Highly Accurate and Energy Efficient Binary-Stochastic Multipliers for Fault-Tolerant Applications

Yongqiang Zhang, *Member, IEEE*, Lingyun Xie, Jie Han, *Senior Member, IEEE*, Xin Cheng, and Guangjun Xie

***Abstract*—Stochastic circuits use randomly distributed bitstreams to represent numbers, so leading to small areas and low power dissipation. However, it does not only result in a long latency and thus increases energy, but also reduces computing accuracy. In this brief, a design of parallel stochastic multipliers with high accuracy and low energy is proposed. To this end, an algorithm for finding optimal multiplicative bitstreams (OpMulbs) is developed for multipliers. Experimental results show that the proposed parallel multipliers using OpMulbs are the most accurate among currently available stochastic multipliers. They also require much less energy with a smaller latency, compared to the others. With a mean squared error of 4.02×10^{-6}, the proposed 8-bit multiplier shows a 42.18%, 48.15%, 20.35%, and 55.56% reduction in area, power, latency, and energy, respectively, compared to an 8-bit exact binary multiplier. The applications in multiply-accumulate units and image processing algorithms show that the proposed multipliers outperform the state-of-the-art stochastic designs in several considered criteria and binary designs in hardware cost.**

***Index Terms*—Stochastic computing, multiplier, bitstream, computing accuracy, hardware cost.**

I. Introduction

STOCHASTIC computing (SC) generally converts n-bit numbers within [0, 2^n-1] into the interval [0, 1] by using randomly distributed bitstreams. The main advantage of carrying out operations through bitstreams is that the area and power dissipation of arithmetic circuits can be significantly reduced; e.g., an AND gate can perform multiplication in SC. Thus, it is suitable for applications containing many multipliers, such as multiply-accumulate (MAC) units [1], filters [2], neural networks (NNs) [3, 4], and some image processing algorithms [1]. It has been shown that a binary-interfaced MAC (BIN-MAC) is more scalable and reconfigurable than a conventional MAC using exclusively stochastic logic (ESL-MAC) [5, 6].

The latency of 2^n clock cycles for the serial multiplier using an AND gate exponentially grows as the bit width n of the binary inputs increases, which leads to a noticeable decrease in energy efficiency. To reduce the latency, a multiplexer (MUX) based multiplier can terminate the operation in advance by detecting each bit in the multiplicator bitstream [5]. At the same time, the multiplicand bitstream is reduced by 1 every clock cycle. If it reaches 0, the multiplier finishes the operations. It shows a high computing accuracy and no longer requires 2^n clock cycles. Assuming that the multiplicand follows a uniform distribution in the interval [0, 2^n-1], the expectation of the required number of clock cycles is $(1+1+2+3\ldots+2^n\text{-}1)/2^n=2^{n\text{-}1}\text{-}1/2$, where each element in the parenthesis gives the number of clock cycles for computing each number. This shows that the multiplier approximately reduces the number of clock cycles by half on average, compared to the serial one. Thus, the energy efficiency of this method can still be improved.

To further reduce the latency, the parallel thermometer code has been introduced for multipliers to reduce the latency from 2^n clock cycles to 1 clock cycle [7, 8]. To increase computing accuracy, a deterministic approach changes the length of bitstreams from 2^n to 2^{2n} bits for a completely accurate computation [9]. This leads to an excessive number of parallel AND gates in multipliers and results in very high energy consumption. Moreover, almost half of the elements are 1 in the truth table of the thermometer code, leading to a high design complexity of stochastic number generators (SNGs) [7]. This also reduces the multipliers' energy efficiency.

To address the above issues, a method to find the bitstreams with the least mean squared error (MSE), referred to as optimal multiplicative bitstreams (OpMulbs), is proposed and instantiated in detail in this work for highly accurate and energy efficient multipliers. Computing accuracy is evaluated using the MSE and mean absolute error (MAE). Simulation results show that the multipliers using OpMulbs produce the same MSEs and MAEs as the state-of-the-art MUX-based designs in [5] while reaching much lower hardware costs. Several applications are then implemented to validate the advantages of the proposed multipliers in lowering hardware costs and improving computing accuracy.

This work was supported by the Fundamental Research Funds for the Central Universities (No. JZ2020HGQA0162 and No. PA2021KCPY0043), by the Natural Sciences and Engineering Research Council (NSERC) of Canada (No. RES0048688), and by Natural Science Foundation of Anhui Province (No. 2108085MF226).

(*Corresponding author: Guangjun Xie*)

Y. Zhang, L. Xie, C. Xin, and G. Xie are with the School of Microelectronics, Hefei University of Technology, Hefei 230009, China (e-mail: ahzhangyq@hfut.edu.cn; 2098425338@qq.com; xcheng@hfut.edu.cn; gjxie8005@hfut.edu.cn)

J. Han is with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada (e-mail: jhan8@ualberta.ca)

Algorithm 1 Algorithm for finding optimal multiplicative bitstreams generated by an n-bit binary number

Input: n (bit width of a binary number)
Output: A (matrix to store all permutations without duplications)
Initialization: $N = C_2^1 \times C_4^2 \times \cdots \times C_{2^n}^{2^{n-1}}$, A=zeros(N,2^n)

```
for i=1 to n do
    A(1,2^(i-1)+1:2^i)=i;
for p=2 to N do
    t=A(p-1,:);
    for p1=2 to 2^n do
        if t(p1-1)<t(p1) then break;
    for p2=1 to 2^n do
        if t(p2)<t(p1) then break;
    exchange(t(p1),t(p2));
    for k=1 to p1/2 do
        exchange(t(k),t(p1-k));
    A(p,:)=t;
```

The main contributions of this brief are summarized as follows. 1) An algorithm for finding the bitstreams with the smallest MSE and MAE is developed. 2) Parallel stochastic multipliers with various bit widths are proposed to reach a higher accuracy and lower hardware costs. 3) The multipliers are applied to several algorithms to verify their practicability in fault-tolerant applications.

This brief proceeds as follows. Section II describes the algorithm and the proposed parallel multipliers. Section III illustrates the experimental results and applications of the multipliers. Section IV concludes this work.

II. Bitstreams and Multipliers

A. The Optimal Multiplicative Bitstream (OpMulb)

Let us take a 3-bit binary number $b_2b_1b_0$ (ranging in 000 ~ 111) to explain the algorithm for finding OpMulbs. The weights of b_2, b_1, and b_0 are respectively 4, 2, and 1. The number of 1s in a stochastic bitstream with a length of 8 bits generated by $b_2b_1b_0$ is approximately $4b_2+2b_1+b_0$. Assume that an initial bitstream is $b_2b_2b_2b_2b_1b_1b_00$. Then, a full permutation is performed on the bitstream.

For ease of description, the initial bitstream is denoted as an initial vector [3 3 3 3 2 2 1 0], where each element maps a bit in $b_2b_2b_2b_2b_1b_1b_00$. Then, the full permutation of the initial bitstream becomes that of the initial vector. It is worth noting that it will be possible to obtain the reverse lexicographic order [10], in total $(2^3)!$=40320 cases, using the *perms* function in MATLAB. The number of cases exponentially increases as the bit width of binary numbers increases. Meanwhile, the computation time must also be considered. Note that, elements 2 and 3 in this vector are duplications, thus leading to a large number of duplications in all 40320 cases. After removing these duplications, the number of cases is $C_8^4 \times C_4^2 \times C_2^1$=840. Thus, an algorithm is required to directly obtain all cases without duplications, at least the best permutation with the least MSE has to be found.

Algorithm 1 describes the principle of finding OpMulbs for an n-bit binary number. Fig. 1(a) illustrates the permutation process of Algorithm 1 for an initial vector. The initial vector can be arbitrarily chosen, e.g., [3 3 3 3 2 2 1 0], while the starting vector is fixed to be in an ascending order, e.g., A_1=[0 1 2 2 3 3 3 3]. Consider the process from the previous vector

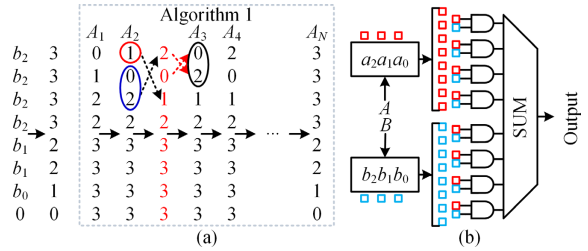

Fig. 1. (a) An example of algorithm 1. (b) A 3-bit parallel multiplier.

A_2=[1 0 2 2 3 3 3 3] to the next vector A_3=[0 2 1 2 3 3 3 3] as an example; it is divided into four steps.

1) Find the position index p_1 of the larger number in the first ascending pair of numbers in A_2. That is, (0 2) enclosed in a blue circle as the first ascending pair, and $A_2(p_1=3)=2>0$.

2) Find the position index p_2 of the first number smaller than the number indexed by p_1 in A_2. That is, 1 in the red circle less than 2 found in the first step, and $A_2(p_2=1)=1<2$.

3) Swap $A_2(p_1=3)=2$ and $A_2(p_2=1)=1$ to obtain an intermediate vector [2 0 1 2 3 3 3 3], as the dotted arrows show.

4) Reverse the order of numbers with indexes smaller than p_1=3 in the intermediate vector, as the red arrows indicate. That is, change (2 0) to (0 2) in the black circle.

The permutation process is finished to get the targeted vector A_3=[0 2 1 2 3 3 3 3]. All others are performed in the same way to reach the last permutation $A_N=A_{840}$=[3 3 3 3 2 2 1 0].

Not only do we record the results of each permutation but also perform a multiplication of each permutation by the initial bitstream, and then compute their MSEs. The bitstreams with the smallest MSEs are saved to be OpMulbs, which will be used for the stochastic multipliers later.

B. The Proposed Parallel Multiplier

With the found OpMulbs, parallel stochastic multipliers are proposed to reduce latency from 2^n clock cycles to 1 clock cycle. Take a 3-bit multiplier with the binary inputs $A=a_2a_1a_0$ and $B=b_2b_1b_0$ as an example, as shown in Fig. 1(b). The multiplier uses hard-wired connections to generate parallel bitstreams, followed by $2^n=2^3$=8 AND gates and a SUM unit to obtain 3-bit binary results in one clock cycle.

The pairing of two hard-wired connections to each AND gate is dominated by OpMulbs. An input of the multiplier can be coded from an initial bitstream, while the other one is coded from an OpMulb. For signed multiplication, it is only necessary to add a sign bit to each binary input and to use two's complement to operate.

The multipliers in [8] share binary input and output interfaces and AND gates to implement parallel operations. The use of hard-wired connections in the proposed multipliers exempts the generation of stochastic bitstreams from dedicated SNGs, i.e., thermometer code generators. These simple connections benefit from OpMulbs, which makes the multipliers more hardware efficient. However, the SNGs and the deterministic approach in the multipliers in [8] lead to a significant increase in bitstream length and thus reduce energy efficiency if the multipliers perform exact computation, which is optional in many fault-tolerant applications, such as NNs, filters, and image processing. We show in the following that OpMulbs provide sufficient computing accuracy for the proposed multipliers.

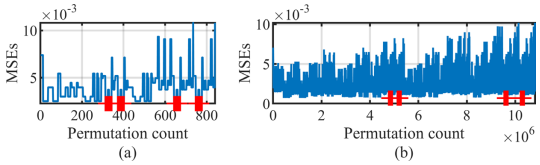

Fig. 2. The MSEs of each permutation for n-bit numbers. (a) 3-bit. (b) 4-bit.

TABLE I
THE MSES OF 3-BIT TO 8-BIT MULTIPLIERS

n-bit / MSE	3 (10^{-3})	4 (10^{-4})	5 (10^{-4})	6 (10^{-5})	7 (10^{-5})	8 (10^{-6})
MUX [5]	2.26	6.75	1.93	5.38	1.48	4.02
Thermo [8]	11.5	112.2	111.4	1111.8	1111.3	11111.5
The_det [8]	0	0	0	0	0	0
Shared [11]	2.37	9.20	3.37	11.78	3.97	12.94
Binary	0	0	0	0	0	0
Proposed	2.26	6.75	1.93	5.38	1.48	4.02

TABLE II
THE MAES OF 3-BIT TO 8-BIT MULTIPLIERS

n-bit / MAE	3 (10^{-2})	4 (10^{-2})	5 (10^{-2})	6 (10^{-3})	7 (10^{-3})	8 (10^{-3})
MUX [5]	3.52	2.02	1.10	5.89	3.10	1.62
Thermo [8]	8.20	8.30	8.33	83.31	83.33	83.33
The_det [8]	0	0	0	0	0	0
Shared [11]	3.59	2.23	1.44	8.82	5.26	3.07
Binary	0	0	0	0	0	0
Proposed	3.52	2.02	1.10	5.89	3.10	1.62

III. EXPERIMENTAL RESULTS AND APPLICATIONS

The MSE for n-bit multipliers is computed by

$$\mathrm{MSE}=\sum_{x=0}^{2^n-1}\sum_{y=0}^{2^n-1}\frac{\left(P_{xy}-P_x\times P_y\right)^2}{2^n\times 2^n}, \quad (1)$$

where P_x and P_y respectively represent the binary values of a permutation and the initial bitstream, and P_{xy} denotes their ANDed result. The mean absolute error (MAE) is also employed to evaluate the computing accuracy. The MAE is obtained by one million Monte Carlo trials for each input within the range of [0, 1] randomly generated by using the *rand* function in MATLAB. All circuits are synthesized by the Synopsys Design Compiler with a TSMC 65 nm library at the typical corner and with a nominal supply voltage of 1.2V. The command 'compile_ultra' is used to ungroup all components and automatically synthesize the circuits according to timing constraints. The power is measured by PrimePower using a vector-free power analysis model.

A. Multiplier

Computing accuracy: Fig. 2 shows the MSEs of all possible permutations without duplication for 3-bit and 4-bit binary inputs. The values marked in red are the MSEs generated by using OpMulbs, which have the smallest values. TABLE I and TABLE II list the smallest MSEs and MAEs of 3-bit to 8-bit multipliers designed using the obtained OpMulbs and previous designs in [5, 8, 11]. 'Shared' means the outputs of a linear feedback shift register are reversed and shared between two SNGs to realize high computing accuracy [11]. The proposed multipliers provide the same MSEs and MAEs as the MUX-based designs in [5], while the sharing-based multipliers reach higher values of these metrics. The designs in [8] using the deterministic approach (denoted by the_det) produce the same results as binary multipliers. If the deterministic approach is not applied in the designs in [8] (thermo for short), the largest MSEs and MAEs occur, since all 1s in the thermometer code proceed to 0s and the 0s involved in multiplication do not affect the number of 1s in results.

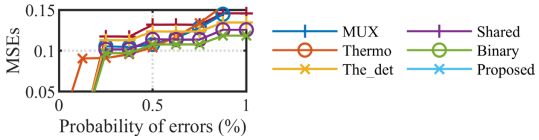

Fig. 3. The MSEs of 3-bit multipliers versus the probability of errors.

TABLE III
THE HARDWARE MEASUREMENTS OF 3-BIT MULTIPLIERS

Multiplier	Area (μm^2)	Power (mW)	CPD (ns)	Latency (ns)	Energy (pJ)	EE
MUX [5]	134.3	0.012	0.46	1.61	0.019	6.5
Thermo [8]	85.7	0.005	0.71	0.71	0.004	1.3
The_det [8]	516.6	0.022	1.56	1.56	0.035	11.8
Shared [11]	112.7	0.012	0.11	0.88	0.010	3.5
Binary	100.4	0.009	0.50	0.50	0.005	1.5
Proposed	74.2	0.005	0.57	0.57	0.003	1.0

TABLE IV
THE HARDWARE MEASUREMENTS OF 8-BIT MULTIPLIERS

Multiplier	Area (μm^2)	Power (mW)	CPD (ns)	Latency (ns)	Energy (pJ)	EE
MUX [5]	379.1	0.033	0.71	90.52	2.95	76.5
Thermo [8]	3543.5	0.113	6.25	6.25	0.70	18.3
The_det [8]	694092.9	23.659	8.29	8.29	196.13	5082.1
Shared [11]	274.7	0.021	0.17	43.52	0.93	24.1
Binary	701.6	0.054	1.72	1.72	0.09	2.4
Proposed	405.7	0.028	1.37	1.37	0.04	1.0

Fault tolerance: The aforementioned method [12] to inject various probabilities of errors into the inputs of multipliers is applied. Each bit in the inputs is randomly selected and flipped with a given probability within [0, 1]. Fig. 3 shows the MSEs of 3-bit multipliers versus the probability of bit flip-induced errors. Apparently, the MSEs increase as the errors increase. The proposed 3-bit multiplier presents lower MSEs than other designs, so it has a higher fault tolerance than others. The same observation is true for higher-bit multipliers.

Hardware cost: TABLE III shows the hardware measurements of 3-bit multipliers, including an exact 3-bit binary multiplier. 3×3=9 AND gates are used to generate the partial products, which are then compressed by using the 4-2 compressor in [13], half adders, and full adders into two rows, based on the Dadda tree structure. A ripple carry adder is then used to compute the final addition.

The MUX-based design terminates operations in advance to reduce latency according to the inputs. Its latency is computed by the critical path delay (CPD) multiplied by the expected number of required clock cycles, i.e., $0.46\times(2^{n-1}-1/2)=0.46\times(2^{3-1}-1/2)$. In the same way, the sharing-based multiplier is also a serial design, of which the latency is computed by the CPD multiplied by the number of clock cycles, i.e., $0.11\times 2^n=0.11\times 2^3$.

The latencies of the thermo, the_det, binary, and proposed multipliers are respectively equal to their CPD since they are designed to operate in parallel. Energy efficiency (EE) is denoted as the ratio of the energy of previous multipliers to that of the proposed design in TABLE III. With these data, the proposed multiplier is more energy efficient than its stochastic counterparts and the binary design. This benefits from the parallel design and the hard-wired connections dominated by OpMulbs to avoid complex SNGs.

TABLE V
THE HARDWARE MEASUREMENTS OF M-INPUT MACS USING 6-BIT MULTIPLIERS

	m	2	4	8	**9**	16	32	64	128	**256**	Average	Normalization	Gate
MUX [5]	Area ($\mu m^2 10^4$)	0.07	0.13	0.27	**0.30**	0.54	1.07	2.14	4.27	**8.53**	0.0333	1.11	232
	Power (mW)	0.06	0.11	0.22	**0.24**	0.43	0.85	1.70	3.40	**6.83**	0.0267	1.31	
	Latency (ns)	22.68	21.42	22.68	**23.31**	25.83	30.87	64.89	75.60	**87.26**	0.7216	30.58	
	Energy (pJ)	1.36	2.36	5.00	**5.61**	11.14	26.31	110.59	257.69	**597.49**	1.9554	43.01	
Thermo [8]	Area ($\mu m^2 10^4$)	0.19	0.38	0.75	**0.85**	1.50	2.99	5.98	11.96	**23.92**	0.0935	3.13	649
	Power (mW)	0.07	0.14	0.27	**0.30**	0.53	1.04	2.07	4.16	**8.46**	0.0328	1.61	
	Latency (ns)	1.95	1.94	1.89	**2.07**	2.05	1.99	2.00	2.10	**2.09**	0.0348	1.47	
	Energy (pJ)	0.14	0.27	0.50	**0.62**	1.09	2.06	4.14	8.75	**17.67**	0.0679	1.50	
The_det [8]	Area ($\mu m^2 10^4$)	8.12	16.23	32.45	**36.50**	64.90	129.85	259.69	519.54	**1039.70**	4.0597	135.58	28192
	Power (mW)	4.20	8.18	16.54	**18.54**	33.06	66.20	132.75	266.21	**534.63**	2.0815	102.11	
	Latency (ns)	5.78	5.86	5.67	**5.47**	5.68	5.95	6.01	6.04	**6.00**	0.1011	4.28	
	Energy (pJ)	24.28	47.93	93.78	**101.42**	187.78	393.89	797.83	1607.91	**3207.78**	12.4520	274.30	
Shared [11]	Area ($\mu m^2 10^4$)	0.07	0.14	0.29	**0.32**	0.58	1.15	2.29	4.56	**9.25**	0.0359	1.20	250
	Power (mW)	0.05	0.09	0.18	**0.20**	0.35	0.68	1.35	2.85	**5.98**	0.0226	1.11	
	Latency (ns)	14.08	14.72	69.76	**81.28**	90.24	108.80	132.48	154.88	**183.68**	1.6376	69.38	
	Energy (pJ)	0.70	1.32	12.56	**16.26**	31.58	73.98	178.85	441.41	**1098.41**	3.5743	78.74	
Binary	Area ($\mu m^2 10^4$)	0.10	0.20	0.40	**0.45**	0.80	1.62	3.24	6.44	**12.87**	0.0503	1.68	350
	Power (mW)	0.08	0.15	0.29	**0.32**	0.57	1.08	2.24	4.74	**9.90**	0.0373	1.83	
	Latency (ns)	1.28	1.28	1.28	**1.29**	1.28	1.42	2.68	3.11	**3.47**	0.0329	1.40	
	Energy (pJ)	0.10	0.19	0.37	**0.41**	0.73	1.53	6.00	14.74	**34.35**	0.1126	2.48	
Proposed	Area ($\mu m^2 10^4$)	0.06	0.12	0.24	**0.27**	0.48	0.96	1.92	3.83	**7.66**	0.0299	1.00	208
	Power (mW)	0.04	0.08	0.16	**0.18**	0.32	0.62	1.26	2.58	**5.34**	0.0204	1.00	
	Latency (ns)	0.97	1.02	1.01	**1.00**	1.01	1.00	1.13	2.37	**2.74**	0.0236	1.00	
	Energy (pJ)	0.04	0.08	0.16	**0.18**	0.32	0.62	1.42	6.11	**14.63**	0.0454	1.00	

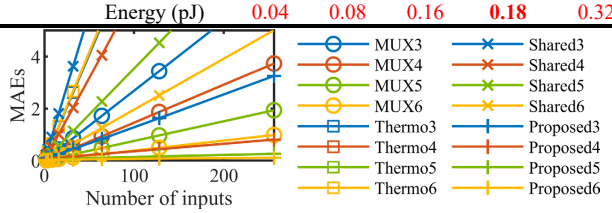

Fig. 4. The MAEs for MACs using different stochastic multipliers.

In addition, the hardware costs for 8-bit multipliers are listed in TABLE IV, which indicate the energy efficiency of the proposed design of stochastic multipliers.

B. Multiply-Accumulate Unit

The proposed 6-bit multiplier shows an MSE of 5.38×10^{-5}, which is low enough for many fault-tolerant applications [11]. So 3-bit to 6-bit multipliers are investigated in various applications to verify the efficacy of the proposed multipliers.

An *m*-input MAC using the proposed multipliers requires only cascading *m* multipliers in parallel. All products are summed by using a binary adder, so the results do not need to be scaled.

Computing accuracy: Fig. 4 shows the MAEs of MACs using different stochastic multipliers versus the number of inputs *m* ($2\le m\le 2^8$), where the terms 'MUX3', 'Thermo3', 'Shared3', and 'Proposed3' means a MAC composed of respective 3-bit stochastic multipliers.

The proposed 6-bit multiplier-based MACs show almost the same MAEs as those denoted as MUX6, whereas Thermo6 and Shared6 present higher MAEs. The MACs using the proposed lower-bit multipliers show lower MAEs than other designs, because of the utilization of OpMulbs. The data also show that the MAEs linearly increase as the number of the inputs of MACs increases. The increasing rates of the designed MACs are smaller than others.

Hardware cost: TABLE V lists the hardware measurements of *m*-input MACs using various 6-bit multipliers ($2\le m\le 2^8$), where the 256-input and 9-input MACs are respectively used for image multiplication and image smoothing later. The energy of the proposed multiplier-based MACs is respectively reduced by 97.68%, 33.14%, 99.64%, 98.73%, and 59.68% on average compared with the MUX, thermo, the_det, sharing, and binary designs, as can be seen from the average and normalized measures. The 'Gate' means the number of equivalent NAND gates for the average area, which also shows the efficiency of the proposed designs.

C. Image Processing

Five images including the airplane, cameraman, clock, Lena, and moon from the USC-SIPI Image Database are multiplied in pairs and smoothened to assess the practicability of multipliers [14]. The peak signal to noise ratio (PSNR) and mean structural similarity index (MSSIM) are used to evaluate the processed image quality. The the_det and exact binary designs provide accurate results for computing the PSNR and MSSIM. The PSNR is given by

$$PSNR = 10\log_{10}\left(w\times r\times MAX^2 \Big/ \sum_{i=0}^{w-1}\sum_{j=0}^{r-1}[S'(i,j)-S(i,j)]^2 \right), \quad (2)$$

where *w* and *r* are the length and width of images; *MAX* is the maximum value of pixels; and $S'(i,j)$ and $S(i,j)$ are the exact and stochastic results for each pixel. The MSSIM is defined as

$$MSSIM(X,Y) = \frac{1}{k}\sum_{i=1}^{k}\frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \quad (3)$$

where the descriptions for these parameters can be found in [15].

Multiplication: An algorithm for multiplying two images pixel by pixel by using the stochastic and binary multipliers is developed in MATLAB.

Smoothing: The smoothing algorithm is given by [16]

$$Y(i,j) = \sum_{m=-1}^{1}\sum_{n=-1}^{1} X(i+m, j+n)K(m+2, n+2), \quad (4)$$

where *X* and *Y* are respectively an input image to be smoothened and an output image, *K* is the smoothing kernel given by

Fig. 5. The processed images. (a) Multiplication. (b) Smoothing.

TABLE VI
THE AVERAGE PSNRS OF IMAGES USING VARIOUS MULTIPLIERS (DB)

Multiplier	3-bit	4-bit	5-bit	6-bit
MUX [5]	23.34/14.69	29.56/24.01	35.31/28.83	41.17/36.48
Thermo [8]	17.66/9.06	17.97/9.04	18.04/9.06	18.05/9.08
Shared [11]	21.25/9.71	25.92/17.29	31.27/23.70	36.81/29.83
Proposed	24.77/15.29	29.68/24.03	35.74/28.83	41.21/37.24

TABLE VII
THE AVERAGE MSSIMS OF IMAGES USING VARIOUS MULTIPLIERS

Multiplier	3-bit	4-bit	5-bit	6-bit
MUX [5]	0.687/0.532	0.816/0.819	0.930/0.909	0.979/0.959
Thermo [8]	0.683/0.642	0.789/0.671	0.838/0.682	0.857/0.686
Shared [11]	0.692/0.442	0.809/0.735	0.928/0.872	0.976/0.944
Proposed	0.725/0.697	0.861/0.846	0.944/0.909	0.982/0.962

$$K = \frac{1}{8}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \quad (5)$$

Computing accuracy: TABLE VI and TABLE VII list the average PSNRs and MSSIMs for the processed images, where the data before/after the slashes respectively are the values for the image multiplication and smoothing. The PSNRs of images multiplied using the proposed multipliers are respectively improved by 1.56%, 83.21%, and 14.00% on average, while the MSSIMs are improved by 5.60%, 10.89%, and 5.84% compared with those in [5, 8, 11]. The PSNRs and MSSIMs of the smoothened images are also improved by 76.03% and 18.48% on average compared with the others. These results also show that the proposed 6-bit multiplier is accurate enough in image processing because a PSNR larger than 30 dB is considered sufficient [17]. Fig. 5 shows examples of the multiplied airplane and moon images and the smoothened cameraman images, generated by the considered 6-bit multipliers.

Hardware cost: The circuits for multiplying and smoothing images are similar to those of 256-input and 9-input MACs, respectively, as listed in TABLE V. Thus, the proposed multipliers achieve a high computing accuracy, while significantly lowering the hardware costs in fault tolerant applications, compared with the state-of-the-art designs.

IV. CONCLUSION

An algorithm for finding OpMulbs is proposed for stochastic multipliers. Experimental results show that the proposed parallel stochastic multipliers using OpMulbs reach the same computing accuracy while incurring much smaller hardware costs, compared with previous designs. Applications in MAC units and image processing also illustrate the effectiveness of the proposed multipliers. These designs use binary input and output interfaces, so they can be easily applied to neural networks by using the designed MACs. This will be investigated in our future work to accelerate neural networks.

REFERENCES

[1] P. Schober, M. Najafi, and N. Taherinejad, "High-accuracy multiply-accumulate (MAC) technique for unary stochastic computing," *IEEE Trans. Comput.,* vol. 71, no. 6, pp. 1425-1439, Jun. 2021.

[2] H. Ichihara, T. Sugino, S. Ishii, T. Iwagaki, and T. Inoue, "Compact and accurate digital filters based on stochastic computing," *IEEE Trans. Emerging Top. Comput.,* vol. 7, no. 1, pp. 31-43, Mar. 2019.

[3] S. Hyeonuk, and L. Jongeun, "A new stochastic computing multiplier with application to deep convolutional neural networks," in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 2017, pp. 29-34.

[4] Y. Hu, Y. Zhang, R. Wang, Z. Zhang, J. Song, X. Tang, W. Qian, Y. Wang, Y. Wang, and R. Huang, "A 28-nm 198.9-TOPS/W fault-tolerant stochastic computing neural network processor," *IEEE Solid-State Circuits Lett.,* vol. 5, pp. 198-201, Aug. 2022.

[5] H. Sim, and J. Lee, "Cost-effective stochastic MAC circuits for deep neural networks," *Neural Netw.,* vol. 117, pp. 152-162, Sep. 2019.

[6] H. Abdellatef, M. Khalil-Hani, N. Shaikh-Husin, and S. Ayat, "Low-area and accurate inner product and digital filters based on stochastic computing," *Signal Process.,* vol. 183, Jun. 2021.

[7] Y. Zhang, R. Wang, X. Zhang, Z. Zhang, J. Song, Z. Zhang, Y. Wang, and R. Huang, "A parallel bitstream generator for stochastic computing," in 2019 Silicon Nanoelectronics Workshop, Kyoto, Japan, 2019, pp. 1-2.

[8] Y. Zhang, R. Wang, X. Zhang, Y. Wang, and R. Huang, "Parallel hybrid stochastic-binary-based neural network accelerators," *IEEE Trans. Circuits Syst., II, Exp. Briefs,* vol. 67, no. 12, pp. 3387-3391, Dec. 2020.

[9] H. Najafi, and D. Lilja, "High quality down-sampling for deterministic approaches to stochastic computing," *IEEE Trans. Emerging Top. Comput.,* vol. 9, no. 1, pp. 7-14, Mar. 2018.

[10] Saul I. Gass, and M. C. Fu, *Encyclopedia of operations research and management science*, Springer New York, NY: Springer New York, NY, 2013.

[11] S. Salehi, "Low-cost stochastic number generators for stochastic computing," *IEEE Trans. Very Large Scale Integr. VLSI Syst.,* vol. 28, no. 4, pp. 992-1001, Apr. 2020.

[12] W. Qian, X. Li, M. Riedel, K. Bazargan, and D. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Comput.,* vol. 60, no. 1, pp. 93-105, Jan. 2011.

[13] C. Chang, J. Gu, and M. Zhang, "Ultra low-voltage low-power CMOS 4-2 and 5-2 compressors for fast arithmetic circuits," *IEEE Trans. Circuits Syst. I-Regul. Pap.,* vol. 51, no. 10, pp. 1985-1997, Oct. 2004.

[14] "The usc-sipi image database," https://sipi.usc.edu/database/.

[15] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.,* vol. 13, no. 4, pp. 600-12, Apr. 2004.

[16] C. Solomon, and T. Breckon, *Fundamentals of digital image processing: A practical approach with examples in matlab*, The Atrium, Southern Gate, Chichester: John Wiley & Sons Ltd, 2011.

[17] M. Ansari, H. Jiang, B. Cockburn, and J. Han, "Low-power approximate multipliers using encoded partial products and approximate compressors," *IEEE J. Emerging Sel. Top. Circuits Syst.,* vol. 8, no. 3, pp. 404-416, Sep. 2018.