Computation on Stochastic Bit Streams: Digital Image Processing Case Studies

Peng Li, *Student Member, IEEE,* David J. Lilja, *Fellow, IEEE,* Weikang Qian, *Member, IEEE,* Kia Bazargan, *Senior Member, IEEE,* and Marc Riedel, *Senior Member, IEEE,*

***Abstract*—Maintaining the reliability of integrated circuits as transistor sizes continue to shrink to nanoscale dimensions is a significant, looming challenge for the industry. Computation on stochastic bit streams, which could replace conventional deterministic computation based on a binary radix, allows similar computation to be performed more reliably and often with less hardware area. Prior work discussed a variety of specific stochastic computational elements (SCEs) for applications such as artificial neural networks and control systems. Recently, very promising new SCEs have been developed based on finite-state machines (FSMs). In this paper, we introduce new SCEs based on FSMs for the task of digital image processing. We present five digital image processing algorithms as case studies of practical applications of the technique. We compare the error tolerance, hardware area, and latency of stochastic implementations to that of conventional deterministic implementations using binary radix encoding. We also provide a rigorous analysis of a particular function, the stochastic linear gain function, which had only been validated experimentally in prior work.**

***Index Terms*—Fault tolerance, stochastic computing, finite state machine, digital image processing.**

I. Introduction

Future device technologies, such as nanoscale CMOS transistors, are expected to become ever more sensitive to system and environmental noise and to process, voltage, and thermal variations [1], [2]. Conventional design methodologies typically over-design systems to ensure error-free results. For example, at the circuit level, transistor sizes and the operational voltage can be increased in critical circuits to better tolerate the impacts of noise. At the architecture level, fault tolerant techniques, such as triple modular redundancy (TMR), can further enhance system reliability. The trade-off for the fault tolerance is typically more hardware resources. As device scaling continues, this overdesign methodology will consume even more hardware resources, which can limit the advantages of further scaling.

In the paradigm of computation on stochastic bit streams, logical computations are performed on values encoded in randomly streaming bits [3], [4], [5], [6], [7], [8], [9]. The technique can gracefully tolerate high levels of errors. Furthermore, complex operations can often be performed with remarkably simple hardware. The images in Fig. 1 illustrate the fault tolerance capability of this technique for the kernel density estimation (KDE)-based image segmentation algorithm. A stochastic implementation is compared to both a conventional deterministic implementation and a TMR-based implementation [10]. As the soft error injection rate increases, the output of the conventional deterministic implementation rapidly degrades until the image is no longer recognizable. The TMR-based implementation, shown in the second row of the figure, can tolerate higher soft error rates, although its output also degrades until the image is unrecognizable. The last row, however, shows that the implementation using stochastic bit streams is able to produce the correct output even at higher error rates.

The concept of computation on stochastic bit streams was first introduced in 1960s by Gaines [3]. He discussed basic stochastic computational elements (SCEs) such as multiplication and (scaled) addition. These SCEs can be implemented using very simple combinational logic. For example, multiplication can be implemented using an AND gate, and (scaled) addition can be implemented using a multiplexer. Subsequently, researchers developed SCEs using sequential logic to implement more sophisticated operations such as division, the exponentiation function, and the tanh function [4]. These SCEs were used in applications such as artificial neural networks (ANNs), control systems, and communication systems. For example, Brown and Card implemented the soft competitive learning algorithm Zhang et al. [11] implemented a proportional-integral (PI) controller for an induction motor drive. Low-density parity-check (LDPC) decoders used in communication systems have been implemented stochastically [12], [13], [14], [15], [16], [17]. Recently, novel SCEs have been proposed based on finite-state machines (FSMs) for functions such as absolute value, exponentiation on an absolute value, comparison, and a two-parameter stochastic linear gain [18].

In this paper, we discuss the application of such SCEs for specific digital image processing algorithms as case studies for the technique. The algorithms are: image edge detection, median filter-based noise reduction, image contrast stretching, frame difference-based image segmentation, and KDE-based image segmentation. We analyze the error tolerate of the technique and compare it to conventional techniques based on binary radix encoding. We also analyze and compare the hardware cost, latency and energy consumption. Another contribution of the paper is a rigorous analysis of a particular function, the stochastic linear gain function proposed by Brown and Card [4]: it had only been validated experimentally in prior work.

Peng Li, David J. Lilja, Kia Bazargan, and Marc Riedel are with the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, MN, 55455 USA e-mail: {lipeng, lilja, kia, mriedel}@umn.edu.

Weikang Qian is with University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, 200240 China e-mail: qianwk@sjtu.edu.cn.

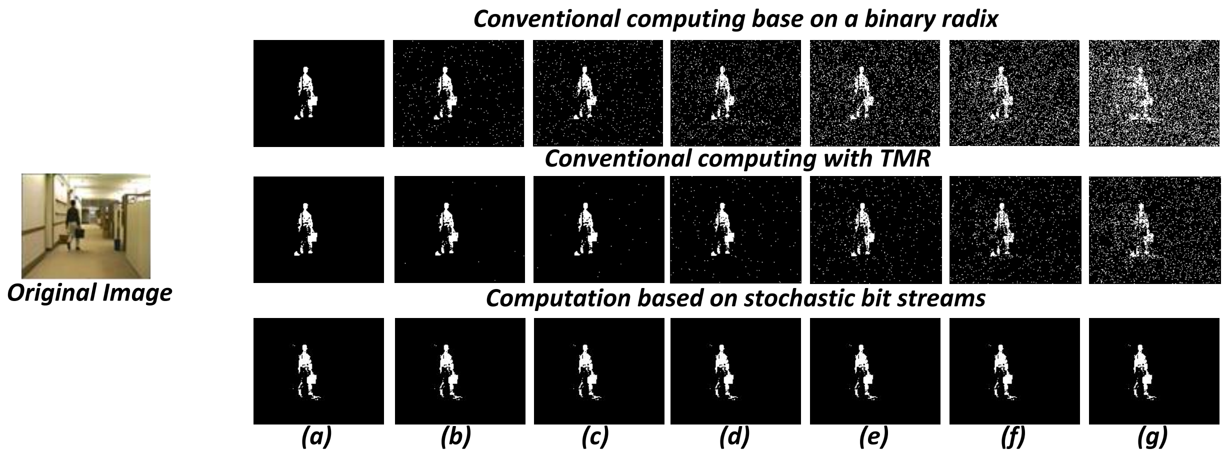
Conventional computing base on a binary radix
Conventional computing with TMR
Original Image
Computation based on stochastic bit streams
(a)
(b)
(c)
(d)
(e)
(f)
(g)

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

The remainder of this paper is organized as follows. Section II introduces the SCEs. Section III demonstrates the stochastic implementations of the five digital image processing algorithms. Section IV describes the experimental methodology and measurement results. In that section, we also discuss the solution to the latency issue and analyze why the stochastic computing technique is more fault-tolerant. Conclusions are drawn in Section V. The proof of the stochastic linear gain function can be found in the appendix.

II. Stochastic Computational Elements

Before we introduce the SCEs, it is necessary to explain some basic concepts in stochastic computing, including coding formats and conversion approaches. In stochastic computing, computation in the deterministic Boolean domain is transformed into probabilistic computation in the real domain [3], [4]. Gaines [3] proposed both a unipolar coding format and a bipolar coding format for stochastic computing. These two coding formats are the same in essence, and can coexist in a single system. The trade-off between these two coding formats is that the bipolar format can deal with negative numbers directly, while given the same bit stream length *L*, the precision of the unipolar format is twice that of the bipolar format.

In the unipolar coding format, a real number x in the unit interval (i.e., $0 \leq x \leq 1$) corresponds to a bit stream X. The probability that each bit in the stream is a one is $P(X = 1) = x$. For example, the value $x = 0.3$ would be represented by a random stream of bits such as 0100010100, where 30% of the bits are "1" and the remainder are "0."

In the bipolar coding format, the range of a real number x is extended to $-1 \leq x \leq 1$, however, the probability that each bit in the stream is a one is $P(X = 1) = \frac{x+1}{2}$. For example, the same random stream of bits used above, 0100010100, will represent $x = -0.4$ in the bipolar coding format.

Some interface circuits, such as the digital to stochastic converter proposed by Brown and Card [4] and the randomizer unit proposed by Qian et al. [19], can be used to convert a digital value x to a stochastic bit stream *X*. A counter, which counts the number of "1" bits in the stochastic bit stream, can be used to convert the stochastic bit stream back to the corresponding digital value [4], [19].

With stochastic computing, some basic arithmetic operations can be very simply implemented using combinational logic, such as multiplication, scaled addition, and scaled subtraction. More complex arithmetic operations can be implemented using sequential logic, such as the exponentiation and tanh functions. We will introduce each of these SCEs as follows.

A. Multiplication

Multiplication can be implemented using an AND gate for the unipolar coding format or an XNOR gate for the bipolar coding format, which had been explained by Brown and Card [4].

B. Scaled Addition and Subtraction

In stochastic computing, we cannot compute general addition or subtraction, because these operations can result a value greater than 1 or less than 0, which cannot be represented as a probability value. Instead, we perform scaled addition and subtraction. As shown in Fig. 2, we can implement scaled addition using a multiplexer (MUX) for both the unipolar and the bipolar coding formats, and scaled subtraction using a NOT gate and a MUX. In Fig. 2(a), with the unipolar coding format, the values represented by the stream A, B, S, and C are $a = P(A = 1)$, $b = P(B = 1)$, $s = P(S = 1)$, and $c = P(C = 1)$. Based on the logic function of the MUX,

$$\begin{aligned}
c &= P(C = 1) \\
&= P(S = 1 \text{ and } A = 1) + P(S = 0 \text{ and } B = 1) \\
&= P(S = 1) \cdot P(A = 1) + P(S = 0) \cdot P(B = 1) \\
&= s \cdot a + (1 - s) \cdot b.
\end{aligned}$$

Thus, with the unipolar coding format, the computation performed by a MUX is the scaled addition of the two inputs a and b, with a scaling factor of s for a and $1 - s$ for b.

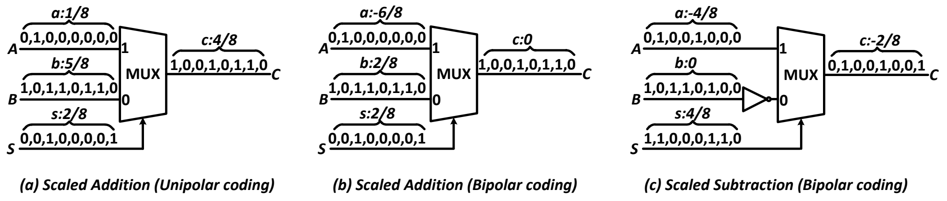

Fig. 2. Scaled addition and subtraction. (a) Scaled addition with the unipolar coding. Here the inputs are $a = 1/8$ and $b = 5/8$. The scaling factor is $s = 2/8$. The output is $2/8 \times 1/8 + (1 - 2/8) \times 5/8 = 4/8$, as expected. (b) Scaled addition with the bipolar coding. Here the inputs are $a = -6/8$ and $b = 2/8$. The scaling factor is $s = 2/8$. The output is $2/8 \times (-6/8) + (1 - 2/8) \times 2/8 = 0$, as expected. (c) Scaled subtraction with the bipolar coding. Here the inputs are $a = -4/8$ and $b = 0$. The scaling factor is $s = 4/8$. The output is $4/8 \times (-4/8) + (1 - 4/8) \times 0 = -2/8$, as expected.

In Fig. 2(b), the values of the stochastic bit streams A, B, and C are encoded with the bipolar coding format. However, the value of the stochastic bit stream S is encoded with the unipolar coding format. Thus, we have $a = 2P(A = 1) - 1$, $b = 2P(B = 1) - 1$, and $c = 2P(C = 1) - 1$. Note that we still have $s = P(S = 1)$. Based on the logic function of the MUX, we have

$$\begin{aligned} P(C = 1) = P(S = 1) \cdot P(A = 1) \\ + P(S = 0) \cdot P(B = 1), \end{aligned}$$

i.e.,

$$\frac{c+1}{2} = s \cdot \frac{a+1}{2} + (1 - s) \cdot \frac{b+1}{2}.$$

Thus, we still have $c = s \cdot a + (1 - s) \cdot b$.

The scaled subtraction can be implemented with a MUX and a NOT gate, as shown in Fig. 2(c). However, it only works for bipolar coding because subtraction can result negative output value and the unipolar coding format cannot represent negative values. Similar to the case of scaled addition with the bipolar coding, the stochastic bit streams A, B, and C use the bipolar coding format and the stochastic bit stream S uses the unipolar coding format. Based on the logic function of the circuit,

$$\begin{aligned} P(C = 1) = P(S = 1) \cdot P(A = 1) \\ + P(S = 0) \cdot P(B = 0), \end{aligned}$$

i.e.,

$$\frac{c+1}{2} = s \cdot \frac{a+1}{2} + (1 - s) \cdot \frac{1-b}{2}.$$

Thus, we have $c = s \cdot a - (1 - s) \cdot b$. It can be seen that, with the bipolar coding format, the computation performed by a MUX and a NOT gate is the scaled subtraction with a scaling factor of s for a and $1 - s$ for b.

C. Stochastic Exponentiation Function

The stochastic exponentiation function was developed by Brown and Card [4]. The state transition diagram of the FSM implementing this function is shown in Fig. 3. It has totally N states, which are S_0, S_1, $\cdots$, and S_{N-1}. X is the input of this state machine. The output Y of this state machine only depends on the current state S_i $(0 \leq i \leq N - 1)$. If $i \leq N - G - 1$ $(G \ll N)$, $Y = 1$; else $Y = 0$. Assume that the input X is a Bernoulli sequence. Define the probability that each bit in the input stream X is one to be P_X, and the probability that each bit in the corresponding output stream Y is one to be P_Y. Define x to be the bipolar coding of the bit stream X, and y to be the unipolar coding of the bit stream Y, i.e.,

$$x = 2P_X - 1, \quad y = P_Y.$$

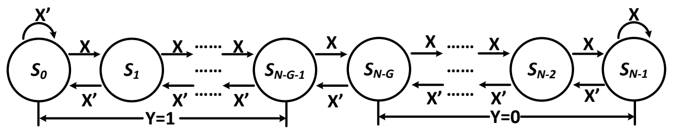

Fig. 3. State transition diagram of the FSM implementing the stochastic exponentiation function.

Brown and Card proposed that,

$$y \approx \begin{cases} e^{-2Gx}, & 0 < x \leq 1, \\ 1, & -1 \leq x \leq 0. \end{cases} \tag{1}$$

The corresponding proof can be found in [18].

D. Stochastic Exponentiation on an Absolute Value

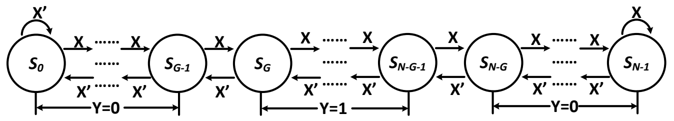

Fig. 4. State transition diagram of the FSM implementing the stochastic exponentiation on an absolute value.

Based on the stochastic exponentiation function proposed by Brown and Card [4], Li and Lilja [10] developed a stochastic exponentiation function on an absolute value, i.e.,

$$y = e^{-2G|x|}, \tag{2}$$

where x is the bipolar coding of the bit stream X, and y is the unipolar coding of the bit stream Y. The state transition diagram of the FSM is shown in Fig. 4. Note that $G \ll N$. The proof of this function can be found in [10].

E. Stochastic Tanh Function

The stochastic tanh function was also developed by Brown and Card [4]. The state transition diagram of the FSM implementing this function is shown in Fig. 5. If x and y are the bipolar coding of the bit streams X and Y, respectively, i.e.,

$x = 2P_X - 1$ and $y = 2P_Y - 1$, Brown and Card proposed that the relationship between x and y was,

$$y = \frac{e^{\frac{N}{2}x} - e^{-\frac{N}{2}x}}{e^{\frac{N}{2}x} + e^{-\frac{N}{2}x}}. \quad (3)$$

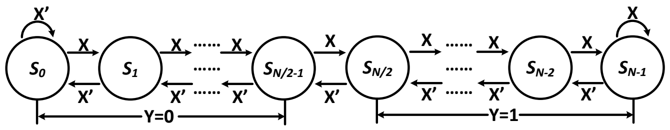

Fig. 5. State transition diagram of the FSM implementing the stochastic tanh function.

The corresponding proof can be found in [18]. In addition, Li and Lilja [18] proposed to use this function to implement a stochastic comparator. Indeed, the stochastic tanh function approximates a threshold function if N approaches infinity,

$$\lim_{N\to\infty} P_Y = \begin{cases} 0, & 0 \le P_X < 0.5, \\ 0.5, & P_X = 0.5, \\ 1, & 0.5 < P_X \le 1. \end{cases}$$

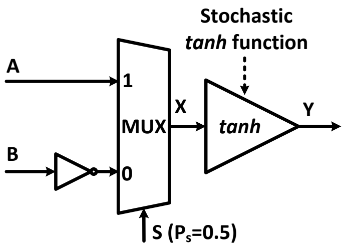

Fig. 6. The stochastic comparator.

The stochastic comparator is built based on the stochastic tanh function and the scaled subtraction as shown in Fig. 6. $P_S = 0.5$ in the selection bit S of the MUX denotes a stochastic bit stream in which half of its bits are ones. Note that the input of the stochastic tanh function is the output of the scaled subtraction. Based on this relationship, the function of the circuit shown in Fig. 6 is:

$$if\ (P_A < P_B)\ then\ P_Y \approx 0; else\ P_Y \approx 1,$$

where P_A, P_B, and P_Y are the probabilities of ones in the stochastic bit streams A, B, and Y, respectively.

F. Stochastic Linear Gain Function

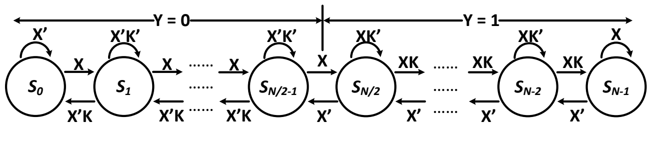

Fig. 7. State transition diagram of the FSM implementing the stochastic linear gain function.

Brown and Card also introduced another FSM-based SCE to compute the linear gain function [4]. The state transition diagram of this FSM is shown in Fig. 7. Note that this FSM has two inputs, X and K. The input K, which is also a stochastic bit stream, is used to control the linear gain. Brown and Card only empirically demonstrated that the configuration in Fig. 7 performs the linear gain function stochastically. However, if we define P_K as the probability that each bit in the stream K is one, the relationship between P_K and the value of the linear gain was not provided in their previous work. Li and Lilja [18] showed that (without proof) the configuration in Fig. 7 performs the following function,

$$P_Y = \begin{cases} 0, & 0 \le P_X \le \frac{P_K}{1+P_K}, \\ \frac{1+P_K}{1-P_K} \cdot P_X - \frac{P_K}{1-P_K}, & \frac{P_K}{1+P_K} \le P_X \le \frac{1}{1+P_K}, \\ 1, & \frac{1}{1+P_K} \le P_X \le 1. \end{cases} \quad (4)$$

In this paper, we give the corresponding proof in the appendix.

G. Two-Parameter Stochastic Linear Gain Function

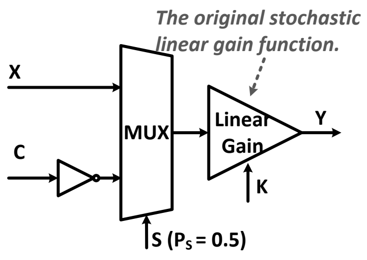

Fig. 8. A two-parameter stochastic linear gain function.

Based on the stochastic linear gain function proposed by Brown and Card [4], Li and Lilja [18] developed a two-parameter linear gain function, which can control both the slope and the position of the linear gain. The circuit is shown in Fig. 8. If we define

$$g_l = max\{0, \quad P_C - \frac{1-P_K}{1+P_K}\},$$

$$g_r = min\{1, \quad P_C + \frac{1-P_K}{1+P_K}\},$$

where P_C and P_K are the probabilities of ones in the stochastic bit streams C and K, the circuit in Fig. 8 performs the following function,

$$P_Y = \begin{cases} 0, & 0 \le P_X < g_l, \\ \frac{1}{2} \cdot \frac{1+P_K}{1-P_K} \cdot (P_X - P_C) + \frac{1}{2}, & g_l \le P_X \le g_r, \\ 1, & g_r < P_X \le 1, \end{cases}$$

where P_X and P_Y are the probabilities of ones in the stochastic bit streams X and Y. In Fig. 8, note that the input of the original stochastic linear gain function is the output of the scaled subtraction. Thus, the above equation can be proved based on this relationship. It can be seen that the center of this two-parameter stochastic linear gain function is moved to P_C. However, the new gain changes to one half of the original gain: in the original stochastic linear gain function, the gain is $\frac{1+P_K}{1-P_K}$; in the new one, the gain is $\frac{1}{2}\cdot\frac{1+P_K}{1-P_K}$.

H. Stochastic Absolute Value Function

Li and Lilja [18] also developed a stochastic absolute value function. The state transition diagram is shown in Fig. 9. The output Y of this state machine is only determined by the current state S_i $(0 \le i \le N-1)$. If $0 \le i < N/2$ and i is even, or $N/2 \le i \le N-1$ and i is odd, $Y = 1$; else $Y = 0$. The approximate function is,

$$y = |x|, \tag{5}$$

where x and y are the bipolar coding of P_X and P_Y. The proof of this function can be found in Li and Lilja [18].

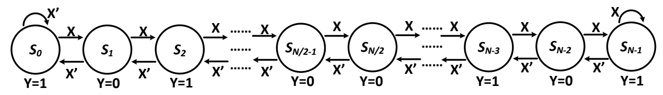

Fig. 9. State transition diagram of the FSM implementing the stochastic absolute value function.

III. Stochastic Implementations of Image Processing Algorithms

In this section, we use five digital image processing algorithms as case studies to demonstrate how to apply the SCEs introduced in the previous section in practical applications. These algorithms are image edge detection, median filter-based noise reduction, image contrast stretching, frame difference-based image segmentation, and KDE-based image segmentation [20], [21]. We use a 16-state finite-state machine (FSM) to compute the absolute value, and 32-state FSMs to compute the tanh function, the exponentiation function, and the linear gain function in all the five algorithms.

A. Edge Detection

Classical methods of edge detection involve convolving the image with an operator (a 2-D filter), which is constructed to be sensitive to large gradients in the image while returning values of zero in uniform regions [20]. There are a large number of edge detection operators available, each designed to be sensitive to certain types of edges. Most of these operators can be efficiently implemented by the SCEs introduced in this paper. Here we consider only Robert's cross operator as shown in Fig. 10 as an example [20].

This operator consists of a pair of 2×2 convolution kernels. One kernel is simply the other rotated by 90°. An approximate magnitude is computed using: $G = |G_X| + |G_Y|$, i.e.,

$$s_{i,j} = \frac{1}{2}(|r_{i,j} - r_{i+1,j+1}| + |r_{i,j+1} - r_{i+1,j}|),$$

where $r_{i,j}$ is the pixel value at location (i,j) of the original image and $s_{i,j}$ is the pixel value at location (i,j) of the processed image. Note that the coefficient $\frac{1}{2}$ is used to scale $s_{i,j}$ to [0, 255], which is the range of the grayscale pixel value. The conventional implementation of this algorithm is shown in Fig. 11(a).

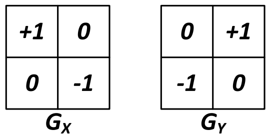

Fig. 10. Robert's cross operator for edge detection.

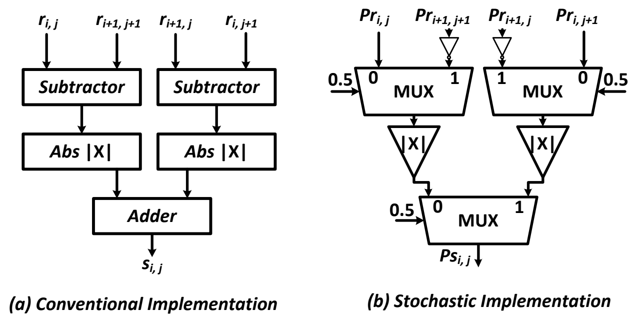

Fig. 11. The conventional implementation and the stochastic implementation of the Robert's cross operator based edge detection.

The stochastic implementation of this algorithm is shown in Fig. 11(b), in which $P_{r_{i,j}}$ is the probability of ones in the stochastic bit stream which is converted from $r_{i,j}$, i.e., $P_{r_{i,j}} = \frac{r_{i,j}}{256}$. So are $P_{r_{i+1,j}}$, $P_{r_{i,j+1}}$, and $P_{r_{i+1,j+1}}$. Suppose that under the bipolar encoding, the values represented by the stochastic bit streams $P_{r_{i,j}}$, $P_{r_{i+1,j}}$, $P_{r_{i,j+1}}$, $P_{r_{i+1,j+1}}$, and $P_{s_{i,j}}$ are $a_{r_{i,j}}$, $a_{r_{i+1,j}}$, $a_{r_{i,j+1}}$, $a_{r_{i+1,j+1}}$, and $a_{s_{i,j}}$, respectively. Then, based on the circuit, we have

$$a_{s_{i,j}} = \frac{1}{4}(|a_{r_{i,j}} - a_{r_{i+1,j+1}}| + |a_{r_{i,j+1}} - a_{r_{i+1,j}}|).$$

Because $a_{s_{i,j}} = 2P_{s_{i,j}} - 1$ and $a_{r_{i,j}} = 2P_{r_{i,j}} - 1$ ($a_{r_{i+1,j}}$, $a_{r_{i,j+1}}$, $a_{r_{i+1,j+1}}$ are defined in the same way), we have

$$\begin{aligned} P_{s_{i,j}} &= \frac{1}{4}\left(\left|P_{r_{i,j}} - P_{r_{i+1,j+1}}\right| + \left|P_{r_{i,j+1}} - P_{r_{i+1,j}}\right|\right) + \frac{1}{2} \\ &= \frac{s_{i,j}}{512} + \frac{1}{2}. \end{aligned}$$

Thus, by counting the number of ones in the output bit stream, we can convert it back to $s_{i,j}$.

B. Noise Reduction Based on The Median Filter

The median filter replaces each pixel with the median of neighboring pixels. It is quite popular because, for certain types of random noise (such as salt-and-pepper noise), it provides excellent noise-reduction capabilities, with considerably less blurring than the linear smoothing filters of a similar size [20]. A hardware implementation of a 3 × 3 median filter based on a sorting network is shown in Fig. 12. Its basic unit and the corresponding conventional implementation are shown in Fig. 13, which is used to sort two inputs in ascending order.

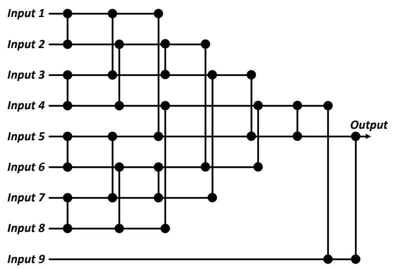

Fig. 12. Hardware implementation of the 3 × 3 median filter based on a sorting network.

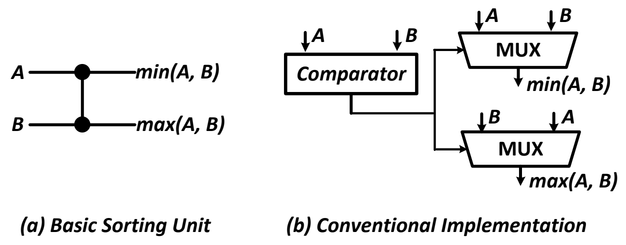

Fig. 13. Basic unit and the corresponding conventional implementation in the sorting network.

The stochastic version of this basic unit is shown in Fig. 14, which is implemented by the stochastic comparator introduced in Section II-E with a few modifications. In Fig. 14, if $P_A > P_B$, $P_S \approx 1$, the probability of ones in the output of "$MUX1$" is P_B, which is the minimum of (P_A, P_B), and the probability of ones in the output of "$MUX2$" is P_A, which is the maximum of (P_A, P_B); if $P_A < P_B$, $P_S \approx 0$, the probability of ones in the output of "$MUX1$" is P_A, which is the minimum of (P_A, P_B), and the probability of ones in the output of "$MUX2$" is P_B, which is the maximum of (P_A, P_B); if $P_A = P_B$, $P_S \approx 0.5$, both the probabilities of ones in the outputs of $MUX1$ and $MUX2$ should be very close to $\frac{P_A+P_B}{2} = P_A = P_B$. Based on this circuit, we can implement the sorting network shown in Fig. 12 stochastically.

C. Contrast Stretching

Image contrast stretching, or normalization, is used to increase the dynamic range of the gray levels in an image. One of the typical transformations used for contrast stretching and the corresponding conventional implementation are shown in Fig. 15.

It can be seen that a closed-form expression of the piecewise linear gain function shown in Fig. 15(a) is,

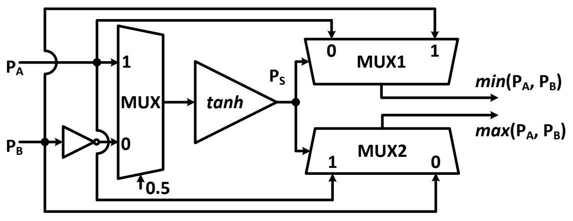

Fig. 14. The stochastic implementation of the basic sorting unit.

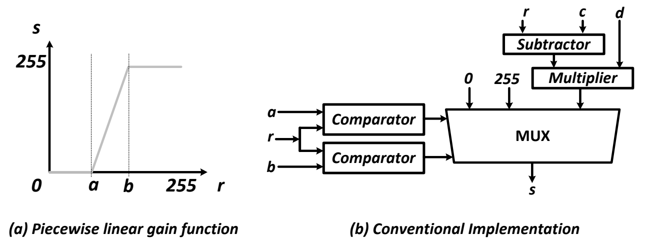
s
255
0
a
b
255
r
r
c
d
Subtractor
0
255
Multiplier
a
r
b
Comparator
Comparator
MUX
s
(a) Piecewise linear gain function
(b) Conventional Implementation

Fig. 15. A piecewise linear gain function used in image contrast stretching and its conventional implementation. In the conventional implementation, we set $d = \frac{255}{b-a}$ and $c = \frac{255a}{b-a}$ based on equation (6).

$$s = \begin{cases} 0, & 0 \le r \le a, \\ \frac{255}{b-a} \cdot (r - 0.5 \cdot (a+b)) + 128, & a < r < b, \\ 255, & b \le r \le 255. \end{cases} \tag{6}$$

The conventional implementation of the above function is shown in Fig. 15(b), which includes some complex arithmetic circuits such as subtractor and multiplier. However, this function can be efficiently implemented stochastically using the two-parameter stochastic linear gain function introduced in Fig.8 of Section II-G. In that circuit, we set

$$P_X = \frac{r}{255}, P_K = \frac{510 + a - b}{510 - a + b}, P_C = \frac{a+b}{510},$$

then,

$$P_Y = \begin{cases} 0, & 0 \le P_X \le \frac{a}{255}, \\ \frac{r-0.5\cdot(a+b)}{b-a} + \frac{1}{2}, & \frac{a}{255} < P_X < \frac{b}{255}, \\ 1, & \frac{b}{255} \le P_X \le 1. \end{cases}$$

It can be seen that $P_Y = \frac{s}{255}$. Thus, by counting the number of ones in the output bit stream, we can convert it back to s.

D. Frame Difference-Based Image Segmentation

If we define the value of a pixel at the current frame as X_t, and the value of a pixel at the same location of the previous frame as X_{t-1}, the frame difference-based image segmentation uses the difference between X_t and X_{t-1} to see if it is greater than a predefined threshold Th. If yes, the pixel at the current frame is set to the foreground; otherwise, it is set to the background.

The stochastic implementation and the conventional implementation of this algorithm are shown in Fig. 17. In the stochastic implementation, we set

$$P_{X_t} = \frac{X_t}{255}, P_{X_{t-1}} = \frac{X_{t-1}}{255}, P_{Th} = \frac{Th}{510} + \frac{1}{2},$$

then the probability (P_D) of the ones in the output bit stream of the stochastic absolute value function is,

$$P_D = \frac{|X_t - X_{t-1}|}{510} + \frac{1}{2}.$$

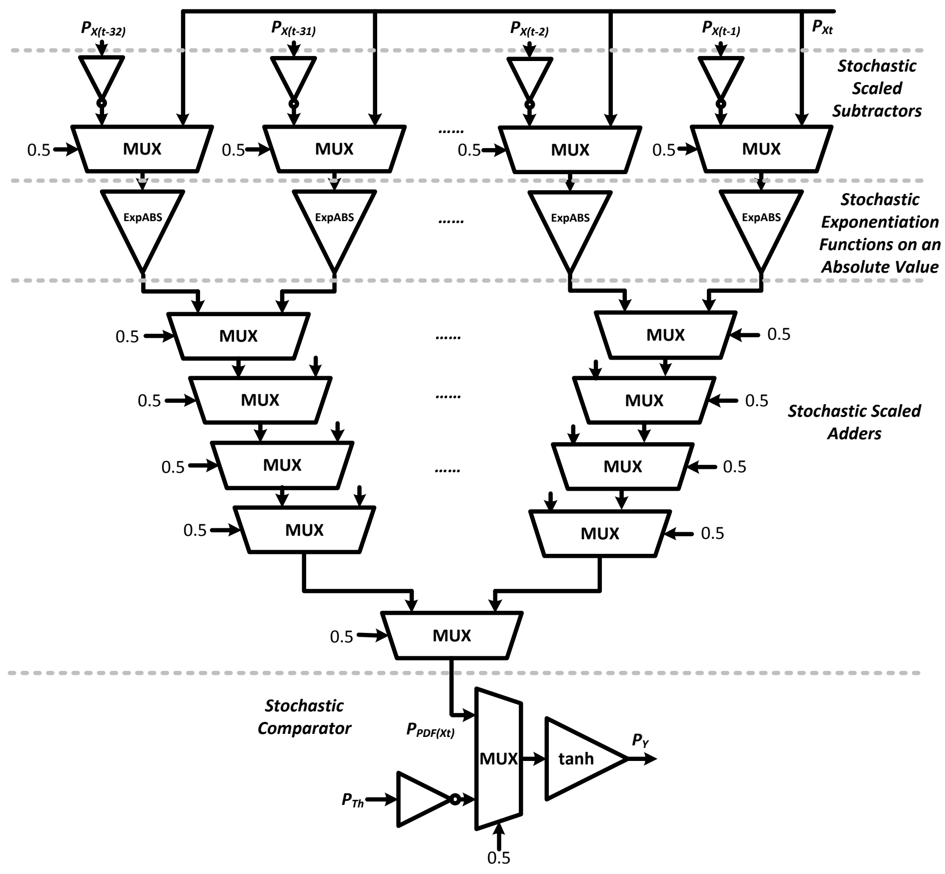

Fig. 16. The stochastic implementation of the KDE-based image segmentation algorithm.

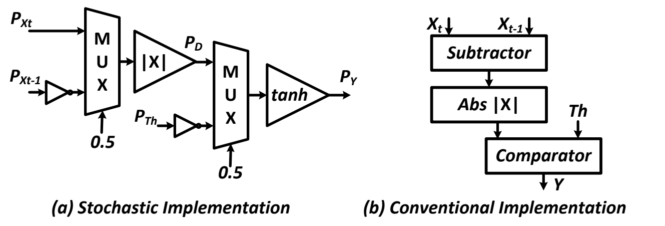

Fig. 17. The stochastic implementation and the conventional implementation of frame difference based image segmentation.

Notice that we convert the bipolar encodings into the probabilities of ones in the bit stream. The bit stream P_Y is the output of the stochastic comparator with inputs P_D and P_{Th}. Based on the function of the stochastic comparator, if $|X_t - X_{t-1}| < Th$, $P_Y = 0$, which means the current pixel belongs to the background; else $P_Y = 1$, which means the current pixel belongs to the foreground.

E. KDE-Based Image Segmentation

KDE is another image segmentation algorithm which is normally used for object recognition, surveillance, and tracking. The basic approach of this algorithm is to build a background model to capture the very recent information about a sequence of images while continuously updating this information to capture fast changes in the scene. This can be used to extract changes in a video stream in real-time, for instance. Since the intensity distribution of a pixel can change quickly, the density function of this distribution must be estimated at any moment of time given only the very recent history information. Let (X_t, X_{t-1}, X_{t-2}, ..., X_{t-n}) be a recent sample of intensity values of a pixel. Using this sample of values, the probability density function (PDF) describing the distribution of intensity values that this pixel will have at time t can be non-parametrically estimated using the kernel estimator *K*,

$$PDF(X_t) = \frac{1}{n}\sum_{i=1}^{n} K(X_t - X_{t-i}).$$

The kernel *K* should be a symmetric function. For example, if we choose our kernel estimator K to be $e^{-4|x|}$, then the PDF can be estimated as:

$$PDF(X_t) = \frac{1}{n}\sum_{i=1}^{n} e^{-4|X_t - X_{t-i}|}. \tag{7}$$

Using this probability estimator, a pixel is considered a background pixel if $PDF(X_t)$ is less than a predefined threshold Th. Both the conventional implementation and the stochastic implementation of this algorithm are mapped from equation (7). For example, Fig. 16 shows the stochastic implementation of this algorithm based on 32 frame parameters (i.e., $n = 32$ in equation (7)). Similar to the previous four algorithms, we can get the conventional implementation of this algorithm by mapping the SCEs to the corresponding conventional computing elements. The block diagram of the conventional implementation is omitted due to space limitation.

In the stochastic implementation shown in Fig.16, the first part is 32 *"Stochastic Scaled Subtracters,"* which are used to compute $0.5 \cdot (P_{Xt} - P_{X(t-i)})$; the second part is 32 *"Stochastic Exponentiation Functions on an Absolute Value,"* which is used to implement the kernel estimator and has been introduced in Section II-D; the third part of this circuit is 31 *"Stochastic Scaled Adders,"* which computes $P_{PDF}(Xt)$; and the last part of this circuit is a *"Stochastic Comparator,"* which is used to produce the final segmentation output. Based on the circuit shown in Fig. 16, if $PDF(X_t) < Th$, $P_Y = 0$, which means the current pixel belongs to the background; else $P_Y = 1$, which means the current pixel belongs to the foreground.

IV. Experimental Results

In this section, we present the experimental results of the stochastic and conventional implementations of the five digital image processing algorithms discussed in Section III. The experiments include comparisons in terms of fault-tolerance and hardware resources. We use the Xilinx System Generator [22] to estimate hardware costs since the systems built by this tool are very close to the real hardware, and can be easily verified using an FPGA.

A. Simulation Results

We use 1024 bits to represent a pixel value in stochastic computing. The simulation results of the KDE-based image segmentation have been shown in Fig. 1. The others are shown in Fig. 18. It can be seen that the simulation results generated by the stochastic implementations are almost the same as the ones generated by the conventional implementations based on binary radix. In fact, computation based on stochastic bit streams does have some errors compared to computation based on a binary radix. For applications such as image/audio processing and artificial neural networks, however, these errors can be ignored because humans can hardly see such small differences.

B. Hardware Resources Comparison

In Section III, we introduced circuit structures of both the stochastic and conventional implementations of the five digital image processing algorithms. The hardware cost of these implementations in terms of the equivalent two-input NAND gates is shown in Table I. It can be seen that the stochastic implementation uses substantially fewer hardware resources than the conventional implementation especially when the algorithm needs many computational elements. This is mainly because SCEs take much less hardware than the ones based on the binary radix encoding used in the conventional implementation. For example, multiplication can be implemented using a single AND gate stochastically, and the exponentiation function can be implemented stochastically using an FSM.

Note that in our hardware evaluation, we did not consider the hardware cost of the interface circuitry, i.e., the hardware cost of encoding values by random bit streams and decoding random bit streams into other representations. In our current implementations, the interface circuitry is designed based on the linear feedback shift register (LFSR) because it is easy to implement and simulate. However, LFSR-based interface circuitry is expensive. In future work, we are going to design the interface circuitry based on a sigma-delta analog to digital conversion technique. The hardware cost for this technique is much less than the LFSR-based technique, and the random bit streams can be generated almost for free.

TABLE I
Hardware resources comparison in terms of the equivalent two-input NAND gates.

	Conventional	Stochastic	Ratio (conv. vs. stoc.)
Edge Detection	776	110	7.05
Frame Difference	486	107	4.54
Noise Reduction	7.2K	1.25K	5.76
Contrast Stretching	896	54	16.6
KDE	400K	1.5K	267

C. Fault-Tolerance Comparison

We test the fault-tolerance of both implementations by randomly injecting soft errors into the internal circuits, and measuring the corresponding average output error for each implementation. The soft errors are injected as shown in Fig. 19. To inject soft errors into a computational element, such as a MUX shown in Fig. 19(a), we insert XOR gates into all of its inputs and output. For each XOR gate, one of its inputs is connected to the original signal of the MUX and the other is connected to a global random soft error source, which is implemented using a linear feedback shift register (LFSR) and a comparator [19]. Note that if the output of the computational element is connected to the input of another computational element, we do not inject the error twice on the intermediate line.

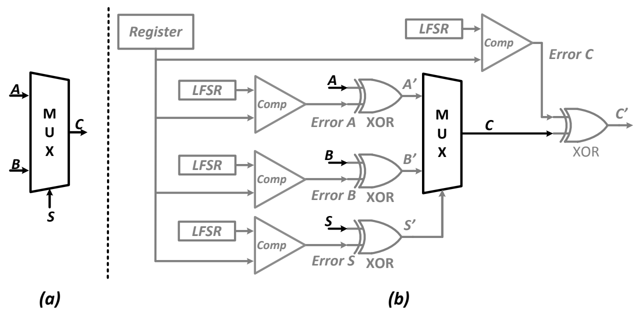

Fig. 19. Soft error injection approach for a scaled addition (MUX) in stochastic computing. (a) The original circuit. (b) The circuit for simulating injected soft errors. A, B, C, and S are the original signals. $Error\ A$, $Error\ B$, $Error\ C$, and $Error\ S$ are the soft error signals generated by the random soft error source. A', B', C', and S' are the signals corrupted by the soft errors.

If the error signal (*Error A*, *Error B*, *Error S*, or *Error C* in Fig. 19(b)) equals one, its corresponding original signal (*A*, *B*, *S*, or *C* in Fig. 19(b)) will be flipped due to the XOR

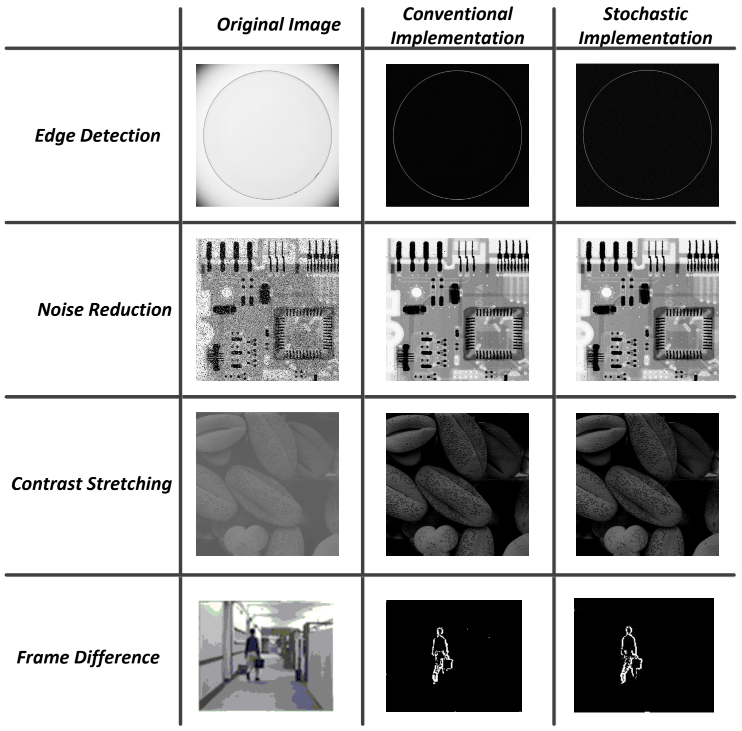

Fig. 18. Simulation results of the conventional and the stochastic implementations of the image processing algorithms. The simulation results of the KDE-based image segmentation algorithm were shown previously in Fig. 1.

TABLE II
FAULT TOLERANCE TEST RESULTS.

	The average output error E of													
	stochastic implementations							conventional implementations						
Noise	0%	1%	2%	5%	10%	15%	30%	0%	1%	2%	5%	10%	15%	30%
Edge Detection	2.05%	1.97%	2.10%	2.10%	2.09%	2.10%	2.54%	0.00%	1.37%	1.73%	3.22%	6.42%	9.92%	20.50%
Frame Difference	0.31%	0.40%	1.17%	0.38%	0.34%	0.49%	0.74%	0.00%	0.82%	2.24%	6.67%	13.71%	20.56%	38.36%
Noise Reduction	1.02%	1.04%	1.05%	1.10%	1.22%	1.34%	1.73%	0.00%	0.07%	0.20%	0.57%	1.13%	1.65%	3.06%
Contrast Stretching	2.55%	2.43%	2.33%	2.27%	2.70%	3.56%	6.88%	0.00%	0.66%	1.93%	5.56%	10.89%	15.49%	25.49%
KDE	0.89%	0.91%	0.91%	0.93%	0.96%	0.99%	1.11%	0.00%	0.54%	1.20%	3.20%	6.60%	9.71%	19.02%

gate. If the error signal equals zero, it has no influence on its corresponding original signal. The *Register* shown in Fig. 19(b) is used to control the soft error injection rate. If we use a k-bit LFSR and want to generate $p\%$ soft errors, we can set the register's value to $2^k \cdot p\%$. For example, in Fig. 19(b), assuming that we use a 10-bit LFSR and want to generate an error rate of 30%, we should set the register's value to $2^{10} \times 30\% = 307$. Note that the LFSR has a parameter called the initial value. By setting different initial values for each of the LFSRs, we can guarantee that the random soft errors generated by them are independent of each other. In addition, at each clock cycle, we only enable one LFSR to generate the soft error.

We apply this approach to each basic computational element in a circuit, such as comparator, adder, subtractor, and multiplier in the conventional implementations; and NOT gate, AND gate, XNOR gate, MUX, stochastic absolute value function, stochastic comparator, and stochastic linear gain function in the stochastic implementations. Fig. 1 uses the KDE-based image segmentation as an example to visualize the effect. We summarize the average output error of the two implementations under different soft error injection rates for all of the five algorithms in Table II. We define the height of the image to be H and the width to be W, and calculate the average output error E in Table II as follows,

$$E = \frac{\sum_{i=1}^{H}\sum_{j=1}^{W}|T_{i,j} - S_{i,j}|}{255 \cdot H \cdot W}, \tag{8}$$

where S is the output image of the conventional implementation of an algorithm without any injected soft errors, and T is the output image of the implementation of the same algorithm with injected soft errors. We define the soft error injection rate to be R, and use a linear regression model [23],

$$E = a + b \cdot R, \tag{9}$$

to analyze the relationship between R and E based on the data in Table II. For the stochastic implementations, we get $a < 5\%$ and $b \approx 0$ for all the five algorithms. This analysis means that the stochastic implementations are not sensitive to the error injection rate R because the slope of the fitted line is $b \approx 0$. However, it does have small errors even though we do not inject any soft errors in the circuit. These errors are due to approximation, quantization, and random fluctuations [19], and can be quantified by the parameter a in (9). For the conventional implementations (except the noise reduction algorithm), we get $a \approx 0$ and $b > 50\%$. This means that the conventional implementations are very sensitive to the soft error injection rate R because the slope of the fitted line $b > 50\%$.

D. Fault-Tolerance Analysis

In this section, we explain why the stochastic computing technique can tolerate more errors than the conventional computing technique based on a binary radix. Assume that the binary radix number has M bits. Consider a binary radix number

$$p = x_1 2^{-1} + x_2 2^{-2} + \cdots + x_M 2^{-M}.$$

Assume that each bit x_i $(1 \le i \le M)$ has probability ϵ being flipped. Since bit flips happen independently, we can use M independent random Boolean variable $R_1, \ldots, R_M$ to indicate the bit flips. Specifically, if $R_i = 1$, then bit x_i is flipped; otherwise, bit x_i is not flipped. We have $P(R_i = 1) = \epsilon$.

Now we can model the overall error with random variables $R_1, \ldots, R_M$. If the i-th bit is flipped, then the bit value becomes $1 - x_i$; otherwise, it is still x_i. Therefore, with bit flips randomly occurring, the value of the i-th bit is also a random variable, which can be represented as

$$X_i = (1 - x_i)R_i + x_i(1 - R_i)$$

It can be easily seen that

$$P(X_i = 1 - x_i) = \epsilon, \quad P(X_i = x_i) = 1 - \epsilon.$$

Now with independent random bit flips occurring at each bit, the binary radix number is

$$\begin{aligned} q &= X_1 2^{-1} + X_2 2^{-2} + \cdots + X_M 2^{-M} \\ &= \sum_{i=1}^{M} [(1 - x_i)R_i + x_i(1 - R_i)]2^{-i}. \end{aligned}$$

The error is

$$\begin{aligned} e &= q - p \\ &= \sum_{i=1}^{M} [(1 - x_i)R_i + x_i(1 - R_i)]2^{-i} - \sum_{i=1}^{M} x_i 2^{-i} \\ &= \sum_{i=1}^{M} (1 - 2x_i)R_i 2^{-i}, \end{aligned} \tag{10}$$

which is also a random variable.

Now we consider the mean of the error e and the variance of the error e. Since $R_1, \ldots, R_M$ are independent, from equation (10), we have

$$E[e] = \sum_{i=1}^{M} (1 - 2x_i)2^{-i} E[R_i]$$

$$Var[e] = \sum_{i=1}^{M} (1 - 2x_i)^2 2^{-2i} Var[R_i]$$

For each R_i, it can be easily shown that

$$E[R_i] = \epsilon, \quad Var[R_i] = \epsilon(1 - \epsilon).$$

Therefore

$$\begin{aligned} E[e] &= \sum_{i=1}^{M} (1 - 2x_i)2^{-i}\epsilon \\ &= \epsilon(\sum_{i=1}^{M} 2^{-i} - 2\sum_{i=1}^{M} x_i 2^{-i}) \approx (1 - 2p)\epsilon \end{aligned}$$

Notice that x_i is either 0 or 1. Thus $(1 - 2x_i)^2 = 1$. Therefore,

$$\begin{aligned} Var[e] &= \sum_{i=1}^{M} (1 - 2x_i)^2 2^{-2i}\epsilon(1 - \epsilon) \\ &= \sum_{i=1}^{M} 2^{-2i}\epsilon(1 - \epsilon) \approx \frac{1}{3}\epsilon(1 - \epsilon). \end{aligned}$$

Now we consider the stochastic encoding of the same value p as in the binary radix case. We need a bit stream of length $N = 2^M$. Suppose that the bit stream is $y_1 y_2 \ldots y_N$. We have

$$p = \frac{1}{N} \sum_{i=1}^{N} y_i.$$

Similarly, we use the random Boolean variable S_i to indicate whether bit y_i is flipped or not. If $S_i = 1$, bit y_i is flipped to $1 - y_i$; otherwise, it stays the same. Assume that the bit flip rate for the stochastic encoding is the same as that for the binary radix encoding, then we have $P(S_i = 1) = \epsilon$.

With bit flips randomly occurring, the value of the i-th bit is also a random variable, which can be represented as

$$Y_i = (1 - y_i)S_i + y_i(1 - S_i).$$

Now with independent random bit flips occurring at each bit, the actual value represented by the stream is

$$q = \frac{1}{N} \sum_{i=1}^{N} Y_i = \frac{1}{N} \sum_{i=1}^{N} [(1 - y_i)S_i + y_i(1 - S_i)].$$

The error is

$$\begin{aligned} e &= q - p \\ &= \frac{1}{N} \sum_{i=1}^{N} [(1 - y_i)S_i + y_i(1 - S_i)] - \frac{1}{N} \sum_{i=1}^{N} y_i \\ &= \frac{1}{N} \sum_{i=1}^{N} (1 - 2y_i)S_i \end{aligned}$$

Now we evaluate $E[e]$ and $Var[e]$. We apply the independence and obtain

$$E[e] = \frac{1}{N}\sum_{i=1}^{N}(1-2y_i)E[S_i]$$
$$= \frac{1}{N}\sum_{i=1}^{N}(1-2y_i)\epsilon = (1-2p)\epsilon.$$

and

$$Var[e] = \sum_{i=1}^{N}\frac{(1-2y_i)^2}{N^2}Var[S_i] = \frac{1}{N}\epsilon(1-\epsilon).$$

It can be seen that bit flips cause errors in both the binary radix encoding and the stochastic encoding. With the same bit flip rate, both the binary radix and the stochastic encoding have the same average error. However, in terms of the variation of the error, they are different. The variation of the error for the binary radix encoding is a constant independent of the number of bits. The variation of the error for the stochastic encoding is inversely proportional to the length of the bit stream. Increasing the length reduces the variation of error.

Now consider error distributions. The error distribution of the binary radix encoding and that of the stochastic encoding essentially have the same center, since they have the same mean value. However, the error distribution of the binary radix encoding is much wider than that of the stochastic encoding, since the former's variance is larger than the latter's. Thus, we have a large chance to obtain a large error by sampling the error distribution of the binary radix encoding. For example, in the KDE-based image segmentation experiment shown in Fig. 1, we can consider it a sample from the error distribution. Thus, in the experimental results, we observe large errors for the binary radix encoding.

Note that the introduction of the finite-state machine computing elements adds interdependence between bits in stochastic computing. The analysis presented above does not consider this, and instead assuming independence among the bit streams. Compared to the error in the combinational logic-based stochastic computing elements, the error due to the effect of soft errors affecting multiple bits in the finite-state machine computing elements is affected by two additional factors: the number of states of the finite-state machine and the relative location of the multiple bit flips. More states cause fewer errors, and adjacent multiple bit flips cause more errors. Based on our experiments, if the finite-state machine has more than 8 states, the adjacent multiple bit flips will not contribute too many errors (less than 1%) in the final results. We plan to make a detailed analysis of the errors in the finite-state machine-based stochastic computing elements in our future work.

E. Dealing with Long Latencies

The main issue of the stochastic computing technique is the long latency, because a pixel value is represented using a long bit stream. But this issue can be solved using a higher operational frequency or a parallel processing approach. We can use a higher operational frequency because the stochastic computing circuitry is very simple and has a shorter critical path than the corresponding conventional implementation. The parallel processing approach can be used to further reduce the latency by trading off hardware area with time. Assume we use L bits to represent a pixel value stochastically. Under the same operational frequency, the latency of the stochastic implementation without parallel processing is L times longer than the conventional implementation. However, many image processing applications exhibit data parallelism, that is, multiple pixels can be processed simultaneously. Thus, we can make multiple copies of a single stochastic implementation to process multiple pixels in parallel. We list the hardware cost in terms of equivalent two-input gates for the single stochastic implementation and the single conventional implementation for the five algorithms in Table I. Assume that n pixels are processed in parallel in the stochastic implementation. If we keep the area of the parallel stochastic implementation to be smaller than a single copy of conventional implementation, then the fourth column ''Ratio'' of Table I will be the maximum value of n for the corresponding algorithms. However, if a pixel value is represented using L bits, the parallel stochastic implementation will still be $\frac{L}{n}$ times slower than the sequential conventional implementation.

F. Energy Consumption Comparison

Although the stochastic implementations of digital image processing algorithms have lower hardware cost and can tolerate more errors, they might consume more energy. It depends on how complex the algorithms are and how many bits are used to represent a pixel value stochastically. If we use L-bit to represent a pixel value stochastically, the processing time of the stochastic implementation will be L times slower than the corresponding conventional implementation. We evaluate the energy consumption using the product of the hardware cost (shown in Table I) and the corresponding processing time. Specifically, for each algorithm,

$$\frac{E_{conv}}{E_{stoch}} = \frac{Area_{conv}}{Area_{stoch}} \cdot \frac{Time_{conv}}{Timestoch} = \frac{Area\ Ratio}{L},$$

where *Area Ratio* is shown in the fourth column of Table III. We show the comparison results in Table III. It can be seen that for a simple algorithm, such as the image edge detection, the stochastic implementation consumes more energy even if $L > 7$. For a complex algorithm, though, such as the KDE-based image segmentation, the stochastic implementation consumes less energy than the conventional implementation if $L \leq 256$.

TABLE III
ENERGY CONSUMPTION COMPARISON.

	Conventional vs. Stochastic
Edge Detection	7.05 : L
Frame Difference	4.54 : L
Noise Reduction	5.76 : L
Contrast Stretching	16.6 : L
KDE	267 : L

We also compare the area-delay product for different implementations, and the results are shown in Table IV. The area has been given in Table I. The delay is evaluated using the number of gates in the critical path of different implementations. Note that the delay of the stochastic implementations are multiplied by 2^10, because we represent the pixel value using 1024 bits in the current experiments. The data in Table IV conclude the same results as the ones in Table III: the area-delay products of different implementations depend on the the algorithm complexity. For a simple algorithm, such as the image edge detection, the area-delay product of the convention implementation is smaller. For a complex algorithm, though, such as the KDE-based image segmentation, the area-delay product of the stochastic implementation is smaller.

TABLE IV
AREA-DELAY PRODUCT COMPARISON.

	delay		area-delay product		ratio
	conv.	**stoc.**	**conv.**	**stoc.**	
Edge Detection	48	$4 \cdot 2^{10}$	37248	450560	0.08
Frame Difference	37	$5 \cdot 2^{10}$	17982	547840	0.03
Noise Reduction	85	$5 \cdot 2^{10}$	612000	6400000	0.10
Contrast Stretching	48	$5 \cdot 2^{10}$	43008	276480	0.16
KDE	85	$5 \cdot 2^{10}$	34000000	7680000	4.43

V. Conclusion and Future Work

In this paper, we first introduced a collection of stochastic computing elements (SCEs). Then, as case studies, we demonstrated the stochastic implementations of five digital image processing algorithms based on these SCEs. Our experimental results show that stochastic implementations are remarkably tolerant of soft errors and have very low hardware cost. However, due to the nature of the encoding, stochastic implementations introduce some variance. Accordingly, this technique in those applications that mandate high reliable but do not require high precision and can tolerate some variance. For such applications, representing a value by a 1024-bit stochastic bit stream will guarantee a sufficiently accurate result [19]. The most significant trade-off with a stochastic implementation is that it generally entails a high latency. However, as we discussed, this issue can be mitigated by using a higher operational frequency or by using parallel processing. In future work, we plan to investigate novel coding schemes which limit the bit stream length while preserving good randomness or pseudo-randomness. We also plan to analyze the errors in the finite-state machine-based stochastic computing elements in more detail.

Acknowledgment

This work is supported in part by National Science Foundation (NSF) grant NO. CCF-1241987. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. Part of this work has been presented in the 22nd IEEE International Conference on Application - Specific Systems, Architectures and Processors [10], and in the 29th IEEE International Conference on Computer Design [18]. The authors would like to thank the reviewers from these two conferences for their comments.

Appendix

In the appendix, we prove the stochastic linear gain function proposed by Brown and Card [4]. Based on the state transition diagram shown in Fig. 7, if the inputs X and K are stochastic bit streams with fixed probabilities, then the random state transition shown in Fig. 7 will eventually reach an equilibrium state, where the probability of transitioning from state S_i to its adjacent state S_{i+1} equals the probability of transitioning from state S_{i+1} to state S_i. Thus, we have

$$\begin{cases} P_i \cdot P_X = P_{i+1} \cdot (1 - P_X) \cdot P_K, & 0 \le i \le \frac{N}{2} - 2, \\ P_{\frac{N}{2}-1} \cdot P_X = P_{\frac{N}{2}} \cdot (1 - P_X), & \\ P_i \cdot P_X \cdot P_K = P_{i+1} \cdot (1 - P_X), & \frac{N}{2} \le i \le N - 2. \end{cases} \tag{11}$$

where P_i is the probability that the current state is S_i in the equilibrium state, P_X is the probability of ones in the input bit stream X, and P_K is the probability of ones in the input bit stream K. Note that $0 < P_X < 1$ and $0 < P_K < 1$. When P_X (or P_K) $= 0$ or 1, we cannot use equation (11) to analyze the state transition diagram shown in Fig. 7. However, in these cases, it is easy to prove the results in equation (4) based on the state transition diagram shown in Fig. 7.

Because the individual state probabilities, P_i, must sum to unity over all S_i, giving

$$\sum_{i=0}^{N-1} P_i = 1. \tag{12}$$

We define $t = \frac{P_X}{1-P_X}$, and A and B as follows,

$$A = \sum_{i=\frac{N}{2}}^{N-1} t^i \cdot P_K^{i+1-N}, \quad B = \sum_{i=0}^{\frac{N}{2}-1} t^i \cdot P_K^{-i}. \tag{13}$$

Based on (11) and (12), we have

$$P_i = \begin{cases} \frac{t^i \cdot P_K^{-i}}{A+B}, & 0 \le i \le \frac{N}{2} - 1, \\ \frac{t^i \cdot P_K^{i+1-N}}{A+B}, & \frac{N}{2} \le i \le N - 1. \end{cases} \tag{14}$$

Based on the state transition diagram shown in Fig. 7, we can represent P_Y, the probability of ones in the output bit stream Y, in terms of P_i as follows,

$$P_Y = \sum_{i=\frac{N}{2}}^{N-1} P_i. \tag{15}$$

If we substitute P_i from (14), we can rewrite (15) as $P_Y = \frac{A}{A+B}$. Note that based on equation 13, we can compute A and B using the formula of the sum of geometric sequence as follows,

$$A = \begin{cases} \frac{t^{\frac{N}{2}} \cdot P_K^{1-\frac{N}{2}} \cdot (1 - t^{\frac{N}{2}} \cdot P_K^{\frac{N}{2}})}{1 - t \cdot P_K}, & t \cdot P_K > 1, \\ \frac{N}{2} \cdot P_K^{1-N}, & t \cdot P_K = 1, \\ \frac{t^{\frac{N}{2}} \cdot P_K^{1-\frac{N}{2}}}{1 - t \cdot P_K}, & t \cdot P_K < 1, \end{cases} \tag{16}$$

$$B = \begin{cases} \frac{1 - t^{\frac{N}{2}} \cdot P_K^{-\frac{N}{2}}}{1 - t \cdot P_K^{-1}}, & \frac{t}{P_K} > 1, \\ \frac{N}{2}, & \frac{t}{P_K} = 1, \\ \frac{1}{1 - t \cdot P_K^{-1}}, & \frac{t}{P_K} < 1. \end{cases} \tag{17}$$

Next we prove equation (4) based on different intervals of P_X.

1) $0 < P_X \le \frac{P_K}{1+P_K}$
Because

$$t = \frac{P_X}{1 - P_X},$$

we have

$$P_X = \frac{t}{1+t}.$$

Since $P_X \le \frac{P_K}{1+P_K}$, we have

$$\frac{t}{1+t} \le \frac{P_K}{1+P_K},$$

i.e., $\frac{t}{P_K} \le 1$, and $t \cdot P_K = \frac{t}{P_K} \cdot P_K^2 \le P_K^2 < 1$.
Thus, based on equation (16) and (17),

$$A = \frac{t^{\frac{N}{2}} \cdot P_K^{1-\frac{N}{2}}}{1 - t \cdot P_K},$$

$$B = \begin{cases} \frac{N}{2}, & \frac{t}{P_K} = 1, \\ \frac{1}{1 - t \cdot P_K^{-1}}, & \frac{t}{P_K} < 1. \end{cases}$$

If $\frac{t}{P_K} < 1$, $\lim\limits_{N\to\infty} \frac{A}{B} = \frac{P_K - t}{1 - t \cdot P_K} \cdot \lim\limits_{N\to\infty} \left(\frac{t}{P_K}\right)^{\frac{N}{2}} = 0.$
If $\frac{t}{P_K} = 1$, $\lim\limits_{N\to\infty} \frac{A}{B} = \frac{2P_K}{(1 - t \cdot P_K)N} = 0.$
Thus, $\lim\limits_{N\to\infty} P_Y = \lim\limits_{N\to\infty} \frac{\frac{A}{B}}{fracAB + 1} = 0.$

2) $\frac{P_K}{1+P_K} < P_X < \frac{1}{1+P_K}$
Because $\frac{P_K}{1+P_K} < P_X$, we have $\frac{t}{P_K} > 1$.
Because $P_X < \frac{1}{1+P_K}$, we have $t \cdot P_K < 1$.
Thus,

$$A = \frac{t^{\frac{N}{2}} \cdot P_K^{1-\frac{N}{2}}}{1 - t \cdot P_K}, \quad B = \frac{1 - t^{\frac{N}{2}} \cdot P_K^{-\frac{N}{2}}}{1 - t \cdot P_K^{-1}}.$$

$$\lim_{N\to\infty} \frac{B}{A} = \frac{\lim\limits_{N\to\infty} (\frac{t}{P_K})^{\frac{N}{2}} - 1}{\lim\limits_{N\to\infty} (\frac{t}{P_K})^{\frac{N}{2}}} \cdot \frac{1 - t \cdot P_K}{t - P_K} = \frac{1 - t \cdot P_K}{t - P_K}.$$

Thus,

$$\lim_{N\to\infty} P_Y = \lim_{N\to\infty} \frac{1}{1 + \frac{B}{A}} = \frac{1}{1 + \frac{1 - t \cdot P_K}{t - P_K}} = \frac{1 + P_K}{1 - P_K} \cdot P_X - \frac{P_K}{1 - P_K}.$$

3) $\frac{1}{1+P_K} \le P_X < 1$
In this case, we have $t \cdot P_K \ge 1$, and $\frac{t}{P_K} \ge P_K^{-2} > 1$.
Thus,

$$A = \begin{cases} \frac{t^{\frac{N}{2}} \cdot P_K^{1-\frac{N}{2}} \cdot (1 - t^{\frac{N}{2}} \cdot P_K^{\frac{N}{2}})}{1 - t \cdot P_K}, & t \cdot P_K > 1, \\ \frac{N}{2} \cdot P_K^{1-N}, & t \cdot P_K = 1. \end{cases}$$

$$B = \frac{1 - t^{\frac{N}{2}} \cdot P_K^{-\frac{N}{2}}}{1 - t \cdot P_K^{-1}}.$$

If $t \cdot P_K > 1$,

$$\lim_{N\to\infty} \frac{B}{A} = \frac{1 - t \cdot P_K}{P_K - t} \cdot \lim_{N\to\infty} \frac{1 - (\frac{t}{P_K})^{\frac{N}{2}}}{(1 - (t \cdot P_K)^{\frac{N}{2}}) \cdot (\frac{t}{P_K})^{\frac{N}{2}}} = \frac{1 - t \cdot P_K}{P_K - t} \cdot 0 = 0.$$

If $t \cdot P_K = 1$,

$$\lim_{N\to\infty} \frac{B}{A} = \lim_{N\to\infty} \frac{P_K^N - (t \cdot P_K)^{\frac{N}{2}}}{(P_K - t) \cdot \frac{N}{2}} = \lim_{N\to\infty} \frac{P_K^N - 1}{(P_K - t) \cdot \frac{N}{2}} = 0.$$

Thus,

$$\lim_{N\to\infty} P_Y = \lim_{N\to\infty} \frac{1}{1 + \frac{B}{A}} = 1.$$

Based on the above discussion, equation (4) has been proved. The simulation result can be found in [4].

References

[1] H. Chen, J. Han, and F. Lombardi, "A transistor-level stochastic approach for evaluating the reliability of digital nanometric cmos circuits," in *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2011 IEEE International Symposium on*, pp. 60–67, IEEE, 2011.

[2] H. Chen and J. Han, "Stochastic computational models for accurate reliability evaluation of logic circuits," in *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, pp. 61–66, ACM, 2010.

[3] B. R. Gaines, "Stochastic computing systems," *Advances in Information System Science, Plenum*, vol. 2, no. 2, pp. 37–172, 1969.

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

[5] B. D. Brown and H. C. Card, "Stochastic neural computation II: Soft competitive learning," *IEEE Transactions on Computers*, vol. 50, pp. 906–920, September 2001.

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

[7] P. Li, D. J. Lilja, W. Qian, and K. Bazargan, "Using a two-dimensional finite-state machine for stochastic computation," in *International Workshop on Logic and Synthesis, IWLS'12*, 2012.

[8] P. Li, W. Qian, M. Riedel, K. Bazargan, and D. Lilja, "The synthesis of linear finite state machine-based stochastic computational elements," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pp. 757–762, IEEE, 2012.

[9] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," in *Computer-Aided Design, 2012. ICCAD 2012. IEEE/ACM International Conference on*, IEEE, 2012.

[10] P. Li and D. J. Lilja, "A low power fault-tolerance architecture for the kernel density estimation based image segmentation algorithm," in *IEEE International Conference on Application - specific Systems, Architectures and Processors, ASAP'11*, 2011.

[11] D. Zhang and H. Li, "A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms," *Industrial Electronics, IEEE Transactions on*, vol. 55, no. 2, pp. 551–561, 2008.

[12] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, vol. 39, pp. 299–301, February 2003.

[13] W. J. Gross, V. C. Gaudet, and A. Milner, "Stochastic implementation of ldpc decoders," in *The 39th Asilomar Conference on Signals, Systems and Computers, ACSSC'05*, pp. 713–717, 2005.

[14] C. Winstead, V. Gaudet, A. Rapley, and C. Schlegel, "Stochastic iterative decoders," in *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, pp. 1116–1120, IEEE, 2005.

[15] S. Sharifi Tehrani, W. Gross, and S. Mannor, "Stochastic decoding of ldpc codes," *Communications Letters, IEEE*, vol. 10, no. 10, pp. 716–718, 2006.

[16] S. Sharifi Tehrani, S. Mannor, and W. Gross, "Fully parallel stochastic LDPC decoders," *Signal Processing, IEEE Transactions on*, vol. 56, no. 11, pp. 5692–5703, 2008.

[17] T. Brandon, R. Hang, G. Block, V. Gaudet, B. Cockburn, S. Howard, C. Giasson, K. Boyle, P. Goud, S. Zeinoddin, *et al.*, "A scalable ldpc decoder asic architecture with bit-serial message exchange," *Integration, the VLSI journal*, vol. 41, no. 3, pp. 385–398, 2008.

[18] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *29th IEEE International Conference on Computer Design, ICCD'11*, 2011.

[19] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, pp. 93–105, January 2010.

[20] R. C. Gonzalez and R. E. Woods, "Digital image processing, 3rd edition," *Prentice Hall*, 2008.

[21] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," in *FRAME-RATE WORKSHOP, IEEE*, pp. 751–767, 2000.

[22] "System generator for dsp user guide," *Xilinx Inc.*, vol. version 12.3, 2010.

[23] D. J. Lilja, "Measure computer performance: a practitioner's guide, 1st edition," *Cambridge University Press*, 2000.

PLACE PHOTO HERE

Peng Li received the BSci degree in Electrical Engineering from North China Electric Power University, Beijing, in 2005; the MEng degree from the Institute of Microelectronics, Tsinghua University, Beijing, in 2008. He is currently pursuing the Ph.D. degree in the Electrical and Computer Engineering Department, University of Minnesota Twin Cities, Minneapolis, MN. His main research interests are in the area of VLSI design for digital signal processing, stochastic computing, and high performance storage architecture.

PLACE PHOTO HERE

David J. Lilja received the BS degree in Computer Engineering from Iowa State University in Ames, and the MS and PhD degrees in Electrical Engineering from the University of Illinois at Urbana-Champaign. He is currently a Professor and Head of Electrical and Computer Engineering at the University of Minnesota in Minneapolis. He has worked as a Development Engineer at Tandem Computers Inc. and has chaired and served on the program committees of numerous conferences. He was elected a Fellow of the IEEE and the AAAS. His main research interests include computer architecture, computer systems performance analysis, and high-performance storage systems, with a particular interest in the interaction of computer architecture with software, compilers, and circuits.

PLACE PHOTO HERE

Weikang Qian received his Ph.D. degree in Electrical Engineering at the University of Minnesota in 2011 and his B.Eng. degree in Automation at Tsinghua University, Beijing, China in 2006. He has been an Assistant Professor at the University of Michigan-Shanghai Jiao Tong University Joint Institute since 2011. He has research interests in the fields of computer-aided design of integrated circuits, circuit design for emerging technologies, and fault-tolerant computing. He is a member of the IEEE.

PLACE PHOTO HERE

Kia Bazargan received the BSci degree in Computer Science from Sharif University, Tehran, Iran, and the MS and PhD degrees in Electrical and Computer engineering from Northwestern University, Evanston, Illinois, in 1998 and 2000, respectively. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis. He was a guest coeditor of the ACM Transactions on Embedded Computing Systems Special Issue on Dynamically Adaptable Embedded Systems in 2003. He has served on the technical program committee of a number of IEEE/ACM-sponsored conferences (e.g., Field Programmable Gate Array (FPGA), Field Programmable Logic (FPL), Design Automation Conference (DAC), International Conference on Computer-Aided Design (ICCAD), and Asia and South Pacific DAC). He was an Associate Editor of the IEEE Transactions on CAD of Integrated Circuits and systems from 2005-2012. He was a recipient of the US National Science Foundation (NSF) Career Award in 2004. He is a Senior Member of the IEEE and the IEEE Computer Society.

PLACE PHOTO HERE

Marc D. Riedel received the BEng degree in Electrical Engineering with a minor in Mathematics from McGill University, and the MSc and PhD degrees in Electrical Engineering from Caltech. He is currently an Associate Professor of Electrical and Computer Engineering at the University of Minnesota. He is also a member of the Graduate Faculty in Biomedical Informatics and Computational Biology. From 2004 to 2005, he was a Lecturer in Computation and Neural Systems at Caltech. He has held positions at Marconi Canada, CAE Electronics, Toshiba, and Fujitsu Research Labs. His PhD dissertation titled "Cyclic Combinational Circuits" received the Charl H. Wilts Prize for the Best Doctoral Research in Electrical Engineering at Caltech. His paper "The Synthesis of Cyclic Combinational Circuits" received the Best Paper Award at the Design Automation Conference. He is a recipient of the US National Science Foundation (NSF) CAREER Award. He is a Senior Member of the IEEE and the IEEE Computer Society.