Alaghi, A.; Hayes, J. P. (2013). "Survey of Stochastic Computing". ACM Transactions on Embedded Computing Systems. 12 (2s): 1.

ACM TECS
To appear, 2012

Survey of Stochastic Computing

ARMIN ALAGHI
JOHN P. HAYES
University of Michigan

Stochastic computing (SC) was proposed in the 1960s as a low-cost alternative to conventional binary computing. It is unique in that it represents and processes information in the form of digitized probabilities. SC employs very low-complexity arithmetic units which was a primary design concern in the past. Despite this advantage and also its inherent error tolerance, SC was seen as impractical because of very long computation times and relatively low accuracy. However, current technology trends tend to increase uncertainty in circuit behavior, and imply a need to better understand, and perhaps exploit, probability in computation. This paper surveys SC from a modern perspective where the small size, error resilience, and probabilistic features of SC may compete successfully with conventional methodologies in certain applications. First, we survey the literature and review the key concepts of stochastic number representation and circuit structure. We then describe the design of SC-based circuits and evaluate their advantages and disadvantages. Finally, we give examples of the potential applications of SC, and discuss some practical problems that are yet to be solved.

Categories and Subject Descriptors: B.2 Arithmetic and Logic Structures, B.8.1 Reliability, Testing, and Fault-Tolerance, C.1 Processor Architectures
Additional Key Words and Phrases: Probabilistic computation, Stochastic computing, Stochastic logic

1. INTRODUCTION

Modern computing hardware is constrained by stringent application requirements like extremely small size, low power consumption, and high reliability. It is also subject to physical phenomena like manufacturing process variations and soft errors, which give rise to error-prone behavior that can best be described in probabilistic terms. Consequently, unconventional computing methods that directly address these issues are of increasing interest. In this paper, we examine one such technique known as *stochastic computing* (SC) [Gaines 1967] [Poppelbaum et al. 1967]. A basic feature of SC is that numbers are represented by bit-streams that can be processed by very simple circuits, while the numbers themselves are interpreted as probabilities under both normal and faulty conditions. For example, a bit-stream S containing 25 percent 1s and 75 percent 0s denotes the number $p = 0.25$, reflecting the fact that the probability of observing a 1 at an arbitrary bit position is p. Neither the length nor the structure of S need be fixed; for example, (1,0,0,0), (0,1,0,0) and (0,1,0,0,0,1,0,0) are all possible representations of 0.25. Note that p depends on the ratio of 1s to the length of the bit-stream, not on their

This work was supported by Grant CCF-1017142 from the U.S. National Science Foundation.
Authors' addresses: Armin Alaghi, John P. Hayes, Department of Electrical Engineering and Computer Science, Computer Science and Engineering Division, University of Michigan, 2260 Hayward Street, Ann Arbor, MI 48109-2121, USA. E-mail: {alaghi, jhayes}@eecs.umich.edu.

positions, which can, in principle, be chosen randomly. We will refer to bit-streams of this type and the probabilities they represent as *stochastic numbers*.

The main attraction of SC when it was first introduced in the 1960's [Gaines 1967] [Poppelbaum et al. 1967] [Ribeiro 1967] is that it enables very low-cost implementations of arithmetic operations using standard logic elements. For example, multiplication can be performed by a stochastic circuit consisting of a single AND gate. Consider two binary bit-streams that are logically ANDed together. If the probabilities of seeing a 1 on the input bit-streams are p_1 and p_2, then the probability of 1 at the output of the AND gate is $p_1 \times p_2$, assuming that the two bit-streams are suitably uncorrelated or independent. Figure 1 illustrates the multiplication of two stochastic numbers in this way. As can be seen, the inputs to the AND gate represent the numbers 4/8 and 6/8 exactly. In the case of Figure 1(a), we get an output bit-stream denoting 4/8 × 6/8 = 3/8. Figure 1(b) shows two of the many possible alternative SC representations of the same input numbers 4/8 and 6/8. In this case, the output bit-stream denotes 2/8, which can be interpreted as an approximation to the exact product 3/8. This example, illustrates a key problem which we will examine later: How do we generate "good" stochastic numbers for a particular application?

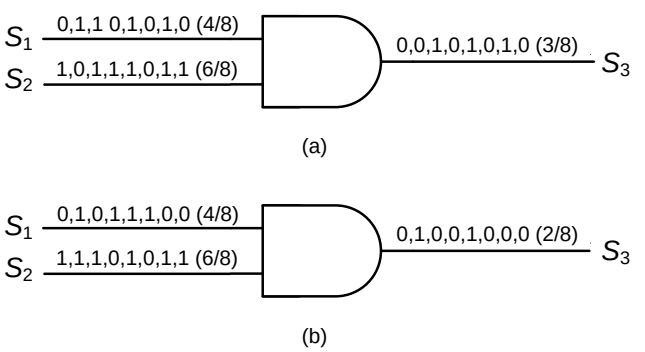


Figure 1. AND gate used as a stochastic multiplier: (a) exact and (b) approximate computation of 4/8 × 6/8.

Another attractive feature of SC is a high degree of error tolerance, especially for transient or soft errors caused by process variations or cosmic radiation. A single bit-flip in a long bit-stream will result in a small change in the value of the stochastic number represented. For example, a bit-flip in the output of the multiplier of Figure 1(a) changes its value from 3/8 to 4/8 or 2/8, which are the representable numbers closest to the correct result. But if we consider the same number 3/8 in conventional binary format 0.011, a single bit-flip causes a huge error if it affects a high-order bit. A change from 0.011 to

0.111, for example, changes the result from 3/8 to 7/8. Stochastic numbers have no high-order bits as such since all bits of a stochastic bit-stream have the same weight.

On the other hand, SC has several problems that have severely limited its practical applicability to date. An increase in the precision of a stochastic computation requires an exponential increase in bit-stream length, implying a corresponding exponential increase in computation time. For instance, to change the numerical precision of a stochastic computation from 4 to 8 bits requires increasing bit-stream length from $2^4 = 16$ bits to $2^8 = 256$ bits. As illustrated by Figure 1, variations in the representation of the numbers being processed, can lead to inaccurate results. An extreme case occurs when identical bit-streams denoting some number p are applied to the AND gate: the result then is p, rather than the numerically correct product $p \times p = p^2$.

Over the years, SC has been recognized as potentially useful in specialized (and often embedded) systems, where small size, low power, or soft-error tolerance are required, and limited precision or speed are acceptable. Besides its intrinsic suitability for certain computation tasks, SC seems worth reexamining because it copes with some complex probabilistic issues that are becoming an unavoidable part of conventional technologies [Krishnaswamy et al. 2007] and are as yet poorly understood.

Table 1: Timeline for the development of stochastic computation.

Dates	**Items**	**References**
1956	Fundamental concepts of probabilistic logic design.	[Von Neumann 1956]
1960-79	Definition of SC and introduction of basic concepts. Construction of general-purpose SC computers.	[Gaines 1967; 1969] [Poppelbaum 1976]
1980-99	Advances in the theory of SC. Studies of specialized applications of SC, including artificial neural networks and hybrid controllers.	[Jeavons et al. 1994] [Kim and Shanblatt 1995] [Toral et al. 1999]
2000-present	Application to efficient decoding of error-correcting codes. New general-purpose architectures.	[Gaudet and Rapley 2003] [Qian et al. 2011]

Table 1 provides a brief historical perspective on SC. Von Neumann [1956] defined fundamental ideas concerning probabilistic, error-tolerant design, and greatly influenced subsequent research. In the mid-1960s, further influenced by developments in both analog and digital computers, SC was defined and explored concurrently in the U.K. [Gaines 1967; 1969] and the U.S.A. [Poppelbaum et al. 1967] [Ribeiro 1967]. Several of the few general-purpose stochastic computers ever actually implemented were built around that time, and they uncovered numerous shortcomings of the technology.

Poppelbaum [1976] observed that "short sequences are untrustworthy" and that a major drawback of SC is low bandwidth and therefore low computational speed.

It is interesting to note that the first—and also the last—International Symposium on Stochastic Computing and its Applications was held in Toulouse in 1978. Since then, interest in SC has greatly diminished as conventional "binary" circuits have become smaller, cheaper, faster and more reliable. SC research has focused on a narrow range of specialized applications such as neural networks, [Kim and Shanblatt 1995] [Brown and Card 2001], control circuits [Toral et al. 1999] [Marin et al. 2002], and reliability calculations [Chen and Han 2010] [Aliee and Zarandi 2011]. There were, however, some important theoretical discoveries [Gupta and Kumaresan 1988] [Jeavons et al. 1994] relating to stochastic number generation, that have attracted little attention, but nevertheless have positive implications for SC, as we explain in Section 3.

A recently-discovered practical application of SC is to the decoding of low-density parity-check (LDPC) and related ECC codes [Gaudet and Rapley 2003]. LDPC codes constitute a family of linear codes that are increasingly used for communication over noisy, error-prone channels, for instance, in the IEEE WiFi standard [IEEE 2009]. Because of its use of extremely long codewords (often containing thousands of bits), LDPC decoding requires massive computational resources using conventional approaches [Zhang et al. 2010]. Moreover, some of the better decoding algorithms are probabilistic or "soft" rather than deterministic. All these features suggest that LDPC decoding can take advantage of the compactness, error tolerance, and inherently probabilistic nature of SC circuits, as has been demonstrated [Naderi et al. 2011].

There are other probabilistic methods in the computing literature that we do not consider here, some of which use the term "stochastic." They typically aim to achieve power-reliability trade-offs by means of probabilistic or statistical design, and differ substantially from what we call SC. For example, Shanbhag et al. [2010] and Akgul et al. [2006] focus on supply-voltage overscaling and methods of reducing the effect of any resulting errors. Other examples of probabilistic computing hardware are found in [Nepal et al. 2005] [Vigoda 2003]. The terms "stochastic numbers" and "stochastic arithmetic" appear in [Alt el al. 2006] which, however, is concerned with numerical errors in conventional binary computation.

In this paper, we attempt to survey and critique SC from a modern perspective. After reviewing the basic ideas behind SC, we examine its major advantages and disadvantages with emphasis on accuracy issues. Then, some recent representative applications of SC,

including LDPC decoding, are discussed. Finally, we draw some conclusions and suggest topics for future research.

2. BASIC CONCEPTS

Since stochastic numbers are treated as probabilities, they fall naturally into the interval [0,1]. This makes the normal add operation inconvenient because the sum of two numbers from [0,1] lies in [0,2]. For this reason, special *scaled add* operations are used in SC in order to map results from [0,2] to [0,1]. As illustrated in Figure 2, a two-way multiplexer can compute the sum of two stochastic numbers $p(S_1)$ and $p(S_2)$ applied to its data inputs S_1 and S_2. A third number with the constant value $p(S_3) = 1/2$ is also required, and is applied to the multiplexer's third (select) input; this can be supplied by a (pseudo) random number generator. The probability of a 1 appearing at the output S_4 is then equal to the probability of 1 at *s* multiplied by probability of 1 at S_1, plus the probability of 0 at *s* multiplied by the probability of 1 at S_2. More formally,

$$p(S_4) = p(S_3)\,p(S_1) + (1 - p(S_3))\,p(S_2) = (p(S_1) + p(S_2))/2$$

so that S_3 effectively scales the sum by 1/2. For the stochastic numbers shown in Figure 2, we obtain the result $p(S_4) = (7/8 + 3/8)/2 = 5/8$.

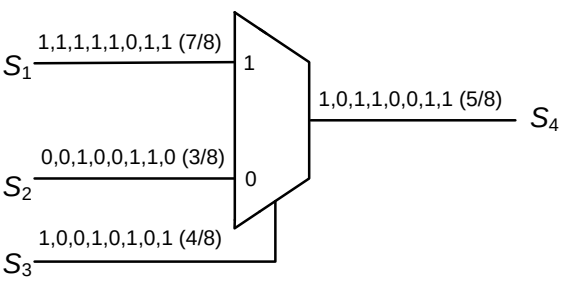


Figure 2. Multiplexer used as a scaled stochastic adder.

Circuits that convert binary numbers to stochastic numbers, and vice versa, are fundamental elements of SC. Figure 3(a) illustrates a widely-used binary-to-stochastic conversion circuit, which we will refer to as a *stochastic number generator* (SNG). The conversion process involves generating an *m*-bit random binary number in each clock cycle by means of a random or, more likely, a pseudo-random number generator, and comparing it to the *m*-bit input binary number. The comparator produces a 1 if the random number is less than the binary number and a 0 otherwise. Assuming that the random numbers are uniformly distributed over the interval [0,1], the probability of a 1

appearing at the output of the comparator at each clock cycle is equal to the binary input of the converter interpreted as a fractional number.

Converting a stochastic number to binary is much simpler. The stochastic number's value p is carried by the number of 1s in its bit-stream form, so it suffices to count these 1s in order to extract p. Figure 3(b) shows a counter that performs this conversion.

Figure 4 shows a stochastic circuit that implements the arithmetic function $z = x_1x_2x_4 + x_3(1 - x_4)$ [Li et al. 2009]. The inputs x_1, x_2, x_3 and x_4 are provided in conventional binary form and must be converted to stochastic numbers via SNGs. Suppose that the corresponding bit-streams S_1, S_2, S_3 and S_4 have the probability values 4/8, 6/8, 7/8 and 2/8, respectively. We know that the AND gate is a multiplier, so (with high probability) it outputs S_5 = 4/8 × 6/8 = 3/8. The probability of 1 at S_6 is the probability of 1 at both S_4 and S_5 plus the probability of 0 at S_4 and a 1 at S_3. This can be written as

$$p(S_6) = p(S_4 \wedge S_5) + p(\overline{S_4} \wedge S_3) = p(S_4)p(S_1)p(S_2) + \big(1 - p(S_4)\big)p(S_3)$$

Hence, S_6 is a stochastic representation of the number $x_1x_2x_4 + x_3(1 - x_4)$, and the counter at the output converts it to conventional binary form.

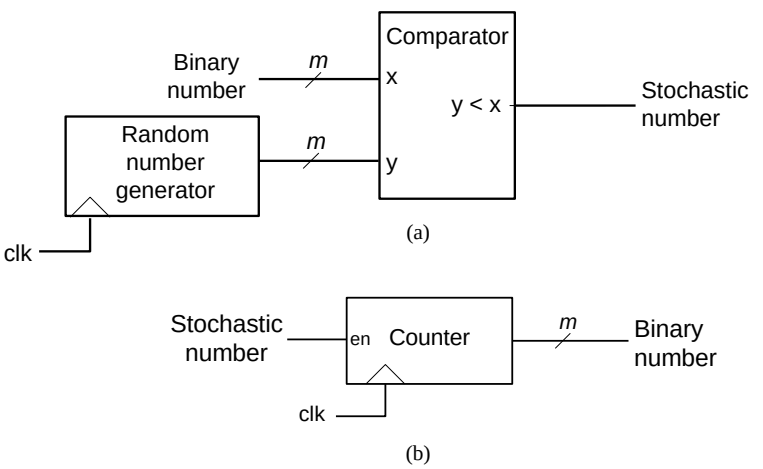


Figure 3. Number conversion circuits: (a) binary-to-stochastic; (b) stochastic-to-binary.

The result appearing at z in Figure 4 is 6/8 only if we get six 1s at S_6 in 8 clock cycles, otherwise the counter outputs a stochastic number other than 6/8. The probability of obtaining exactly six 1s is $p\{z = 6/8\} = \binom{8}{6}(6/8)^6(2/8)^2 \cong 0.31$, implying there is a 69% chance that we do not get six 1s, and the computation has some inaccuracy.

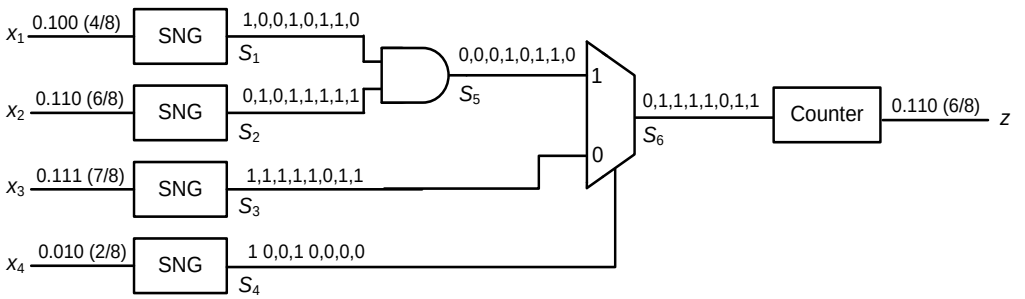


Figure 4. Stochastic circuit realizing the arithmetic function $z = x_1x_2x_4 + x_3(1 - x_4)$.

The stochastic numbers in Figure 4 have been chosen to avoid inaccuracy. Indeed, even if we use a high-quality random number source such as the "one million random digits" table [RAND Corp. 1955] to generate the stochastic numbers, we will probably still find some inaccuracy. For example, using the first four lines of this table to generate stochastic representations for x_1, x_2, x_3 and x_4, we obtain S_1 = (0,1,1,0,0,0,1,0), S_2 = (0,0,1,1,1,1,1,1), S_3 = (1,1,1,1,1,1,1,1) and S_4 = (0,0,0,0,0,1,0,0). Applying these numbers to the circuit in Figure 4 yields $S_6 = (1,1,1,1,0,1,1) = 7/8 \neq 6/8$.

Accuracy concerns arise in SC for several reasons. The one discussed above is due to the fluctuations inherent in random numbers. Correlations among the stochastic numbers being processed also lead to inaccuracies. Surprisingly, it is generally *not* desirable to use truly random number sources to derive or process stochastic numbers. As explained in the next section, deterministic or pseudo-random number generators form the best driving sources for SNGs from both a practical and theoretical point of view. They can be used to implement SC operations with high and, in some special cases, complete accuracy.

3. ACCURACY ISSUES

A stochastic number is a binary sequence of length n with n_1 1s and $1 - n_1$ 0s that represents the number $n_1/n \in [0,1]$; n_1 is the number's *weight*. Clearly, the stochastic representation of a number is not unique. SC uses a redundant number system in which there are $\binom{n}{n_1}$ possible representations for each number n_1/n. For example, with $n = 4$, there are six ways to represent 1/2: (0,0,1,1), (0,1,0,1), (0,1,1,0), (1,0,0,1), (1,0,1,0) and (1,1,0,0). Moreover, an n-bit sequence can only represent numbers in the set $\{0/n, 1/n, 2/n, \ldots, (n-1)/n, n/n\}$, so only a small subset of the real numbers in [0,1] can be expressed exactly in SC. More formally, a binary sequence S of length n is a *stochastic number* $\hat{p}$ if it is interpreted as the rational number $p = n_1/n$, where n_1 is the weight of S.

Several variant formats for stochastic numbers have been proposed. Gaines [1967] considers mapping the natural range of stochastic numbers, i.e., the interval [0,1] to different symmetric ranges, and discusses the computation elements needed. One such mapping is from $x \in [0,1]$ to the range $y \in [-1,1]$ via the function $y = 2x - 1$. This is the *bipolar* stochastic representation, and an XNOR gate performs multiplication in this case. We shall not consider such representations any further, as their properties are quite similar to those of the basic stochastic numbers used here.

Inaccuracy in SC has several distinct sources: random fluctuations in stochastic number representation, similarities (correlations) among the numbers that are being combined, and physical errors that alter the numbers. Jeavons et al. [1994] define two n-bit binary sequences $S_1 = (S_1(1),S_1(2), \ldots, S_1(n))$ and $S_2 = (S_2(1),S_2(2), \ldots, S_2(n))$ as *uncorrelated* or *independent* if and only if

$$\sum_{i=1}^{n} S_1(i)S_2(i) = \frac{\sum_{i=1}^{n} S_1(i) \times \sum_{i=1}^{n} S_2(i)}{n}$$

Otherwise, the sequences are called correlated. The following example shows how correlation can lead to inaccuracy. The 8-bit stochastic numbers S_1 = (1,1,1,1,0,0,0,0) and S_2 = (0,1,0,1,0,1,0,1) both representing 1/2, are uncorrelated according to the above definition. Their product $S_1 \times S_2$, obtained by ANDing them as in Figure 1, is (0,1,0,1,0,0,0,0) = 1/4. In contrast, S_1 and S_3 = (0,0,0,0,1,1,1,1) are correlated, and their product (0,0,0,0,0,0,0,0) = 0, which is far from the correct result.

To reduce such inaccuracies, SNGs are needed which produce stochastic numbers that are sufficiently random and uncorrelated. These requirements can be met by linear feed-back shift registers (LFSRs), which have been proposed for number generation in many SC designs The preferred LFSRs have m flip-flops and cycle through $n = 2^m - 1$ distinct states, the maximum possible since the all-0 state is excluded. The n-bit binary sequences produced are usually called *pseudo*-random because, although they are deterministic, they pass various randomness tests, e.g., they contain (almost) equal numbers of 0s and 1s, as well as runs of 0s and 1s whose numbers and lengths correspond to those of a Bernoulli sequence. In addition, shifted versions of LFSR sequences have low correlation [Golomb 1982] [Jeavons et al. 1994]. Poppelbaum [1976] noted that if LFSRs are large enough, they resemble ideal random sources. Hence, noise-like random fluctuations appear as errors in the generated stochastic numbers. These errors can be reduced at a rate $n^{-1/2}$ by increasing the number length n. Much less is known about how this error size changes as stochastic numbers go through a sequence of operations.

Surprisingly, LFSRs can be used to generate stochastic numbers that are, in certain cases, guaranteed to be exact. This was demonstrated for multiplication by Gupta and Kumaresan [1988] who introduced a new type of SNG that we call a weighted binary SNG. Figure 5 shows a 4-bit version, which converts a 4-bit binary number x to a stochastic number $\hat{x}$ of length 16. The pseudo-random source is a 4-bit LFSR to which the all-0 state is artificially added (details not shown). The SNG's behavior with x = 11/16 is illustrated in Table 2. The bit-streams L_i generated by the LFSR, the intermediate signals W_i, and the output stochastic number $\hat{x}$ are all shown. The key features of the Gupta-Kumaresan design, which immediately imply that $\hat{x}$ exactly represents x, is that the W_3, W_2, W_1 and W_0 bit-streams have non-overlapping 1s, and weights of 1/2, 1/4, 1/8 and 1/16, respectively. Gupta and Kumaresan further used this SNG to design a circuit that multiplies two stochastic numbers accurately. As shown in Figure 6 for the 4-bit case, it contains two weighted binary generators connected to a double-length LFSR. It is not hard to see that stochastic multiplication using this approach always yields exact results. Zelkin [2004] employs an essentially similar SNG to design an accurate arithmetic unit for stochastic numbers.

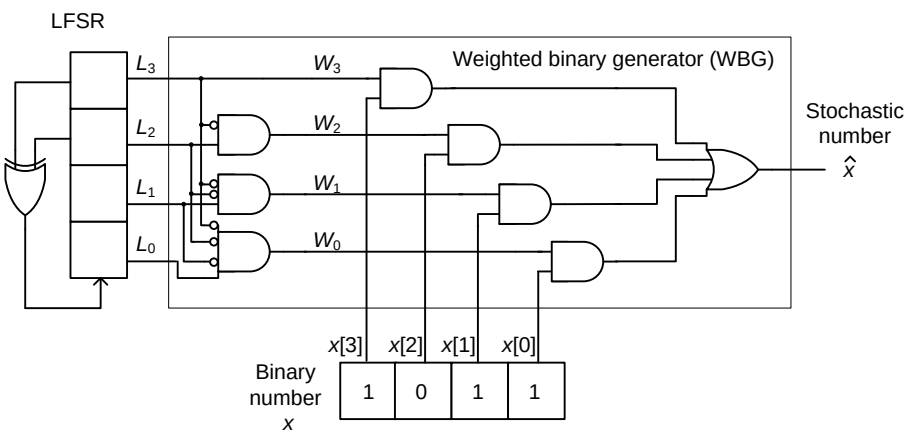


Figure 5. The weighted binary SNG proposed by Gupta and Kumaresan [1988].

The foregoing results reveal an important and perhaps unexpected aspect of SC: pseudo-random stochastic numbers can be better than random. This can be compared to the use of low-discrepancy numbers in quasi-Monte-Carlo sampling methods [Singhee and Rutenbar 2009]. Quasi-Monte Carlo is similar to normal Monte Carlo, but uses carefully chosen deterministic samples instead of purely random ones. Moreover, Gupta and Kumaresan's work shows that small deterministic LFSRs can produce highly

accurate results. In contrast, Gains [1967] and Poppelbaum [1976] rely on very large LFSRs to ensure randomness in the generated stochastic numbers.

Table 2. Bit-streams generated by the circuit of Figure 5.

Signal	Bit-stream	Value
L_3	0 0 1 0 1 0 1 1 1 1 0 0 0 0 1 1	8/16
L_2	0 1 0 1 0 1 1 1 1 0 0 0 0 1 1 0	8/16
L_1	1 0 1 0 1 1 1 1 0 0 0 0 1 1 0 0	8/16
L_0	0 1 0 1 1 1 1 0 0 0 0 1 1 0 0 1	8/16
W_3	0 0 1 0 1 0 1 1 1 1 0 0 0 0 1 1	8/16
W_2	0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0	4/16
W_1	1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	2/16
W_0	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0	1/16
$\hat{x}$	1 0 1 0 1 0 1 1 1 1 0 1 1 0 1 1	11/16

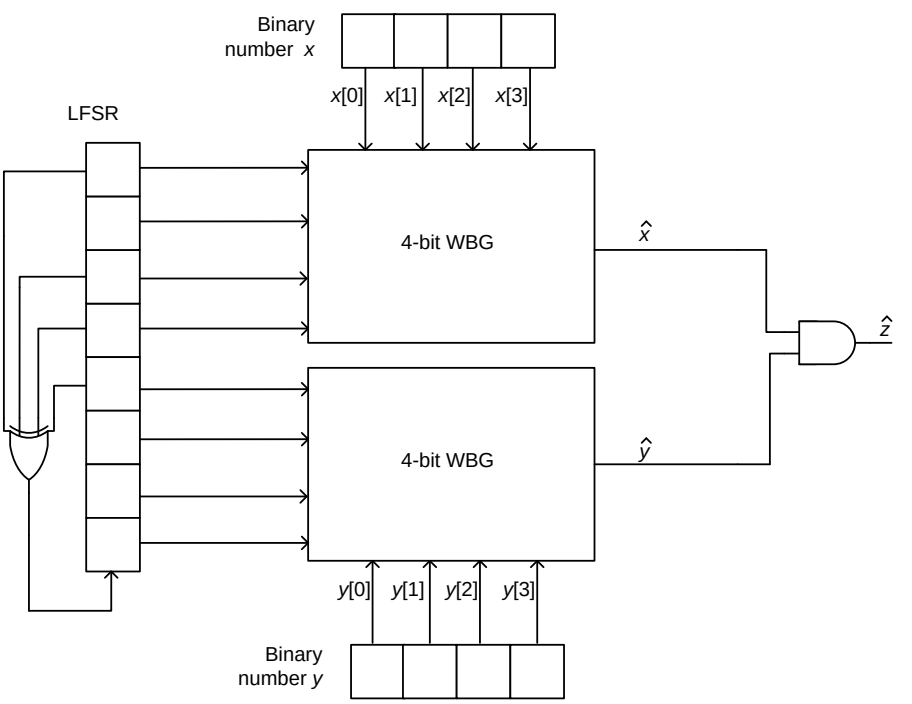
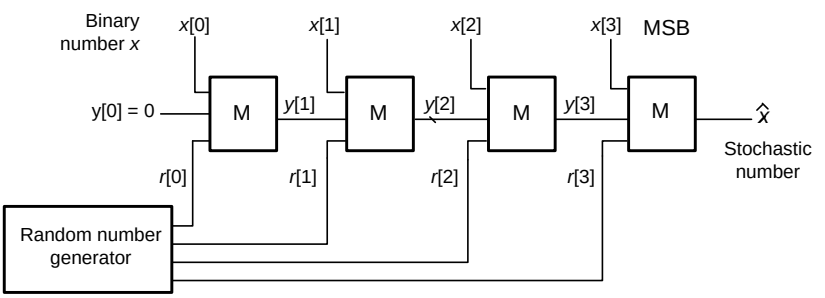


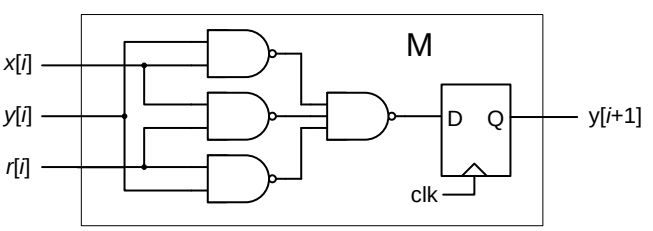
Figure 6. Accurate 4-bit stochastic multiplier of the type proposed by Gupta and Kumaresan [1988].

Jeavons et al. [1994] address the SC accuracy problem by creating a mathematical framework in which another SNG type is introduced. Instead of using comparators as in the traditional SNG of Figure 3(a), their design [Van Daalen et al. 1993] employs a cascade of majority modules *M*, as illustrated in Figure 7. A random number generator feeds the *M*s with uncorrelated stochastic numbers of probability 1/2, and Jeavons et al. [1994] suggest using bits of an LFSR for this purpose. Individual bits of the binary number *x* to be converted are also fed to the *M*s as shown in the figure, and one bit of the stochastic number $\hat{x}$ is generated per clock cycle.

Informally, the *precision* of a value is the number of bits needed to express that value. With *m* bits of precision, we can distinguish between 2^m different numbers. For instance, the numbers in the interval [0,1] when represented with 8-bit precision reduce to the following 256-member set: {0/256, 1/256, 2/256, …, 255/256, 256/256}, and their exact stochastic representation requires bit-streams of length 256. To increase the precision from 8 to 9 bits requires doubling the bit-stream length to 512, and so on. This exponential growth in data length with precision is responsible for the long computation times associated with SC.



(a)



(b)

Figure 7. Four-bit SNG proposed by Van Daalen at al. [1993]: (a) overall structure; (b) module *M*.

A stochastic number $\hat{p}$ of length n has $m = \log_2 n$ bits of precision, which equals that of an $\lceil m \rceil$-bit binary number. For instance, the 16-bit stochastic numbers (1,1,1,1, 1,1,1,1,1,0,0,0,0,0,0,0) and (0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1) both represent 9/16, and have 4-bit precision. There is a difference between them, however. If we consider the first 8 bits of the sequences as a stochastic number of lower precision (3-bit precision in this case), we obtain (1,1,1,1,1,1,1,1) and (0,1,0,1,0,1,0,1) representing 8/8 and 4/8, respectively. The latter provides a low-precision estimate of the full-length stochastic number (i.e., 9/16). This points to a potential advantage of SC: initial subsequences of a stochastic number, if appropriately generated, can provide an estimate of a target number. We say a stochastic number $\hat{p}$ of length n has *consistent precision*, if all the stochastic numbers $\hat{p}_k$, $k = 1,2,3,\ldots,\log_2(n)$, composed of the first 2^k elements of $\hat{p}$ are accurate. In other words, accuracy and precision increase steadily with stochastic number length. Stochastic numbers with consistent precision can therefore be seen as presenting their most significant bits first. Such behavior of stochastic numbers has been implicitly exploited in decoding applications [Gross et al. 2005]. In certain computations, this makes it possible to make decisions early by looking at the first few bits of a result.

Despite the efforts of Jeavons et al. [1994] and Gupta and Kumaresan [1988], accuracy, or the lack thereof, continues to be a major concern in SC. Inaccuracies due to correlation, for example, worsen as the number of inputs increases, the stochastic numbers pass through multiple levels of logic, or feedback is present. Reconvergent fanout, in particular, creates undesirable correlations when signals derived from a common input converge and interact. A possible but expensive solution to this problem is to convert these signals to binary and regenerate new and independent stochastic numbers from them on the fly [Tehrani et al. 2010].

4. APPLICATIONS

Stochastic computing has been investigated for a variety of applications. Besides the basic operations of addition and multiplication, SC has been applied to division and square-rooting [Toral et al. 2000], matrix operations [Mars and Mclean 1976], and polynomial arithmetic [Qian et al. 2011] [Qian and Riedel 2008]. A more specialized application area for SC is reliability analysis [Chen and Han 2010] [Aliee and Zarandi 2011]. Since probabilities are fundamentally analog quantities, SC has been proposed for some analog and hybrid analog-digital computing tasks, often under the heading of digital signal processing [Keane and Atlas 2001] [Pejic and Vujicic 1999]

Neural networks [Brown and Card 2001] [Dickson et al. 1993] [Kim and Shanblatt 1995] [Petriu et al. 2003] and control systems [Dinu et al. 2002] [Marin et al. 2002] are among the earliest and most widely studied applications of SC, and have close connections with analog computing. A recent illustrative example is found in [Zhang and Li 2008] where a control unit for an induction motor is described that integrates several SC-based algorithms and a large neural network. The controller is implemented on an FPGA, and is claimed to exhibit higher performance and lower hardware cost than conventional microprocessor-based designs for the same application. Figure 8 shows a simplified, high-level view of the motor controller. The stochastic integrators execute functions of the form $y(n) = x(n) + y(n - 1)$ on stochastic numbers. The stochastic anti-windup controller incorporates a complex algorithm that limits any changes implied in the input speed command that might lead to improper motor operation. The stochastic neural network estimator implements in real time the key feedback-processing functions of the system, several of which are computation-intensive. An example is the hyperbolic tangent or "tansig" function, which is a typical transform function computed by an artificial neuron, and takes the form

$$tansig(x) = \frac{2}{(1 - e^{-2x})} - 1$$

Zhang and Li [2008] note that besides improved cost-performance figures, their SC-based design has advantages in terms of reduced design and verification effort.

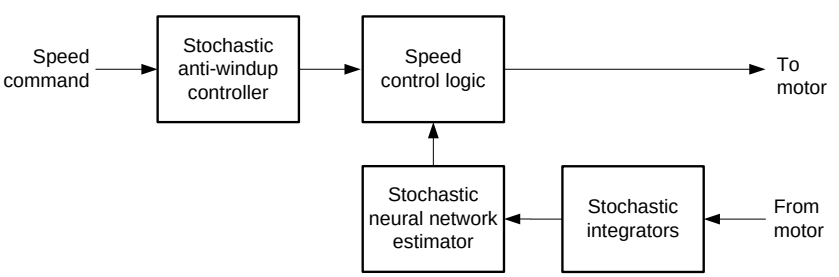


Figure 8. SC-based controller for an induction motor [Zhang and Li 2008].

Image processing is another potential application area for SC of great practical importance. Many imaging applications involve functional transformations on the pixels of an input image. The pixel-level functions are usually simple, but because of the large number of pixels involved, the overall transformation process is extremely computation-intensive. If these functions are implemented using SC, then low-cost, highly parallel

image processing becomes possible, as has been demonstrated in a smart SC-based image-sensing chip [Hammadou et al. 2003].

General-purpose SC has received little attention since the 1970s. It was revisited recently by Qian et al. [2011] and Li et al. [2009], who introduced an SC architecture that is capable of implementing many arithmetic functions. Function implementation in this style has a strong similarity to analog computing [McClennan 2009]. The main idea here, which is based on [Qian and Riedel 2008], is to approximate a given function by a Bernstein polynomial [Lorentz 1986]. A *Bernstein polynomial* of degree *k* has the form

$$z = \sum_{i=0}^{k} b_i \, . \, B_{i,k}(x)$$

where the b_i's are the coefficients of the polynomial, and $B_{i,k}(x)$ is a Bernstein basis polynomial of the form

$$B_{i,k}(x) = \binom{k}{i} x^i (1-x)^{k-i}$$

Qian et al. propose a reconfigurable stochastic architecture that can evaluate any function expressed in the form of a Bernstein polynomial; see Figure 9. The input variable *x* and the constant coefficients b_i are converted to stochastic numbers via the random number generator and the comparators. The inputs of the adder are *k* independent stochastic numbers representing *x* for some realization of a polynomial of degree *k*. The probability of obtaining a number *i* at the output of the adder is

$$P\{sum = i\} = \binom{k}{i} x^i (1-x)^{k-i}$$

Now the probability of having a 1 at *z* is:

$$P\{z = 1\} = b_0 \times P\{sum = 0\} + \ b_1 \times P\{sum = 1\} + \ \ldots + b_k \times P\{sum = k\}$$

which reduces to

$$P\{z = 1\} = \sum_{i=0}^{k} b_i \, . \, B_{i,k}(x)$$

In other words, the probability of outputting a 1 at *z* is a Bernstein polynomial of degree *k* defined by the coefficients b_i calculated at *x*.

Qian et al. have used their SC architecture to implement several image processing functions. An example is gamma correction, which involves applying the function $z = x^{0.45}$ to every pixel of an image. This function is approximated by a degree-6 Bernstein polynomial, which can be implemented with the general structure of Figure 9. The ability to maintain high accuracy in the presence of errors is another feature of SC that can be

exploited in image processing. Figure 10 compares two implementations of the gamma correction function [Qian et al. 2011]. The results of processing a particular image by conventional (binary) circuits are shown in the top row, and by SC circuits in the bottom row. The two implementations were exposed to the same noisy environment. Column (a) shows the results with no errors present. Columns (b - f) show the same results in the presence of random bit-flip errors occurring at rates 1%, 2%, 5%, 10% and 15%, respectively.

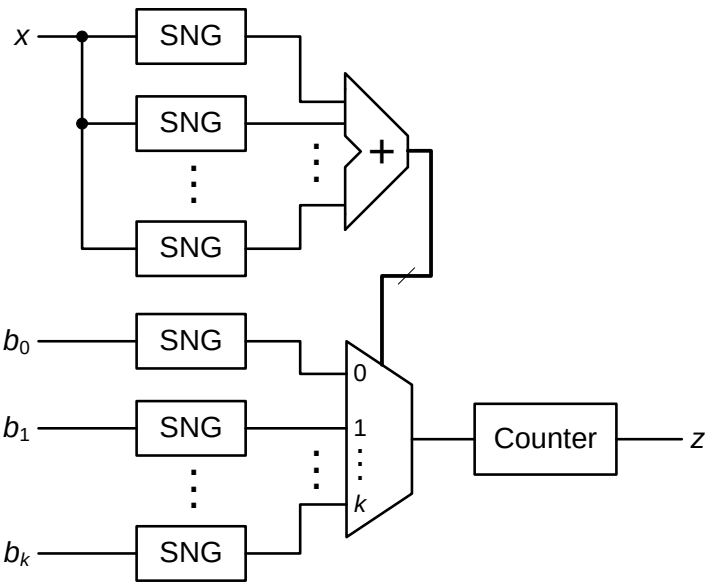


Figure 9. Reconfigurable stochastic circuit realizing a Bernstein polynomial of degree *k* [Qian et al. 2011].

Figure 10 demonstrates that the SC implementation achieves higher error resilience. Moreover, the stochastic implementation uses less hardware. Table 3 shows the cost of several image processing functions implemented on a Xilinx Virtex-II Pro FPGA. The table compares conventional binary design with SC. The leftmost column in the table lists several implemented image processing functions. The gamma function, for instance, is the function mentioned earlier; for details of these functions see [Qian et al. 2011]. The costs in Table 3 are shown in terms of the number of look-up tables (LUTs) used in the FPGA. Interestingly, the table reveals that more than 80% percent of the SC design is used by SNGs and stochastic-to-binary converters.

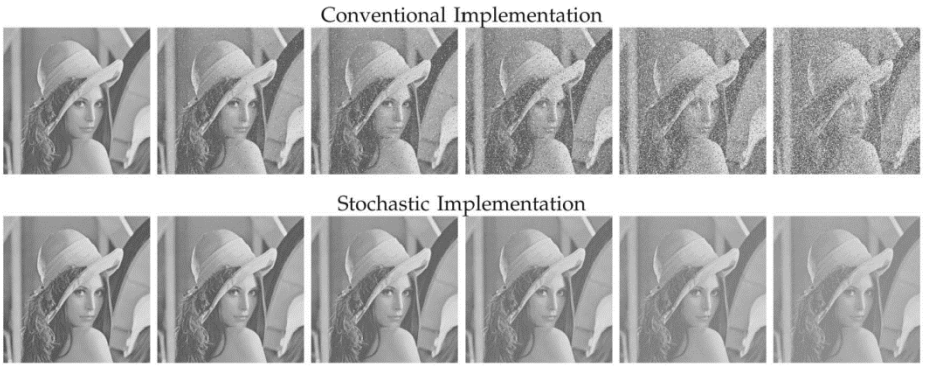


Figure 10. Error-tolerance comparison between conventional and SC implementations of a gamma correction function. © 2011 IEEE. Reprinted, with permission, from [Qian et al. 2011].

Table 3: Comparison between conventional and SC designs; cost = number of LUTs [Qian et al. 2011].

Module	Conventional design cost	Stochastic design cost			
		Complete system*		Stochastic core only**	
	Cost	Cost	Savings (%)	Cost	Savings (%)
Gamma	96	124	-29.2	16	83.3
RGB → XYZ	524	301	42.6	64	87.8
XYZ → RGB	627	301	52.0	66	89.5
XYZ → CIE → L*ab	295	250	15.3	58	80.3
CIE → L*ab → XYZ	554	258	53.4	54	90.3
Geometric	831	299	64.0	32	96.1
Rotation	737	257	65.1	30	95.9
Average	523.4	255.7	37.6	45.7	89.0

* The complete SC design
** Only the stochastic part (without the SNGs and stochastic-to-binary converters)

5. LDPC DECODING

The main reasons for the early interest in SC are the relative simplicity and robustness of SC-based arithmetic units and the possibility of having many units working in parallel. These benefits became less important as the transistors became cheaper but, as the foregoing motor-control application suggests, the benefits continue to be significant, even in some well-established applications. Furthermore, recent research has introduced several entirely new applications for SC. One such application is discussed in detail next: the decoding of low-density parity check (LDPC) codes.

LDPC codes are powerful error-correcting codes which were introduced by Gallager [1962]. They enable data to be sent over noisy channels at rates close to the theoretical maximum (the Shannon limit). Because they are difficult to implement in practice, they were largely ignored until the 1990s when new research [MacKay and Neal 1996] and

semiconductor technology developments made them economically viable. LDPC codes are attractive because there are no global relations among their bits, making it possible to have efficient decoding algorithms for very long codewords, such as the sum-product algorithm (SPA) [Kschischang et al. 2001]. LDPC codes are now utilized in communication standards such as WiFi [IEEE 2009] and digital video broadcasting [ETSI 2005].

Linear codes can be represented by Boolean equations that define parity constraints on the bits of a codeword. For instance, a 9-bit single-error detecting code is defined by the equation

$$x_0 \oplus x_1 \oplus x_2 \oplus \ldots \oplus x_7 \oplus x_8 = 0$$

where $x_0, x_1, \ldots, x_7$ are data bits, x_8 is a parity bit, and $\oplus$ is the XOR (sum modulo 2) operation. The following four equations define a very small LDPC code of length 6:

$$x_0 \oplus x_2 \oplus x_3 = 0$$
$$x_1 \oplus x_2 = 0$$
$$x_0 \oplus x_4 = 0$$
$$x_1 \oplus x_5 = 0$$

These equations can also be written in matrix form:

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = 0$$

where the vector-matrix multiplication uses AND and XOR. Note that this code only has local parity constraints involving small subsets of the bits. In general, an LDPC code of length n with k parity equations has the matrix form

$$\mathbf{H}.\vec{\mathbf{x}} = \begin{bmatrix} b_{0,0} & \cdots & b_{0,n-1} \\ \cdots & \ddots & \cdots \\ b_{k-1,0} & \ldots & b_{k-1,n-1} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix} = 0$$

where the $b_{i,j}$'s are 0s and 1s, the x_i's are code bits. A typical parity matrix **H** of an LDPC code is huge (n and k can be many thousands), but it is sparse in that it has only a few1s in each row and column.

A relatively small subset of all 2^n possible combinations of 0s and 1s on $x_0, x_1, \ldots, x_{n-1}$ satisfy the code equations; these are the valid codewords. In the 6-bit example above, there are 64 possible codewords, only four of which are valid. The process of encoding data involves mapping data words to valid codewords. In the example, it is possible to

encode 2-bit data, because there are only four valid codewords. Since codewords are 6 bits in length, they are redundant and thus provide protection against errors.

LDPC decoding is the process of trying to find the closest valid codeword to a received and possibly erroneous codeword. The SPA algorithm mentioned earlier is a probabilistic algorithm for LDPC decoding based on belief propagation. For each bit x_i of the received code, this algorithm assumes a probability y_i of its being 1, i.e., $y_i = P\{x_i = 1\}$. For instance, if a 1 is received at position x_0 of a codeword passing through a channel with 0.1 probability of error, the SPA algorithm assigns $y_0 = 0.9$. This means that x_0 is assumed to be 1 with probability of 0.9, which is a reasonable assumption. These probabilities are then combined until the best valid codeword candidate is determined. A simplified version of the SPA algorithm appears in Table 4. Here, it is assumed that there are at most three 1s in each row and at most two 1s in each column of the parity matrix H. It can be shown that any LDPC code can be transformed to an LDPC code that satisfies these conditions [Kschischang et al. 2001].

Table 4. Simplified sum-product algorithm (SPA) for LDPC decoding.

Step 1	Initialize all y_i's using $y_i = P\{x_i = 1\} = x_i.(1 - p_e) + (1 - x_i).p_e$ where the x_i's are the received codeword bits and p_e is the error probability of the channel.
Step 2	For each row in the parity matrix **H**: **a.** If the row has two 1s in positions i and j, suggest new values of y_i and y_j for use in Step 3 according to the following rules: $y_i' = y_j$ and $y_j' = y_i$. Use y_i'' instead of y_i' on the second appearance of each position i in the current iteration. **b.** If the row has three 1s in positions i, j and k: $y_i' = y_j.(1 - y_k) + y_k.(1 - y_j)$, $y_j' = y_i.(1 - y_k) + y_k.(1 - y_i)$ and $y_k' = y_i.(1 - y_j) + y_j.(1 - y_i)$.
Step 3	Update each y_i according to the following rules: **a.** If there is a single 1 in column i of **H**, then $y_i = y_i'$. **b.** If there are two 1s in column i of **H**, then $y_i = (y_i' \times y_i'')/[(y_i' \times y_i'') + (1 - y_i')(1 - y_i'')]$.
Step 4	Check if the current y_i's form a valid codeword after converting them to binary numbers by applying a threshold. If a valid codeword is obtained or an iteration count limit is reached, end the procedure. Otherwise go to Step 2.

Consider again the 6-bit LDPC code introduced above, which has the following parity matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Assume that a codeword $x_0x_1\ldots x_5$ = 100111 is received through a channel with error probability of 0.1; note that this codeword does not satisfy the parity constraints defined by **H**. We will follow the procedure in Table 4 to find the closest valid codeword.

Table 5 shows the values of the y_i's in each step of the decoding process. First, the y_i's are initialized in Step 1: $y_0 = 0.9$, $y_1 = 0.1$, $y_2 = 0.1$, $y_3 = 0.9$, $y_4 = 0.9$, $y_5 = 0.9$. In Step 2, the rows of **H** are traversed. The first row has three 1s in positions 0, 2 and 3, so new values y_0', y_2' and y_3' are calculated according to Step 2b. The second row of **H** has two 1s, so Step 2a is executed. Since position 2 has already been visited in the current iteration, y_2'' is calculated. After calculation of all suggested values, the y_i's are updated in Step 3. In Step 4, all the y_i's are converted to x_i's as shown in Table 4. Any y_i greater than 0.75 is rounded to 1, and y_i's smaller than 0.25 are rounded to 0; the rest are marked undecided. In this case, y_1 = 0.5, so x_1 is undecided, and the current codeword is determined to be invalid. The algorithm continues by returning to Step 2. After two more iterations, it finds the valid codeword (x_0, …, x_5) = 100110 and stops. It should be noted that the algorithm may fail to converge to the correct codeword. However, with suitable extensions, its average rate of decoding failure can be reduced to an acceptably low level.

Table 5. LDPC decoding example using the algorithm of Table 4.

Steps	y_0	y_1	y_2	y_3	y_4	y_5
Step 1	0.9	0.1	0.1	0.9	0.9	0.9
Step 2	y_0' = 0.82 y_0'' = 0.9	y_1' = 0.1 y_1'' = 0.9	y_2' = 0.18 y_2'' = 0.1	y_3' = 0.82	y_4' = 0.9	y_5' = 0.1
Step 3	0.98	0.5	0.02	0.82	0.9	0.1
Step 4	x_0 = 1	x_1 = ?	x_2 = 0	x_3 = 1	x_4 = 1	x_5 = 0
New iteration: **Step 2**	y_0' = 0.81 y_0'' = 0.9	y_1' = 0.02 y_1'' = 0.1	y_2' = 0.19 y_2'' = 0.5	y_3' = 0.96	y_4' = 0.98	y_5' = 0.5
Step 3	0.97	0	0.19	0.96	0.98	0.5
Step 4	x_0 = 1	x_1 = 0	x_2 = 0	x_3 = 1	x_4 = 1	x_5 = ?
New iteration: **Step 2**	y_0' = 0.79 y_0'' = 0.98	y_1' = 0.19 y_1'' = 0.5	y_2' = 0.07 y_2'' = 0	y_3' = 0.79	y_4' = 0.97	y_5' = 0
Step 3	0.99	0.19	0	0.79	0.97	0.002
Step 4	x_0 = 1	x_1 = 0	x_2 = 0	x_3 = 1	x_4 = 1	x_5 = 0

Gross et al. [2005] present a stochastic implementation of the SPA approach that employs the basic components shown in Figure 11. The XOR gate of Figure 11(a) implements the function $y_i' = y_j(1 - y_k) + y_k(1 - y_j)$ used in Step 2b, while the circuit of Figure 11(b) implements the update function $y_i = y_i'y_i''/[y_i'y_i'' + (1 - y_i')(1 - y_i'')]$ in Step 3b. Steps 2a and 3a involve no significant computation and are implemented by wires.

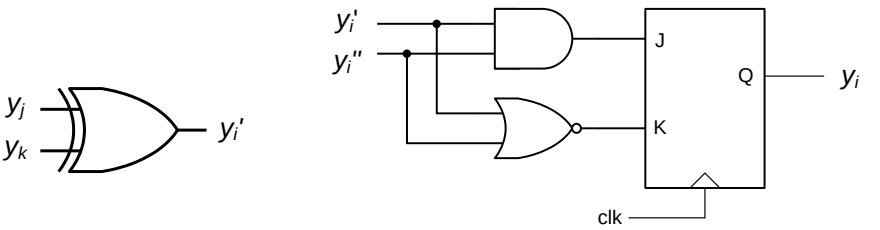


Figure 11. (a) Parity and (b) update blocks used in stochastic LDPC decoding.

Figure 12 shows the complete stochastic LDPC decoder for the above example, designed according to the method of [Gross et al. 2005]. The initial probability calculator corresponds to Step 1 of the SPA algorithm, and it generates initial values using the received bits and the known channel behavior. Next, SNGs convert these probabilities to stochastic numbers. The main SC core implements Steps 2 and 3 of the algorithm. It receives the initial probabilities and generates new probabilities, which are fed back through multiplexers for use in later iterations. The final probabilities are converted to binary numbers and a parity check is performed on them, as in Step 4 of the algorithm.

The main stochastic core is the part of the LDPC decoder that performs most of the computation. Its structure is shown in Figure 13. The XOR gates and the wiring at the beginning of this block correspond to Step 2 of the SPA method; they are followed by blocks that correspond to Step 3. The update blocks shown in the figure are copies of the circuit in Figure 11(b).

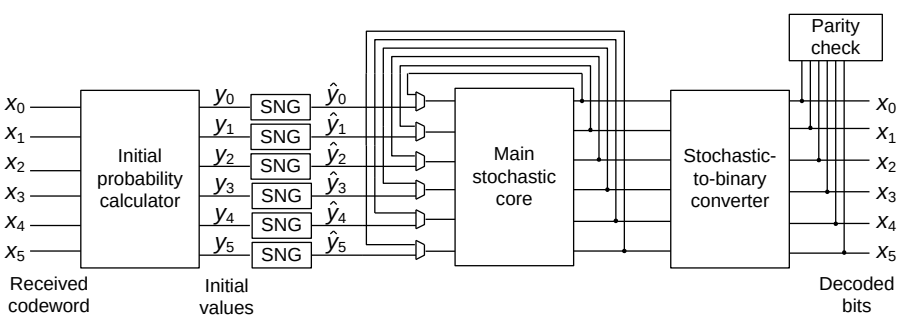


Figure 12. Stochastic LDPC decoder for the small example.

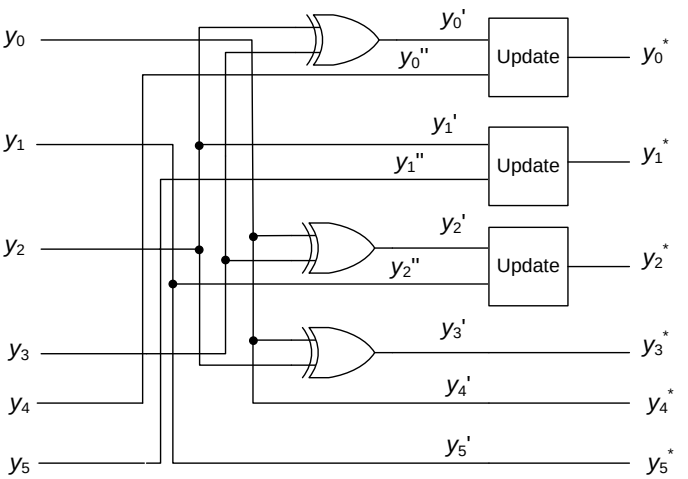


Figure 13. Main stochastic core for the LDPC decoder of Figure 12.

While the probabilistic nature of SC makes it suitable for this decoding application, several other features of SC can be exploited in the decoder design. The low complexity of the basic components enables easy scaling to very large LPDC codes and effectively supports efficient parallel processing for such codes. Also, the precision consistency feature noted in Section 3 speeds up convergence of the decoding process. The early iterations of the algorithm can proceed faster by using rough values provided from the first bits of the stochastic numbers being processed. Unlike the image processing application [Qian et al. 2011] discussed in Section 4, the overhead due to binary-to-stochastic number conversion is small. A recently reported ASIC implementation of an SC-based LDPC decoder [Naderi et al. 2011] claims to achieve higher throughput with less chip area than conventional non-SC decoders.

6. DISCUSSION

Stochastic computing has had a long and checkered history over the last half-century. After an initial burst of interest beginning in the 1960s when a handful of prototype SC-based computers were built, we are seeing a revival of interest in this rather unique approach to computation. The main driving forces continue to be low hardware cost and high error tolerance. SC has been successfully used in various control applications, where it has proven to be an effective replacement for conventional analog or hybrid controllers. It has also proven attractive for some specialized digital applications such as image processing and LDPC decoding, which can exploit massive parallelism. The main drawback of SC is long computation time, which tends to grow exponentially with

respect to precision; a 1-bit increase in precision requires a doubling in stochastic number length. This has limited SC usage to relatively low-precision applications, many of which are found in embedded systems.

SC has considerable theoretical appeal as a very unconventional way to compute. It can also provide insights into the probabilistic aspects of other computing technologies, such as conventional circuits subject to random faults or component variability. In the case of a recently invented and inherently probabilistic technique, quantum computing, the randomness in the possible representations of a stochastic number mirrors the randomness in the measured values of a quantum number [Paler et al. 2011].

Despite its long history, many aspects of SC are still not well understood and call for further research.

- There is a need to build and evaluate more SC-based systems of practical size to gain insight into their performance and behavior in actual use. The success of SC in LPDC decoding suggests that SC may be fruitfully applied to fields like artificial intelligence and robotics where probabilistic inference is widely used, often with special-purpose hardware support [Vigoda 2003] [Mansinghka 2009**].**
- The role of randomness and correlation in stochastic number representation and generation remains poorly understood. As we have seen here, judicious use of pseudo-random numbers leads to more accurate results than purely random ones. Although individual SC operations like multiplication can now be implemented with high precision, the problem of efficiently maintaining such precision across multiple operations and circuits is difficult and largely unexplored.
- Stochastic number generators continue to be among the most costly components of stochastic circuits. The problems of designing and deploying cost-effective SNGs that ensure a desired, system-wide level of accuracy or precision is far from solved.
- Finally, although SC lies at the boundary between traditional analog and digital computation, the theoretical and practical links between these different computational styles have yet to be fully explored in the SC context.

REFERENCES

AKGUL, B.E.S., CHAKRAPANI, L.N, KORKMAZ, P., AND PALEM, K.V. 2006. Probabilistic CMOS technology: a survey and future directions. *Proc. IFIP Conf. VLSI*, 1-6.

ALIEE, H., AND ZARANDI, H.R. 2011. Fault tree analysis using stochastic logic: a reliable and high speed computing. *Proc. Reliability and Maintainability Symp.*, 1-6.

ALT, R., LAMOTTE, J.-L., AND MARKOV, S. 2006. On the solution to numerical problems using stochastic arithmetic. *Proc. Symp. Scientific Computing, Computer Arithmetic and Validated Numerics*, p 6.

BROWN, B.D., AND CARD, H.C. 2001. Stochastic neural computation I: computational elements. *IEEE Trans. Computers*, 50, 891-905.

CHEN, H., AND HAN, J. 2010. Stochastic computational models for accurate reliability evaluation of logic circuits. *Proc. Great Lakes Symp. VLSI*, 61-66.

DICKSON, J.A., MCLEOD, R.D., AND CARD, H.C. 1993. Stochastic arithmetic implementations of neural networks with in situ learning. *Proc. Intl. Conf. Neural Networks*, 711-716.

DINU, A., CIRSTEA, M.N., AND MCCORMICK, M. 2002. Stochastic implementation of motor controllers. *Proc. IEEE Symp. Industrial Electronics*, 639-644.

ETSI. 2005. European Telecommunications Standards Institute Standard TR 102 376 V1.1.1*: Digital Video Broadcasting (DVB). User Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications*. Available from http://www.etsi.org.

GAINES, B.R. 1967. Stochastic computing. *Proc. AFIPS Spring Joint Computer Conf.*, 149-156.

GAINES, B.R. 1969. Stochastic computing systems. *Advances in Information Systems Science*, 2, 37-172.

GALLAGER, R.G. 1962. Low-density parity-check codes. *IRE Trans. Information Theory*, 8, 21-28.

GAUDET, V.C., AND RAPLEY, A.C. 2003. Iterative decoding using stochastic computation. *Electronics Letters*, 39, 299-301.

GOLOMB., S.W. 1982. *Shift Register Sequences*. Rev. ed., Aegean Park Press, Laguna Hills, Calif.

GROSS, W.J., GAUDET, V.C., AND MILNER, A. 2005. Stochastic implementation of LDPC decoders. *Proc. Asilomar Conf. Signals, Systems and Computers*, 713-717.

GUPTA, P.K., AND KUMARESAN, R. 1988. Binary multiplication with PN sequences. *IEEE Trans. Acoustics, Speech and Signal Processing*, 36, 603-606.

HAMMADOU, T., NILSON, M., BERMAK, A., AND OGUNBONA, P. 2003. A 96 × 64 intelligent digital pixel array with extended binary stochastic arithmetic. *Proc. Intl. Symp. Circuits and Systems*, IV-772 – IV-775.

IEEE. 2009. IEEE Standards Association standard *IEEE. 802.11n for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks.* Available from http://standards.ieee.org.

JEAVONS, P., COHEN, D.A., AND SHAWE-TAYLOR, J. 1994. Generating binary sequences for stochastic computing. *IEEE Trans. Information Theory*, 40, 716-720.

KEANE, J.F., AND ATLAS, L.E. 2001. Impulses and stochastic arithmetic for signal processing. *Proc. Intl. Conf. Acoustics, Speech and Signal Processing*, 1257-1260.

KIM, Y-C., AND SHANBLATT, M.A. 1995. Architecture and statistical model of a pulse-mode digital multilayer neural network. *IEEE Trans. Neural Networks*, 6, 1109-1118.

KRISHNASWAMY, S., MARKOV, I.M., AND HAYES, J.P. Tracking uncertainty with probabilistic logic circuit testing. *IEEE Design & Test of Computers*, 24, 312–321.

KSCHISCHANG, F.R., FREY, B.J., AND LOELIGER, H-A. 2001. Factor graphs and the sum-product algorithm. *IEEE Trans. on Information Theory*, 47, 498-519.

LI, X., QIAN, W., RIEDEL, M.D., BAZARGAN, K., AND LILJA, D.J. 2009. A reconfigurable stochastic architecture for highly reliable computing. *Proc. Great Lakes Symp. VLSI*, 315-320.

LORENTZ, G.G. 1986. *Bernstein Polynomials*. 2nd ed., Chelsea Publishing Co., New York

MACKAY D.J.C., AND NEAL, R.M. 1996. Near Shannon limit performance of low density parity check codes. *Electronics Letters*, 32, p 1645.

MACLENNAN, B.J. 2009. Analog computation. *Encyclopedia of Complexity and System Science*, Springer, 271-294.

MANSINGHKA, V. 2009. *Natively Probabilistic Computation*. Ph.D. dissertation, Massachusetts Institute of Technology, Dept. of Brain and Cognitive Sciences.

MARIN, S.L.T., REBOUL, J.M.Q., AND FRANQUELO, L.G. 2002. Digital stochastic realization of complex analog controllers. *IEEE Trans. Industrial Electronics*, 49, 1101-1109.

MARS, P., AND MCLEAN, H.R. 1976 High-speed matrix inversion by stochastic computer. *Electronics Letters*, 12, 457-459.

NADERI, A., MANNOR, S., SAWAN, M., AND GROSS, W.J. 2011. Delayed stochastic decoding of LDPC codes. *IEEE Trans. Signal Processing*. To appear.

NEPAL, K., BAHAR, R.I., MUNDY, J., PATTERSON, W.R., AND ZASLAVSKY, A. 2005. Designing logic circuits for probabilistic computation in the presence of noise. *Proc. Des. Autom. Conf.*, 485-490.

PALER, A., ALAGHI, A., POLIAN, I., AND HAYES, J.P. 2011. Tomographic testing and validation of probabilistic circuits. *Proc. European Test Symp*., 63-68.

PEJIC, D., AND VUJICIC, V. 1999. Accuracy limit of high precision stochastic watt-hour meter. *IEEE Trans. Instrum. and Meas.*, 49, 617-620.

PETRIU, E.M., ZHAO, L., DAS, S.R., GROZA, V.Z., AND CORNELL, A. 2003. Instrumentation applications of multibit random-data representation. *IEEE Trans. Instrum. and Meas.*, 52, 175-181.

POPPELBAUM, W.J. 1976. Statistical processors. *Advances in Computers*, 14, 187-230.

POPPELBAUM, W.J., AFUSO, C., AND ESCH, J.W. 1967. Stochastic computing elements and systems. *Proc. AFIPS Fall Joint Computer Conf.*, 635-644.

QUERO, J.M., TORAL, S.L., CARRASCO, J.M., ORTEGA, J.G., AND FRANQUELO, L.G. 1999. Continuous time controllers using digital programmable devices. *Proc. 25th IECON*, 1, 210-215.

QIAN, W., LI, X., RIEDEL, M.D., BAZARGAN, K., AND LILJA, D.J. 2011. An architecture for fault-tolerant computation with stochastic logic. *IEEE Trans. Computers*, 60, 93-105.

QIAN, W., AND RIEDEL, M.D. 2008. The synthesis of robust polynomial arithmetic with stochastic logic. *Proc. Des. Autom. Conf.*, 648-653.

RAND Corp. 1955. *A Million Random Digits with 100,000 Normal Deviates*. Glencoe, IL: Free Press. Reprinted by RAND Corp. in 2001.

RIBEIRO, S.T. 1967. Random-pulse machines. *IEEE Trans. Electronic Computers*, EC-16, 261-276.

SHANBHAG, N.R., ABDALLAH, R.A., KUMAR, R., AND JONES, D.L. 2010. Stochastic computation. *Proc. Des. Autom. Conf.*, 859-864.

SINGHEE, A., AND RUTENBAR, R.A. 2009. *Novel Algorithms for Fast Statistical Analysis of Scaled Circuits*. Lecture Notes in Electrical Engineering 46, Springer.

TEHRANI, S.S., NADERI, A., KAMENDJE, G.A., HEMATI, S., MANNOR, S., AND GROSS, W.J. 2010. Majority-based tracking forecast memories for stochastic LDPC decoding. *IEEE Trans. Signal Processing*, 58, 4883-4896.

TORAL, S.L., QUERO, J.M., AND FRANQUELO, L.G. 2000. Stochastic pulse coded arithmetic. *Proc. ISCAS*, 1, 599-602.

VAN DAALEN, M., JEAVONS, P., SHAWE-TAYLOR, J., AND COHEN, D. 1993. Device for generating binary sequences for stochastic computing. *Electronics Letters*, 29, p 80.

VIGODA, B. 2003. *Analog Logic: Continuous-Time Analog Circuits for Statistical Signal Processing*, Ph.D. dissertation, Massachusetts Institute of Technology.

VON NEUMANN, J. 1956. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, Princeton Univ. Press, 43-98.

ZELKIN, B. 2004. Arithmetic unit using stochastic data processing. U.S. Patent 6,745,219 B1.

ZHANG, D., AND LI, H. 2008. A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms. *IEEE Trans. Industrial Electronics*, 55, 551-561.

ZHANG, Z., ANANTHARAM, V., WAINWRIGHT, M.J. AND NIKOLIC, B. 2010. An efficient 10GBASE-T Ethernet LDPC decoder design with low error floors. *IEEE J. Solid-State Circuits*, 45, 843-855.