Scaled-CBSC: Scaled Counting-Based Stochastic Computing Multiplication for Improved Accuracy

Shuyuan Yu
University of California, Riverside
Department of Electrical and Computer Engineering
Riverside, California, United States
syu070@ucr.edu

Sheldon X.-D. Tan
University of California, Riverside
Department of Electrical and Computer Engineering
Riverside, California, United States
stan@ece.ucr.edu

ABSTRACT

Stochastic computing (SC) can lead area-efficient implementation of logic designs. Existing SC multiplication, however, suffers a long-standing problem: large multiplication error with small inputs due to its intrinsic nature of bit-stream based computing. In this article, we propose a new scaled counting-based SC multiplication approach, called *Scaled-CBSC*, to mitigate this issue by introducing scaling bits to ensure the bit '1' density of the stochastic number is sufficiently large. The idea is to convert the "small" inputs to "large" inputs, thus improve the accuracy of SC multiplication. But different from an existing stream-bit based approach, the new method uses the binary format and does not require stochastic addition as the SC multiplication always starts with binary numbers. Furthermore, Scaled-CBSC only requires all the numbers to be larger than 0.5 instead of arbitrary defined threshold, which leads to integer numbers only for the scaling term. The experimental results show that the 8-bit Scaled-CBSC multiplication with 3 scaling bits can achieve up to 46.6% and 30.4% improvements in mean error and standard deviation, respectively; reduce the peak relative error from 100% to 1.8%; and improve 12.6%, 51.5%, 57.6%, 58.4% in delay, area, area-delay product, energy consumption, respectively, over the state of art work. Furthermore, we evaluate the proposed multiplication approach in a discrete cosine transformation (DCT) application. The results show that with 3 scaling bits, 8-bit scaled counting-based SC multiplication can improve the image quality with 5.9dB upon the state of art work in average.

ACM Reference Format:
Shuyuan Yu and Sheldon X.-D. Tan. 2022. Scaled-CBSC: Scaled Counting-Based Stochastic Computing Multiplication for Improved Accuracy. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530499

This work is supported in part by NSF grants under No. OISE-1854276, in part by NSF grant under No. CCF-1816361, No. CCF-2007135 and No. CCF-2113928.

1 INTRODUCTION

Approximate computing enables efficient trade-off among accuracy, area, latency and power for more efficient error tolerant applications implementation such as machine learning and multimedia workloads [1]. Those workloads are heavily dominated by the multiplication operations and hence design of hardware-efficient multiplier has been intensively investigated recently. The primary goal of the approximate multiplier design is to trade the accuracy or quality for the power/energy, latency and area.

A number of approximate multiplier designs have been proposed recently [2–10]. Those approximate multipliers employ some adhoc truncation or reduction methods or mathematically formulated approximation schemes. Most of the existing methods, however, lack the systematic configurability for accuracy vs. area/power/latency trade-off.

On the other hand, another viable solution to approximate computing is by means of stochastic computing (SC) in which the multiplication is performed by simple *AND* operation of two random bit streams [11, 12]. SC can provide *inherent progressive* trade-off between accuracy and latency/energy/area by changing the length of bit-streams and it can be extremely low-cost and energy efficient. However, traditional SC hardware implementation suffers from very long latency, strict requirement for randomness of SC numbers, and large area overhead for random number generations. Recently a more efficient and also more accurate SC multiplier was proposed to partially mitigate the two aforementioned problems in traditional SC [13]. First, It replaces the *AND* operation with counting bit '1' in the bit-stream, which can be early terminated without going through the full length of the bit-streams and second, the bit streams no longer need to be random, which significantly reduces SC hardware overhead. However, it still suffers from the nature of sequential counting process. Recently such counting-based SC scheme has been improved by the COSAIM multiplier in which the counting process can be further accelerated by a simple formula [14].

Despite of the improvements for SC computing, one long-standing problem for SC multiplication is that when two numbers are small, the error of the multiplication can be very significant as shown in Fig. 1. As we can see, the relative error in the SC multiplication for 8-bit binary data can be as large as 100% when the inputs are both in the region of [1/256, 15/256] for the range of [1/256, 255/256].

To mitigate this problem, recently [15] proposed a scaled population (SP) arithmetic for SC. The idea is to introduce a scaling term so that one can scale the number within ([0,1]) larger than some given threshold (like 0.7, ex.) to ensure better accuracy for SC multiplication. However, this method still suffers a few drawbacks as

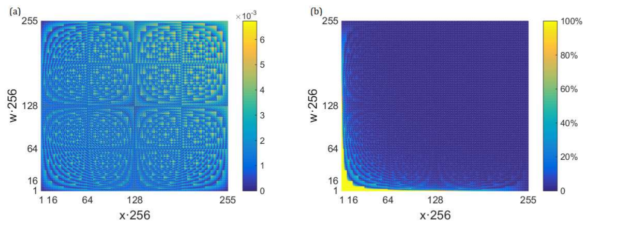

Figure 1: **(a)The absolute error distribution of the counting-based SC method; (b)The relative error distribution of the counting-based SC method.**

it is based on the traditional random-number based SC framework. Also, the SP format is difficult to interface with the commonly used binary number format, which is often required in the image processing and machine learning application. Further, it still operates in the bit-stream level and requires complicated design to reduce random fluctuation errors for random number generation, scaling and random-shuffling operations.

In this work, we try to mitigate the long-standing issues with the stochastic computing by introducing a scaled counting-based stochastic computing scheme, called *Scaled-CBSC*, which can significantly improve the accuracy of SC multiplications for small numbers. The key contributions of this work are listed as follows:

1. In the *Scaled-CBSC* framework, instead of representing the number in a bit-stream as the traditional SC methods do, a scaling factor is introduced so that bit '1' density in the bit-stream (SC part) is always large than 0.5 to ensure sufficient accuracy for SC computing and the scaling factor is guaranteed to be integer. Further more, our new scaled SC format is in the binary form and all the operations like shifting is done in the binary format, which is much more efficient than the bit-stream level operations.
2. Instead of using traditional SC framework, we adopt the counting-based SC (CBSC) scheme [13], in which deterministic bit-stream pattern is used. As a result, all the complicated designs associated with the random number generations, correlation reductions are not needed any more. Further more, no SC addition is required as it will be performed in the binary domain, which is important for introducing the scaling factors as SC addition based on *OR* operations requires low '1' density, which contradicates with high '1' density requirement of SC multiplication.
3. Our numerical results show that the 8-bit Scaled-CBSC multiplication with 3 scaling bits can improve the mean error and standard deviation upon the original CBSC baseline by up to 46.6% and 30.4%, respectively. The 16-bit Scaled-CBSC multiplication with 4 scaling bits can improve the mean error and standard deviation by up to 50.3% and 34.9%. Also with 1, 2, 3 scaling bits, Scaled-CBSC can significantly reduce the peak relative error from 100% to 51.6%, 5.8% and 1.8%, respectively. Furthermore, compared with the state of art work, the 8-bit Scaled-CBSC multiplication with 3 scaling bits improves area, delay, ADP (area-delay product) and energy consumption with 12.6%, 51.5%, 57.6% and 58.4%, respectively.
4. We also evaluate the Scaled-CBSC multiplication in a *discrete cosine transformation* (DCT) application. The results show that for 8-bit precision, 3 scaling bits are required to achieve significant improvement on the image quality (5.9dB in average). For 16-bit precision, 1 scaling bit is enough to achieve 16.8dB (in average) image quality improvement. And with over 2 scaling bits, we can even achieve an output image with no quality loss.

This paper is organized as follows: Section 2 reviews several recently proposed SC multiplication designs. Section 3 presents the proposed *Scaled-CBSC* design including the special data tuple format and the CBSC kernel optimization techniques. Section 4 shows the experimental results for the error metrics, area, delay, power and energy consumption comparison results with the original CBSC baseline and state of art works. Finally, section 5 concludes the paper.

2 REVIEW OF RELATED WORK

Traditionally, SC multiplication consists of 3 parts: (1) stochastic number generators (SNG) which convert the N-bit binary inputs to the 2^N-bit stochastic numbers (SN); (2) SC multiplication core, usually an *AND* or *NOR* gate which is corresponding to unsigned multiplication or signed multiplication; (3) a counter which converts the product SN to binary form back again if needed.

As the computing accuracy of SC multiplication is determined by the SN quality, in other words, the SNG. And the SNG also costs much more area and power than the SC multiplication core. Existing works mainly focused on how to design more area efficient and high quality SNGs, like: Halton sequence generator [16], LFSR (linear feedback shift register) [17], LD (low discrepancy) sequence generation [18].

Among these works, a recent work proposed by Sim [13], named counting-based SC (CBSC) multiplication achieved not only the better accuracy but also the smallest latency by introducing an finite state machine (FSM) based SNG with counting scheme. The CBSC multiplication design is shown in Fig. 2. Different from the traditional SC multiplication, CBSC only requires one FSM-based SNG to convert one of the two binary inputs, ex. x, into a bit-stream with deterministic pattern first. The FSM-based SNG evenly distributes the x_{i-1}, which is the ith bit of x, based on its binary weight 2^{i-1}. For instance, if $i = 3$, then x_2 will appear 4 times in the resulting SN. Such SN generation can be simplified and implemented by an FSM and a MUX. The FSM is actually an up counter counts from 0 to $2^N - 1$, assuming x is N-bit. The MUX then outputs x_{i-1} based on the output value of the FSM.

If the SN bit-stream for the other input w is set to be series of '1' followed by a bunch of '0' as shown in Fig. 2. As SC multiplication is simply *AND* operation, it is no necessary to count the second half of the output bit-stream. So, the whole counting process only requires $w \cdot 2^N$ cycles to finish, which saves half latency in average. The authors used a down counter to realize the idea. While $w \cdot 2^N$ is used as the initial value, the down counter decreases by one in each clock cycle. When it reaches "zero", the process is terminated. As a result, CBSC leads to a simpler design as one traditional SNG (typically using LFSR) and the *AND* gate are removed in exchange of a down counter, which is much cheaper than an SNG.

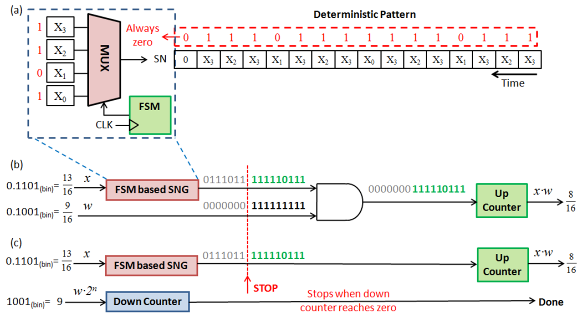

Figure 2: **(a) FSM based SNG which generates SN bit-stream in a deterministic way. (b) CBSC concept. (c) CBSC multiplication method.**

To further improve the computing latency of CBSC method, work in [19] exploits the symmetric properties of the deterministic bit stream pattern so that one can start the counting process from either the end or the beginning of the SN bit-stream depending on the value of the weight to reduce the clock cycles needed. The counting process can be further improved by recent *COSAIM* work in [14], in which a simple formula is used to compute the number of bit '1' in just a single clock cycle with minimum hardware overhead.

These aforementioned works indeed helped reduce the latency, but the long-standing low accuracy issue for small numbers as shown in Fig. 1 for SC still remains. Basically this is due to the intrinsic SC multiplication property that two N-bit inputs for SC multiplication still generate N-bit output product instead of $2N$-bit output.

To mitigate this issue, Zhou in [15] has proposed a scaled population (SP) arithmetic concept. This method indeed improved the error metrics in the "small" range by inserting bit '1's into the input SN bit-stream. As a result, the bit '1' density of the SN bit-stream is then raised to a certain level, like 0.7, for example. Since the SN value is changed during the bit '1' insertion process, the SP method introduced an exponent term so that the scaled SN number will remain approximately the same. The resulting SP number is a 2-tuple including a scaling term and a SN term. But SP method still suffers from a few problems. First, the method is still based on the traditional random-number based SC scheme. The format is difficult to interface with the commonly used binary number format, which is often required in the image processing and machine learning application. Second, it operates in the bit-stream level and requires complicated design to reduce random fluctuation errors for random number generation, scaling and random-shuffling operations. Third, it requires the bit '1' density to be any given value such as 0.7, which can lead to no integer number in the scaling term. Fourth, it uses the *OR* operation for SC addition, which however favors the low bit '1' density (small value) in the SC bit stream, and it contradicts the high bit '1' density requirement in the SC multiplication.

Based on the analysis and observation, we propose our scaled counting-based SC multiplication approach, *Scaled-CBSC* to mitigate the long-standing low accuracy issues of SC at the same time, and try to maintain the advantages of the latest CBSC method.

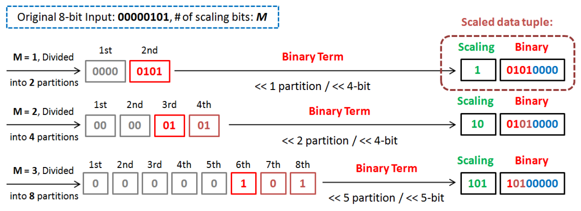

Figure 3: **The data tuple including the scaling and the binary terms.**

3 PROPOSED SCALED SC MULTIPLICATION APPROACH

In this section, we present the details of our proposed scaled counting-based SC (Scaled-CBSC) multiplication approach to mitigate the large errors for small number multiplication.

3.1 Scaled counting-based SC multiplication method

As shown in Fig. 1, SC multiplication will have very large relative error with "small" inputs, especially in the range of [1/256, 15/256], due to the low bit '1' density of the input SN bit-stream, which is an intrinsic property of SC multiplication. So a simple idea to improve the error metrics of SC multiplication is to avoid these "small" inputs, in other words, increasing the bit '1' density of the input SN bit-stream. Unlike SP method [15] which inserts bit '1' into the SN bit-stream, our proposed Scaled-CBSC multiplication directly enlarge the binary number inputs, which is also equivalent to raising the bit '1' density level of the SN bit-stream.

In the Scaled-CBSC multiplication, an N-bit binary number data is represented by a 'scaling-binary' 2-tuple. The tuple consists of two parts, an M-bit scaling term and an N-bit binary term. The data representation and how an N-bit binary number be converted to 2-tuple $\{M, N\}$ is shown in Fig. 3. Specifically the scaling term has M bits, the original binary number is then divided into 2^M partitions. To mitigate the SC multiplication error, we want the binary term to be as large as possible to avoid the "small" inputs regions. So the leading bit '1' in the binary term, which is marked with red in Fig. 3, should appear in the most significant partition. We then left shift $N/2^M$ bits, or one partition each time. The value of the scaling term records number of times the shift operation being carried out. We can easily derive that the maximum number of the scaling bits, or M_{MAX}, equals to $\log_2^N$. As the binary term will always promise the leading bit '1' in the most significant partition. Each partition consists of $N/2^M$ bits, and the least significant bit in the most significant partition weights $(1/2)^{N/2^M}$. Thus it can be verified that after bit partition-based shifting operations, the value of the binary term should be no less than $(1/2)^{N/2^M}$. We show an 8-bit example in Fig. 4. Since we use the binary term of the scaled data presentation as the input of the CBSC kernel in our proposed Scaled-CBSC method (will be discussed later in Sec. 3.2), our binary only representation is more compact than the SP format [15], which is a mixed 2-tuple of bit-stream and binary numbers.

The resulting Scaled-CBSC multiplication approach is shown in Fig. 5. The Scaled-CBSC multiplication consists of two blocks. One is the simple M-bit binary summation to sum the scaling terms (green blocks in Fig. 5) of the two input tuples (x and w) up. The

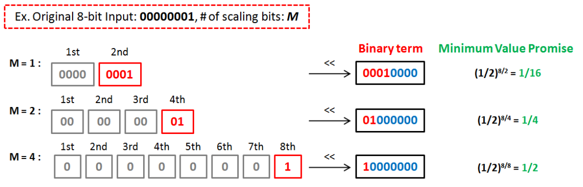

Figure 4: **An 8-bit example to show the minimum promised value of the binary term.**

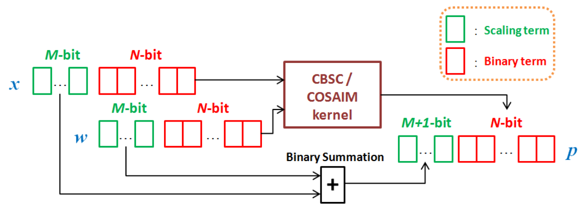

Figure 5: **The Scaled-CBSC/COSAIM method.**

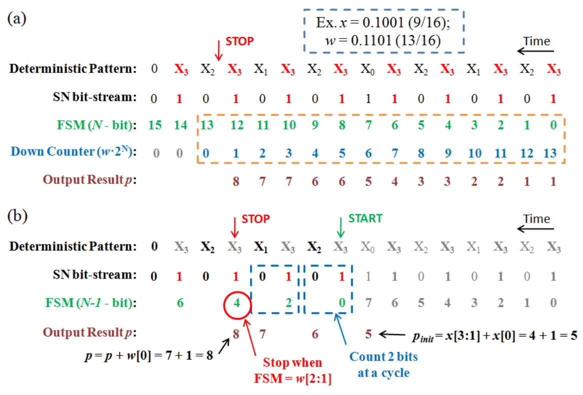
(a)
Ex. x = 0.1001 (9/16); w = 0.1101 (13/16)
STOP
Time
Deterministic Pattern: 0 X3 X2 X3 X1 X3 X2 X3 X0 X3 X2 X3 X1 X3 X2 X3
SN bit-stream: 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1
FSM (N-bit): 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Down Counter (w·2^N): 0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13
Output Result p: 8 7 7 6 6 5 4 3 3 2 2 1 1
(b)
STOP
START
Time
Deterministic Pattern: 0 X3 X2 X3 X1 X3 X2 X3 X0 X3 X2 X3 X1 X3 X2 X3
SN bit-stream: 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1
FSM (N-1-bit): 6 4 2 0 7 6 5 4 3 2 1 0
Output Result p: 8 7 6 5
p_init = x[3:1] + x[0] = 4 + 1 = 5
p = p + w[0] = 7 + 1 = 8
Stop when FSM = w[2:1]
Count 2 bits at a cycle

Figure 6: **(a) The output value of the original CBSC kernel components. (b) The output value of the optimized CBSC kernel components with the maximum number of the scaling bits.**

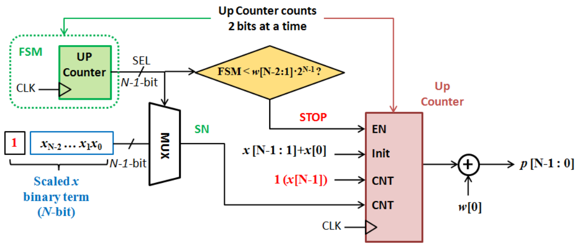

Figure 7: **The structure of the optimized CBSC kernel with the maximum number of scaling bits.**

other block is the CBSC kernel which is shown in Fig. 2. The inputs of the CBSC kernel are the binary terms (red blocks in Fig. 5) of the two input data tuples. The output product p is also a 2-tuple, but with a $M+1$-bit scaling term and an N-bit binary term. Furthermore, the Scaled-CBSC multiplication could be further extended to *Scaled-COSAIM* with the CBSC kernel in Fig. 5 simply exchanged to COSAIM kernel proposed in [14].

3.2 Hardware design and optimization

Assume a $\{M, N\}$-bit 2-tuple has the maximum number of the scaling bits M_{MAX}, as mentioned in Sec. 3.1, the minimum value of the binary term should be 0.5. From Fig. 6(a), we notice that if w is promised to be no less than 0.5, the computing latency of the SC multiplication should be no less than $2^N/2$. And since the SN bit-stream of x is "centrosymmetric", the up counter output value at the $2^N/2$th clock cycle always equals to $x/2+x[0]$. So the counting process before the $2^N/2+1$th cycle is promised, which means we don't need to count the first $2^N/2$ cycles anymore. We can simply start the counting process from the $2^N/2+1$th cycle with an initial value of $x/2+x[0]$ ($x[N-1:1]+x[0]$). Thus we can save $2^N/2$ clock cycles of the computing latency of CBSC method, which is shown in Fig. 6(b).

With the maximum number of scaling bits, the minimum value of the binary term of x equals to 0.5, which means the most significant bit (MSB) of x is bit '1'. Based on the deterministic SN generation pattern of CBSC method, bit '1' will appear once every two clock cycles. So we can improve the up counter to count 2 bits (bit '1' and an SN bit) at a clock cycle, which will further reduce the computing latency. And since one of these two bits is determined, we don't need to use the FSM-based SNG to generate it. The N-bit FSM could be shrunk to $N-1$-bit. Note that as we count 2 bits at a cycle, the up counter of the FSM will increase by 2 at each cycle, which is shown in Fig. 6(b).

To further improve the hardware performance of the Scaled-CBSC multiplication approach, we try to optimize the CBSC kernel upon the original design under the case of the maximum number of the scaling bits. By observing the 3rd and the 4th line in Fig. 6(a), we notice that the output value of FSM in the SNG is just a reversal series of the output value of the down counter before the SC multiplication terminates. This means that when the FSM output value equals to the initial value of the down counter, the whole counting process will be ended. So the down counter is not required anymore, we simply remove it from the CBSC structure, thus further reduce the area and power of our proposed scaled CBSC multiplication. We show the improved scaled CBSC multiplication structure in Fig. 7. Note that since we don't need to count the first $2^N/2$ cycles and count 2 bits at each cycle now, the counting process should not be stopped when $FSM < \lfloor (w-2^N/2)/2 \rfloor$, which is equivalent to $FSM < w[N-2:1]$. For the odd number case, we need to count an extra "half" cycle. In this case, as w is an odd number, $w[0]=1$. We just need to add one to the final output result. Finally, to combine the odd and even cases together, the output result will be added by $x[N-1]\cdot w[0]$ when the process stops. And since $x[N-1]=1$, $x[N-1]\cdot w[0]=w[0]$. We show the optimization in Fig. 7.

4 EXPERIMENTAL RESULTS AND DISCUSSIONS

In this section, we evaluate the performance of the proposed scaled counting-based stochastic computing multiplication approach, named *Scaled-CBSC* multiplication under 8-bit and 16-bit precision. We also compare the proposed approach against the original CBSC (counting-based stochastic computing) multiplication baseline [13], and other state of art works with the same precision.

Table 1
Hardware performance comparison.

	M	Area /μm^2	Delay /ns	ADP	Power /μW	Energy /pJ
8-bit						
CBSC [13]	0	487.96	215.04	104930	26.85	13.75
Improved CBSC[19]	0	657.72	130.56	85873	39.04	10.00
Scaled-CBSC	1	460.51	176.64	81344	22.60	11.57
	2	483.38	176.64	85385	24.15	12.36
	3	575.13	63.36	36440	32.50	4.16
COSAIM [14]	0	660.77	2.05	1355	36.20	0.14
Scaled-COSAIM	1	692.29	2.07	1433	37.57	0.15
	2	710.33	2.07	1470	38.34	0.15
	3	707.79	2.07	1465	39.33	0.16
16-bit						
COSAIM [14]	0	2575.50	3.85	9916	163.95	0.66
Scaled-COSAIM	1	2622.00	3.87	10472	165.33	0.66
	2	2691.13	3.88	10144	157.34	0.63
	3	2700.53	3.87	10216	157.86	0.63
	4	2655.55	3.88	10130	159.86	0.64

4.1 Experimental setup

To evaluate the performance of the proposed Scaled-CBSC multiplication design, we first compare the error metrics and the hardware performance of Scaled-CBSC with the original baseline version: the CBSC multiplication proposed by Sim [13]. We also compare Scaled-CBSC multiplication with a state of art work: an optimized CBSC multiplication [19]. To make a complementary, We show the hardware performance of another state of art work, *COSAIM* [14] which could be treated as an approximate multiplication form of the original CBSC multiplication, with different number of scaling bits. For the 16-bit Scaled-CBSC multiplication, as CBSC multiplication will take 2^{15} clock cycles in average to finish the multiplication, which is too long and doesn't make sense. We just demonstrate the results of COSAIM and "Scaled-COSAIM" designs, which only require a single clock cycle to finish the process.

All the aforementioned multipliers are implemented in Verilog HDL and synthesized with Synopsys Design Compiler using EDK 32nm standard cell library [20] as single-cycle designs. For fair comparison, the power and energy consumption of all the aforementioned multipliers are measured under the same working frequency (250MHz).

For the error metrics evaluation, we developed behavioral simulation models for CBSC and Scaled-CBSC with all the possible number of scaling bits in MATLAB and measured the accuracy using 1 million random inputs uniformly distributed over the set $\{0, 1, ..., (2^N - 1)\}$, $(N = 8, 16)$. The errors are reported with respect to the exact results. The error metrics used to report the error behavior includes: mean error (mean of the absolute value of the error) and standard deviation. Note that both of the improved versions of the CBSC method only do improvements on the computing latency and the hardware performance (area/power/energy), and the error metrics keeps the same as the CBSC baseline. Thus for the error metrics, we just compare our proposed Scaled-CBSC multiplication with the original CBSC baseline.

4.2 Performance evaluation

We show the error metrics for CBSC and Scaled-CBSC multiplication with 8-bit and 16-bit in Fig. 8 and Fig. 9, respectively. Note that the original CBSC method can be treated as Scaled-CBSC with '0' scaling bit, which is shown at the leftmost location in Fig. 8 and Fig. 9.

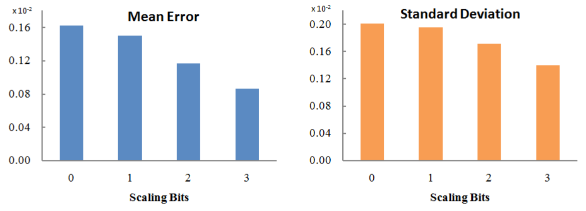

Figure 8: **The mean error and standard deviation of 8-bit Scaled-CBSC multiplication approach.**

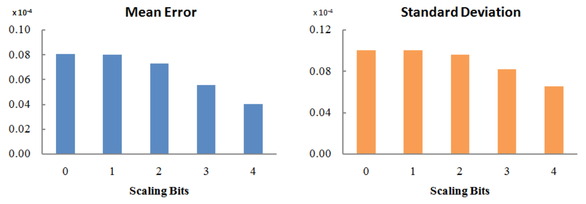

Figure 9: **The mean error and standard deviation of 16-bit Scaled-CBSC multiplication approach.**

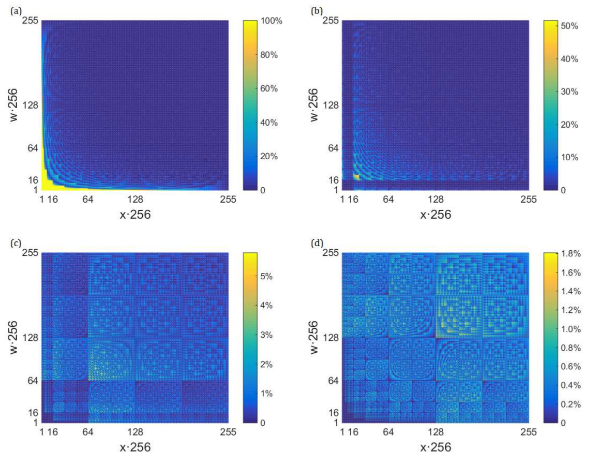

Figure 10: **The absolute relative error distribution of Scaled-CBSC with different number of scaled bits: (a)** $M = 0$; **(b)** $M = 1$; **(c)** $M = 2$; **(d)** $M = 3$.

Fig. 8 shows that with 3 scaling bits, the Scaled-CBSC multiplication can improve the mean error and the standard deviation upon the CBSC baseline by up to 46.6% and 30.4%. Fig. 9 shows that with 4 scaling bits, Scaled-CBSC multiplication can improve the mean error and the standard deviation upon the CBSC baseline by up to 50.3% and 34.9%. Furthermore, from Fig. 10, we can see that with scaling bits, Scaled-CBSC can improve the relative error profile of the original CBSC significantly. With 1, 2, 3 scaling bits, Scaled-CBSC can reduce the maximum relative error from 100% to 51.6%, 5.8% and 1.8%, respectively.

We demonstrate the hardware performance of the Scaled-CBSC multiplication and compare with the CBSC baseline and several state of art works in Table 1. M is the number of scaling bits we used in the multiplication. By observing Table 1, we can see that with 8-bit precision, Scaled-CBSC can improve the delay upon the CBSC baseline by up to 70.5% with 17.9% area overheads, and improve the ADP (area-delay product) and energy consumption upon the CBSC baseline by up to 65.3% and 69.7%, respectively; and improve all the four metrics (area, delay, ADP, energy) upon the state of art work by 12.6%, 51.5%, 57.6% and 58.4%, respectively, with 3 scaling bits. For the Scaled-COSAIM design, no matter 8-bit or 16-bit case, with scaling term, though the Scaled-COSAIM will introduce a little bit hardware resource overheads, it will significantly improve the error metrics as mentioned before.

4.3 An image processing application evaluation

Now, we show how the proposed Scaled-CBSC multiplication approach compare to state of art methods in a multimedia application. Discrete cosine transformation (DCT) is a commonly used lossy image compression method. The quality of the compressed images is usually evaluated using metrics such as PSNR (peak signal noise ratio) and higher PSNR value represents better image quality. We implement the proposed Scaled-CBSC multiplication approach with different number of scaling bits in the DCT-iDCT (inverse DCT) workloads, and compare with the CBSC baseline on five example images, considering both 8-bit and 16-bit precision. As mentioned before, $M = 0$ represents the CBSC baseline. We show the results of image compression in Table 2. For 8-bit precision, we can see that 1 or 2 scaling bits will not improve the image quality much. With 3 scaling bits, Scaled-CBSC can improve the image quality of 5.9dB in average upon the CBSC baseline. For 16-bit precision, 1 scaling bit can improve the image quality of 16.8dB in average, and 2 scaling bits are enough to achieve the best image quality improvement. Here “INF” in Table 2 represents “infinity large”, which means the final output image is completely the same as the input with no quality loss.

Table 2
PSNR (dB) for images after DCT-iDCT using CBSC multiplications.

8-bit						
M	**Lena**	**Boat**	**Barbara**	**House**	**Pepper**	**Avg. (dB)**
0	32.24	31.57	32.70	31.95	32.62	32.22
1	32.43	31.77	32.93	32.17	32.93	32.45
2	32.47	31.85	33.04	32.32	33.03	32.54
3	38.17	37.92	38.58	37.80	37.97	38.09
16-bit						
M	**Lena**	**Boat**	**Barbara**	**House**	**Pepper**	**Avg. (dB)**
0	78.23	76.08	77.54	77.32	74.99	76.83
1	INF	91.52	96.30	93.29	93.29	93.60
2	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF	INF

5 CONCLUSION

In this work, we have proposed a novel scaled counting-based stochastic computing multiplication design, named *Scaled-CBSC*. The proposed design introduced scaling bits to promise the inputs of the SC multiplication kernel to be larger than 0.5, avoiding the “small” number region which led to large relative error of SC multiplication. Numerical results showed that 8-bit *Scaled-CBSC* with 3 scaling bits could achieve up to 46.6% and 30.4% improvements in mean error and standard deviation, respectively; reduced the peak relative error from 100% to 1.8%; and improved 12.6%, 51.5%, 57.6%, 58.4% in delay, area, area-delay product, energy consumption, respectively, over the state of art work. For discrete cosine transformation (DCT) application, 3 scaling bits are required for 8-bit *Scaled-CBSC* multiplication to significantly improve the image quality by 5.9dB.

REFERENCES

[1] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Approximate computing and the quest for computing efficiency,” in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2015.

[2] P. Kulkarni, P. Gupta, and M. Ercegovac, “Trading accuracy for power with an underdesigned multiplier architecture,” in *2011 24th Internatioal Conference on VLSI Design*, pp. 346–351, IEEE, 2011.

[3] K. Bhardwaj, P. S. Mane, and J. Henkel, “Power-and area-efficient approximate wallace tree multiplier for error-resilient systems,” in *Fifteenth International Symposium on Quality Electronic Design*, pp. 263–269, IEEE, 2014.

[4] B. S. Prabakaran, S. Rehman, M. A. Hanif, S. Ullah, G. Mazaheri, A. Kumar, and M. Shafique, “Demas: An efficient design methodology for building approximate adders for fpga-based systems,” in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 917–920, IEEE, 2018.

[5] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar, “Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators,” in *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

[6] S. Ullah, S. S. Murthy, and A. Kumar, “Smapproxlib: library of fpga-based approximate multipliers,” in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[7] S. Hashemi, R. I. Bahar, and S. Reda, “Drum: A dynamic range unbiased multiplier for approximate applications,” in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 418–425, IEEE, 2015.

[8] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, “Energy-efficient approximate multiplication for digital signal processing and classification applications,” *IEEE transactions on very large scale integration (VLSI) systems*, vol. 23, no. 6, pp. 1180–1184, 2014.

[9] H. Saadat, H. Bokhari, and S. Parameswaran, “Minimally biased multipliers for approximate integer and floating-point multiplication,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2623–2635, 2018.

[10] H. Saadat, H. Javaid, A. Ignjatovic, and S. Parameswaran, “Realm: reduced-error approximate log-based integer multiplier,” in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1366–1371, IEEE, 2020.

[11] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, “Delayed stochastic decoding of ldpc codes,” *IEEE Transactions on Signal Processing*, vol. 59, no. 11, pp. 5617–5626, 2011.

[12] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, “Computation on stochastic bit streams digital image processing case studies,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 3, pp. 449–462, 2013.

[13] H. Sim and J. Lee, “A new stochastic computing multiplier with application to deep convolutional neural networks,” in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2017.

[14] S. Yu, Y. Liu, and S. X.-D. Tan, “COSAIM: Counter-based stochastic-behaving approximate integer multiplier for deep neural networks,” in *Proc. Design Automation Conf. (DAC)*, pp. 1–6, Dec. 2021.

[15] H. Zhou, S. P. Khatri, J. Hu, and F. Liu, “Scaled population arithmetic for efficient stochastic computing,” in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 611–616, 2020.

[16] A. Alaghi and J. P. Hayes, “Fast and accurate computation using stochastic circuits,” in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–4, IEEE, 2014.

[17] A. Alaghi and J. P. Hayes, “Survey of stochastic computing,” *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s, pp. 1–19, 2013.

[18] M. H. Najafi, D. J. Lilja, and M. Riedel, “Deterministic methods for stochastic computing using low-discrepancy sequences,” in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2018.

[19] Z. Chen, Y. Ma, and Z. Wang, “Optimizing stochastic computing for low latency inference of convolutional neural networks,” in *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–7, 2020.

[20] R. Goldman, K. Bartleson, T. Wood, K. Kranen, V. Melikyan, and E. Babayan, “32/28nm educational design kit: Capabilities, deployment and future,” in *2013 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, pp. 284–288, IEEE, 2013.