0.856	0.841
	4	0.759	0.763	0.782	0.865	0.877
	5	0.906	0.856	0.865	0.866	0.840
	6	0.897	0.841	0.877	0.840	0.814
10	2	0.469	0.654	1.086	1.296	1.284
	3	0.654	0.659	1.092	1.225	1.203
	4	1.086	1.092	1.120	1.238	1.255
	5	1.296	1.225	1.238	1.240	1.203
	6	1.284	1.203	1.255	1.203	1.166
11	2	0.695	0.968	1.608	1.920	1.901
	3	0.968	0.976	1.617	1.814	1.782
	4	1.608	1.617	1.658	1.833	1.859
	5	1.920	1.814	1.833	1.836	1.781
	6	1.901	1.782	1.859	1.781	1.726

B. Comparison of Circuit Performance with Noisy Input Data

We compare the performance on polynomial evaluations when the input data is corrupted with noise. Suppose that the input data of a deterministic implementation is $M = 10$ 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.

To illustrate that our method is applicable to digital signal processing, we choose the *Butterworth polynomial* as the objective function of our implementation.

Butterworth filters are commonly used in signal processing [8]. The transfer function of an n-th order low-pass Butterworth with cutoff frequency $\omega_c = 1$ can be represented as

$$H(s) = \frac{G_0}{B_n(s)}, \tag{19}$$

where G_0 is the DC gain (i.e., gain at zero frequency) and $B_n(s)$ is an n-th order *normalized Butterworth polynomial*.

The n-th order normalized Butterworth polynomial has the general form

$$B_n(s) = s^n + a_{n_1}s^{n-1} + \cdots + a_1 s + 1. \tag{20}$$

Table III gives the coefficients a_i of the first 6 nontrivial Butterworth polynomials.

TABLE III
COEFFICIENTS a_i OF THE FIRST 6 NONTRIVIAL BUTTERWORTH POLYNOMIALS. ALL COEFFICIENTS ARE UP TO FOUR DECIMAL PLACES.

n	a_1	a_2	a_3	a_4	a_5	a_6
2	1.4142	-	-	-	-	-
3	2.0000	2.0000	-	-	-	-
4	2.6131	3.4142	2.6131	-	-	-
5	3.2361	5.2361	5.2361	3.2361	-	-
6	3.8637	7.4641	9.1416	7.4641	3.8637	-
7	4.4940	10.0978	14.5918	14.5918	10.0978	4.4940

If we set s to be the complex angle frequency $\sigma + j\omega$ then each $B_n(\sigma + j\omega)$ can be represented as

$$B_n(\sigma + j\omega) = Re_n(\sigma, \omega) + jIm_n(\sigma, \omega),$$

where $Re_n(\sigma, \omega)$ and $Im_n(\sigma, \omega)$ are the real and imaginary part of $B_n(\sigma + j\omega)$, respectively.

In this experiment, we synthesize stochastic logic to implement the computation of 8 polynomials: Re_n and Im_n, for $n = 2, 3, 4$ and 5, which are all bivariate polynomials. Table IV shows the polynomials Re_2, Im_2, Re_3 and Im_3. Due to the space limitations, the other 4 polynomials are omitted.

TABLE IV
POLYNOMIAL Re_n AND Im_n, FOR $n = 2, 3, 4$ AND 5, THEIR DEGREES AND THEIR MINIMUMS AND MAXIMUMS IN THE INTERVAL $[0, 1]^2$.

poly. name	polynomial	degree of σ and ω	min	max
Re_2	$1-\omega^2+1.4142\sigma+\sigma^2$	$(2, 2)$	0	3.414
Im_2	$1.4142\omega + 2\sigma\omega$	$(1, 1)$	0	3.414
Re_3	$1-2\omega^2+2\sigma-3\sigma\omega^2+2\sigma^2+\sigma^3$	$(3, 2)$	-1.113	6
Im_3	$2\omega-\omega^3+4\sigma\omega+3\sigma^2\omega$	$(2, 3)$	0	8
Re_4	omitted	$(4, 4)$	-5.661	10.641
Im_4	omitted	$(3, 3)$	0	14.668
Re_5	omitted	$(5, 4)$	-23.180	18.944
Im_5	omitted	$(4, 5)$	-1.197	23.877

Table IV also shows the degree of each bivariate polynomial. Here, σ is treated as the first argument and ω as the second argument in the bivariate polynomial. In the experiment, we assume that these polynomials are defined in the interval $[0, 1]^2$. The minimums and maximums of these polynomials are also listed in Table IV. Since all of their minimums are less than or equal to 0 and all of their maximums are greater than 1, we need to apply Equation (17) to transform them into proper objective polynomials.

TABLE V
THE DEGREE OF THE EQUIVALENT MULTIVARIATE BERNSTEIN POLYNOMIAL WITH ALL COEFFICIENTS IN THE UNIT INTERVAL.

poly. name	degree of equiv. Bern. poly.	poly. name	degree of equiv. Bern. poly.
Re'_2	$(2,2)$	Re'_4	$(6,4)$
Im'_2	$(1,1)$	Im'_4	$(3,3)$
Re'_3	$(4,2)$	Re'_5	$(5,4)$
Im'_3	$(2,3)$	Im'_5	$(4,14)$

We apply the synthesis steps 1 to 4 given in Section II-D to obtain the equivalent multivariate Bernstein polynomial with coefficients in the unit interval. Table V gives the degree of those equivalent Bernstein polynomials. From Table V, we can see that the degrees of the Bernstein polynomials equivalent to Re_2, Im_2, Im_3, Im_4 and Re_5 equal that of the original power-form polynomials. However, the degrees of the Bernstein polynomials equivalent to Re_3, Re_4 and Im_5 are higher than those of the original power-form polynomials.

We evaluated each Butterworth polynomial on 169 points: $(t_1, t_2) \in \{0.2, 0.25, 0.3, \ldots, 0.8\}^2$. For each error ratio ϵ, each objective polynomial, and each evaluation point, we simulated both the stochastic and the deterministic implementations 5000 times. We averaged the relative errors over all simulations. Finally, for each error ratio ϵ, we averaged the relative errors over all polynomials and all evaluation points.

Table VI shows the average relative error of the stochastic implementation and the deterministic implementation versus different error ratios ϵ. This data is plotted in Figure 4.

TABLE VI
RELATIVE ERROR FOR THE STOCHASTIC AND DETERMINISTIC IMPLEMENTATION OF BUTTERWORTH POLYNOMIAL COMPUTATION VERSUS THE ERROR RATIO ϵ IN THE INPUT DATA.

error ratio ϵ	rel. error of stoch. impl.(%)	rel. error of deter. impl.(%)
0.0	3.54	0.00
0.001	3.55	0.97
0.002	3.57	1.93
0.005	3.70	4.74
0.01	4.11	9.16
0.02	5.48	17.06
0.05	10.92	36.87
0.1	20.33	62.44

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

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

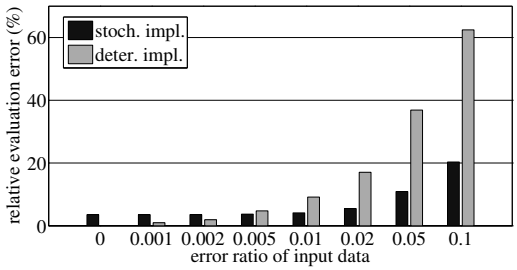

Fig. 4. A plot of the relative error for the stochastic and the deterministic implementation of Butterworth polynomial computation versus the error ratio ϵ in the input data.

IV. CONCLUSION AND FUTURE WORK

In this work, we provide a general method to synthesize stochastic logic to compute an arbitrary multivariate polynomial. The synthesis results for the stochastic implementation of polynomial arithmetic are convincing. The area-delay product is comparable to that of deterministic implementations with adders and multipliers. However, the circuits are much more error tolerant. The precision of the results is dependent only on the statistics of the streams that flow through the datapaths, and so the computation can tolerate errors gracefully. Here we have only considered stochastic logic obtained from combinational circuits. In future work, we will generalize the synthesis methodology to stochastic logic implemented by sequential circuits with random Boolean inputs.

REFERENCES

[1] Z. Asgar, S. Kodakara, and D. Lilja, "Fault-Tolerant Image Processing using Stochastic Logic," *Technical Report*,http://www.zasgar.net/zain/publications/publications.php, 2005.

[2] J. Berchtold and A. Bowyer, "Robust Arithmetic for Multivariate Bernstein-form Polynomials," *Computer-Aided Design*, Vol. 32, No. 11, pp. 681–689, 2000.

[3] C.M. Bishop, *Neural Networks for Patten Recognition*, Clarendon Press, 1995.

[4] B. Brown and H. Card, "Stochastic Neural Computation I: Computational Elements," *IEEE Transactions on Computers*, Vol. 50, No. 9, pp. 891–905, 2001.

[5] S.R. Deiss, R.J. Douglas, and A.M. Whatley, "A Pulse Coded Communications Infrastructure for Neuromorphic Systems," *Pulsed Neural Networks*, W. Maass and C.M. Bishop, eds., *MIT Press*, 1999.

[6] B.R. Gaines, "Stochastic Computing Systems," *Advances in Information Systems Science*, J.F. Tou, ed., Vol. 2, pp. 37–172, *Plenum*, 1969.

[7] C.L. Janer, J.M. Quero, J.G. Ortega, and L.G. Franquelo, "Fully Parallel Stochastic Computation Architecture," *IEEE Transactions on Signal Processing*, Vol. 44, No. 8, pp. 2110–2117, 1996.

[8] T.W. Parks and C.S. Burrus, *Digital Filter Design,* John Wiley & Sons, 1987.

[9] W. Qian, J. Backes and M.D. Riedel, "The Synthesis of Stochastic Circuits for Nanoscale Computation," *International Workshop on Logic and Synthesis*, pp. 176–183, 2007.

[10] W. Qian and M.D. Riedel, "The Synthesis of Robust Polynomial Arithmetic with Stochastic Logic," *Design Automation Conference*, Anaheim, CA, 2008.

[11] S.L. Toral, J.M. Quero and L.G. Franquelo, "Stochastic Pulse Coded Arithmetic," *International Symposium on Circuits and Systems*, Vol. 1, pp. 599–602, 2000.