Polynomial Arithmetic Using Sequential Stochastic Logic

Naman Saraf and Kia Bazargan
Department of Electrical and Computer Engineering
University of Minnesota Twin Cities
200 Union St SE, Minneapolis, Minnesota, USA 55455
{saraf012,kia}@umn.edu

ABSTRACT

We present the design of stochastic computing systems based on sequential logic to implement arbitrary polynomial functions. Stochastic computing is an emerging alternative computing paradigm that performs arithmetic operations on real-valued data represented as random bitstreams using digital logic gates. Stochastic computing systems are capable of realizing complex mathematical operations using a small number of hardware resources by expressing the computation in terms of probabilities. Moreover, the stochastic representation of data using random bitstreams is extremely robust against bit errors. We present a systematic approach to implement arbitrary polynomial functions in stochastic computing using sequential logic, and compare our approach against prior conventional and stochastic implementations.

1. INTRODUCTION

CMOS technology has been instrumental in the proliferation of digital computing with submicron CMOS transistors realizing complex computing systems operating at GHz clock frequencies. However, concerns about increasing power dissipation and declining reliability in recent nanometer scale CMOS technologies have prompted research into unconventional computing paradigms. Stochastic computing [2] is an emerging alternative to conventional computing that represents data as random bitstreams, and performs arithmetic operations using digital logic gates by expressing the computation in terms of probabilities. Stochastic computing systems, in general, have lower hardware requirements than conventional systems, and the random bitstream representation improves the fault tolerance of the computation. We present the design of stochastic computing systems based on sequential logic to implement arbitrary polynomial functions, and compare our proposed approach against prior conventional and stochastic implementations.

As previously mentioned, stochastic computing relies on random bitstreams to represent data. In particular, a real number x ($0 \leq x \leq 1$) is represented in a stochastic comput-

Figure 1: Addition and multiplication operations in stochastic computing.

ing system as a sequence of random bits, where each bit in the sequence is logic ‘1’ with a probability x . The random bitstream representation of numbers enables us to express a computation in terms of probabilities. As an example, consider a two-input AND gate driven by **independent** random bitstreams A and B representing real numbers a and b .

$$Prob(A = 1) = a, \quad Prob(B = 1) = b \quad (1)$$

The real number c represented by the output random bitstream of the AND gate is given by,

$$Prob(C = 1) = c = Prob(A = 1, B = 1) \quad (2)$$

$$\Rightarrow Prob(A = 1)Prob(B = 1) = ab \quad (3)$$

A two-input AND gate, therefore, performs multiplication in stochastic computing. Similarly, a 2 : 1 multiplexer performs scaled addition, as depicted in Figure 1. The stochastic implementations of addition and multiplication, therefore, require far fewer hardware resources than a conventional implementation. The simple illustrations likewise extend to more complex arithmetic operations in image processing [8, 5] and digital communications [11].

The random bitstream representation of numbers, moreover, is extremely robust against bit errors. Consider an M -bit weighted binary number system, representing 2^M fixed-point numbers, $0.b_1 b_2 \dots b_M$. An error of magnitude $|1/2|$ results when the most significant bit b_1 of a fixed-point number is corrupted by noise. In contrast, the random bitstream representation is unweighted. Specifically, a random bitstream of length L , $[b_1 b_2 \dots b_L]$, represents the number $p = \sum_{i=1}^L b_i/L$. Therefore, a smaller error of magnitude $1/L$ results if a bit in the random bitstream is erroneous.

The high fault tolerance and low hardware overhead have fueled a growing interest in stochastic computing with an

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GLSVLSI'16, May 18-20, 2016, Boston, MA, USA

© 2016 ACM. ISBN 978-1-4503-4274-2/16/05...\$15.00

DOI: <http://dx.doi.org/10.1145/2902961.2902981>

increasing number of successful stochastic implementations of artificial neural networks [4], digital filters [6], and image processing systems [8, 5]. We present the design of stochastic computing systems based on sequential logic to realize arbitrary polynomial functions.

2. PRIOR RESEARCH

Polynomials are an important class of functions that frequently arise in numerous scientific disciplines, and are often employed to approximate other complicated functions such as the trigonometric functions. Specifically, a polynomial function $f(x)$ of degree n is an expression of the form,

$$f(x) = \sum_{i=0}^n a_i x^i \quad (4)$$

where x is the known as the input variable and a_i are the $(n + 1)$ polynomial coefficients.

A previous stochastic implementation of polynomial functions was presented in [9], where the authors described a method to realize arbitrary polynomial functions using purely combinational logic circuits. However, a serious drawback of their proposed architecture (Figure 2) is the requirement of **multiple independent** random bitstreams for the input variable x . Concretely, their architecture requires at least n independent random bitstreams representing the input variable x , in order to implement a polynomial function of degree n . The linear dependence of the input interface complexity on the polynomial degree presents a significant hardware overhead as multiple independent random bitstreams ought to be generated corresponding to the input variable x , thereby restricting the practical applicability of the approach to low degree polynomials.

In contrast, we propose sequential logic circuits to implement polynomial functions in stochastic computing. Specifically, we present finite state machines (FSMs) in the form of binary counters that realize arbitrary polynomial functions. Our proposed approach requires an FSM containing at least $(n + 1)$ states, and a **single** random bitstream for the input variable x , in order to implement a polynomial function of degree n . Using a binary encoding for the FSM states, we notice that the apparent linear dependence of the FSM complexity on the polynomial degree only translates into a logarithmic dependence in hardware area. As an example, a 7^{th} degree polynomial function is realized by an 8-state FSM in our approach that only requires 3 flip flops. However, of greater significance is the massive simplification in the input interface, and the corresponding reduction in the auxiliary circuitry that is required to generate the multiple independent input random bitstreams in the purely combinational solution. To summarize, our proposed approach realizes an arbitrary polynomial function of degree n using a single random bitstream for the input variable x that drives an FSM containing at least $(n + 1)$ states.

3. BERNSTEIN POLYNOMIALS

The theory of Bernstein polynomials [7] forms the bedrock of the stochastic implementations of polynomial functions. In particular, the Bernstein **basis** polynomials of degree n are a family of $(n + 1)$ polynomials given by,

$$B_{i,n}(x) = \binom{n}{i} x^i (1-x)^{n-i}, \quad i = 0, 1, \dots, n \quad (5)$$

Figure 2: A stochastic implementation of a degree n polynomial using combinational logic [9].

and a linear combination of the Bernstein basis polynomials, $g_n(x)$, is called a Bernstein polynomial of degree n .

$$g_n(x) = \sum_{i=0}^n b_i B_{i,n}(x) \quad (6)$$

where b_i are known as the $(n + 1)$ Bernstein coefficients.

In general, a **power-form** polynomial of degree n , as depicted in Equation (4), has an equivalent representation as a Bernstein polynomial of degree n , where the polynomial coefficients a_i are related to the Bernstein coefficients b_i as,

$$b_i = \sum_{j=0}^i \binom{i}{j} a_j, \quad 0 \leq i \leq n \quad (7)$$

The Bernstein coefficients for an arbitrary polynomial, however, seldom lie inside the unit interval $[0, 1]$ as required for a stochastic implementation. Fortunately, by elevating the degree of the Bernstein polynomial representation, we can express a power-form polynomial of degree n as a Bernstein polynomial of degree q ($q \geq n$) with the Bernstein coefficients $b_i \in [0, 1]$. In fact, due to a theorem [10], a power-form polynomial $f(x)$ of degree n can be converted into a Bernstein polynomial of degree q ($q \geq n$) with the Bernstein coefficients inside the unit interval $[0, 1]$ if,

1. $f(x) = 0$ or $f(x) = 1 \quad \forall x \in [0, 1]$, or
2. $f(x) \in (0, 1) \quad \forall x \in (0, 1)$ and $0 \leq f(0), f(1) \leq 1$

The details of the conversion method are provided in [9]. Following a successful conversion of a power-form polynomial to a Bernstein polynomial, we proceed with the stochastic implementation of the Bernstein representation by,

1. generating the Bernstein basis polynomials (Equation 5),
2. forming a linear combination of the Bernstein basis polynomials with the Bernstein coefficients (Equation 6)

4. MARKOV CHAINS

Before describing our stochastic implementation of polynomial functions, we present a brief review of Markov chains. Markov chains are crucial to the understanding of sequential logic based stochastic computing systems because an FSM with random bitstream inputs is mathematically equivalent to a Markov chain.

A Markov chain is a random process that models probabilistic transitions between states on a **state space**. We restrict our attention to **finite-state** Markov chains since only finite memory systems are physically realizable.

Consider a finite-state Markov chain with a state space S containing $(n + 1)$ states,

$$S = \{0, 1, \dots, n\} \quad (8)$$

Let X_t denote the state of the Markov chain at time instant t . The next state of the Markov chain, X_{t+1} , depends only on X_t , and is independent of the past system history.

$$Pr(X_{t+1} = j | X_t = i, \dots, X_0 = s_0) \quad (9)$$

$$= Pr(X_{t+1} = j | X_t = i) = p_{i,j}, \forall t, \forall i, j \in S \quad (10)$$

The probability that a Markov chain transitions from a state i to another state j in the state space is known as the **transition probability**, and is denoted as $p_{i,j}$.

$$p_{i,j} \geq 0, \forall i, j \in S; \sum_{j=0}^n p_{i,j} = 1, \forall i \in S \quad (11)$$

Contrary to the transition probabilities $p_{i,j}$, $\forall i, j \in S$ that capture the probabilistic transitions between the states, the **stationary probabilities** π_i , $\forall i \in S$ of a Markov chain determine the probability of observing the states at equilibrium. Concretely, starting from a random initial state, a Markov chain reaches equilibrium after making a number of state transitions, at which point the probability of observing the Markov chain in a particular state i becomes a constant π_i , independent of the initial state.

$$\pi_i \geq 0, \forall i \in S; \sum_{i=0}^n \pi_i = 1 \quad (12)$$

The relationship between the stationary probabilities π_i and the transition probabilities $p_{i,j}$ of a Markov chain, in general, is complicated. However, by restricting a Markov chain to a **linear** arrangement of states with only nearest neighbor transitions, as illustrated in Figure 3, a powerful set of **detailed balance conditions** become available. The detailed balance conditions relate the stationary probabilities of neighboring states to the transition probabilities between the states. Specifically,

$$\pi_i p_{i,i+1} = \pi_{i+1} p_{i+1,i}, \quad i = \{0, 1, \dots, n-1\} \quad (13)$$

$$\implies \frac{\pi_{i+1}}{\pi_i} = \frac{p_{i,i+1}}{p_{i+1,i}} = r_{i+1,i} \quad (14)$$

We notice that the stationary probabilities of neighboring states in a **Reversible** Markov chain only depend on the ratio of the transition probabilities between the states.

To summarize, the stationary probabilities of a Reversible Markov chain are conveniently obtained by applying Equation 12 and the detailed balance conditions listed in Equation 14. We next describe a specific Reversible Markov chain construct where the stationary probabilities equal the Bernstein basis polynomials.

Figure 3: A 3-state Reversible Markov chain. The vertex labels denote the state while the edge annotations depict the transition probabilities.

5. THE STOCHASTIC INTEGRATOR

Consider the $(n + 1)$ -state Reversible Markov chain depicted in Figure 4. Using Equation 14, the stationary probabilities of the Reversible Markov chain are related to the transition probabilities as,

$$\begin{aligned} \frac{\pi_{i+1}}{\pi_i} &= \frac{x}{1-x} \frac{n-i}{i+1} \\ \implies \pi_{i+1} &= \pi_i \frac{x}{1-x} \frac{n-i}{i+1}, \quad 0 \leq i \leq n-1 \end{aligned} \quad (15)$$

Let us select π_0 as the reference stationary probability.

$$\pi_0 = \binom{n}{0} \left[\frac{x}{1-x} \right]^0 \pi_0 \quad (16)$$

$$\pi_1 = \pi_0 \frac{x}{1-x} n = \binom{n}{1} \left[\frac{x}{1-x} \right]^1 \pi_0 \quad (17)$$

$$\begin{aligned} \pi_2 &= \pi_1 \frac{x}{1-x} \frac{n-1}{2} = \pi_0 \left[\frac{x}{1-x} \right]^2 \frac{n(n-1)}{2} \\ &= \binom{n}{2} \left[\frac{x}{1-x} \right]^2 \pi_0 \end{aligned} \quad (18)$$

...

$$\pi_i = \binom{n}{i} \left[\frac{x}{1-x} \right]^i \pi_0 \quad (19)$$

...

$$\pi_n = \binom{n}{n} \left[\frac{x}{1-x} \right]^n \pi_0 \quad (20)$$

Invoking Equation 12,

$$\sum_{i=0}^n \pi_i = \pi_0 \sum_{i=0}^n \binom{n}{i} \left[\frac{x}{1-x} \right]^i = 1 \quad (21)$$

and solving for the reference stationary probability π_0 ,

$$\pi_0 = \frac{1}{\sum_{i=0}^n \binom{n}{i} \left[\frac{x}{1-x} \right]^i} = \frac{1}{\left[1 + \frac{x}{1-x} \right]^n} = [1-x]^n \quad (22)$$

Therefore, the stationary probabilities of the Reversible

Figure 4: Reversible Markov chain representation of the stochastic integrator. The transition probabilities on the self-loops are omitted for clarity.

Markov chain in Figure 4 are given as,

$$\pi_0 = [1 - x]^n = \binom{n}{0} x^0 (1 - x)^n \quad (23)$$

$$\pi_1 = \binom{n}{1} \left[\frac{x}{1-x} \right]^1 [1-x]^n = \binom{n}{1} x^1 (1-x)^{n-1} \quad (24)$$

...

$$\pi_i = \binom{n}{i} x^i (1-x)^{n-i} \quad (25)$$

...

$$\pi_n = \binom{n}{n} x^n (1-x)^0 \quad (26)$$

Comparing with Equation 5, we conclude that the stationary probabilities of the Reversible Markov chain in Figure 4 form the Bernstein basis polynomials of degree n .

A hardware realization of the Reversible Markov chain in Figure 4 is a **stochastic integrator** [3] with parameter n . As illustrated in Figure 5, a stochastic integrator consists of an up/down counter, a binary comparator, and source of random numbers. The input to the stochastic integrator is a random bitstream X representing the input variable x .

The up/down counter of a stochastic integrator with parameter n stores binary numbers from 0 through n in unit integer increments, and saturates at the maximum or minimum count. A summary of the up/down counter operation is presented in Table 1.

The binary comparator compares the present counter state with a random number sampled from a uniform distribution on the integers $\{1, 2, \dots, n\}$, and produces a logic '1' bit if the counter state equals or exceeds the random number. Thus, if the present counter state is i , $0 \leq i \leq n$, the probability of the comparator output to be logic '1' is given by,

$$Prob(Z = 1) = \frac{i}{n} \implies Prob(Z = 0) = \frac{n-i}{n} \quad (27)$$

Consequently, the probability that the up/down counter increments to $i + 1$ at the next clock becomes,

$$\begin{aligned} p_{i,i+1} &= Prob(X = 1, Z = 0) \\ &= Prob(X = 1) Prob(Z = 0) = x \frac{n-i}{n} \end{aligned} \quad (28)$$

Similarly, the probability of the counter state to decrement to $i - 1$ is given by,

$$\begin{aligned} p_{i,i-1} &= Prob(X = 0, Z = 1) \\ &= Prob(X = 0) Prob(Z = 1) = (1-x) \frac{i}{n} \end{aligned} \quad (29)$$

where we assume that the random bitstreams X and Z are independent.

Table 1: Operation of the up/down counter.

up	down	present state	next state	probability
1	0	i	$i + 1$	$x(n-i)/n$
0	1	i	$i - 1$	$(1-x)i/n$
1	1	i	i	xi/n
0	0	i	i	$(1-x)(n-i)/n$

Figure 5: A hardware realization of the stochastic integrator.

Comparing the probabilities of the up/down counter to increment (Equation 28) and decrement (Equation 29) with the transition probabilities of the Reversible Markov chain in Figure 4, we ascertain that a stochastic integrator with parameter n implements the Bernstein basis polynomials of degree n . Specifically, the probability of observing counter state i at equilibrium equals the Bernstein basis polynomial $B_{i,n}(x)$ (Equation 5).

6. HARDWARE ARCHITECTURE

As previously mentioned, a stochastic implementation of a Bernstein polynomial representation consists of,

1. generating the Bernstein basis polynomials (Equation 5),
2. forming a linear combination of the Bernstein basis polynomials with the Bernstein coefficients (Equation 6)

We have demonstrated in the previous section that a stochastic integrator with parameter n generates the Bernstein basis polynomials of degree n . The remaining task of forming the linear combination is performed by an $(n+1) : 1$ multiplexer. The inputs to the multiplexer are random bitstreams B_i , $0 \leq i \leq n$ representing the $(n+1)$ Bernstein coefficients b_i , while the multiplexer select input is driven by the count, cnt , of the up/down counter of the stochastic integrator. As the counter states are mutually exclusive and exhaustive, the probability of the output random bitstream of the multiplexer to be logic '1' is obtained by invoking the law of total probability as,

$$\begin{aligned} Prob(Y = 1) &= \sum_{i=0}^n Prob(Y = 1 | cnt = i) Prob(cnt = i) \\ &= \sum_{i=0}^n Prob(B_i = 1) Prob(cnt = i) = \sum_{i=0}^n b_i B_{i,n}(x) \end{aligned} \quad (30)$$

Comparing with Equation 6, we conclude that our proposed architecture, depicted in Figure 6, realizes an arbitrary polynomial of degree n expressed as a Bernstein polynomial.

Furthermore, our stochastic implementation of polynomial functions (Figure 6) using sequential logic requires a single random bitstream for the input variable x , compared to the combinational implementation (Figure 2) that relies on multiple independent random bitstreams.

7. EXPERIMENTAL RESULTS

Finally, we compare the fault tolerance and the hardware costs of our proposed stochastic implementation of polynomial functions using sequential logic with the combinational implementation proposed in [9], and a conventional architecture based on the weighted binary number system.

Figure 6: The proposed stochastic implementation of a degree n polynomial using sequential logic.

7.1 Fault Tolerance

In order to compare the fault tolerance of the different polynomial implementations, we consider the gamma correction function. Gamma correction is a non linear operation used to efficiently encode image luminance as,

$$f(x) = x^\gamma \quad (31)$$

Selecting $\gamma = 0.45$, the non linear gamma correction function is approximated by a degree 6 Bernstein polynomial with the following Bernstein coefficients [8],

$$\begin{aligned} b_0 &= 0.0955, & b_1 &= 0.7207, & b_2 &= 0.3476, & b_3 &= 0.9988 \\ b_4 &= 0.7017, & b_5 &= 0.9695, & b_6 &= 0.9939 \end{aligned} \quad (32)$$

We construct a conventional implementation of the gamma correction function using binary multipliers and adders performing arithmetic on 8-bit data, and generating 8-bit results. The stochastic implementations, on the contrary, were realized using random bitstreams of length 256, as the 8-bit weighted binary representation is equivalent to a random bitstream representation of length 256. We performed simulations on a test image by adding random noise to the input data and measuring the average output error. Specifically, we randomly corrupted $\epsilon\%$ of the input data bits, computed the gamma correction function using the conventional and the stochastic implementations, and measured the average percent error between the result and the correct output in the absence of noise. Figure 7 presents the results of the simulation experiments. The results clearly indicate that the stochastic implementations outperform the conventional implementation in terms of fault tolerance. The stochastic implementations do produce small errors in the absence of noise ($\epsilon = 0$) as a result of the random nature of the computation. However, unlike the conventional implementation, the average output error of the stochastic implementations grows gradually as the input error increases, confirming the robustness of the unweighted random bitstream representation over the weighted binary number system.

The simulation results also reveal that between the two stochastic implementations, the combinational implementation [9] is slightly better at tolerating random noise than our proposed sequential implementation. The lower fault tolerance of the sequential implementation is a result of the correlation

Figure 7: A comparison of the fault tolerance of the different polynomial implementations of the gamma correction function.

Figure 8: Gamma correction results at input error $\epsilon = 15\%$ a) Original image b) Conventional implementation c) Combinational implementation d) Sequential implementation.

relation between successive counter states of the stochastic integrator. Specifically, the counter state at time t is not independent of the state at time $t - 1$. Consequently, unlike the combinational implementation (Figure 2), successive bits in the output random bitstream Y of the sequential implementation (Figure 6) exhibit correlation. The correlation in the random bitstream Y leads to a higher variance in the real-valued output $g_n(x)$. However, the multiplexer and the Bernstein coefficient random bitstreams B_i mitigate the impact of correlation in the counter states, as evidenced by the small difference in the fault tolerance of the combinational and the sequential implementations. The average output error of the sequential implementation is within 1% of the combinational implementation (Figure 7). Moreover, the gamma correction results at input error $\epsilon = 15\%$ in Figure 8 reinforce our assertion as the output images of the combinational and sequential implementations are virtually indistinguishable, whereas the output image of the conventional implementation is barely recognizable.

7.2 Hardware Costs

To compare the hardware costs of the different polynomial implementations, we follow the approach outlined in [9]. We synthesize and optimize our proposed architecture in Figure 6 using *ABC* [1], and map the optimized implementation to a circuit consisting of only inverters, and 2-input

Table 2: A comparison of the area and delay of the stochastic polynomial implementations.

n	combinational implementation [9]			proposed sequential implementation		
	area	delay	product	area	delay	product
3	22	10	220	45	7	315
4	40	17	680	75	10	750
5	49	20	980	79	9	711
6	58	20	1160	85	9	765

NAND, NOR, AND, OR, XOR, and XNOR logic gates. We assume that each logic gate requires a unit area and contributes a unit delay to the circuit. Moreover, each memory element is composed of 6 2-input NAND logic gates. Table 2 compares the area and delay of our proposed sequential implementation with the combinational implementation [9]. The presence of memory elements does increase the area of our sequential implementations, but reduces the critical path delay, resulting in a better **area-delay** product as the Bernstein polynomial degree n increases.

Finally, we compare the different polynomial implementations using the **area-delay-latency** product metric. The latency of a stochastic implementation corresponding to an M -bit weighted conventional implementation is 2^M , since a random bitstream of length 2^M is equivalent to an M -bit binary number. Table 3 presents the area-delay-latency products (q_{xx}) of the different polynomial implementations. The ratios of the area-delay-latency products in the final two columns of Table 3 indicate that our proposed sequential implementation better competes with the conventional implementation as the Bernstein polynomial degree n and the data resolution M increase.

To conclude, we have presented a stochastic implementation of polynomial functions using sequential logic. Our proposed approach is able to realize arbitrary polynomials with a better area-delay-latency product than the previously described combinational implementation [9]. Moreover, our architecture requires a single random bitstream for the input variable, unlike the combinational implementation that relies on multiple independent random bitstreams. The fault tolerance of our sequential implementation, albeit slightly lower than the combinational implementation, is still better than the conventional implementation. Overall, our proposed sequential stochastic computing system is a competitive solution to realize arbitrary polynomial functions.

8. ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under grant CCF-1408123.

9. REFERENCES

- [1] Berkeley logic synthesis and verification group, ABC: A system for sequential synthesis and verification, release 10216. <http://www.eecs.berkeley.edu/~alanmi/abc/>.
- [2] A. Alaghi and J. P. Hayes. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):1–19, May 2013.
- [3] B. Gaines. Stochastic computing systems. *Advances in information systems science*, pages 37–172, 1969.

Table 3: A comparison of the area-delay-latency products of the polynomial implementations. The bold entries indicate the points of equal performance between the conventional and the stochastic implementations.

n	M	q_{conv} [9]	q_{comb} [9]	q_{seq}	$\frac{q_{comb}}{q_{conv}}$	$\frac{q_{seq}}{q_{conv}}$
3	7	99207	28160	40320	0.284	0.406
3	8	152745	56320	80640	0.369	0.528
3	9	222615	112640	161280	0.506	0.724
3	10	310977	225280	322560	0.724	1.037
3	11	419991	450560	645120	1.073	1.536
4	7	132276	87040	96000	0.658	0.726
4	8	203660	174080	192000	0.855	0.943
4	9	296820	348160	384000	1.173	1.294
4	10	414636	696320	768000	1.679	1.852
4	11	559988	1392640	1536000	2.487	2.743
5	7	165345	125440	91008	0.759	0.550
5	8	254575	250880	182016	0.986	0.715
5	9	371025	501760	364032	1.352	0.981
5	10	518295	1003520	728064	1.936	1.405
5	11	699985	2007040	1456128	2.867	2.080
6	7	198414	148480	97920	0.748	0.494
6	8	305490	296960	195840	0.972	0.641
6	9	445230	593920	391680	1.334	0.880
6	10	621954	1187840	783360	1.910	1.260
6	11	839982	2375680	1566720	2.828	1.865

- [4] Y. Ji, F. Ran, C. Ma, and D. J. Lilja. A hardware implementation of a radial basis function neural network using stochastic logic. In *Proc. Design, Automation & Test in Europe Conf. (DATE)*, pages 880–883, 2015.
- [5] P. Li, D. Lilja, W. Qian, K. Bazargan, and M. Riedel. Computation on stochastic bit streams digital image processing case studies. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 22(3):449–462, Mar. 2014.
- [6] Y. Liu and K. Parhi. Lattice FIR digital filter architectures using stochastic computing. In *IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, pages 1027–1031, 2015.
- [7] G. Lorentz. *Bernstein polynomials*. University of Toronto Press, 1953.
- [8] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Trans. Comput.*, 60(1):93–105, Jan. 2011.
- [9] W. Qian and M. D. Riedel. The synthesis of robust polynomial arithmetic with stochastic logic. In *Proc. Annu. Design Automation Conf. (DAC)*, pages 648–653, 2008.
- [10] W. Qian, M. D. Riedel, and I. Rosenberg. Uniform approximation and bernstein polynomials with coefficients in the unit interval. *European Journal of Combinatorics*, 32(3):448–463, 2011.
- [11] S. Sharifi Tehrani, S. Mannor, and W. J. Gross. Fully parallel stochastic LDPC decoders. *IEEE Trans. Signal Process.*, 56(11):5692–5703, Nov. 2008.