A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks*

Hyeonuk Sim and Jongeun Lee
School of Electrical and Computer Engineering, UNIST, Ulsan, 44919 South Korea
{detective,jlee}@unist.ac.kr

ABSTRACT

Stochastic computing (SC) allows for extremely low cost and low power implementations of common arithmetic operations. However inherent random fluctuation error and long latency of SC lead to the degradation of *accuracy* and *energy efficiency* when applied to convolutional neural networks (CNNs). In this paper we address the two critical problems of SC-based CNNs, by proposing a novel SC multiply algorithm and its vector extension, SC-MVM (Matrix-Vector Multiplier), under which one SC multiply takes just a few cycles, generates much more accurate results, and can be realized with significantly less cost, as compared to the conventional SC method. Our experimental results using CNNs designed for MNIST and CIFAR-10 datasets demonstrate that not only is our SC-based CNN more accurate and 40X∼490X more energy-efficient in computation than the conventional SC-based ones, but ours can also achieve lower area-delay product and lower energy compared with *bitwidth-optimized* fixed-point implementations *of the same accuracy*.

1 INTRODUCTION

Stochastic computing (SC) provides an alternative, more error-resilient way of representing numbers for when device reliability is no longer guaranteed. At the same time SC versions of multiplication and addition operations, as well as other more complex ones, can be designed with extremely low cost and low power than their conventional binary versions. Consequently SC has been successfully applied to some applications including edge detection [2] and LDPC decoding [6], where exact computation is not required.

SC has also been applied to deep neural networks (DNNs) [3, 8, 17], demonstrating its unique advantages such as early decision termination [8], low cost, and high energy efficiency. However in order to achieve the highest energy efficiency, the previous work on SC-DNNs [3, 8, 17] assumes a fully-parallel architecture, which can only be designed when the target DNN is fully specified *and* the design fits the area budget. For large DNNs or when the target DNN is not known, one must use a more general architecture, which essentially executes an array of MAC (multiply-accumulate) operations repeatedly while simultaneously accessing on/off-chip memory for intermediate results.

However this kind of architecture—that includes not only computation but also a copious amount of data transfer to/from memory—causes an extremely high overhead if done in SC. This is because SC is as inefficient with storage as it is efficient with computing. And the fundamental reason for that is the exponentially longer SN (stochastic number) bitstreams than the equivalent BNs (binary numbers).[1] This exponential overhead of SC can be avoided by using BNs for memory access, which implies that BN-to-SN and SN-to-BN conversions must be added before/after every SC process. We call this style of computing *Binary-Interfaced Stochastic Computing (BISC)*, which was first introduced in [18]. The main challenge of scalable SC-DNN is how to maintain high energy efficiency for BISC despite the conversion overhead.

To minimize the large overhead of BISC, one may consider sharing conversion circuitry. However, sharing even a small part of the conversion circuit may affect the accuracy of SC significantly. This fundamental trade-off between accuracy and efficiency in SC is what needs to be improved in order to achieve better SC designs.

In this paper we propose a novel SC multiply algorithm for BISC, turn it into a vectorized form called *BISC-MVM (Matrix-Vector Multiplier)*, and show its applicability to deep convolutional neural networks (DCNNs), which is an important workload today. Compared with conventional SC, our SC multiply algorithm and BISC-MVM architecture improve *both* the efficiency and accuracy of SC considerably. Efficiency is improved by simplifying and restructuring the entire chain of computation from BN-to-SN conversion to SC process and to SN-to-BN conversion. Accuracy is enhanced over previous work (i) by our new SC multiply algorithm and (ii) through the use of our novel low-discrepancy SNG (Stochastic Number Generator) scheme. Our vectorized version, BISC-MVM, also minimizes the overhead of SNG without *any* degradation in accuracy, which contrasts with that of conventional SC.

We have implemented and evaluated our proposed BISC-MVM in Verilog RTL, and also evaluated the accuracy of our scheme with Caffe [7] using DCNNs designed for MNIST and CIFAR-10 datasets. Our experimental results demonstrate that for CNN acceleration, our new SC method can be 40X∼490X more energy-efficient in the compute array than the conventional SC while generating more accurate result, and can achieve lower area-delay product and lower energy compared with *bitwidth-optimized* fixed-point binary implementations *of the same accuracy*.

This paper makes the following contributions.

- A low-latency, low-cost, and high-accuracy SC multiplier for BISC with guaranteed error bound.
- A low-latency BISC-MVM and its application to DCNNs.
- Evaluation of the proposed BISC-MVM for DCNNs using MNIST and CIFAR-10, with comparisons with previous work.

2 OUR PROPOSED SC-MAC FOR BISC

2.1 Conventional SC Multiplication

Fig. 1(a) illustrates conventional SC multiplication. In SC, a number (called *stochastic number*, or *SN*) is represented by a bitstream, whose signal probability, or the frequency of 1, determines its value depending on the range, which is known *a priori*. Popular choices for the range include $[0, 1]$ called *unipolar* and $[-1, 1]$ called *bipolar*. An SNG (Stochastic Number Generator), which is synonymous to

*This work was supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-IT1501-08. *(Corresponding author: Jongeun Lee)*

DAC '17, Austin, TX, USA

978-1-4503-4927-7/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3061639.3062290

[1] In this paper we use the term "binary number" in the sense of a radix-2 number. A stochastic bitstream is radix-1, or a unary number.

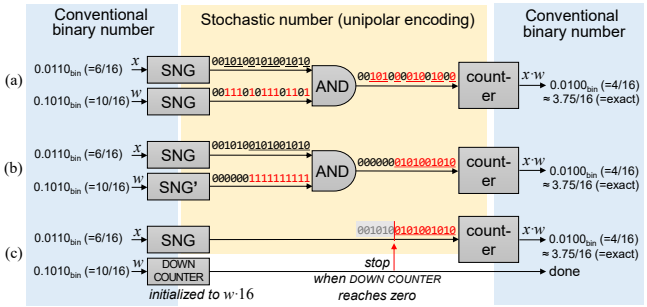


Figure 1: How our new SC multiply works. (a) Conventional multiplication of two SN bitstreams, (b) Reordering the bits for w, (c) Our low-latency multiplication for BISC.

BN-to-SN converter, takes an N-bit binary number (BN) and generates an SN bitstream, and it typically consists of a random number generator such as an N-bit LFSR (Linear Feedback Shift Register) and an N-bit comparator, which generates 1 if the random number is less than the input BN, and 0 otherwise [1]. An AND gate can perform multiplication for unipolar encoding if the input SN bitstreams are statistically uncorrelated with each other. An XNOR gate does the same for bipolar encoding. Finally a bit-counter converts a unipolar SN to a BN. An up-down counter does the same for bipolar.

2.2 Our Proposed SC-MAC

Suppose we reorder the bits for one input w such that all the 1s appear first as illustrated in Fig. 1(b). Certainly it does not affect the *value* of SN for w or that of the resulting SN after the AND operation, if the two SNs are still statistically uncorrelated. Therefore the BN outcome in (b) is expected to be the same as that of (a). Note that the order of SN bits for the other input x does not affect the outcome either as long as it is randomized.

Now since we know that all the zeros in the SN bitstream for w and the corresponding bits for x will have zero contribution to the final outcome, we can skip those bits altogether. This observation leads to the alternative method illustrated in (c), which connects an SNG directly to the bit-counter that is activated for $w \cdot 2^N$ cycles only.

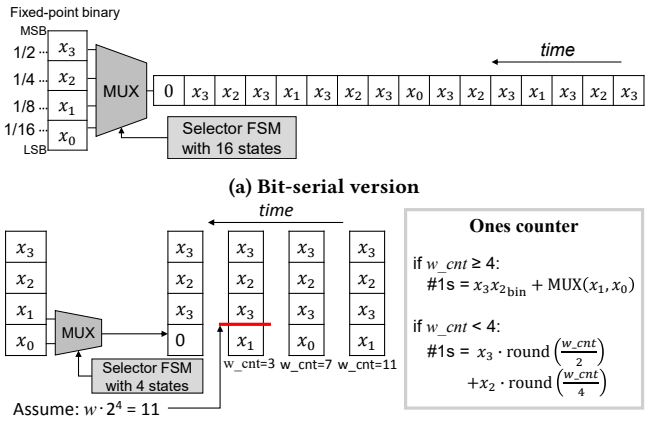
This new SC multiplication method works only for unipolar encoding and is relevant to BISC (Binary-Interfaced SC) only, but it has the following important advantages. First, its design is simpler as it eliminates an SNG and an AND gate in exchange for a down-counter, which is much less than an SNG. Second, it enables sharing of some circuitry in an array version without losing accuracy at all, as given in Section 3. Third, it has a shorter average latency, again without losing any accuracy, as compared to the conventional method.

Our SC multiplier illustrated in Fig. 1(c) can also be called *SC-MAC*, since the counter naturally accumulates results from consecutive multiplications. The counter only needs to have a wider width.

2.3 Enhancing Accuracy via Low-Discrepancy Code

The accuracy of the BN output in Fig. 1(c) depends on how uniformly the 1s are distributed in the SN bitstream, suggesting a good use of low-discrepancy code such as Halton sequences [2]. Though low-discrepancy code has already been used in SC [2, 9], in previous work low-discrepancy code is limited to improving the accuracy of SNG only, with no guarantee on the accuracy of an *SC operation's output*. However in our scheme the value of an SN *is* the outcome of SC; therefore, the use of low-discrepancy code can directly improve the accuracy of our SC multiplier.

In addition to the strong guarantee on the accuracy of SC multiplier itself, which we quantify empirically, our SC multiplication



(b) Bit-parallel version: 4-bit parallel processing example (unsigned)

Figure 2: Simple and accurate low-discrepancy code generation using FSM.

result depends only on the distribution of bits, not on the order at all. This allows us to use a simple and deterministic bit shuffling scheme via an N-bit FSM (Finite-State Machine) and one mux, which is in fact simpler than the conventional LFSR-comparator-based SNG and far simpler than a Halton sequence generator [2] (see Section 4.3.1).

Given an N-bit fractional number $w \in [0, 1)$, let $k = 2^N w$. The accuracy objective for our SC multiplication in Fig. 1(c) dictates that the partial sum, P_k, of the SN sequence $\{X_i\}$ for x must satisfy $P_k = \sum_{i=0}^{k-1} X_i \simeq xk$ for $\forall k$. Since x is an N-bit BN, $x_{N-1} \cdots x_0$, we have $x = \sum_{i=1}^{N} 2^{-i} x_{N-i}$, from which we can write the reference output, xk, as follows: $xk = k \sum_{i=1}^{N} 2^{-i} x_{N-i} = \sum_{i=1}^{N} k/2^i \cdot x_{N-i}$. A good approximation of this is $\sum_{i=1}^{N} \text{round}(k/2^i) x_{N-i}$.

We can design an FSM-MUX circuit such that the partial sum always equals this approximation, as illustrated in Fig. 2(a). The essence of the pattern generated by the FSM-MUX circuit is that x_{N-i} first appears at cycle 2^{i-1}, and thereafter in every 2^i cycles. Though omitted due to page limitation, it can be proved that with this pattern, the number of times x_{N-i} appears within the first k cycles equals round$(k/2^i)$. The theoretical maximum error of our SC multiply is $\sum_{i=1}^{N} 1/2 = N/2$ for xk, or $N/2^{N+1}$ for wx. But this error bound is not tight; instead, we show maximum error empirically in Section 4.1.

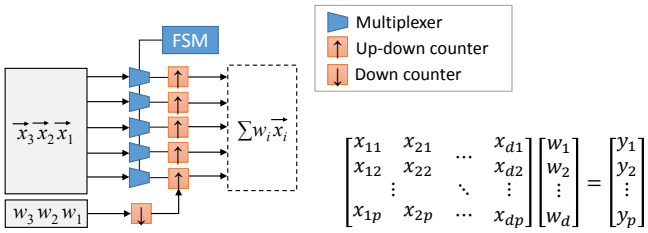
2.4 Extension to Support Signed Multiplication

Our scheme can be extended to support signed multiplications, where both x and w as well as the output are represented in two's complement. The only major change is that the bitstream counter now becomes an up-down counter, counting up for input '1' and down for '0'. The other changes are minor. The sign bit of input x is flipped and XOR-ed with the sign bit of the other input w after being converted to sign-magnitude representation, and the magnitude part is fed to the down counter as before. The FSM-based bitstream generator can be used without modification.

To see how it works, let us consider the values of x and w listed in Table 1. In this example, N, the number of bits of each operand including the sign bit, which we call *multiplier precision*, is 4. Thus the examples are for the max/min values of w. Our SC multiplier generates an N-bit two's complement number as output. Column 3 is the binary representation of x. After sign bit flipping, the MUX out is XOR-ed with the sign bit of w, which is fed to the up-down counter, whose value is read at cycle $|2^{N-1} w|$ as the result of multiplication. When compared with the true multiplication result with sufficient precision in the last column, one can see that they are very close.

Table 1: Signed multiplication example

2^3w	2^3x	Binary	Sign-flipped	MUX out	Counter	Ref. (2^3wx)
	0	0000	1000	10101010	0	0
−8	7	0111	1111	11111111	−8	−7
	−8	1000	0000	00000000	8	8
	0	0000	1000	1010101	1	0
7	−7	0111	1111	1111111	7	6.125
	−8	1000	0000	0000000	−7	−7



(a) (b)

Figure 3: (a) Our SC matrix-vector multiplier (simplified) and (b) the operation it performs, where $y_j = \sum_i w_i x_{ij}$.

2.5 Bit-parallel Processing Optimization

To further reduce the latency of our SC multiplier, we propose bit-parallel processing. Let us consider the example in Fig. 2(b), where the degree of bit-parallelism, b, is 4, meaning that we process this bitstream in 4 cycles instead of 16. We first rearrange the 2^N-bit sequence into a b-row, $2^N/b$-column matrix, and process each column in one cycle. Let w be the other operand (i.e., multiplier) of this multiplication. If $w \geq b$, we only need to know how many ones are included in the current column. Otherwise we need to count the number of ones in the top w bits. And we repeat this for the next column after decrementing w by b.

Counting the number of ones (i) in a column and (ii) in a sub-column, can be done by the formulas in the inset (called *ones counter*). To understand why, first notice that half the bits are x_3, and half the remaining ones are x_2. Thus for (i), the only variation is in the last row, which we can easily provide using a small FSM with $2^N/b$ states. For (ii), we need to multiply w to the number of ones in the column, which we do using the approximation formula we derived in Section 2.3. Thus our bit-parallel computation result is exactly the same as our bit-serial result.

Increasing bit-parallelism can reduce multiplier latency at the cost of hardware overhead. Therefore the degree of bit-parallelism needs to be chosen carefully.

3 OUR PROPOSED BISC-MVM AND SC-CNN ACCELERATOR

3.1 BISC-MVM: Vectorization of Our SC-MAC

Fig. 3(a) illustrates our BISC-MVM, which contains p parallel SC-MACs of N-bit multiplier precision. Each SC-MAC requires a mux and an up-down counter, whose width is $N+A$ bits (A additional bits are for accumulation). All muxes share the same control input, hence the same FSM. The down counter can be shared as well if the other operand, w, is common to all, as is the case with our BISC-MVM. This SC multiplier array can perform one scalar-vector multiplication, $w\vec{x}$, in $|2^{N-1}w|$ cycles. Moreover it can be used to calculate accumulation, $\sum_{i=1}^{d} w_i\vec{x}_i$, simply by feeding a sequence of $\vec{x}_i$ and w_i; no additional hardware is necessary. Then the accumulation result can be read from the array of up-down counters at cycle $\sum_{i=1}^{d} |2^{N-1}w_i|$.

Mathematically this is a matrix-vector multiplication in the form of Fig. 3(b). Our BISC-MVM has the following features.

- All SC multipliers share both the down counter and the FSM, but this causes no accuracy degradation, which is quite contrary to conventional SC.

```
for (m1 = 0; m1 < M; m1+=TM)
 for (r1 = 0; r1 < R; r1+=TR)
  for (c1 = 0; c1 < C; c1+=TC)
   for (z = 0; z < Z; z++)
    for (i = 0; i < K; i++)
     for (j = 0; j < K; j++)
      for (m = m1; m < min(M, m1 + TM); m++)
       for (r = r1; r < min(R, r1 + TR); r++)
        for (c = c1; c < min(C, c1 + TC); c++)
         B[m][r][c] += W[m][z][i][j] × A[z][Sr + i][Sc + j];
```

Figure 4: Convolution layer tiled along three loop levels. Arrays A, B, and W are input feature map, output feature map, and weight parameters, respectively, and S is stride.

- By sharing w, all SC multiplications finish simultaneously, which enables our BISC-MVM to retain the latency reduction feature of our single SC multiplier.

The high accuracy of our BISC-MVM can be attributed to the following. First our SC multiplier itself is highly accurate. Second, accumulation does not introduce any error, given that the up-down counter is wide enough. Third, sharing the FSM and the down counter does not introduce any error. At the same time, sharing certain resources makes our BISC-MVM more cost-efficient than a set of SC multipliers.

One potential downside of our BISC-MVM is that the particular matrix-vector multiplication in the form of Fig. 3(b) may not be how a neural-net layer is typically described mathematically. We next discuss how this BISC-MVM can be applied to accelerate DCNNs.

3.2 Applicability to DCNN

The computation in convolution layers is typically represented as a 6-deep nested loop of MAC operations. There are different ways to design an accelerator for the loop nest, but recent work [15] suggests that superior performance can be achieved by accelerating along three dimensions including the output feature map (M), the output width (C), and the output height (R). This is equivalent to tiling the loop as shown in Fig. 4, where the 3 innermost loops are executed by a hardware accelerator as fully unrolled (i.e., simultaneously).

This accelerator requires $T_M T_R T_C$ number of MAC units, out of which every $T_R T_C$ MACs use the same weight parameter $W[m][z][i][j]$, which does not depend on either r or c. Thus our BISC-MVM is well-suited for this kind of architecture, and can be configured as $p = T_R T_C$ and $d = K^2 Z$, generating p output feature map values in every t cycles, where

$$t = \sum_{z=0}^{Z-1} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \left| 2^{N-1} W[m][z][i][j] \right|.$$

The actual latency reduction as compared to conventional SC (which requires 2^N cycles for each multiplication) depends on the value of weight parameters. But it is a well-known fact that weight parameter values in a typical neural network layer including convolution layers are distributed in a bell-shaped form centered around zero, in which the average (of absolutes) is far less than the maximum. This leads to significant latency reduction as demonstrated in our experiments, which reinforces the suitability of our BISC-MVM for DCNN acceleration.

3.3 Our SC-CNN Accelerator Architecture

Our SC-CNN accelerator architecture is by design very similar to previous CNN accelerators [15, 19] based on conventional binary. In fact, there should be no difference in the top-level architecture from that of [15] in particular, since we use the same parallelization

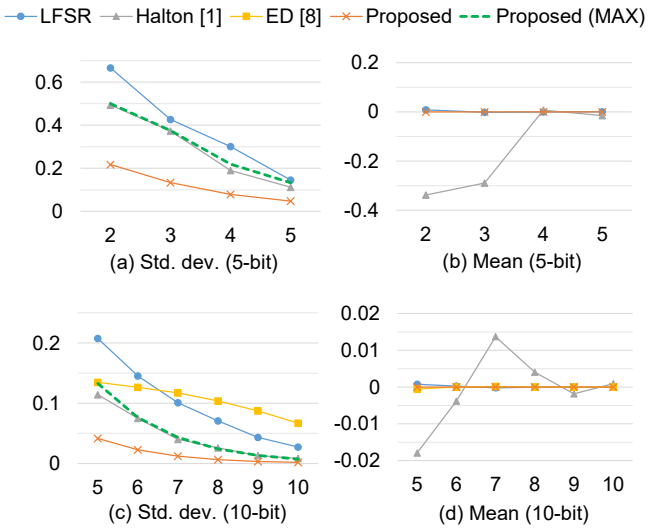


Figure 5: Error statistics of various SC multipliers.

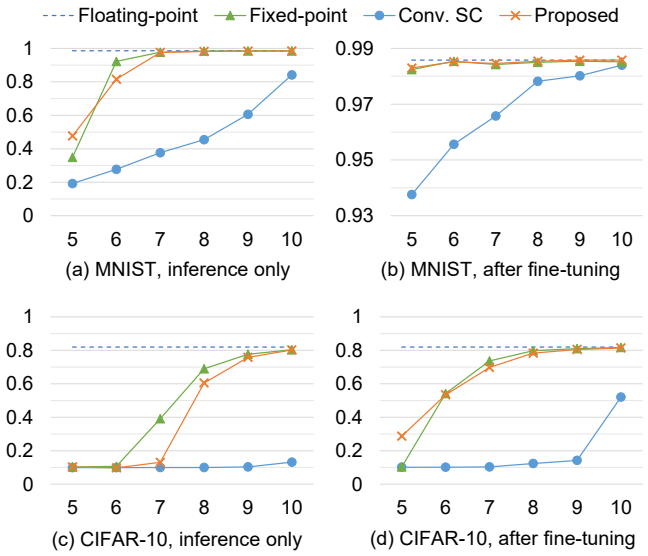


Figure 6: MNIST, CIFAR-10 recognition accuracy (the first 5,000 test images), where x-axis is multiplier precision including the sign bit.

scheme for the nested loop. Even the on-chip memory sizes for input/output/weight buffers are exactly the same, which should make our comparison with binary implementation more credible. As in the previous work [18] we apply SC to convolution layers only, which account for 90~99% of computation of a DCNN, with no restriction on how the other layers are implemented.

4 EXPERIMENTS

We compare our proposed SC with **conventional SC** and **fixed-point binary** (short-handed as binary), in the context of a CNN accelerator, where most computation occurs in the MAC array.

4.1 Accuracy Analysis of Our Proposed SC-Multiply Algorithm

The conventional SC has different flavors depending on the SNG: (1) LFSR and comparator, (2) Halton [2], and (3) even-distribution-based low-discrepancy code abbreviated as ED [9]. To evaluate accuracy we simulate various SC multiply algorithms in software, testing for all input combinations from 5- and 10-bit fixed-point binary numbers. Fig. 5 shows error statistics, where error is defined as the difference from the fixed-point multiplication result without rounding (thus having twice the precision). The graphs show the running statistics of error at cycle 2^x, where x is the x-coordinate value.[2] Thus it shows not only the statistics at the end of the bitstream, but also how fast the output converges. ED [9] is applied to the 10-bit case only, since it generates 32 bits per cycle. Note that our bit-parallel version and vector version (i.e., BISC-MVM) generate the same output as our SC multiplier, only faster.

The graphs suggest that among the conventional SC methods the Halton method is the most accurate and converges fast. However, ours has much less error, about 1/3 of Halton, at all times. In addition the figure shows the *maximum absolute error* of our proposed scheme, which can be calculated easily because our scheme does not rely on LFSR. Interestingly the max error of our scheme roughly coincides with the standard deviation of error of Halton, which unequivocally shows the high accuracy of our scheme. Finally the *mean* graphs confirm that ours is zero-biased.[3]

[2]For our proposed method, at cycle $|w|/2^{5-x}$ or $|w|/2^{10-x}$.
[3]The result for the Halton case depends on the prime number base. We use 2 and 3 for x and w, respectively.

4.2 Recognition Accuracy of Our Proposed SC-CNN

To evaluate the recognition performance of our SC-based CNNs we use the Caffe framework [7], in which the convolution layer is extended for fixed-point and SC. We use two CNNs designed for MNIST and CIFAR-10 datasets, comparing three cases: (1) fixed-point binary, (2) conventional SC based on LFSR, and (3) our proposed SC. We use the network definitions and training parameters included in the Caffe distribution. For the CIFAR-10 net we scale the input feature map before/after convolution by 128 so that the values mostly come in the $[-1, 1]$ range. We vary multiplier precision (N) from 5 to 10, with 2 additional bits for accumulation ($A = 2$). We use a saturating accumulator/up-down counter. For the binary case the multiplication result is truncated before accumulation.

The upper graphs in Fig. 6 show the test accuracy of the MNIST net. The left one is the accuracy when using the weight parameters obtained from the training of the original floating-point net. The right one is after fine-tuning for 5,000 iterations (with the same learning rate) atop the original training, which runs for 10,000 iterations. During fine-tuning, fixed-point or SC-based convolution is used in the forward pass.

The graphs reveal many interesting points. First, fixed-point binary shows very good recognition performance, and 5- or 7-bit precision seems to be enough for MNIST depending on whether fine-tuning is done. Second, without fine-tuning conventional LFSR-based SC can have much lower accuracy, though fine-tuning can recover most of the accuracy loss. Third, under the same precision setting, our SC-CNN achieves almost the same accuracy as the fixed-point binary. While this is significant, MNIST is relatively easy and a similar result is achieved in previous work as well [8].

We run a similar experiment with CIFAR-10, the result of which is summarized in Fig. 6(c)-(d). Again we use the same precision setting across different methods. For the fixed-point case, achieving the floating-point recognition rate requires 9- to 10-bit precision without fine-tuning or 8- to 9-bit with fine-tuning. On the other hand, conventional LFSR-based SC shows a very poor performance even with fine-tuning, whereas our proposed SC shows almost the same performance as binary, especially with fine-tuning. Specifically, the fact that our SC-CNN can achieve near-fixed-point performance

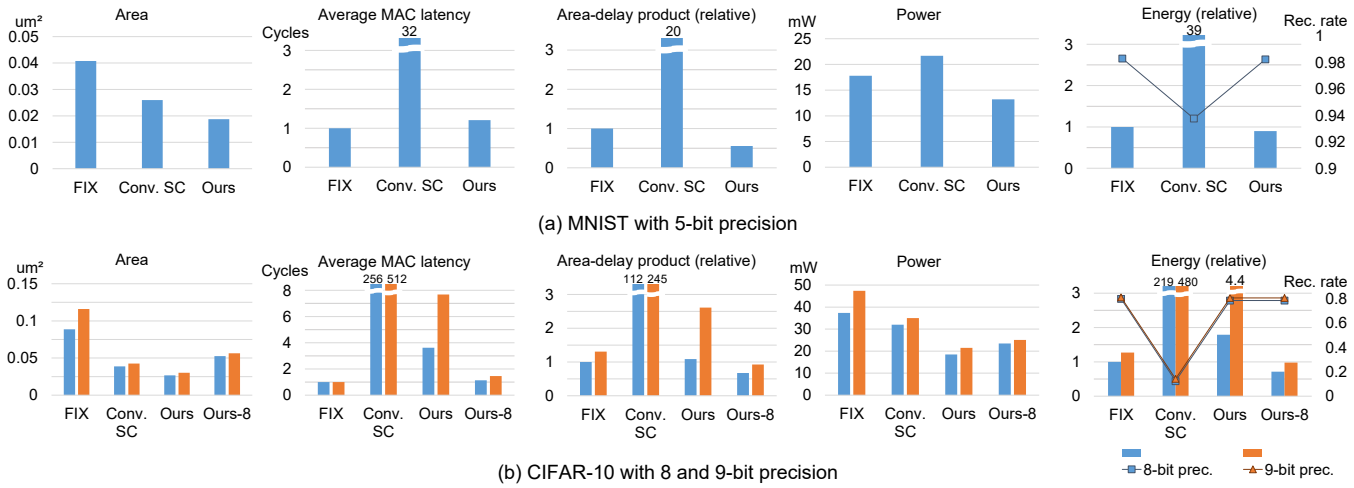


Figure 7: Comparison of MAC arrays: fixed-point binary version ("FIX"), LFSR-based conventional SC version ("Conv. SC"), and our proposed BISC-MVM (*Ours* and *Ours-8* are our bit-serial and 8-bit-parallel versions, respectively).

even without fine-tuning at 9~10 bits underscores the high accuracy of our BISC-MVM. Eventually, however, without retraining the small error of our SC multiplier creates a performance gap at 7~8 bits; nontheless, it is successfully filled via retraining, making our SC-CNN virtually indistinguishable from the fixed-point version in terms of accuracy.

4.3 Implementation Efficiency of Our Proposed SC-CNN

To evaluate implementation efficiency we have designed MAC arrays in Verilog RTL based on our proposed (i) BISC-MVM, (ii) LFSR-based conventional SC, and (iii) fixed-point binary, and synthesized them with Synopsys Design Compiler vD-2010.03 using TSMC 45nm technology. The three cases are designed to use the common setting as much as possible, including size (256 MACs), input/output data representation (two's complement), and multiplier/accumulator precisions. Specifically, multiplier precision, N, is set to 5 bits for MNIST, and varied to 8~9 bits for CIFAR-10. The accumulator is saturating and A is 2 bits as before. All three cases are synthesized for the same clock frequency of 1GHz.

SNG sharing is enabled for the SC cases. In particular in the conventional SC case, the SNG for the weight parameter is shared across all SC-MACs within the MAC array. Similarly for our BISC-MVM, a FSM and a down counter are shared across all SC-MACs.

Results are summarized in Fig. 7, where we include the bit-parallel processing option for the CIFAR-10 experiment, with the parallelism of 8 bits.

4.3.1 Area-Delay Product. As expected, SC designs require smaller area than the binary, with our proposed scheme (in particular, the bit-serial version) being the smallest. Also the area difference between SC and binary is larger when the precision is higher, which is due to the quadratic relationship between precision and binary multiplier complexity. What may be surprising is that the area difference is not nearly as high as the latency difference between SC and binary. This is because of the large conversion overhead from BN to SN and back, as confirmed by area breakdown in Table 2.

Table 2 shows detailed area breakdown of a single MAC for two multiplier precision (MP) settings: 5 bits and 9 bits. We add a number of other designs not included in the CNN-level comparison, but it is important to note that these numbers are area only, and latency must be taken into account when comparing different designs. For instance, the ED case [9], which is evaluated for the 9-bit precision setting only, uses a bit-parallel SNG that generates 32 bits per cycle, requiring

Table 2: Area breakdown of a MAC (synthesis result)

MP	Case	Design	SNG		Mult./ XNOR[a]	Par. CNT/ 1s CNT	Accum./ UD CNT	Total
			Reg/FSM	Combi.				
5	Binary	Fixed-point	–		88.9	–	66.3	155.2
	Conv. SC	LFSR	51.5	19.1	1.8	–	64.9	137.2
		Halton [2]	87.7	18.3	1.8	–	64.9	172.7
	Proposed	Bit-serial	31.2	6.0	38.8	–	66.7	142.7
9	Binary	Fixed-point	–		305.0	–	110.1	415.1
	Conv. SC	LFSR	89.6	37.0	1.8	–	104.4	232.8
		Halton [2]	203.7	33.9	1.8	–	108.0	347.3
		ED [9]	346.8	226.3	57.9	136.0	124.9	891.9
	Proposed	Bit-serial	60.9	11.8	80.6	–	103.4	256.7
		8b-par.	38.6	–[b]	78.7	108.5	111.1	336.9
		16b-par.	37.7	–[b]	80.6	174.1	112.2	404.7
		32b-par.	23.8	–[b]	76.9	239.4	107.4	447.5

Note: a. This is the down counter for our proposed method.

b. This is not zero, but includes a small mux, whose area is included in the ones counter (1s CNT).

32X as many XNOR gates and a parallel counter (Column 7) while simultaneously reducing latency by 32X. Similarly our proposed SC-MAC, even for the bit-serial version, has a very low latency compared to the conventional SC (see Fig. 7).

From the table we can make the following observations. First, ED is quite cost-efficient. In fact it has the lowest area-delay-product (ADP) among the conventional SC methods. However ED has also the lowest quality in terms of multiplication accuracy (see Fig. 5(c)). Second, Halton has a very good accuracy but also the highest area per throughput. Third, while in the previous work accuracy and ADP are only a trade-off, ours improves both of them simultaneously. The average delay of ours is data-dependent, but very small as shown in Fig. 7. In particular the bit-serial version has a latency of up to 7.7 cycles for CIFAR-10, but it is effectively suppressed by the bit-parallel version. Fourth, in the 9-bit precision setting, increasing the bit-parallelism for our proposed SC-MAC increases the total area, only modestly. However the 8-bit parallelism already achieves very low average latency and thus has the lowest ADP (not shown in the graph). Finally unlike the binary case, our proposed scheme becomes more cost-efficient when vectorized due to the sharing of the FSM and down counter. This helps explain the larger gap between the binary and our proposed designs (and between Ours-8 and Ours) in Fig. 7 than in Table 2.

In summary our proposed BISC-MVM can achieve 29~44% lower ADP even when compared with the fixed-point binary design *of the same accuracy*, thanks to the very low average MAC latency of our scheme.

4.3.2 Power and Energy Efficiency. Since we use the same clock frequency for all designs, power dissipation as reported by the synthesis tool is largely proportional to the area result, with one exception.

Table 3: Comparison with previous neural network accelerators

		Frequency	Area*	Power*	GOPS	GOPS/mm^2	GOPS/W	Tech.	Scope for area & power
Binary	MWSCAS'12 [14]	400 MHz	12.50	570.00	160.00	12.80	280.70	45nm	Total chip
	ISSCC'15 [13]	200 MHz	10.00	213.10	411.30	41.13	1930.08	65nm	Total chip
	ASPLOS'14 [5]	980 MHz	0.85	132.00	501.96	592.94	3802.73	65nm	NFU** only
	GLSVLSI'15 [4]	700 MHz	0.98	236.59	274.00	278.85	1158.11	65nm	SoP (≃ MAC) units only
SC	ArXiv'15 [3]	400 MHz	0.09	14.90	1.01	11.91	67.93	65nm	One neuron
	DAC'16 [8]	1000 MHz	0.06	3.60	75.74	1262.33	21038.79	45nm	One neuron with 200 inputs
	Proposed (9b-precision)	1000 MHz	0.06	25.06	351.55	6242.37	14029.72	45nm	MAC array (size: 256)

*Note 1: See the rightmost column for the scope of area (in mm^2) and power (in mW).
**Note 2: NFU can also perform pooling and activation functions.

We found that LFSRs have unusually high power dissipation per area, negatively impacting power efficiency of the conventional SC case. As a result the conventional SC case turns out to be about as high power-dissipating as the binary case even before considering its high toll on latency. Of course, this weakness of conventional SC is mostly due to the conversion overhead between SN and BN, and ultimately because we are targeting BISC, and things could be very different if we exclude such overheads as would be more relevant for a fully-parallel architecture.

Our new proposed SC-CNN has the lowest power consumption and extremely low latency, which make it about 40X (for MNIST) and 300X~490X (for CIFAR-10) more energy-efficient in the MAC array than the conventional SC while at the same time being more accurate. Our proposed solution is also slightly more energy-efficient (23~29% for CIFAR-10 and 10% for MNIST) than the fixed-point binary while having nearly the same accuracy. Note that this comparison is without considering the inherent advantages of SC such as dynamic energy-quality tradeoff and better error tolerance. For future technologies in which variability and noise are expected to grow, the advantages of SC may be greater.

4.3.3 Comparison with Previous DNN Accelerators. Table 3 provides a brief comparison with previous neural-net accelerators. Due to differences in many aspects including target neural networks, we compare performance in GOPS, with 1 MAC counted as 2 operations. SC's (long) latency is taken into account when computing GOPS. SNGs are included for area and power calculation except for ArXiv'15. Note also that the first two cases (MWSCAS'12 and ISSCC'15) are not directly comparable with the rest, since they include large on-chip buffers, which should dominate area and power.

Compared to a recent SC design [8], ours has much higher area efficiency but dissipates more power. However the previous work is a fully-parallel architecture, which is one reason for its supreme energy efficiency. Instead, ours has scalability, which cannot be provided by the previous work. Compared to the others, our architecture is more energy-efficient in addition to having the highest area-efficiency.

There are other recent SC-based DNNs [11, 12], which however do not provide area/power numbers (focusing on accuracy or targeting FPGAs), as well as DNNs based on concepts similar to SC [16]. In particular XNOR-Net [16] shows that through clever learning tricks, the same recognition accuracy as that of a floating-point network can be achieved using XNOR computation only even for AlexNet [10]. Such training methods are orthogonal to our contributions, and can bolster the case for SC-CNNs in general.

5 CONCLUSION

We presented a highly accurate, low-latency, and cost-efficient SC multiply algorithm and its vector version, BISC-MVM, for binary-interfaced SC, where input/output must be represented in a conventional binary format. Binary-interfaced SC is not only essential for scalable architectures but also enables us to easily compare SC-based CNN accelerators with conventional binary ones on an equal footing. Our work is distinguished from previous work on SC-CNN in important ways. First our SC-CNN architecture targets BISC, and thus is more flexible in terms of post-fabrication changes in hyper-parameters of a CNN. Second we show that, for the first time, SC-CNNs can have essentially the same recognition accuracy as fixed-point implementations even for not-so-easy benchmarks such as CIFAR-10, while at the same time consuming less energy. Third while our aggressive optimization targeting BISC has blurred the boundary between SNG and SC process in our SC-MAC, unary encoding is the key to enabling its very efficient operation (viz., low average latency). On the other hand, our variable-latency MAC operation may make memory subsystem more difficult to implement. Also included in the future work is the evaluation of our SC-CNN for larger-scale benchmarks and for error resilience.

REFERENCES

[1] Armin Alaghi and John P. Hayes. 2013. Survey of Stochastic Computing. *ACM Trans. Embed. Comput. Syst.* 12, 2s, Article 92 (May 2013), 19 pages.
[2] Armin Alaghi and John P. Hayes. 2014. Fast and Accurate Computation Using Stochastic Circuits. In *DATE '14*. Article 76, 4 pages.
[3] Arash Ardakani, François Leduc-Primeau, Naoya Onizawa, Takahiro Hanyu, and Warren J. Gross. 2015. VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing. *CoRR* (2015).
[4] Lukas Cavigelli, David Gschwend, Christoph Mayer, Samuel Willi, Beat Muheim, and Luca Benini. 2015. Origami: A Convolutional Network Accelerator. In *GLSVLSI '15*. ACM, 199–204.
[5] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning. In *ASPLOS '14*. 269–284.
[6] Warren J Gross, V Gaudet, and Aaron Milner. 2005. Stochastic implementation of LDPC decoders. In *ASILOMAR '05*, Vol. 28. 713–717.
[7] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).
[8] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. 2016. Dynamic Energy-accuracy Trade-off Using Stochastic Computing in Deep Neural Networks. In *DAC '16*. ACM, Article 124, 6 pages.
[9] K. Kim, J. Lee, and K. Choi. 2016. An energy-efficient random number generator for stochastic circuits. In *ASP-DAC '16*. 256–261.
[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS '12*. 1097–1105.
[11] Bingzhe Li, M. Hassan Najafi, and David J. Lilja. 2016. Using Stochastic Computing to Reduce the Hardware Requirements for a Restricted Boltzmann Machine Classifier. In *FPGA '16*. ACM, 36–41.
[12] Zhe Li, Ao Ren, Ji Li, Qinru Qiu, Yanzhi Wang, and Bo Yuan. 2016. DSCNN: Hardware-Oriented Optimization for Stochastic Computing Based Deep Convolutional Neural Networks. In *ICCD '16*.
[13] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H. J. Yoo. 2015. A 1.93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications. In *ISSCC '15*. 1–3.
[14] P. H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello. 2012. NeuFlow: Dataflow vision processing system-on-a-chip. In *MWSCAS '12*.
[15] A. Rahman, J. Lee, and K. Choi. 2016. Efficient FPGA acceleration of Convolutional Neural Networks using logical-3D compute array. In *DATE '16*.
[16] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *CoRR* (2016).
[17] K. Sanni, G. Garreau, J.L. Molin, and A.G. Andreou. 2015. FPGA implementation of a Deep Belief Network architecture for character recognition using stochastic computation. In *CISS '15*. 1–5.
[18] H. Sim, D. Nguyen, J. Lee, and K. Choi. 2017. Scalable stochastic-computing accelerator for convolutional neural networks. In *ASP-DAC '17*. 696–701.
[19] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *FPGA '15*. 161–170.