electronics

MDPI

Article

Novel Stochastic Computing for Energy-Efficient Image Processors

Hounghun Joe [1] and Youngmin Kim [2,*]

[1] School of Computer and Information Engineering, Kwangwoon University, Seoul 01897, Korea; johuki34@gmail.com
[2] School of Electronic and Electrical Engineering, Hongik University, Seoul 04066, Korea
* Correspondence: youngmin@hongik.ac.kr; Tel.: +82-2-320-1665

Received: 21 May 2019; Accepted: 21 June 2019; Published: 25 June 2019

Abstract: Stochastic computing, which is based on probability, involves a trade-off between accuracy and power and is a promising solution for energy-efficiency in error-tolerance designs. In this paper, adder and multiplier circuits based on the proposed stochastic computing architecture are studied and analyzed. First, we propose an efficient yet simple stochastic computation technique for multipliers and adders by exchanging the wires used for their operation. The results demonstrate that the proposed design reduces the relative error in computation compared with the conventional designs and has smaller area compared to conventional designs. Then, a new energy-efficient and high-performance stochastic adder with acceptable error metrics is investigated. The proposed multiplier shows better error metrics than other existing stochastic multipliers, and significantly improves area utilization and power consumption compared to the exact binary multiplier. Finally, we apply the proposed stochastic architecture to an edge detection algorithm and achieve a significant reduction in area utilization (64%) and power consumption (96%). It is therefore demonstrated that the proposed stochastic architecture is suitable for energy-efficient hardware designs.

Keywords: approximate computing; stochastic computing; wire exchange; energy-efficiency; edge detection

1. Introduction

Presently, energy efficiency is one of the major design objectives in electronic devices such as wireless or wearable devices because of limited battery life. Energy efficiency can be improved by reducing both the computation time and power consumption. However, reducing power consumption entails compromising on performance (i.e., slowing down the device). In other words, to reduce power consumption, we need to decrease the performance of the system in general. However, in reality, an increasing number of electronics devices for more functionalities require higher performance. Approximate computing, which provides advantages such as small area and low power by compromising on the accuracy of computation, is one of the techniques used to increase energy efficiency. Approximate computing is suitable for applications based on human senses, such as image and audio processing, because humans cannot distinguish an approximated value from the exact value. Approximate computing involves a trade-off between accuracy and power consumption. Hence, despite its error tolerance, error metrics such as error distance and error rate are important. Many studies have been conducted on approximate computing and the trade-off between power and accuracy has been investigated [1,2].

Stochastic computing—which is an approximate computation method based on probabilities—was introduced in the 1960s [3]. Because of its tolerance toward errors and the use of very simple circuits, stochastic computing is particularly used in applications such as image

processing and digital filters [4–16]. Also, it is possible to merge various inputs and reduce the dimension of the stochastic computing [17]. The arithmetic logic units used in stochastic computing consist of random number generators, comparators, basic logic gates (e.g., AND, XOR), and counters. Stochastic computing, in general, has a simple structure but a random number generator (RNG), such as the linear feedback shift register (LFSR), occupies a considerably large area and affects the accuracy of stochastic computing. In Reference [6], the sharing of random number sources by using a circular shifting technique was proposed for improving both accuracy and simplicity. In References [11–19], various random number generator architectures and real world applications based on stochastic computing were introduced and analyzed in terms of accuracy and area.

In this paper, we propose an efficient and simple technique that improves the accuracy of stochastic computing by using a wire exchanging method. Two RNGs, which occupy a large area in the stochastic computing, are required in conventional stochastic computing. However, the proposed method uses the output of one RNG to generate two uncorrelated stochastic numbers for the arithmetic operation. Thus, we can reduce area utilization and improve efficiency by exchanging the wire of the RNG, which is one of the inseparable elements of stochastic computing, for improved accuracy.

2. Stochastic Computing

Stochastic computing (SC) uses unary bitstreams, in which the probability corresponds to a number to be computed. The value of the bitstream, which is referred to as a stochastic number (SN), is encoded using 0s and 1s. A specific integer number X can be expressed as an L-bit stochastic number, S_X by $S_X = \{S_{X1}, S_{X2}, \cdots, S_{XL}\}$, where $S_{Xi} \subset \{0, 1\}$. If the stochastic number S has a probability of 1 at 75% and a probability of 0 at 25%, we define the probability of occurrence of '1' as $P = 0.75$.

There are two methods of stochastic number encoding—unipolar or bipolar encoding. In unipolar encoding, '1' has a weight of +1 and '0' has a weight of 0 and the encoded value is limited to the range [0, 1], which can be used to express a positive value. On the other hand, in bipolar encoding, '1' has a weight of +1 and '0' has a weight of -1 and the encoded value is limited to the range $[-1, +1]$. Thus, both positive and negative values can be encoded by using bipolar encoding [5].

When performing computations with stochastic numbers, simple and basic logic gates can be used for arithmetic calculations. The AND gate performs multiplication in unipolar encoding. Given inputs A and B and output C, which have probabilities of P_A, P_B and P_C, respectively, the multiplication is defined as

$$P_C = P_A \times P_B \tag{1}$$

For instance, when A = 01101010 and B = 10111011, P_A is 0.5 and P_B is 0.75. According to Equation (1), output C, which is $A \bullet B$, becomes 00101010 and P_C is 0.375. A multiplexer performs an important role in stochastic computing with unipolar adders. When adding two unipolar numbers P_A and P_B, the result lies in the range [0, 2], which is unacceptable for unipolar encoding. Therefore, the result needs to be scaled by a factor using an auxiliary selective function. Given inputs A and B, selective input S, and output C, which have a probability of P_A, P_B, P_S and P_C, respectively, the addition is defined as

$$P_C = (1 - P_S)P_A + P_S P_B = P_S(P_A + P_B) \tag{2}$$

For example, when the stochastic numbers are A = 11111011, B = 00100110, and S = 10010101, their probabilities become P_A = 7/8, P_B = 3/8, and P_S = 4/8, respectively. According to Equation (2), P_C is 5/8. Thus, the addition using a multiplexer in unipolar encoding is treated as a scaled adder due to the selectivity of the multiplexer.

A stochastic number generator (SNG) creates a stochastic number by comparing the integer value and the random values that are produced periodically by a random number generator (RNG) which

uses LFSR in general, as shown in Figure 1a,b. The stochastic bit becomes '1' if the integer number is bigger than the random number generated by the RNG and becomes '0' otherwise. Then, the arithmetic operation is performed with these stochastic numbers (SNs). A counter, which counts the number of '1's in the stochastic computing result, is used to convert the stochastic number into the integer value at the end. Counters and RNGs are inherent overheads of stochastic computing and they consume a significant amount of power to perform stochastic computation. However, in real-world applications, this can be compensated by using simple stochastic computing operations.

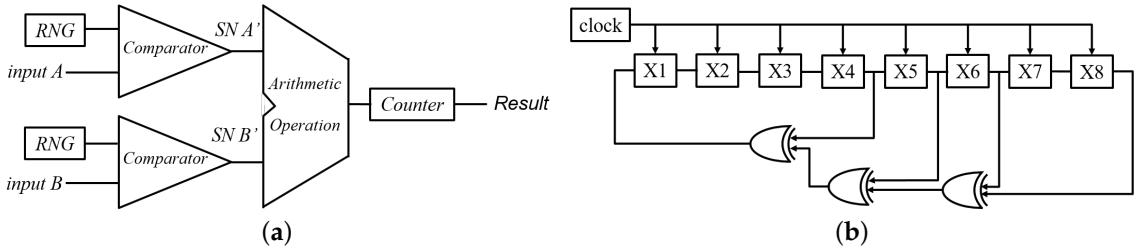

Figure 1. (**a**) Example operation of stochastic computing (SC): random number generator (RNG), comparator, arithmetic operation module, and counter. (**b**) Example of 8-bit linear feedback shift register (LFSR) for conventional random number generator (polynominal = $X^8 + X^6 + X^5 + X^4 + 1$).

Figure 2 shows the various arithmetic operation circuits in stochastic computing: (a) shows an AND gate, which acts as a multiplier; (b) shows an XOR gate, which acts as a subtractor; (c) shows a multiplexer, which acts as a scaled adder; and (d) shows a divider with a multiplexer and a flip-flop.

In stochastic computing, the computation results will be different for different sequences even if they have the same probability values, as shown in Figure 3. It means that a bad-order stochastic number will reduce the accuracy of computation. This is related to a correlation between stochastic numbers in stochastic computing. In stochastic computing, the correlation between two stochastic numbers X and Y is quantified by using the stochastic computing correlation (SCC), which has a value between -1 and +1 and is defined in Reference [16] as

$$SCC(X,Y) = \begin{cases} \dfrac{ad - bc}{N \times \min(a+b, a+c) - (a+b)(a+c)} & ad > bc \\ \dfrac{ad - bc}{(a+b)(a+c) - N \times \max(a-d, 0)} & else \end{cases}$$

In this formula, a is the number of overlaps when both X and Y are '1'. In contrast, d is the number of overlaps when both X and Y are '0'. b is the number of overlaps when X is '0' and Y is '1' and c is the number of overlaps when X is '1' and Y is '0'.

An SCC value of '+1' indicates maximal positive correlation, while '-1' indicates maximal negative correlation. The process of creating the bitstream has an impact on the SCC value and a proper computation module with a correct SCC value should be used in stochastic computing. For example, two stochastic numbers generated by uncorrelated RNGs are uncorrelated. On the other hand, two stochastic numbers produced by the same RNG are positively correlated. As a result, the process of generating the SN affects the SCC value and accuracy. Therefore, a proper arithmetic module with a correct SCC value should be used in SC.

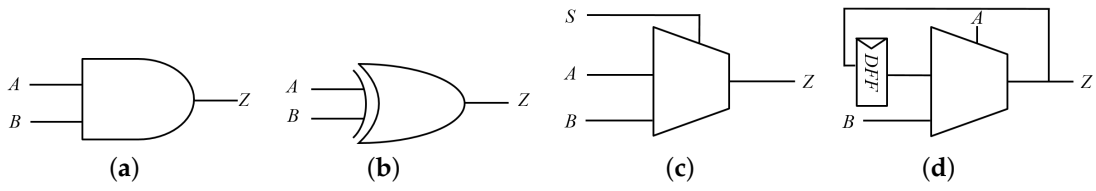

Figure 2. Circuits of each operation in stochastic computing: (**a**) multiplication, (**b**) subtraction, (**c**) addition, and (**d**) division.

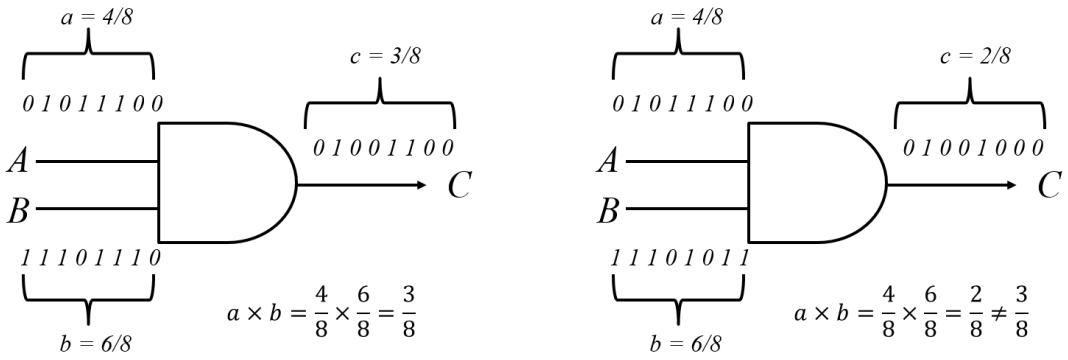

Figure 3. Stochastic multiplication examples using AND gate, which perform $P_A \times P_B = P_C$: (**a**) good result and (**b**) bad result due to the difference in the random number sequence.

3. Stochastic Computing Using a Wire Exchanging Technique

3.1. Basic Structures

In this subsection, we explain the proposed stochastic computing method that uses a wire exchanging technique. A compact and simple LFSR is used in conventional stochastic computing. However, not all LFSR combinations can generate completely uncorrelated stochastic numbers. Therefore, in order to obtain accurate results, different random seeds are required every time for a single LFSR, or separate LFSRs are required for each input number, to generate uncorrelated stochastic numbers. The LFSR is a non-negligible module in stochastic computing and consumes a considerable amount of power since it is relatively large. In addition, if we use dedicated LFSRs for each input as shown in Figure 4a, they consume more power. Therefore, the existing methods involve sharing an LFSR, as shown in Figure 4b, or using an inverter with one LFSR when converting operands with stochastic numbers, as shown in Figure 4c. If the SNG used in the operation uses only one LFSR, the area required for the LFSR can be reduced by half. However, this leads to inaccurate results depending on the SCC that the operation is aiming at.

To solve this problem, we propose sharing an LFSR with a wire exchanging technique to reduce the overhead and to generate uncorrelated random numbers in the SNGs. The exchange function, E, complements the least significant bit of the node in network switching [20], and is defined as:

$$E\left(b_{m-1}b_{m-2}b_{m-3}b_{m-4}\cdots b_3b_2b_1b_0\right) = b_{m-1}b_{m-2}b_{m-3}b_{m-4}\cdots b_3b_2b_1b_0'$$

This converts the bits of the node during network switching. Assuming that the bits of the node is one of the wires from LFSR's output, however, the difference between the exchanged value and the value from the LFSR is small if it is used in the SNG because it only pairs two bits and then changes the order. This is defined as follows:

$$E_{wire}\left(b_{n-1}b_{n-2}b_{n-3}b_{n-4}\cdots b_3b_2b_1b_0\right) = b_{n-2}b_{n-1}b_{n-4}b_{n-3}\cdots b_2b_3b_0b_1$$

To achieve a large difference (i.e., increase randomness and improve the uncorrelation) after the exchange, we use a random number from one LFSR and the pairs of even and odd wires are exchanged symmetrically, as shown Figure 4d. The bit exchange used in the proposed design is defined as follows.

$$E_{proposed}\left(b_{n-1}b_{n-2}b_{n-3}b_{n-4}\cdots b_3b_2b_1b_0\right) = b_1b_0b_3b_2\cdots b_{n-3}b_{n-4}b_{n-1}b_{n-2}$$

where b is the value of each bit of the random number (RN) from the LFSR. Table 1 shows the original 8-bit output from the LFSR and modified bitstreams according to the E_{wire} and $E_{proposed}$ methods.

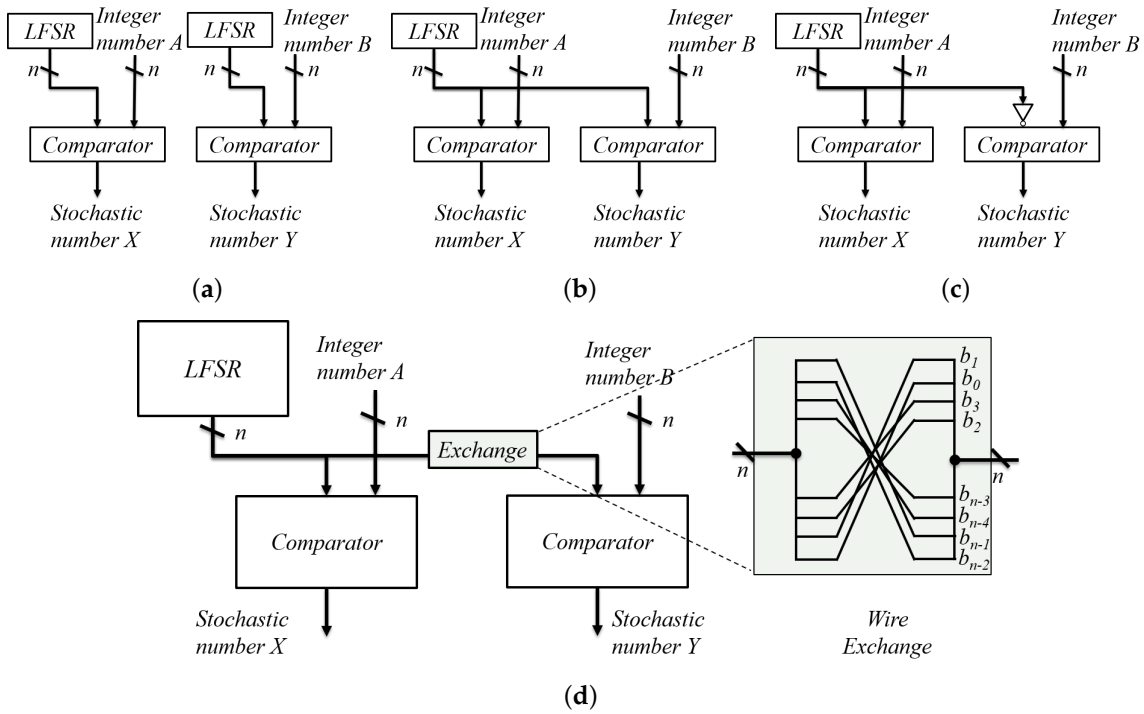

Figure 4. Bitstream generation methods: (**a**) dedicated LFSR, (**b**) sharing, (**c**) sharing with an inverter, and (**d**) proposed sharing with a wire exchanging technique.

Table 1. Bitstream numbering by an 8-bit LFSR for various exchange methods.

	Wire Bit Numbering							
Original	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
E_{wire}	b_6	b_7	b_4	b_5	b_2	b_3	b_0	b_1
$E_{proposed}$	b_1	b_0	b_3	b_2	b_5	b_4	b_7	b_6

3.2. Simulation Results and FPGA Verifications

A logic-level simulation is performed using Quartus 13.1 [21] in order to compare the error characteristics of the various stochastic number generation methods (with multipliers and adders) shown in Figure 4. In order to verify the computation error characteristics, 100,000 randomly generated 8-bit numbers are used. The SCC average is defined as $\frac{\sum_{i=0}^{N} SCC(X_i,Y_i)}{n}$. The absolute relative error (RE) histogram of each adder is shown in Figure 5a, and the result for the multipliers is presented in Figure 5b. The average values of the SCC and the error statistics are summarized in Table 2. The RE is defined as $\frac{|ACC-APP|}{ACC}$, where ACC is the accurate result and APP is the result obtained by the stochastic computation. As shown in Figure 5 and Table 2, the SCC, RE, standard deviation (SD), and coefficient of variance (CV) are reduced significantly by the proposed design ($E_{proposed}$) compared with the conventional method. The SCC average for the inverting method is -0.921, which indicates a highly negative correlation. However, the SCC average for the proposed method is close to zero, which means a lower correlation. In SC adder, the proposed method provides the lower RE than inverting method and the inverting method give the minimum SD number among various SNGs. However, in SC multiplier, the proposed method gives the minimum RE, SD and CV. As shown in Table 2, the SCC is not improved by a simple exchange method (E_{wire}), However, the proposed method ($E_{proposed}$) can achieve an average SCC reduction of up to 6× compared with the sharing method during SN generation. The SD of the proposed adder is reduced by 2× compared with that of the dedicated LFSR method and the multiplier based on the proposed design can reduce the average absolute RE by more than 4× as compared with the multiplier that is driven using the inverter method.

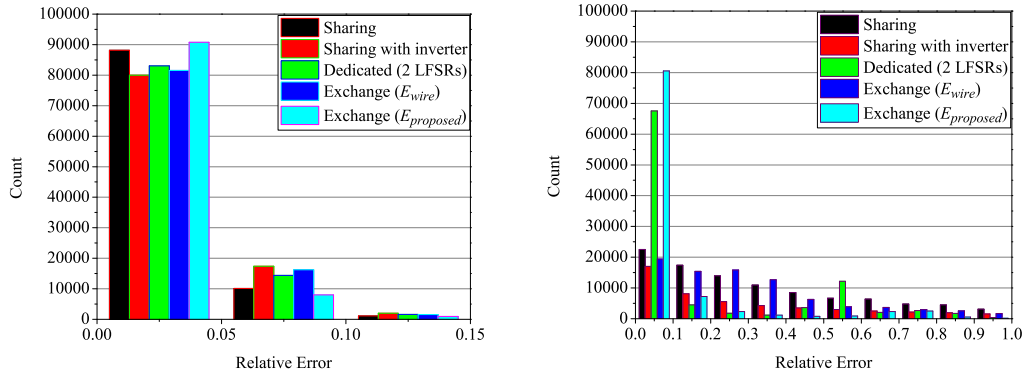

Figure 5. Comparison of absolute relative error distribution: (**a**) stochastic adder, and (**b**) stochastic multiplier with the bitstreams generated in five different methods as shown in Figure 4 and E_{wire}.

For hardware design metric comparison, the operating frequency and the dynamic power of the binary multiplier, Wallace tree multiplier and stochastic multiplier with the proposed method are analysed by using a Cyclone V (5CSEMA5F31C6) FPGA device, included in a DE1-SoC [22]. For an application specific integrated circuit (ASIC), all the designs are synthesized using the 45-nm Nangate library [23] in Design Compiler [24]. These results are summarized in Table 3. As shown, the proposed design is faster and consumes less power than other accurate multipliers. For example, only 1/4 of power is required by the proposed design compared with the Wallace tree multiplier and the proposed stochastic multiplier is 4× faster than the accurate binary multiplier in FPGA. In addition, with ASIC 45-nm simulation, the proposed design enables about 25% reduction in the total area compared to the binary multiplier and requires only 1/11th of the power needed for the Wallace tree multiplier. Also, compared with other stochastic computing methods, the proposed method can save up to 44% of dynamic power with 24% less area than the dedicated method in ASIC.

Table 2. Comparison of the average stochastic computing correlation (SCC AVG), average absolute relative error (ABS RE AVG), standard derivation (SD), and coefficient variability (CV) between various stochastic number generators (SNGs) and the proposed SNG.

		Various SNG				
		Dedicated	**Sharing**	**Inverter**	**Wire (E_{wire})**	**Proposed ($E_{proposed}$)**
SCC AVG		0.042	1	−0.921	0.93	0.164
SC adder	ABS RE AVG	0.030	0.023	0.032	0.032	0.021
	SD	0.053	0.027	0.014	0.065	0.023
	CV	1.740	1.134	3.553	2.026	1.07
SC multiplier	ABS RE AVG	0.187	0.341	0.646	0.514	0.104
	SD	0.274	0.271	0.403	0.595	0.219
	CV	0.682	1.258	1.601	0.864	0.475

Table 3. Comparison of the various design metrics in 45-nm technology and FPGA of the accurate multipliers and the various SC multipliers.

		Accurate			SC with Various SNG					Impr. over Wallace Tree
		Binary	**Wallace Tree**	**Behavior**	**Dedicated**	**Sharing**	**Inverter**	**E_{wire}**	**$E_{proposed}$**	
ASIC 45-nm	Total area	404.65	544.03	434.09	391.99	302.35	303.38	301.55	301.40	45%
	Power (μW)	197.55	329.90	220.98	54.33	31.41	30.75	30.96	30.88	91%
	Delay (ns)	2.15	3.81	1.19	0.42	0.48	0.48	0.48	0.48	88%
FPGA	Frequency	102.51	150.33	310.08	440.6	425.2	427.75	413.87	417.54	2.8×
	Power (mW)	1.10	2.15	2.52	0.67	0.56	0.54	0.57	0.56	74%

4. Stochastic Sobel Edge Detection

4.1. Basic Structures

To verify the energy efficiency of the proposed stochastic computing design methodology in a real-world application, we implemented an approximate (i.e., stochastic) Sobel edge detection by using the proposed stochastic computing structure. In an edge detection algorithm, the first-order derivatives of a digital image were computed by using the horizontal and vertical gradients. The gradient of an image, $\nabla f(x,y)$, in point (x,y) is defined as follows:

$$\nabla f(x,y) = [\partial f/\partial x, \partial f/\partial y] = [Gx, Gy]$$

The magnitude of this vector, which is represented by ∇f, is important in edge detection and is defined as follows:

$$\text{magn}(\nabla f) = \sqrt{Gx^2 + Gy^2} = |Gx| + |Gy|$$

Since data are arranged at regular intervals in an image, the difference between adjacent pixels was calculated without performing a mathematical differential calculation. One method uses a mask in order to perform such a differential calculation on an image. The basic condition for determining a mask is as follows: The mask must have the same width and height and must be odd. In addition, the vertices should be symmetrical with respect to the center point. The value of the center point should always be a positive number or '0'. In addition, the sum of the values at all points should be '0'. Figure 6a shows a mask that meets these conditions. Here, z indicates the number of each pixel about the mask. In this paper, we used two masks (horizontal and vertical) for Sobel edge detection, as shown in Figure 6b. It is efficient since it assigns weights for the center pixel and the horizontal mask (Gx) and vertical mask (Gy) are defined as follows.

$$Gx = \partial f/\partial x = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$
$$Gy = \partial f/\partial y = (z_1 + 2z_2 + z_3) - (z_7 + 2z_8 + z_9)$$

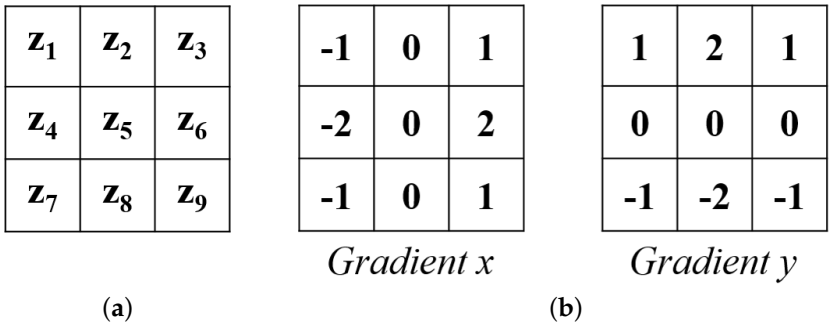

Figure 6. (**a**) 3 by 3 mask example around pixel Z_5, (**b**) Two Sobel masks for horizontal (*Gradient x* (G_x)) and vertical differential (*Gradient y* (G_y)), respectively.

Figure 7 shows the Sobel mask design based on the proposed stochastic computing. Each P is SN for the corresponding pixel value used in the Sobel masks as shown in Figure 6b. P_x is for the G_x and P_y is for the G_y. Two results were passed through the multiplexer by the selection signal, P_S. For mask values of '2' in the z2, z4, z6, and z8 locations, those pixels are AND together with $P = 1/128$, which is the integer '2' in the SN of 8 bits. Since the final mux performs the scaled addition of $P_S = 1/2$, the original integer value can be obtained after 1-bit left shifting. When SNs are generated, various SNGs can be used. The Sobel mask calculation, defined in the above equation, requires a total of 16 operations (e.g., additions and multiplications). Therefore, when creating each SN with a dedicated LFSR method, a total of 16 8-bit LFSRs was used. On the other hand, since the other methods (i.e., sharing the LFSR, sharing with an inverter and the proposed method) require only one LFSR for

16 SNs, one half of the inputs use the original output of the LFSR and the other half use the inverted or wire exchanged values.

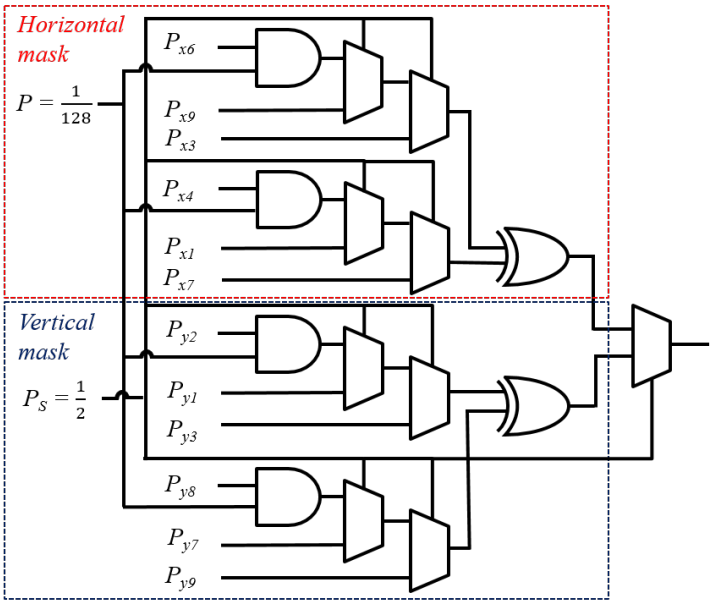

Figure 7. Proposed stochastic Sobel edge detection circuit.

4.2. Simulation Results and FPGA Verifications

The image quality and basic design specifications (i.e., area, delay, power, and energy) were analyzed and compared between a conventional Sobel edge detection method (i.e., using accurate adders and multipliers or conventional stochastic computation) and the proposed stochastic edge detection method. The Sobel edge detection was designed with Verilog and analyzed by using a Cyclone V (5CSEMA4F31C6) FPGA device, included in a DE-1-SoC board from Terasic. In addition, the designs were synthesized using a 45-nm Nangate open cell library in Design Compiler for performance measurement in ASIC. Figure 8 shows the original image (e.g., airplane) of 512 × 512 pixels (a) and the results obtained using various edge detection methods, namely the method using accurate adders and multipliers (b), the previous methods using stochastic multiplier and adder (c, d, and e), and the proposed method (f). As shown, the image obtained using the proposed stochastic computing technique is better than the other stochastic methods and is similar to the image obtained by the accurate edge detection. Table 4 summarizes the design characteristics and noise metrics after synthesis of the Sobel edge detection technique using an accurate arithmetic operation and various stochastic computing methods including the proposed design. As shown, up to 64% of total area and 96% of power were saved in ASIC compared with the accurate edge detection. When compared with the dedicated LFSR based stochastic edge detection, the proposed method can achieve up to 55% reduction in the area, 76% reduction in dynamic power in the 45-nm-Nangate implementation. Also, it can reduce 15% delay compared with the sharing methods. In addition, compared with the accurate design, the proposed design required 50% less logic utilization, consumed up to 75% less power, and was 2.4× faster in FGPA implementation. Compared with the dedicated LFSR based stochastic computing, up to 58% of the power and 67% of the register can be reduced by the proposed design. Compared with the accurate edge detection, the root-mean-square error (RMSE) and the relative power signal noise ratio (PSNR) of the proposed design were 39.48 and 15.13 on average for the five sample images, respectively. The percentage difference between the images of the proposed stochastic Sobel edge detection with respect to the image after accurate Sobel edge detection was 11.45% on average, which is the lowest among stochastic computing methods. Also, the proposed design can reduce RMSE

by 26%, enhance PSNR by 21%, and decrease the percentage difference between the images by 30% compared with the sharing method.

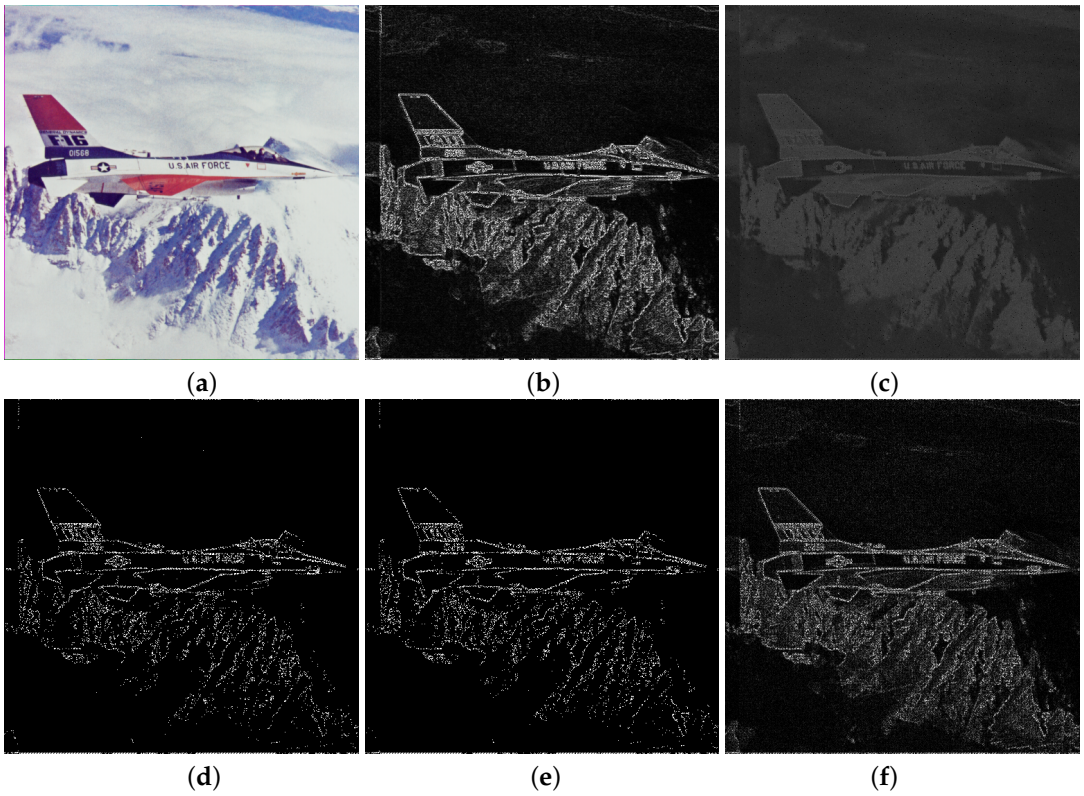

Figure 8. Original image and images obtained using various edge detection techniques: (**a**) original image, (**b**) accurate Sobel edge detection, (**c**) stochastic Sobel edge detection using multiple LFSRs, (**d**) stochastic Sobel edge detection by sharing the LFSR, (**e**) stochastic Sobel edge detection using inverter LFSR, and (**f**) Sobel edge detection using the proposed design.

Table 4. Comparison of the various design and error metrics in 45-nm technology and FPGA of the accurate Sobel edge detection and the stochastic Sobel edge detection with five benchmark images.

		Accurate	Stochastic			
			Dedicated	Sharing	Inverter	Proposed
ASIC 45 nm	Total area (μm^2)	896.61	717.41	319.72	319.53	322.98
	Power (mW)	0.70	0.11	0.03	0.03	0.03
	Data arrival time	0.20	0.46	0.70	0.70	0.60
FPGA	Logic utilization (in ALMs)(/32070)	60	48	31	31	31
	Total register	24	63	21	21	21
	Dynamic power (mW)	3.18	1.86	1.04	1.23	1.29
	Frequency (MHz)	136.35	321.96	347.95	312.40	329.75
RMSE	airplane	-	35.43	47.17	47.18	34.58
	lenna	-	38.04	54.45	54.75	41.66
	pepper	-	35.68	52.37	52.24	38.32
	sailboat	-	47.72	69.80	69.81	50.84
	tiffany	-	38.09	42.84	42.68	32.01
	average	-	38.99	53.33	53.33	39.48
PSNR	airplane	-	15.93	13.45	13.44	16.19
	lenna	-	15.28	12.20	12.16	14.53
	pepper	-	15.87	12.56	12.58	15.29
	sailboat	-	13.32	10.04	10.04	12.79
	tiffany	-	15.29	14.32	14.33	16.85
	average	-	15.14	12.51	12.51	15.13
Percentage difference between images (%)	airplane	-	12.63	13.01	13.01	8.91
	lenna	-	13.68	17.37	17.42	12.86
	pepper	-	11.15	16.52	16.49	11.27
	sailboat	-	15.15	21.92	21.91	14.94
	tiffany	-	14.58	13.33	13.31	9.29
	average	-	13.44	16.43	16.43	11.45

5. Conclusions

In this study, we proposed a novel stochastic computing structure using an wire exchange method and we investigated the design characteristics, such as power, delay, area and various error metrics. The proposed stochastic multiplier can achieve more than 25% reduction in area, 4× increased speed and 85% improvement in power consumption compared to the conventional binary multiplier. The absolute relative error of the proposed stochastic multiplier is only half that of the conventional sharing method. An image processing application was designed using the proposed stochastic computing technique in order to prove the proposed approximate design. The proposed stochastic Sobel edge detection provided significant advantages in terms of area and power compared to the accurate method. Therefore, the proposed stochastic design can be applicable to energy-efficient hardware designs for embedded systems or image processing applications, in which some well-controlled errors are acceptable.

Author Contributions: Conceptualization, H.J. and Y.K.; methodology, H.J.; software, H.J.; validation, H.J. and Y.K.; investigation, H.J.; resources, H.J. and Y.K.; writing–original draft preparation, H.J.; writing–review and editing, Y.K.; supervision, Y.K.; project administration, Y.K.; funding acquisition, Y.K.

Funding: This research was funded by the Ministry of Education grant number NRF-2017R1D1A1B03028065.

Acknowledgments: This research was supported by the Basic Science Research Program, through the National Research Foundation of Korea (NRF), funded by the Ministry of Education (NRF-2017R1D1A1B03028065).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Han, J.; Orshansky, M. Approximate computing: An emerging paradigm for energy-efficient design. In Proceedings of the 2013 18th IEEE European Test Symposium (ETS), Avignon, France, 27–30 May 2013; pp. 1–6.
2. Agrawal, A.; Choi, J.; Gopalakrishnan, K.; Gupta, S.; Nair, R.; Oh, J.; Prener, D.A.; Shukla, S.; Srinivasan, V.; Sura, Z. Approximate computing: Challenges and opportunities. In Proceedings of the IEEE International Conference on Rebooting Computing (ICRC), San Diego, CA, USA, 17–19 October 2016; pp. 1–8.
3. Gaines, B.R. Stochastic computing systems. In *Advances in Information Systems Science*; Springer: Boston, MA, USA, 1969; Volume 2, pp. 37–172.
4. Alaghi, A.; Hayes, J.P. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst. TECS* **2012**, *12*, 92. [CrossRef]
5. Parhi, K.K. Analysis of stochastic logic circuits in unipolar, bipolar and hybrid formats. In Proceedings of the International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA, 28–31 May 2017; pp. 1–4.
6. Ichihara, H.; Ishii, S.; Sunamori, D.; Iwagaki, T.; Inoue, T. Compact and accurate stochastic circuits with shared random number sources. In Proceedings of the International Conference on Computer Design (ICCD), Seoul, Korea, 19–22 October 2014; pp. 361–366.
7. Ranjbar, M.; Salehi, M.E.; Najafi, M.H. Using stochastic architectures for edge detection algorithms. In Proceedings of the Iranian Conference on Electrical Engineering, Tehran, Iran, 10–14 July 2015; pp. 723–728.
8. Alaghi, A.; Li, C.; Hayes, J.P. Stochastic circuits for real-time image-processing applications. In Proceedings of the Design Automation Conference, Austin, TX, USA, 29 May–7 July 2013; pp. 1–6.
9. Qian, W.; Li, X.; Riedel, M.D.; Bazargan, K.; Lilja, D.J. An architecture for fault-tolerant computation with stochastic logic. *IEEE Trans. Comput.* **2011**, *60*, 93–105. [CrossRef]
10. Seva, R.; Metku, P.; Choi, M. Energy-efficient FPGA-based parallel quasi-stochastic computing. *J. Low Power Electron. Appl.* **2017**, *7*, 29 [CrossRef]
11. Alaghi, A.; Hayes, J.P. Fast and accurate computation using stochastic circuits. In Proceedings of the Conference on Design, Automation & Test in Europe, Dresden, Germany, 24–28 March 2014; p. 76.
12. Liu, S.; Han, J. Energy efficient stochastic computing with Sobol sequences. In Proceedings of the Conference on Design, Automation & Test in Europe, Lausanne, Switzerland, 27–31 March 2017; pp. 650–653.

13. Vahapoglu, E.; Altun, M. Accurate synthesis of arithmetic operations with stochastic logic. In Proceedings of the 2016 IEEE Computer Society Annual Symposium VLSI(ISVLSI), Pittsburgh, PA, USA, 11–13 July 2016; pp. 415–420.
14. Yuan, B.; Zhang, C.; Wang, Z. Design space exploration for hardware-efficient stochastic computing: A case study on discrete cosine transformation. In Proceedings of the 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Shanghai, China, 20–25 March 2016; pp. 6555–6559.
15. Lee, V.T.; Alaghi, A.; Ceze, L. Correlation manipulating circuits for stochastic computing. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Dresden, Germany, 19–23 March 2018; pp. 1417–1422.
16. Alaghi, A.; Hayes, J.P. Exploiting correlation in stochastic circuit design. In Proceedings of the 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, USA, 6–9 October 2013; pp. 39–46.
17. Alaghi, A.; Hayes, J.P. Dimension reduction in statistical simulation of digital circuits. In Proceedings of the Symposium on Theory of Modeling & Simulation (TMS DEVS), Alexandria, VA, USA, 12–15 April 2015; pp. 1–8.
18. Alaghi, A.; Qian, W.; Hayes, J.P. The Promise and Challenge of Stochastic Computing. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 1515–1531. [CrossRef]
19. Li, P.; Lilja, D.J.; Qian, W.; Bazargan, K.; Riedel, M.D. Computation on Stochastic Bit Streams Digital Image Processing Case Studies. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **2014**, *22*, 449–462. [CrossRef]
20. Lang, T.; Stone, H.S. A shuffle-exchange network with simplified control. *IEEE Trans. Comput.* **1976**, *100*, 55–65. [CrossRef]
21. Quartus 13.1 User Manual. Available online: https://www.intel.com/ (accessed on 21 August 2018).
22. DE1-SoC User Manual 1.2.2. Available online: http://www.terasic.com/ (accessed on 5 July 2017).
23. Nangate: 45 nm Open Cell Library. Available online: https://www.silvaco.com/products/nangate/FreePDK45_Open_Cell_Library (accessed on 4 January 2019).
24. Design Compiler ver.M-2016.12-SP5-5. Available online: https://www.synopsys.com/ (accessed on 16 May 2018).