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 lower power dissipation. However, it does not 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 more accurate as well as reducing the number of multipliers in multipliers to a greater extent. Thus, the energy efficiency of this method can still be improved.

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 [5]. 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 multiplier 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 \((1+1+2+3+\ldots+2^n-1)/2^n=2^{n-1}-1/2\), 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 \( 2^m \) 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. J2020HGQA0162 and No. PA2021KCPY0043), by the Natural Sciences and Engineering Research Council (NSERC) of Canada (No. RES004868), 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:** \( N = C_0^0 \times C_1^0 \times \cdots \times C_{n-1}^{n-1} \), \( A = \text{zeros}(N,2^n) \)

for \( i = 1 \) to \( n \) do  
\( A(1,2^{i-1}+1:2^i) = i \);

for \( p = 2 \) to \( N \) do  
\( i = \pi(p-1) \)  
for \( p_i = 2 \) to \( 2^n \) do  
if \( t(p_{i-1}) < t(p_i) \) then break;  
for \( p_1 = 1 \) to \( 2^n \) do  
if \( t(p_1) < t(p_i) \) then break;  
exchange \((p_1), (p_i))\);  
for \( k = 1 \) to \( p_1 / 2 \) do  
exchange \((k), (p_1-k))\);  
\( A(p, \pi(p)) = i \);

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 \( b_2b_2b_1b_1b_1b_0 \). Then, a full permutation is performed on the bitstream.

For ease of description, the initial bitstream is denoted as an initial vector \([3 3 3 2 2 1 0]\), where each element maps a bit in \( b_2b_2b_1b_1b_1b_1b_0 \). 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 \((2^n)! = 40320\) cases, using the \textit{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 \( C_3^1 \times C_4^1 \times C_7^1 = 840 \). 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., \([3 3 3 2 2 1 0]\), while the starting vector is fixed to be in an ascending order, e.g., \( A = [0 1 2 2 3 3 3] \). Consider the process from the previous vector

\( A_1 = [1 0 2 2 3 3 3 3] \) to the next vector \( A_2 = [0 2 1 2 3 3 3 3] \) 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) 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 \( A_2(p_2) = 1 \).

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

4) Reverse the order of numbers with indexes smaller than \( p_1 \) 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 \( A_8 = [3 3 3 2 2 1 0] \).

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 \( A = a_0a_1a_2 \) 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.
The deterministic approach (denoted by the \text{det}) produce the same higher values of these metrics. The designs in \cite{8} using the sharing-based designs in \cite{5}, while the sharing-based multipliers reach the same MSEs and MAEs as the MUX-based multipliers. The values marked in red are the MSEs generated by using OpMulbs, which have the smallest values.

\begin{table}[h]
\centering
\caption{The MSEs of 3-bit to 8-bit multipliers}
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
\textbf{n-bit} & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
\textbf{MSE} & \( (10^2) \) & \( (10^3) \) & \( (10^4) \) & \( (10^5) \) & \( (10^6) \) & \( (10^7) \) & \( (10^8) \) \\
\hline
\text{MUX} \cite{5} & 2.26 & 6.75 & 1.93 & 5.38 & 1.48 & 4.02 \\
\text{Thermo} \cite{8} & 11.5 & 11.22 & 111.4 & 1111.8 & 11113.3 & 111111.5 \\
\text{The}_{\text{det}} \cite{8} & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Shared} \cite{11} & 0.37 & 2.90 & 3.37 & 11.78 & 3.97 & 12.94 \\
\text{Binary} & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Proposed} & 2.26 & 6.75 & 1.93 & 5.38 & 1.48 & 4.02 \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\caption{The MAEs of 3-bit to 8-bit multipliers}
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
\textbf{n-bit} & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
\textbf{MAE} & \( (10^2) \) & \( (10^3) \) & \( (10^4) \) & \( (10^5) \) & \( (10^6) \) & \( (10^7) \) \\
\hline
\text{MUX} \cite{5} & 3.52 & 2.02 & 1.10 & 5.89 & 3.10 & 1.62 \\
\text{Thermo} \cite{8} & 8.20 & 8.30 & 8.33 & 83.31 & 83.33 & 83.33 \\
\text{The}_{\text{det}} \cite{8} & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Shared} \cite{11} & 3.59 & 2.23 & 1.44 & 8.82 & 5.26 & 3.07 \\
\text{Binary} & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Proposed} & 3.52 & 2.02 & 1.10 & 5.89 & 3.10 & 1.62 \\
\hline
\end{tabular}
\end{table}

III. EXPERIMENTAL RESULTS AND APPLICATIONS

The MSE for \( n \)-bit multipliers is computed by
\begin{equation}
\text{MSE} = \sum_{x=0}^{2^n-1} \sum_{y=0}^{2^n-1} \left( P_{xy} - P_x \right)^2 2^{-2^n},
\end{equation}
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 \text{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 \text{‘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

\text{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 \cite{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 \cite{11}. The proposed multipliers provide the same MSEs and MAEs as the MUX-based designs in \cite{5}, while the sharing-based multipliers reach higher values of these metrics. The designs in \cite{8} using the deterministic approach (denoted by the \text{det}) produce the same results as binary multipliers. If the deterministic approach is not applied in the designs in \cite{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.

\text{Fault tolerance:} The aforementioned method \cite{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.

\text{Hardware cost:} TABLE III shows the hardware measurements of 3-bit multipliers, including an exact 3-bit binary multiplier. \(3 \times 3=9\) AND gates are used to generate the partial products, which are then compressed by using the 4-2 compressor in \cite{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., \(0.46 \times (2^{n-1} - 1/2)\). 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 \text{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.
Lower-bit multipliers show lower MAEs than other designs, whereas Thermo6 and only cascading applications to verify the efficacy of the proposed multipliers.

An 6-bit multiplier shows an MSE of $5.38 \times 10^{-5}$, where the 256-input and 9-input MACs are respectively used for image multiplication and image smoothing later. The energy cost, the_thet and exact binary designs provide accurate results for computing the PSNR and MSSIM. The PSNR is given by

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

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

$$MSSIM(X, Y) = \frac{1}{k} \sum_{i=0}^{k} \left( \mu_i \mu_i + C_1 \right) \left( \sigma_i \sigma_i + C_2 \right) \left( \rho_{i, i} \sigma_i \sigma_i + C_3 \right),$$

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

### B. Multiply-Accumulate Unit

The proposed 6-bit multiplier shows an MSE of $5.38 \times 10^{-5}$, 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 $m$ ($2 \leq m \leq 2^8$), 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 ($2 \leq m \leq 2^8$), where the 256-input and 9-input MACs are respectively used for image multiplication and image smoothing later. The energy cost, the_thet and exact binary designs, as can be seen from the average and normalized measurements. 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 smoothed 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_thet and exact binary designs provide accurate results for computing the PSNR and MSSIM. The PSNR is given by

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

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

$$MSSIM(X, Y) = \frac{1}{k} \sum_{i=0}^{k} \left( \mu_i \mu_i + C_1 \right) \left( \sigma_i \sigma_i + C_2 \right) \left( \rho_{i, i} \sigma_i \sigma_i + C_3 \right),$$

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

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

**Smoothing:** The smoothing algorithm is given by

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

where $X$ and $Y$ are respectively an input image to be smoothened and an output image, $K$ is the smoothing kernel given by [16].
Investigated in our future work to accelerate neural networks.

Images are similar to those of 256-input and 9-input MACs, output interfaces, so they can be easily applied to neural applications, compared with the state-of-the-art designs.

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, respectively.

The circuits for multiplying and smoothing are similar to those of 256-input and 9-input MACs, output interfaces, so they can be easily applied to neural applications, compared with the state-of-the-art designs.

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% 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


