COSAIM: Counter-based Stochastic-behaving Approximate Integer Multiplier for Deep Neural Networks

Shuyuan Yu*, Yibo Liu*, Sheldon X.-D. Tan*
* Department of Electrical and Computer Engineering, University of California, Riverside, CA 92521 stan@ece.ucr.edu

2021 58th ACM/IEEE Design Automation Conference (DAC) | 978-1-6654-3274-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/DAC18074.2021.9586097

***Abstract*—In this work, we propose a new counter-based stochastic-behaving approximate integer unsigned multiplier, called *COSAIM*, for many emerging error tolerant application workloads such as deep neural networks. Unlike existing approximate multipliers, which are based on some deterministic ad-hoc methods or mathematical formula, the new design is an improved stochastic multiplier, which performs improved sequential counting for multiplication operation in a deterministic way. In this work, we further improve the counting efficiency by introducing approximate schemes to significantly speed up the counting process, which leads to significant clock cycle reduction with no accuracy loss. *COSAIM* bears all the advantages of stochastic computing such as built-in configurability for progressive performance-accuracy trade-off. At the same time, it shows very small latency and high energy efficiency. Our evaluation shows that the *COSAIM* with error improvement operation can achieve very low error bias (0.06%), along with lower mean error (0.30% to 3.49%), and low peak errors (around 1.81%) with variance of $1.47 \times 10^{-4}\%$. Experimental results obtained from Xilinx ISE show that compared with the 8-bit exact multiplier baseline, *COSAIM* can save up to 53.95%, 32.84%, 52.24%, 21.05% in area, power, energy and the product $Area \cdot 1/Throughput$, respectively. Furthermore, by doing shared parallel design, *COSAIM* can further lead to improvements in area, power and energy reduction by 60.44%, 53.33% and 68.54%, respectively compared to the baseline. We also implement *COSAIM* in a *Convolution Neural Network* (CNN) and test it on CIFAR10 dataset and find that CNN with *COSAIM* delivers similar inference accuracy compared to some state of art approximate multipliers.**

I. Introduction

Approximate computing allows the trade-off between area, latency and power for more efficient implementation for error tolerant applications such as machine learning and multimedia applications [1]. Typically those applications are dominated by the multiplications operation and the major design constraints in these applications are energy dissipation and latency due to massive data intensive computation in limited energy and resource budgets. Multipliers are one of the most energy-hungry units and area and the energy efficient multiplier design is paramount important for efficient processing of deep neural networks which require intensive multiply-accumulate operation (MAC) for matrix/tensor based operations [2]. In addition to the error-resilient nature, studies show that DNNs are robust to reduction in the precision of the arithmetic operations. 16-bit fixed point is demonstrated to be sufficient for training neural networks with no loss in classification accuracy [3]. 8-bit precision is sufficient for inference with minimal accuracy error loss [4].

A number of approximate multiplier designs have been proposed recently [5]–[13]. Those approximate multipliers employ some ad-hoc 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 [14], [15]. SC can provide *inherent progressive* trade-off between accuracy and latency/energy/area trade-off 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 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 [16]. First, It replaces the *AND* operation with a '1' counting process in the bit stream, which can be reduced without going through the full length of the bit streams and second, the bit streams no longer need to be random. However, it still suffers from the sequence nature of counting process.

In this work, we further explore the counter-based SC multiplier (CBSC-MUL) for further efficiency improvement. Due to its deterministic nature, the resulting multiplier, *COSAIM* can be viewed as a general approximate multiplier. At the same time, *COSAIM* bears all the advantages of stochastic computing such as built-in configurability for progressive performance-accuracy trade-off. At the same time, it shows very small latency and high energy efficiency compared to cutting-edge approximate multiplier designs. As a result, we would like to present a new design in light of more general approximate multipliers to give a new perspective of the stochastic computing based on some common performance metrics. The main contributions of this work are as follows:

1. First, in *COSAIM*, instead of counting the bit '1' sequentially, we propose to compute the number of '1' in the bit stream using a simple formula for different positions in corresponding to the binary number, which can significantly speed up the counting process and further lead to significant clock cycle reductions with no additional accuracy loss.
2. We show the circuit implementation of resulting *COSAIM* multiplier and its MAC designs for machine learning applications in which weight sharing can be further leveraged for area, power and energy reduction.
3. Compared to the 8-bit exact multiplier as the baseline, our evaluation shows that the *COSAIM* with error improvement operation can achieve very low error bias (0.06%), along with lower mean error (0.30% to 3.49%), and low peak errors (around 1.81%) with extremely low variance of $1.47 \times 10^{-4}\%$.
4. Experimental results obtained from Xilinx ISE shows that compared with the 8-bit exact multiplier baseline, *COSAIM* can save up to 53.95%, 32.84%, 52.24%, 21.05% in area, power, energy and the product $Area \cdot 1/Throughput$, respectively. Furthermore, by doing parallel design, *COSAIM* can further do improvements in area, power and energy reduction by 60.44%, 53.33% and 68.54%, respectively.
5. We also evaluate the *COSAIM* in a *Convolution Neural Network* (CNN) and do test on CIFAR10 dataset. The result shows that *COSAIM* can deliver similar inference classification accuracy compared to state of art approximate multipliers, which justify its hardware implementation advantages as mentioned earlier.

This paper is organized as follows: Section II reviews several recently proposed approximate multiplication designs. Section III reviews the CBSC method and the design of CBSC-MUL from which our proposed approximate multiplier is inspired. Section IV presents the proposed *COSAIM* design including techniques to improve error metrics and to save area. Section V shows the error and hardware design metrics of *COSAIM*. Experimental results for the area, power, energy dissipation, the product $Area \cdot 1/Throughput$ of *COSAIM*

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

are summarized. Besides, we also evaluate the performance of *COSAIM* when implemented in a CNN doing classification for CIFAR10 dataset. Finally, section VI concludes the paper.

II. Review of Related Work

In this section, we review a few related works concerning our proposed *COSAIM* design.

Recently, various designs of approximate unsigned integer multipliers have been proposed. Earlier designs often involve ad-hoc based approximations, such as recursive multipliers [5] which consist of 2×2 multiplication blocks, simplification of Wallace tree [6], simplifying partial product generation/summation [7]–[9], Others use a smaller multiplier by extracting m-bit fragment from the N-bit precision inputs. Such as [10], [11].

Among these methods, many recent cutting edge approximate multipliers are developed based on the classical approximate log-based multiplier, the MA multiplier, proposed by Mitchell as it shows good overall performance and has flexibility for error compensation [17]. Specifically, for the MA multiplier, the two inputs A and B are first represented by the following format: $2^{ka} \cdot (1+x)$ and $2^{kb} \cdot (1+y)$, respectively. Then the multiplication result C can be approximated as (1).

$$C = \begin{cases} 2^{ka+kb} \cdot (1+x+y), & x+y < 1, \\ 2^{ka+kb+1} \cdot (x+y), & x+y \geq 1 \end{cases} \quad (1)$$

MA design requires four steps to finish the multiplication process. First it utilizes leading-one detectors (LOD) to find the leading one as the integer part; second, barrel shifters are used to re-align the rest of the bits as the fraction part; then it sums the two fraction parts up; and finally it shifts back with the same bits. Although MA suffers from high MRED (absolute mean relative error) and peak relative error of 3.76% and 11.11%, respectively. It can achieve very high area, power and energy reduction.

To further improve the accuracy of the MA method, several derivative works have been proposed by means of different error compensation mechanisms. For instance, the MBM design proposes to add a fixed single error-correction term to the result [12]; This is further improved by the LeAp multiplier, which adds different error coefficients to the fraction parts based on the value ranges of the results [18]; The REALM multiplier design further improves such compensation scheme by using a look-up table to store $M \times M$ coefficients for $M \times M$ partitions of inputs at some hardware overhead [13]. These works indeed improved the error metrics of the MA multiplication without incurring too much area overhead, power and energy cost.

One observation is that the log-scaled based Mitchell multiplier and their derivative perform well in area, power and energy reduction for traditional integer precision (such as 16-bit or 32-bit data). The benefits of log based computing in terms of hardware design metrics, however, become less significant for 8-bit precision, which is widely used in today's DNN networks. For instance, the MA multiplier, which is the most area efficient multiplier among all of the Mitchell based multipliers, only leads to 25% saving in area compared to the accurate baseline. Furthermore, in terms of power and energy, log-based approximate multiplier and its derivative methods even exceed that of the exact multiplier with 8-bit inputs. In this work, we will focus on the 8-bit precision multiplier design and demonstrate the superior performance of the proposed new design against two state of art Mitchell-based multiplier designs: The LeAp [18] and the REALM [13] designs.

III. Preliminaries

In this section, we present some relevant backgrounds for stochastic computing (SC) and the recently proposed counter-based SC. Let's assume the bit width is N for the given two binary numbers (BN) x and w, which are both in the range of [0, 1]. The conventional SC multiplier using *AND* gate (for unipolar encoding) will take 2^N, which is the length of bit-stream of *stochastic number* (SN), cycles to finish the work. To improve this, Sim *et al.* in [16] proposed a counter-based SC (CBSC) multiplier design. Here, the multiplier mainly consists of two counters. The down counter counts the binary value of input w and the up counter counts the result $x \cdot w$. So the operation only takes $w \cdot 2^N$ cycles to finish. More importantly, the SN of input x can be generated in a deterministic way without hurting the accuracy (actually more accurate). As a result, such design is simpler as we eliminate the two traditional stochastic number generators or SNG (typically using Linear Feedback Shift Registers) and *AND* gates in exchange of a down counter and an up counter, which is much cheaper than SNG.

To generate the SN of x, which can be a conventional low-discrepancy random number, the authors proposed a deterministic way to do this. The method evenly distributes the x_{i-1}, which is the ith bit of x, based on its binary weight 2^i. For instance, if $i = 3$, then x_2 will appear 4 times in the resulting SN as shown in Fig. 1. Such SN generation can be simplified and implemented by an FSM and a MUX. The whole CBSC-MUL design is shown in Fig. 1.

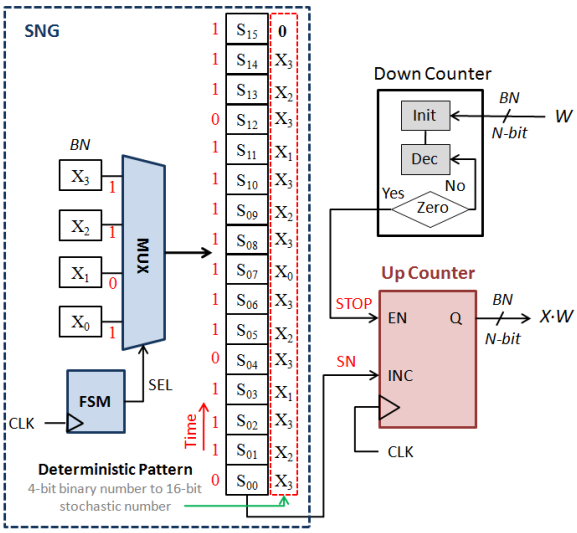

Fig. 1: The counter-based SC multiplication design diagram.

CBSC-MUL though outperforms traditional SC method in accuracy, area and computing latency, it still requires 2^{N-1} clock cycles for N-bit binary inputs in average. We observe that the SN generated by CBSC method follows a deterministic pattern. At the same time, the computation latency, or the number of bits need to be counted in the SN equals to the value of the other input, which is already known. As both the SN bit distribution and the number of bits need to be counted are already known, we can directly calculate the multiplication result without the counting process based on the two inputs. As a result, the latency of the resulting SC multiplication can be further improved. Based on these observation, we propose *COSAIM*, a new counter-based stochastic-behaving approximate integer multiplier.

IV. Proposed Counter-Based Approximate Multiplier

In this section, we elaborate our proposed integer approximate multiplier: *COSAIM*.

A. The proposed accelerated counter-based multiplication

Suppose we have a stochastic number (SN) converted from a binary number x, x is N-bit. The product of CBSC-MUL equals to how many times the bit '1' appears in the stochastic number bit-stream

of x in the first w cycles, while w is the other input. So the product of CBSC can also be represented as (2).

$$Product = \sum_{i=0}^{N-1} N_{x_i} \cdot x_i \tag{2}$$

Following the SN deterministic pattern of CBSC, x_{N-1}, which is the most significant bit (MSB), appears once every 2 cycles. Since the counting process stops at wth cycle, then we can easily prove that $N_{x_{N-1}}$ equals to $w/2$. When w is even, there is no remainder after $w/2$, we simply do 1-bit right shift operation to w to obtain $w/2$. But when w is odd, we need to count one more cycle, then $N_{x_{N-1}}$ equals to $\lfloor w/2 \rfloor + 1$. As w being odd or even is determined by w_0, then $N_{x_{N-1}}$ can be simply represented in the form of $\lfloor w/2 \rfloor + w_0$. In the same manner, as x_{N-2} appears once every 4 cycles, $N_{x_{N-2}}$ equals to $\lfloor w/4 \rfloor + w_1$, and finally arbitrary N_{x_i} can be calculated by (3).

To further illustrate this, we walk through an example of the 4-bit multiplication shown in Fig. 2.

$$N_{x_i} = \lfloor w/2^{N-i} \rfloor + w_{N-1-i} \tag{3}$$

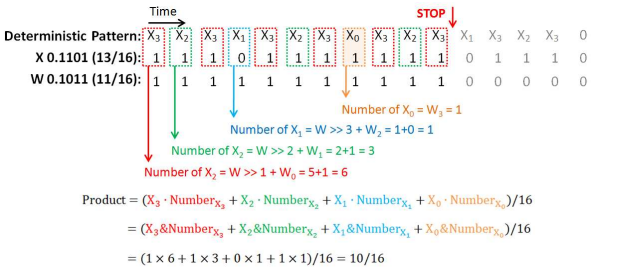

Fig. 2: 4-bit approximate multiplication example.

B. The circuit design for the proposed COSAIM multiplier

We show the structure of our proposed COSAIM design in Fig. 3. The proposed design is composed of shift registers, *AND* Gates and adders. $\lfloor w/2^{N-i} \rfloor$ is realized by simply using bit shifting operation. After added by w_{N-1-i}, the number of x_i is obtained. As one bit multiplication is actually a simple *AND* operation, we use an *AND* Gate to obtain the value of $N_{x_i} \cdot x_i$. Then to sum up $N_{x_i} \cdot x_i$ together, we utilize an adder tree. The number of the adder tree levels is actually also the number of the pipeline stages. Increasing the number of the pipeline stages will increase the speed of the COSAIM design, but also increase the required area at the same time. For N-bit inputs, if we use the largest possible pipeline stages, the computation latency of the adder tree is of $\mathcal{O}(Log_2(N))$. That means we can do latency reconfigurable design based on the input bit-width just like stochastic computing.

To achieve better hardware performance, the area and the throughput should both be taken into consideration. We use the product $Area \cdot 1/Throughput$ to evaluate the trade-off between the area and the throughput. Obviously, less the product is, better the hardware performance will the design achieve. The proposed COSAIM design will improve its hardware performance by increasing the number of pipeline stages, showing the best improvement over the baseline (the exact 8-bit multiplier) and other related works with 8-bit inputs. We'll prove this in Sec. V-C.

C. Accuracy improvement for COSAIM

Different from Mitchell based approximate multipliers REALM [13] and LeAp [18], COSAIM shows very high relative error when the inputs are small as shown in Fig. 4(a). Note that the relative error shown in the figure is the absolute value and in percentage. This behavior actually is well known for stochastic computing as the random errors in multiplication become significant

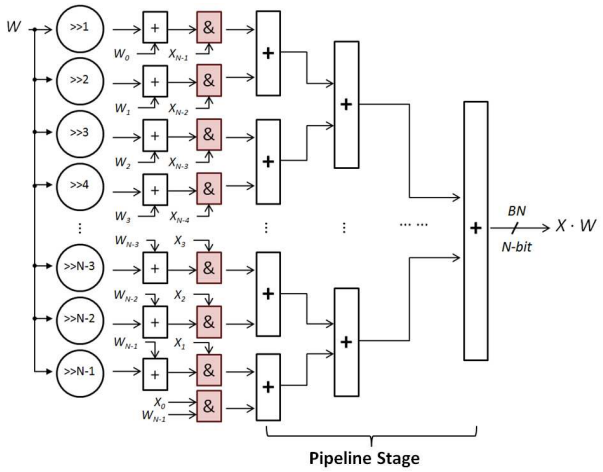

Fig. 3: COSAIM design.

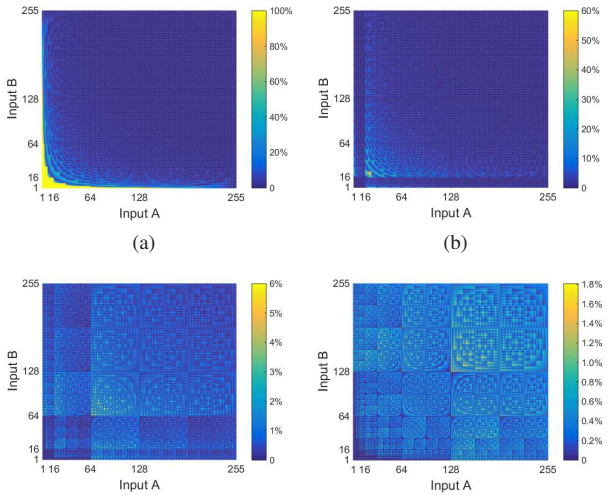

Fig. 4: COSAIM relative error profile: (a) with no error configuration. (b) M=2. (c) M=4. (d) M=8

when the result is close to zero [19]. For SC multiplication, N-bit inputs will also obtain an N-bit output. Compared to the exact multiplier and Mitchell based approximate multiplier, whose output is $2N$-bit precision with N-bit inputs, COSAIM will lose the low N-bit precision of the $2N$-bit output. Thus leads to the error profile shown in Fig. 4(a).

There are many mitigation techniques to resolve this issue. For instance, one can re-train the DNN networks to remove near-zero weight [19] or deploy the dynamic scaling for the input data [20].

In this work, we propose to enlarge the inputs to avoid "*high relative error area*". To this line, we first perform left shifting operation when the input is small. We ensure the new inputs are large enough to achieve the output with small relative error. Of course, the output is larger than the actual value now. As a result, we need to shift the output back by doing right shifting operation after the multiplication.

Before enlarging the input, we first need to judge whether the input is "small enough". The input is separated into multiple partitions. The error configure parameter M is used to represent the number of the partitions. Note that when $M = 1$, the input is not separated at all, meaning that we don't do any operation to the input then. We use "leading-one detector" (LOD) to detect the location of the

leading bit '1' in the input. When the leading bit '1' is not in the most significant partition (the partition which the MSB belongs to), we perform left shifting operation to enlarge the input based on the size of the partition. Based on the error profile shown in Fig. 4(a), to achieve the best error improvement, we left shift the input until the leading bit '1' is in the most significant partition. We sort the partitions in an order from left to right, for example, the most significant partition is the first partition and the least significant partition is the Mth (the last) one. When the leading bit '1' is in the ith partition, we left shift $N/M \cdot (i-1)$ bits as each partition consists of N/M bits. We use 3 examples shown in Fig. 5 with M = {2,4,8} to help understanding. The improved error metrics of COSAIM with M = {2,4,8} are shown in Fig. 4(b)˜ 4(d). We can observe that the relative error is significantly reduced.

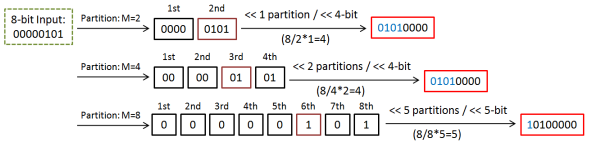

Fig. 5: COSAIM error improvement example.

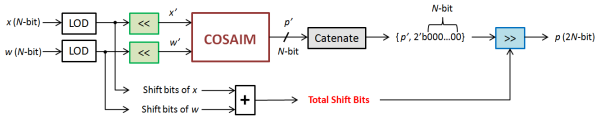

Fig. 6: The structure of COSAIM design with accuracy improvement.

The structure of the optimized COSAIM with accuracy improvement method is shown in Fig. 6. The two inputs x and w are firstly converted to the numbers that are large enough with LODs and shift registers. After data processing, we also need 2 registers to record the number of bits that each input is shifted. We add them together to obtain the number of bits the output need to right shift back. Note that since the output of COSAIM is N-bit actually, we concatenate the output with N-bit '0' to achieve the $2N$-bit precision. Finally, we use a shift register again to do the right shift of the concatenated output.

We remark that the additional accuracy improvement design will incur additional area and power overheads. On the other hand, the error improvement scheme at the circuit level may not be necessary as the operations can be performed as the application levels as demonstrated in the DNN network re-training and weight regulation work mentioned earlier. As a result, the implementation costs associated with this SC-inherent accuracy issue is application and problem specific. But the accuracy improved COSAIM proposed in this subsection can demonstrate error metrics of COSAIM design under those ideal data inputs as shown in the experimental result section.

D. Further efficiency improvement by area sharing

For DNN's applications, we can have or arrange the multiplications (for instance convolution operation) such that one can have one multiplicant (like weights) share with many multipliers. Such shared-multiplicant multiplication is very amicable for counter-based stochastic computing as demonstrated in the parallel multiplier design in the BISC-MVM design [16].

In this work, we also propose a design which contains parallel multipliers to do multiple multiplications at the same time. Like BISC-MVM, to save area, one of the inputs need to be the same, we use w to represent it. Then w can be shared by the parallel multipliers. Supposed we have m parallel multipliers, j is in the range of (1,m), x_j and w are the two inputs of the jth parallel multiplier. Since w is shared, and N_{x_i} is only determined by the value of w. $N_{x\_j_i}$ actually keeps the same for all the x_j. In this manner, the N_{x_i} calculation part can be shared by all of the parallel multipliers. Thus, save the total design area. We show the structure of this parallel design in Fig. 7. We will show the area and power reduction of the parallel design in Sec. V-C.

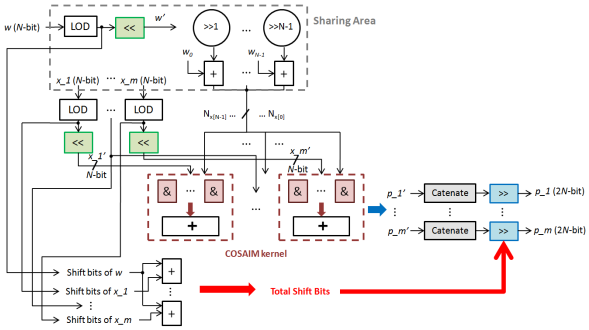

Fig. 7: The parallel COSAIM design.

V. Experimental Results and Discussions

In this section, we perform a comprehensive evaluation of the error behavior and the performance metrics of the proposed 8-bit *COSAIM* multiplier design. We first compare it against the accurate unsigned integer multiplier. Then we compare it with two state-of-the-art approximate multiplication designs: REALM [13], LeAp [18]. Furthermore, we compared it against the stochastic based CBSC-MUL [16], as well as the traditional Mitchell's multiplication design MA [17]. Finally, we evaluate those multipliers for a convolution neural network (CNN) application and compare *COSAIM* against the same CNN design using MA and LeAp multipliers.

A. Experimental setup

We implemented the 8-bit version of these aforementioned multipliers in Verilog and synthesized using Xilinx ISE 14.7 for XC6SLX45 device of Spartan-6 family. The area, delay and power consumption are directly reported from ISE Simulations and Xilinx XPower Analyzer. To make a fair comparison for the power consumption for all the approximate multipliers, the clock frequency is set to be 100MHz. The energy dissipation is measured based on the total execution clock cycles and the power consumption. Since CBSC-MUL [16] is actually stochastic computing design and requires 2^{N-1} clock cycles (N-bit precision) in average to finish the process, the throughput can not be simply calculated by inverting the value of delay and should be calculated as (4), note that $Latency$ is the number of the required clock cycles.

$$Throughput = 1/(Delay \cdot Latency) \tag{4}$$

Also, to make the trade-off between the area and the throughput more obviously, we use the product of $1/Throughput$ and LUTs (area) as the performance metrics to evaluate the hardware performance of the multipliers.

For the error characterization, we develope behavioral simulation models of the multipliers in MATLAB. Note that the input for CBSC-MUL should always be within the range of (0,1) because of the limitations of stochastic computing. So one million random inputs are uniformly distributed over the set {1/256,...,255/256}. The errors are calculated with respect to the exact 8-bit multiplication result. We follow the similar error metrics to report the error behavior: error bias [10] (mean of relative error), MRED [21] (mean of absolute relative error), variance [10], peak error (maximum of the absolute relative error). All error metrics are in percentage.

B. Accuracy evaluation

The error metrics of the approximate multipliers compared with the exact result are shown in Table I. As discussed in Sec. IV-C, the

proposed COSAIM design can improve the error metrics by adjusting the error configure parameter M. With M = {1,2,4,8}, we actually have four choices to do the bit-shifting operation. M is then used as a suffix, for example, COSAIM-2 means $M = 2$. Note that when $M = 1$, COSAIM will not perform any bit-shifting operation and simply represented as *COSAIM* in the table. In this situation, COSAIM is essentially an accelerated approximate computing form of CBSC-MUL: its values of error metrics are exactly the same as that of CBSC-MUL shown in the 6th row in Table I. For the two state-of-the-art approximate multipliers REALM and LeAp, we choose the error configure parameters which can lead to the smallest possible error in the design to make the fair comparison.

Error metrics (%)				
	Error Bias	MRED	Peak Error	Variance
COSAIM	-0.63	3.49	100	**4.02E-04**
COSAIM-2	-0.08	1.29	51.61	**3.68E-04**
COSAIM-4	0.11	0.53	5.79	**2.57E-04**
COSAIM-8	0.06	0.30	1.81	**1.47E-04**
CBSC-MUL	-0.63	3.49	100	**4.02E-04**
LeAp [18]	0.37	1.36	4.71	0.0036
REALM [13]	-0.11	2.63	9.10	0.0086
MA [17]	-3.76	3.76	11.11	0.0227

TABLE I
ERROR METRICS OF 8×8 APPROXIMATE MULTIPLIERS.

From Table I, we can see that the smallest Error Bias, MRED, Peak Error and variance that COSAIM-8 can achieve are 0.06%, 0.3%, 1.81% and 4.02×10^{-4}%, respectively with the accuracy improvement operation. The plain COSAIM also has pretty decent Error Bias and MRED compared to the three approximate multipliers. We also notice that COSAIM has ultra low variance compared to the existing works, even without error improvement, the variance of COSAIM is about an order of magnitude better than those published three multipliers.

C. Hardware performance metrics

Next, we evaluate the hardware performance metrics of the approximate multipliers and the results are shown in Table II. We choose the Xilinx default exact multiplier (8-bit) IP core [22] as our comparison baseline. The number of pipeline stages of the Xilinx IP multiplier is set to be 3, which is the suggested one. The number in bracket following COSAIM in Table II represents the number of pipeline stages COSAIM takes (for instance, *COSAIM (2)* means the design has 2 pipeline stages). Compared with the exact multiplier baseline, COSAIM design can save up to 53.95% in area. Also, COSAIM design can achieve the same throughput (delay) with 21.05% in area reduction when we increase the number of pipeline stages of the design.

The 4th column of Table II shows the product $Area \cdot 1/Throughput$, which is improved when the pipeline stages of COSAIM increases. *COSAIM (3)* performs best when compared with all of the approximate multipliers. Note that though CBSC-MUL requires very small area and delay, the average clock cycles the stochastic computing needs is 128 for 8-bit precision. Hence the product $Area \cdot 1/Throughput$ is pretty large. And different from the 16-bit or 32-bit multiplication, MA [17] shows a minor improvement in area reduction and throughput when the inputs are 8-bit, as shown in Table II. In the same manner, LeAp and REALM, which are both based on Mitchell's multiplication also show small improvement in area and delay compared to the baseline.

For power and energy comparison, COSAIM (2) can save up to 32.84% of power. But COSAIM (1) is more energy efficient, doing up to 52.24% in energy dissipation. From Table II, we observe that though the power of CBSC-MUL is lower than COSAIM, the total energy consumption is much more than COSAIM because of the long latency. And compared to MA, LeAp and REALM, COSAIM outperforms them both in power and energy consumption.

	LUT	Delay (ns)	Area1/Throughput	Power (mw)	Energy (nJ)
Xilinx IP	76	2.666	202.616	67	2.01
CBSC-MUL	24	2.666	8189.952	39	49.92
COSAIM (1)	35	5.351	187.285	48	0.96
COSAIM (2)	50	3.718	185.900	45	1.35
COSAIM (3)	60	2.666	159.960	56	2.24
MA [17]	57	3.753	213.921	83	4.15
LeAp [18]	68	3.799	258.332	81	4.86
REALM [13]	71	3.789	269.019	83	4.98
Area sharing by parallel multiplication					
Xilinx IP	76	2.666	202.616	30	0.89
COSAIM	30(16)	6.992	210.197	14	0.28
CBSC-MUL	13(11)	2.666	4450.887	1	1.18
MA [17]	53(14)	2.666	141.444	44	2.20
LeAp [18]	52(16)	3.799	197.785	41	2.43
REALM [13]	52(19)	3.789	197.309	44	2.64

TABLE II
PERFORMANCE METRICS OF 8×8 APPROXIMATE MULTIPLIERS.

As discussed in Sec. IV-C, COSAIM improves the error metrics by enlarging the inputs when they are small to avoid the "large error area" shown in the error profile (Fig. 4). The error configuration, actually the bit shifting operation is done at circuit level. But the error configuration at circuit level has high hardware cost, which will incur area overhead for COSAIM-8. We consider this approach be inefficient and should not be done at circuit level. So we prefer to do the error configuration at application level. In fact, if the inputs could be adjusted to avoid the "large error area" in the COSAIM error profile, it would achieve the same effect as doing the error configuration process. We'll prove this later in Sec. V-D.

We further show the performance metrics of the parallel design at the bottom part of Table II. For the *LUT* column, the numbers in round brackets are the shared LUTs for each multiplier. To be fair, not only CBSC-MUL and COSAIM save area when one of the inputs is shared; REALM [13], LeAp [18] and MA [17] can also save area by sharing the input. And for the exact multiplier, though it cannot save area by sharing the input, the average power for each multiplier will also decrease with the parallel design due to the architecture of FPGA. So we evaluate the average power and energy dissipation of the exact multiplier in the parallel design as the reasonable baseline.

We follow the BISC-MVM structure, which use 256 parallel CBSC-MUL as discussed in the related work [16], we also use the same number of multipliers in the parallel design. The average area of the parallel multipliers is calculated by (5).

$$Area_{Avg.} = Area_{Private} + Area_{Common}/256 \quad (5)$$

Where $Area_{Private}$ is the number of LUTs which cannot be shared, and $Area_{Common}$, which is the value shown in the round brackets following $Area_{Private}$, is the number of LUTs which can be shared. From the bottom part of Table II, we find that COSAIM can lead to further area reduction: it saves up to 60.44% of area now, which is much better than MA [17], REALM [13] and LeAp [18], which only have about 31.58% in area reduction. Considering the power and energy, though the average power and energy dissipation also goes down for the exact multiplier, our proposed COSAIM design still shows improvements compared with the single multiplier situation, saving up to 53.33% and 68.54% in power and energy consumption, respectively.

D. A CNN based performance evaluation

To further evaluate the performance of COSAIM in applications which are multiplication-intensive, we implement COSAIM, LeAp [18] and MA [17] in Convolution Neural Network (CNN) and compared with the 8-bit exact multiplier baseline, which is developed on a Python platform.

We use CIFAR10 [23] dataset to test the CNN application with approximate multipliers. The network consists of two convolution

(CONV) layers and three fully (FC) connected layers. The network is trained with 2500 steps, using double-precision floating point numbers with a batch size of 128 in one step. We then use the the approximate multipliers and exact multiplier with 8-bit precision to do the inference. Since the CONV layer is computation-intensive, and FC layer is memory-extensive. We simply replace the multipliers in CONV layers. Fig. 8 shows the classification accuracy comparison of the exact multiplier, COSAIM, MA and LeAp on 1000 images. The accuracy is in percentage.

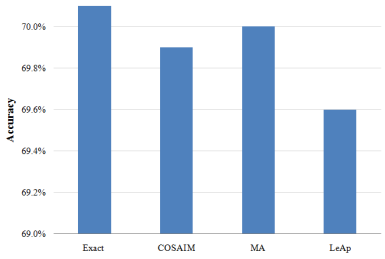

Fig. 8: CIFAR10 dataset inference accuracy based on different approximate multipliers.

From Fig. 8, we can see that the exact multiplier and approximate multipliers almost achieve the same inference accuracy. From Table I, we know that plain COSAIM performs worse than MA and LeAp in MRED and Peak Error. But since CNN is an error tolerant application and when the multiplication product is small, it will not affect the final convolution result much even though the relative error is big. From Fig. 4, we find that COSAIM only has high relative error when the inputs are small. The relative error for large inputs which have much larger influence on the convolution result is actually very small. If we just focus on these large multiplications, which are more important, COSAIM will become very accurate in this case. From Table I, we can see that the relative error is pretty small and hence the CNN with COSAIM can obtain the inference result without losing much accuracy compared to the exact multiplier. As a result, we can achieve both area and energy efficient neural network without losing much inference accuracy by implementing the plain COSAIM.

VI. Conclusion

In this article, we have proposed a new counter-based stochastic-behaving approximate integer unsigned multiplier, *COSAIM* for many emerging machine learning hardware implementation. *COSAIM* is an accelerated design of recently proposed counter-based stochastic multiplier. As a result, it still kept the advantages of stochastic computing such as built-in progress reconfigurability for progressive performance-accuracy trade-off. Experimental results show that the proposed *COSAIM* design with error improvement can achieve very low error bias (0.06%), along with lower mean error (0.30% to 3.49%), and low peak errors (around 1.81%) with variance of $1.47 \times 10^{-4}\%$. Compared with the 8-bit exact multiplier baseline, *COSAIM* could save up to 53.95%, 32.84%, 52.24%, 21.05% in area, power, energy and the product $Area \cdot 1/Throughput$, respectively. Furthermore, by doing parallel design, *COSAIM* could further do improvements in area, power and energy reduction by 60.44%, 53.33% and 68.54%, respectively. We also implemented *COSAIM* in a *Convolution Neural Network* and did test on CIFAR10 dataset. We have found that CNN with *COSAIM* could achieve similar accuracy compared to the baseline.

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] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks," *Synthesis Lectures on Computer Architecture*, vol. 15, no. 2, pp. 1–341, 2020.

[3] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, pp. 1737–1746, 2015.

[4] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," 2011.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[18] Z. Ebrahimi, S. Ullah, and A. Kumar, "Leap: Leading-one detection-based softcore approximate multipliers with tunable accuracy," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 605–610, IEEE, 2020.

[19] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2016.

[20] 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.

[21] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 4, pp. 1–34, 2017.

[22] I. LogiCORE, "Multiplier v12. 0 product guide," *PG108*, 2014.

[23] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.