Case Studies of Logical Computation on Stochastic Bit Streams

Peng Li[1], Weikang Qian[2], David J. Lilja[1], Kia Bazargan[1], and Marc D. Riedel[1]

[1] Electrical and Computer Engineering, University of Minnesota, MN, USA, 55455,
{lipeng, lilja, kia, mriedel}@umn.edu,
[2] University of Michigan-Shanghai Jiao Tong University Joint Institute, China, 200241,
qianwk@sjtu.edu.cn.

Abstract. Most digital systems operate on a positional representation of data, such as binary radix. An alternative is to operate on random bit streams where the signal value is encoded by the probability of obtaining a one versus a zero. This representation is much less compact than binary radix. However, complex operations can be performed with very simple logic. Furthermore, since the representation is uniform, with all bits weighted equally, it is highly tolerant of soft errors (i.e., bit flips). Both combinational and sequential constructs have been proposed for operating on stochastic bit streams. Prior work has shown that combinational logic can implement multiplication and scaled addition effectively; linear finite-state machines (FSMs) can implement complex functions such as exponentiation and tanh effectively. Building on these prior results, this paper presents case studies of useful circuit constructs implement with the paradigm of logical computation on stochastic bit streams. Specifically, it describes finite state machine implementations of functions such as edge detection and median filter-based noise reduction.

1 Introduction

In a paradigm advocated by Gaines [1], logical computation is performed on stochastic bit streams: each real-valued number x $(0 \le x \le 1)$ is represented by a sequence of random bits, each of which has probability x of being one and probability $1-x$ of being zero. Compared to a binary radix representation, a stochastic representation is not very compact. However, it leads to remarkably simple hardware for complex functions; also it provides very high tolerance to soft errors.

There are two possible coding formats: a unipolar format and a bipolar format [1]. These two coding formats are the same in essence, and can coexist in a single system. In the unipolar coding format, a real number x in the unit interval (i.e., $0 \le x \le 1$) corresponds to a bit stream $X(t)$ of length L, where $t = 1, 2, ..., L$. The probability that each bit in the stream is one is $P(X = 1) = x$. For example, the value $x = 0.3$ could 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 $-1 \le x \le 1$. However, the probability that each bit in the stream is one is $P(X = 1) = \frac{x+1}{2}$. 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. (Unless stated otherwise, our examples will use the unipolar format.)

The synthesis strategy is to cast logical computations as arithmetic operations in the probabilistic domain and implement these directly as stochastic operations on datapaths. Two simple arithmetic operations – multiplication and scaled addition – are illustrated in Figure 1.

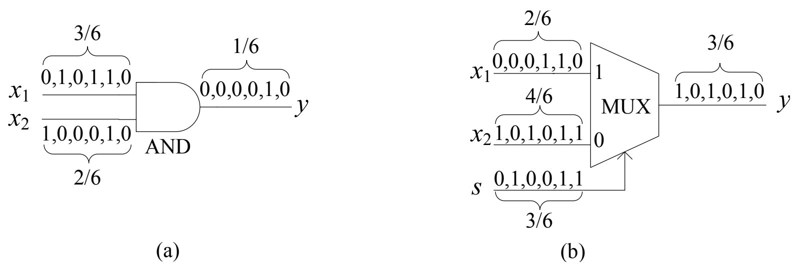

Fig. 1. Stochastic implementation of arithmetic operations: (a) Multiplication; (b) Scaled addition.

- **Multiplication**. Consider a two-input AND gate, shown in Figure 1(a). Suppose that its inputs are two independent bit streams X_1 and X_2. Its output is a bit stream Y, where

$$\begin{aligned} y &= P(Y = 1) = P(X_1 = 1 \text{ and } X_2 = 1) \\ &= P(X_1 = 1)P(X_2 = 1) = x_1 x_2. \end{aligned}$$

  Thus, the AND gate computes the product of the two input probability values.

- **Scaled Addition**. Consider a two-input multiplexer, shown in Figure 1(b). Suppose that its inputs are two independent stochastic bit streams X_1 and X_2 and its selecting input is a stochastic bit stream S. Its output is a bit stream Y, where

$$\begin{aligned} y &= P(Y = 1) \\ &= P(S = 1)P(X_1 = 1) + P(S = 0)P(X_2 = 1) \\ &= sx_1 + (1 - s)x_2. \end{aligned}$$

  (Note that throughout the paper, multiplication and addition represent *arithmetic* operations, not Boolean AND and OR.) Thus, the multiplexer computes the scaled addition of the two input probability values.

In the decades since Gaines' original work, there have been numerous papers discussing the paradigm. Most notable has been the work by Brown and Card [2]. They demonstrated efficient constructs for a wide variety of basic functions, including multiplication, squaring, addition, subtraction, and division. Further, they provided elegant constructs for complex functions such as tanh, linear gain, and exponentiation.[3] They

[3] Such functions were of interest to the artificial neural networks community. The tanh function, in particular, performs a non-linear, sigmoidal mapping; this is used to model activation function of a neuron.

used combinational logic to implement simple functions such as multiplication and scaled addition; the used sequential logic in the form of linear finite-state machines (FSMs) to implement complex functions such as tanh.

More recently, Qian et al. presented a general synthesis method for logical computation on stochastic bit streams [3][4][5]. They showed that combinational logic can be synthesized to implement arbitrary polynomial functions, provided that such polynomials map the unit interval onto the unit interval. Their method is based on novel mathematics for manipulating polynomials in a form called Bernstein polynomials. In [4] Qian et al. showed how to convert a general power-form polynomial into a Bernstein polynomial with coefficients in the unit interval. In [3] they showed how to realize such a polynomial with a form of "generalized multiplexing." In [5], they demonstrated a reconfigurable architecture for computation on stochastic bit streams. They analyzed cost as well as the sources of error: approximation, quantization, and random fluctuations; also they studied the effectiveness of the architecture on a collection of benchmarks for image processing. Li and Lilja demonstrated a stochastic implementation of a kernel density estimation-based image segmentation algorithm [6].

After an introduction to the concepts and a review of implementations of functions such as tanh and exponentiation, this paper presents case studies of useful circuit constructs implemented with the paradigm of logical computation on stochastic bit streams. Specifically, it describe finite state machine implementations of functions for image processing such as edge detection and median filter-based noise reduction.

1.1 Stochastic Exponentiation Function

When operating on stochastic bit streams, combinational logic can only implement polynomial functions of a specific form – namely those that map the unit interval to the unit interval [4]. Non-polynomial functions can be approximated by combinational logic, for instance with MacLaurin expansions [3]. However, highly non-linear functions such as exponentiation and tanh cannot be approximated effectively with this approach. This limitation stems from the fact combinational logic can only implement scaled addition in the stochastic paradigm. The implementation of polynomials with coefficients not in the unit interval is sometimes not possible and is generally not straightforward [5].

Gaines [1] described the use of an ADaptive DIgital Element (ADDIE) for generation of arbitrary functions. The ADDIE is based on a saturating counter, that is, a counter which will not increment beyond its maximum state value or decrement below its minimum state value. In the ADDIE, the state of the counter is controlled in a closed loop fashion. The problem is that ADDIE requires that the output of the counter to be converted into a stochastic bit stream in order to implement the closed loop feedback [1]. This is potentially inefficient and hardware intensive.

In 2001, Brown and Card [2] presented the stochastic exponentiation (SExp) function, with the state transition diagram shown in Figure 2. This configuration approximates an exponentiation function stochastically as follows,

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

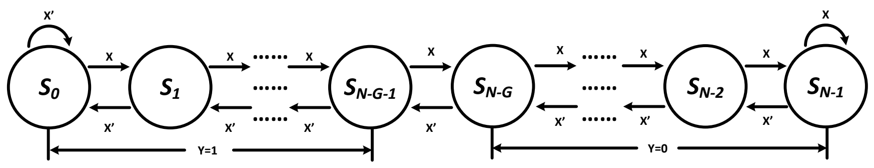

Fig. 2. State transition diagram of the FSM-based stochastic exponentiation function.

where x is the bipolar encoding of the input bit stream X and y is the unipolar encoding of the output bit stream Y.

The FSM shown in Figure 2 is similar to Gaines' ADDIE. The difference is that this linear FSM does not use a closed loop [1][2]; accordingly this construct is much more efficient.

1.2 Scaled Subtraction

The scaled subtraction can be implemented with a MUX and a NOT gate, as shown in Fig. 3.

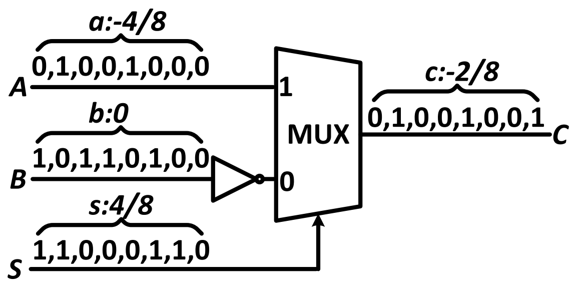

Fig. 3. Scaled subtraction with the bipolar coding. Here the inputs are $a = -4/8$ and $b = 0$. The scaling factor is $s = 4/8$. The output is $4/8 \times (-4/8) + (1 - 4/8) \times 0 = -2/8$, as expected.

The scaled subtraction only works for bipolar coding, since 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, i.e.,

$$\begin{aligned}
a &= 2P(A = 1) - 1, \\
b &= 2P(B = 1) - 1, \\
s &= P(S = 1), \\
c &= 2P(C = 1) - 1.
\end{aligned}$$

Based on the logic function of the circuit, we have

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

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 of the two input values a and b, with a scaling factor of s for a and $1-s$ for b.

1.3 Stochastic Tanh Function

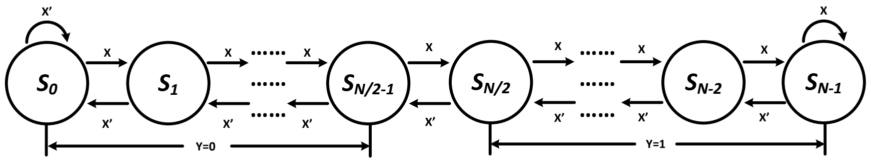

Fig. 4. State transition diagram of the FSM implementing the stochastic tanh function.

The stochastic tanh function is also developed by Brown and Card [2]. The state transition diagram of the FSM implementing this function is shown in Fig. 4. 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,

$$y = \frac{e^{\frac{N}{2}x} - e^{-\frac{N}{2}x}}{e^{\frac{N}{2}x} + e^{-\frac{N}{2}x}}. \tag{2}$$

The corresponding proof can be found in [7]. In addition, Li and Lilja [7] proposed to use this function to implement a stochastic comparator. Indeed, the stochastic tanh function approximates a threshold function as follows if N approaches infinity,

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

The stochastic comparator is built based on the stochastic tanh function and the scaled subtraction as shown in Fig. 5. $P_S = 0.5$ in the selection bit S of the MUX stands for 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. 5 is:

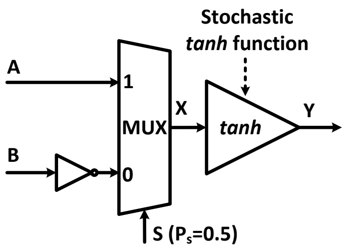

Fig. 5. The stochastic comparator.

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

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

1.4 Stochastic Absolute Value Function

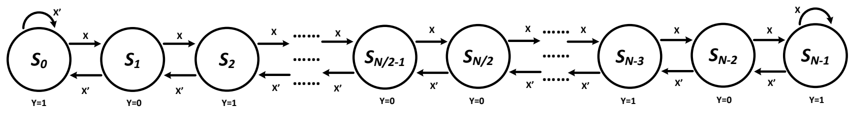

Fig. 6. State transition diagram of the FSM implementing the stochastic absolute value function.

Li and Lilja [7] also developed a stochastic absolute value function. The state transition diagram is shown in Fig. 6. The output Y of this state machine is only determined by the current state S_i $(0 \leq i \leq N-1)$. If $0 \leq i < N/2$ and i is even, or $N/2 \leq i \leq N-1$ and i is odd, $Y = 1$; else $Y = 0$. The approximate function is,

$$y = |x|, \tag{3}$$

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 [7].

2 Case Studies

In this section, we demonstrate circuit constructs for common image processing tasks as case studies illustrating our method: image edge detection and median filter-based noise reduction [8].

2.1 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 [8]. There are an extremely 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. 7 as an example [8].

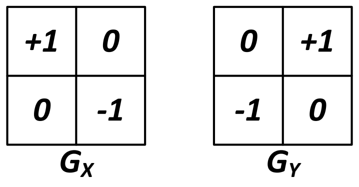

Fig. 7. Robert's cross operator for edge detection.

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.

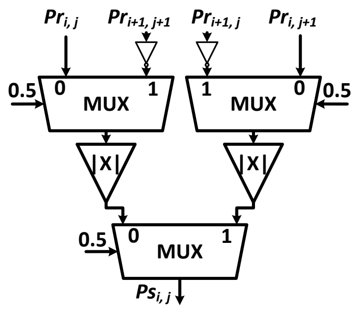

Fig. 8. The stochastic implementation of the Robert's cross operator based edge detection.

The stochastic implementation of this algorithm is shown in Fig. 8. $P_{r_{i,j}}$ 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 $s_{i,j}$.

2.2 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 the similar size [8]. A hardware implementation of a 3×3 median filter based on a sorting network is shown in Fig. 9. Its basic unit is used to sort two inputs in ascending order. It can be implemented by a comparator in a conventional deterministic implementation.

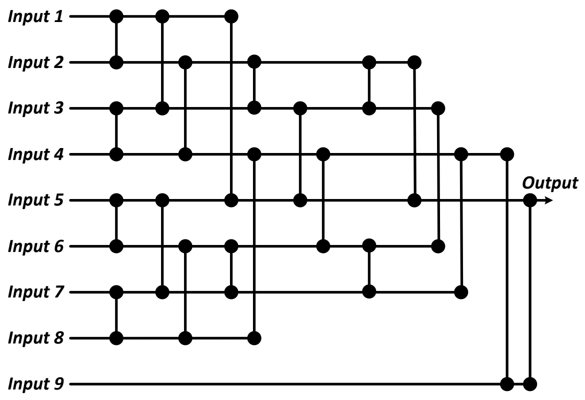

Fig. 9. Hardware implementation of the 3×3 median filter based on a sorting network.

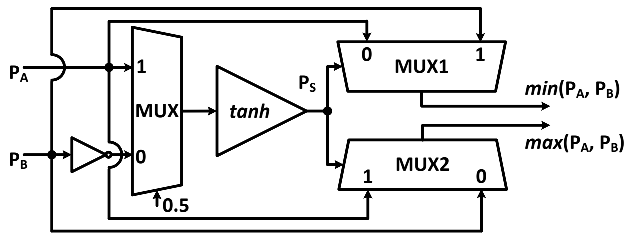

Fig. 10. The stochastic implementation of the basic sorting unit.

The stochastic implementation of this basic unit can be implemented by the stochastic comparator introduced in Section 1.3 with a few modifications. The circuit shown in Fig. 10 has the following functions:

- 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. 9 stochastically.

3 Discussions and Conclusions

The stochastic paradigm offers a novel view of digital computation: instead of transforming definite inputs into definite outputs, circuits transform probability values into probability values; so, conceptually, real-valued probabilities are both the inputs and the outputs. The computation has a pseudo *analog* character, reminiscent of computations performed by physical systems such as electronics on continuously variable signals such as voltage. Here the variable signal is the probability of obtaining a one versus a zero in a stochastic yet digital bit stream. The circuits can be built from ordinary digital electronics such as CMOS. And yet they computed complex, continuous-valued transfer functions. Prior work has shown constructs for a variety of interesting functions. Most intriguing among these are the complex functions produced by linear finite-state machines: exponentiation, tanh, and absolute value.

Because a stochastic representation is uniform, with all bits weighted equally, it is highly tolerant of soft errors (i.e., bit flips). Computation on stochastic bit streams offers tunable precision: as the length of the stochastic bit stream increases, the precision of the value represented by it also increases. Thus, without hardware redesign, one has the flexibility to trade off precision and computation time. In contrast, with a conventional binary-radix implementation, when a higher precision is required, the underlying hardware must be redesigned.

A significant drawback of the paradigm is the long latency of the computations. The accuracy depends on the length of the bit streams; with long bit streams, each operation requires many clock cycles to complete. However, potentially the operations could be performed at a much faster clock rate, mitigating the latency issue.

The accuracy of the computation also depends on the quality of the randomness. If the stochastic bit streams are not statistically independent, the accuracy will drop. Furthermore, *correlation* is an issue in any circuit that has feedback or reconvergent paths. If the circuit has multiple outputs, these will have correlated probability values. In future work, we will study how to design circuits with multiple outputs – and so correlations in space. Also, we will study the impact of feedback – and so correlations in time.

Also, in future work we will study the *dynamic* behavior of stochastic constructs. We have observed that, using bit streams of length L to represent the inputs values, the output values of FSM-based stochastic constructs are always correct and stable after L clock cycles, no matter what the initial state. We will justify this claim mathematically. Finally, we will study a variety of FSM topologies, including 2D and 3D meshes, tori, and circulant graphs.

Acknowledgment

This work was supported in part by an NSF CAREER Award, No. 0845650.

References

1. Gaines, B.R.: Stochastic computing systems. Advances in Information System Science, Plenum **2**(2) (1969) 37–172
2. Brown, B.D., Card, H.C.: Stochastic neural computation I: Computational elements. IEEE Transactions on Computers **50**(9) (September 2001) 891–905
3. Qian, W., Riedel, M.: The synthesis of robust polynomial arithmetic with stochastic logic. In: 45th ACM/IEEE Design Automation Conference, DAC'08. (2008) 648–653
4. Qian, W., Riedel, M.D., Rosenberg, I.: Uniform approximation and Bernstein polynomials with coefficients in the unit interval. European Journal of Combinatorics **32** (2011) 448–463
5. Qian, W., Li, X., Riedel, M., Bazargan, K., Lilja, D.: An architecture for fault-tolerant computation with stochastic logic. IEEE Transactions on Computers **60**(1) (January 2010) 93–105
6. Li, P., Lilja, D.J.: 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)
7. Li, P., Lilja, D.J.: Using stochastic computing to implement digital image processing algorithms. In: 29th IEEE International Conference on Computer Design, ICCD'11. (2011)
8. Gonzalez, R.C., Woods, R.E.: Digital image processing, 3rd edition. Prentice Hall (2008)