



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

Doctoral Dissertation

Low-Cost Deep Convolutional Neural Network
Acceleration with Stochastic Computing and
Quantization

Hyeonuk Sim

Department of Computer Science and Engineering

Ulsan National Institute of Science and Technology

2021

Low-Cost Deep Convolutional Neural Network Acceleration with Stochastic Computing and Quantization

Hyeonuk Sim

Department of Computer Science and Engineering

Ulsan National Institute of Science and Technology

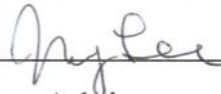
Low-Cost Deep Convolutional Neural Network Acceleration with Stochastic Computing and Quantization

A dissertation submitted to
Ulsan National Institute of Science and Technology
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Hyeonuk Sim

12/07/2020 of submission

Approved by



Advisor

Jongeun Lee

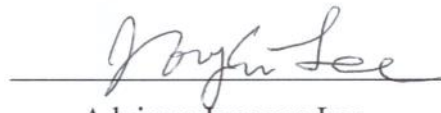
Low-Cost Deep Convolutional Neural Network Acceleration with Stochastic Computing and Quantization

Hyeonuk Sim

This certifies that the dissertation of Hyeonuk Sim is approved.

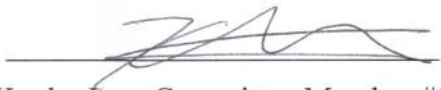
12/07/2020 of submission

Signature



Advisor: Jongeun Lee

Signature



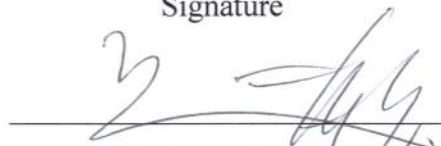
Kyuho Lee: Committee Member #1

Signature



Myeongjae Jeon: Committee Member #2

Signature



Sam H. Noh: Committee Member #3

Signature



Woongki Baek: Committee Member #4;

Abstract

For about a decade, image classification performance led by deep convolutional neural networks (DCNNs) has achieved dramatic advancement. However, its excessive computational complexity requires much hardware cost and energy. Accelerators which consist of a many-core neural processing unit are appearing to compute DCNNs energy efficiently than conventional processors (e.g., CPUs and GPUs). However, a huge amount of general-purpose precision computations is still tough for mobile and edge devices. Therefore, there have been many researches to simplify DCNN computations, especially multiply-accumulate (MAC) operations that account for most processing time.

Apart from conventional binary computing and as a promising alternative, stochastic computing (SC) was studied steadily for low-cost arithmetic operations. However, previous SC-DCNN approaches have critical limitations such as lack of scalability and accuracy loss. This dissertation first offers solutions to overcome those problems. Furthermore, SC has additional advantages over binary computing such as error tolerance. Those strengths are exploited and assessed in the dissertation.

Meanwhile, quantization which replaces high precision dataflow by low-bit representation and arithmetic operations becomes popular for reduction of DCNN model size and computation cost. Currently, low-bit fixed-point representation is popularly used. The dissertation argues that SC and quantization are mutually beneficial. In other words, efficiency of SC-DCNN can be improved by usual quantization as the conventional binary computing does and a flexible SC feature can exploit quantization more effectively than the binary computing. Besides, more advanced quantization methods are emerging. In accordance with those, novel SC-MAC structures are devised to attain the benefits.

For each contribution, RTL implemented SC accelerators are evaluated and compared with conventional binary implementations. Also, a small FPGA prototype demonstrates the viability of SC-DCNN. In a rapidly changing and developing deep learning world headed by conventional binary computing, multifariously enhanced SC, though not as popular as binary, is still competitive implementation with its own benefits.

Contents

I. Introduction	7
II. Background and Related Work	10
2.1. Deep Convolutional Neural Network Accelerators	10
2.2. Stochastic Computing	11
2.3. Dynamic Precision and Logarithmic Quantization	12
III. Scalable, Accurate, and Efficient SC-DCNNs	14
3.1. Binary-Interfaced Stochastic Computing	14
3.2. New Stochastic Computing Multiplication Algorithm	17
3.3. Evaluation	23
IV. Stochastic Computing in Synergy with Quantization	36
4.1. Dynamic Precision Scaling	36
4.2. Log-Quantized Stochastic Computing	42
4.3. Successive Log-Quantization and Its Application to SC	46
4.4. Evaluation	53
V. Conclusion	71
References	72

List of Figures

1. A generic accelerator for convolution layers. -----	10
2. SC multiplication example. -----	11
3. Signed and bit-parallel SC multiplication example. -----	11
4. Structure of a MAC unit. -----	16
5. How new SC multiplication works. -----	17
6. Simple and accurate low-discrepancy code generation using FSM. -----	19
7. Our SC-MVM and operation. -----	21
8. Convolution layer tiled along three loop levels. -----	22
9. Area breakdown of a MAC. -----	24
10. Recognition accuracy vs. stochastic bitstream length. -----	26
11. Area-delay product comparison (SBL: 128-bit). -----	27
12. Error statistics of various SC multipliers. -----	28
13. MNIST, CIFAR-10 recognition accuracy. -----	29
14. Comparison of MAC arrays. -----	31
15. Block diagram of MVM-based accelerator implementation on VC707. ---	34
16. DPS SC-MAC. -----	37
17. Supporting HRS mode for x along with the normal mode. -----	39
18. Dynamic precision can give huge boost in efficiency for SC. -----	41
19. Proposed log-quantized SC-MAC. -----	42
20. Logic implementation comparison of 1s counter. -----	43
21. Proposed log-quantized SC-MVM. -----	44
22. Multiplication error against floating-point weights. -----	45
23. Quantization error comparison. -----	47
24. New SC-MAC. -----	48
25. RMS error vs. L . -----	50
26. Encoding schemes. -----	50
27. Conventional-binary MAC with SLQ. -----	51
28. SC-MVM using SLQ for weight parameters. -----	52
29. Area overhead of DPS-MVM. -----	54
30. ADP vs. software precision. -----	55
31. ADP vs. hardware precision. -----	56
32. Comparison with the digital baseline and Stripe [35]. -----	57

33. Comparison with previous SC-DCNN on AlexNet. -----	58
34. Fault tolerance comparison among different schemes, for AlexNet. -----	59
35. Area and power breakdown of MVMs. -----	61
36. Recognition test on target DCNNs fine-tuned for 5,000 iterations. -----	62
37. Area and latency Pareto-optimal points of MVMs. -----	64
38. RMSE vs. effective resolution. -----	65
39. Area breakdown (um ²). -----	68
40. Latency and energy comparison of SC-DCNNs. -----	68

List of Tables

1. Logarithmic data representation example. -----	13
2. Signed multiplication example. -----	20
3. Compute tile synthesis results. -----	25
4. Area breakdown of a MAC (synthesis result). -----	31
5. Comparison with previous neural network accelerators. -----	33
6. Comparison of various MVM implementations on FPGA. -----	35
7. Comparison of linear vs. logarithmic quantization. -----	46
8. Quantization examples. -----	49
9. Recognition accuracy (for 10K images) and DPS precision setting. -----	57
10. Geo-mean latency and energy consumption comparison. -----	63
11. SC-DCNN classification accuracy (top-5, unit: %). -----	66
12. Model size comparison (convolution layers only). -----	67
13. SLQ on SC vs. conventional binary DCNN. -----	69

I. Introduction

Deep convolutional neural networks (DCNNs) [1]-[4] are showing dominant image classification performance. However, those abilities are not free. In fact, computation costs are expensive. DCNNs contain dozens of layers which consist of millions of neurons and weights. The complexity of appearing networks is increasing fast even now. In energy constrained system, such as a mobile phone, the problem becomes more significant while the demand for the deep learning on edge devices rises. That is why the necessary of new computing paradigms, for instance, approximate computing, is being magnified.

Stochastic computing (SC) is also well known as a low-cost computing paradigm. Stochastic number is generated from stochastic number generator (SNG) and represented as a bitstream, whose positive signals (1s in digital implementation) occur with the frequency of its value. Unlike multi-bit binary numbers that is without time domain, the bitstream can be carried on a single wire over a period of time, which is reminiscent of analog computing, yet at the same time, SC circuits can be entirely made out of digital components. The bitstream is also called signal probability. The placement of 1s is assumed as random and there are no positional weights among bits, that is, all positions have the equal weight. This property gives SC circuits unique strengths in terms of (i) better error resilience. Because the number is represented as frequency of 1s, the longer bitstream can produce the more precise number. It enables (ii) free, dynamic trade-off between performance, accuracy, and energy. The bitstream itself may be arranged in series, in parallel, or in some combination of the two. The most popular advantage of SC is (iii) simple multiplication which consumes much costs in conventional binary computing, where the majority of computations are multiply-accumulate (MAC) operations in DCNNs. In SC, an AND gate simply outputs the bitstream which has the probability that equals to the multiplication of two input streams, in contrast to a complex binary multiplier. The outcome might not be exact but DCNNs tolerate some level of computation error. There are weaknesses too. SC inherently contains computation error coming from the probability. When high precision or accuracy is required, processing cycles can be increased without hardware modification, that is an advantage, but it ultimately leads to long latency. When it comes to precision, an n -bit fixed-point number requires 2^n cycles to be represented as a stochastic number.

SC is seen as a promising approach to accelerating certain applications including deep neural networks [5]-[9] where exact computation is not required. Previous work on SC-based neural networks has shown the viability of SC in the context of neural networks, and proposed ideas on overcoming certain limitations of SC [7]-[9]. However previous work has some critical limitations, such as the fully-parallel architecture assumption and using fully-connected networks only. This prevents the previous solutions from being applicable to recent DCNNs. There are at least three problems in applying previous SC-based neural network solutions to DCNNs. First, DCNNs typically consist of many layers, with

GoogLeNet [4] having more than a hundred layers. It is virtually impossible to implement all the layers in parallel using dedicated hardware on a single chip even in SC. Second, DCNNs have many layer types whose SC versions are unknown. Examples include max-pooling layer (due to the difficulty of doing comparison in SC), ReLU (Rectified Linear Unit) layer, and local response normalization (LRN) layer. Third, even if one solves the first two problems, it remains uncertain whether the accuracy will be high enough. No one has shown that SC-DCNNs can achieve even similar recognition performance as conventional binary designs. This is a very critical issue, as low-performing neural network has very little value. As a first step towards evaluating the feasibility of SC for DCNNs, the dissertation considers a hybrid design where only part of the architecture is in SC. Clearly this marks an important departure from earlier work on SC-based neural networks to support DCNNs at scale. For large DCNNs or when the target DCNN is not known, one must use a more general architecture, which essentially executes an array of MAC operations repeatedly while simultaneously accessing on/off-chip memory for intermediate results. Still, there should be also a solution to produce stable and better accuracy by SC.

On the other hand, quantization becomes crucial and takes a significant role for reducing both model size and the complexity of DCNN calculations [10]-[11]. The idea is prompted by the fact that neural networks do not require single or double precision of floating-point all the time. As a matter of fact, the requirements depend on the application, layer, etc. As DCNNs grow more complex and diverse, there is a need for a more reconfigurable hardware architecture that can run various DCNNs with different precision requirements at high efficiency. This is particularly useful for SC, where 1-bit saving could reduce computation latency by 50%, suggesting a great potential for higher efficiency on DCNNs with layers of different precision requirements. This dissertation presents a SC-DCNN accelerator that is not only highly efficient for large DCNNs but also very flexible in terms of supporting various DCNNs with different precision requirements.

Besides, there are advanced quantization methods appearing. One particular representation of interest is logarithmic quantization [12]-[13], which is to place quantization points in logarithmic scale. This can be achieved by quantizing the exponent of a number (throwing away the mantissa part). The latest work on this shows [14] that logarithmic quantization can reduce the precision of weight parameters by a few bits as compared to linear (fixed-point) quantization with the comparable accuracy of large DCNNs. The dissertation proposes to use logarithmic quantization to SC-DCNNs. It goes one step further which combines the advantages of both schemes, simpler hardware of logarithmic quantization and high precision of linear quantization, by suggesting a new quantization method.

This dissertation makes following contributions which are based on a series of published papers authored by the writer of this dissertation [15]-[20].

- It is argued that for scalability, it is essential to have some elements designed in conventional binary representation. Based on this, a scalable SC-based accelerator architecture for DCNNs, called binary-interfaced SC (BISC), is proposed.
- A novel SC multiply algorithm for BISC is presented and turned into a vectorized form called BISC-MVM (Matrix-Vector Multiplier). It improves both the efficiency and accuracy of SC considerably.
- A dynamic precision scaling (DPS) SC-DCNN accelerator is designed, which extends BISC-MVM, such that the precision of input/output data can be arbitrarily changed at runtime. The extension has very little overhead, but allows SC-MAC to be efficiently parsimonious in terms of precision, with an exponential reward in latency due to SC.
- Logarithmic quantization is applied to SC-DCNN and design improvement of the accelerator correlated to the quantization is made demonstrating that also in SC, there is no handicap for logarithmic quantization compared to linear (fixed-point binary).
- A new quantization method, called successive log-quantization (SLQ), is presented which solves the problem that, increasing resolution, or the bitwidth of log-quantized words quickly saturates accuracy due to uneven precision distribution.
- Experimental results demonstrate the feasibility of proposed SC-based DCNNs basically in terms of area-delay-product (ADP) and network accuracy. There are additional experiments on error resiliency which is a benefit of SC. Finally, a small FPGA prototype, validates our new SC multiplication algorithm's correctness as well as its implementation efficiency on an FPGA.

Section II provides the backgrounds and related works on DCNN acceleration, SC, and quantization. In Section III, we presents methods for enhancing SC-DCNN with those evaluations. Section IV demonstrates how SC can take the advantage of quantization effectively and presents state-of-the-art efficiency over fixed-point binary implementation in terms of ADP. Lastly, the dissertation concludes in Section V.

II. Background and Related Work

2.1. Deep Convolutional Neural Network Accelerators

Let us consider one convolution layer as convolution layers in a typical DCNNs account for more than 90% of the MAC operations. The computation in a convolution layer can be seen as a transformation from one 3D array of data called input feature maps to another 3D array called output feature maps, as illustrated in Figure 1-(a). Each element of the output array is computed by the 3D convolution between the input array and the weight matrix, followed by a bias addition. Different output array elements are created by applying the same convolution operation for different input subarrays or different weight subarrays. With limited resources, not all neurons can be simultaneously active, causing resources to be shared among different neurons. Hence using a good architecture can make a difference in terms of performance and energy efficiency. Several architectures for neural network acceleration have been proposed in the domain of conventional binary computing [21]-[24]. One important point here is that the input/output feature maps can be very large, and is stored in the off-chip memory in the general case. The off-chip memory may have sufficiently large capacity, but accessing it requires memory bandwidth, which is often quite limited. Another factor is on-chip buffer size. In general, there can be three on-chip buffers, for input, output, and weight parameters (see Figure 1-(b)), and unless the entire data can fit in the buffer, buffers need to be double-buffered to hide the off-chip memory access latency. The computation itself is very simple. We need no operation but a number of MAC operations. Accelerators are often built around an array of hardware MAC units, which can be organized as simply as a 1D array or a 3D array. The exact shape (such as 1D vs. 2D) and the size of the MAC array, along with the hyper-parameters of each layer (number of neurons, size of filters, etc.), affect the performance of the accelerator.

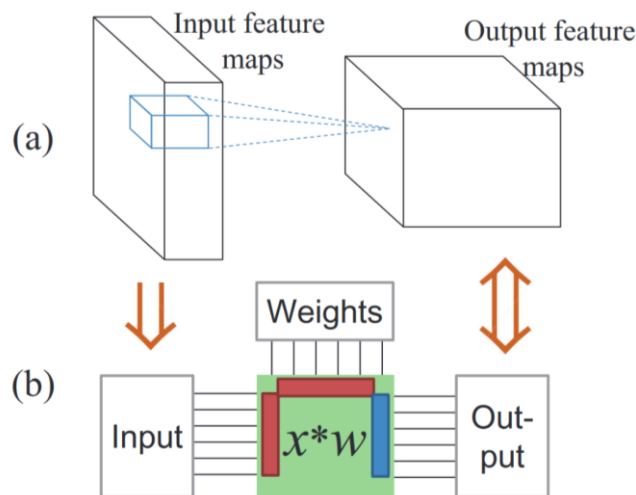


Figure 1: A generic accelerator for convolution layers.

2.2. Stochastic Computing

SC is not grown-up technology yet so that here we introduce SC backgrounds. Considering Figure 2, stochastic number is represented by bitstream which have the target probability. Stochastic number generator (SNG) converts conventional binary number into bitstream when the inputs are given as binary numbers. An AND (XNOR in case of signed multiplication) gate computes multiplication between 2 input streams. 1s in the output stream can be counted as converted into the conventional binary number of 1s' frequency. Computation with negative values can be done with bipolar format easily. To reduce the long latency of SC-MAC, bit-parallelism which computes several stochastic cycles at one time is applied as Figure 3. A parallel counter is used to count 1s of parallel streams.

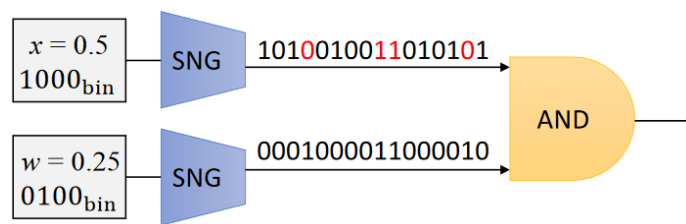


Figure 2: SC multiplication example.

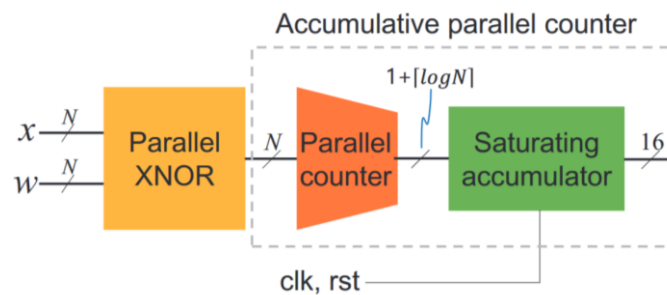


Figure 3: Signed and bit-parallel SC multiplication example.

As early as in 1994, SC was applied to implementing neural networks on a then resource-limited FPGA [25]. Circuit elements for SC are described in [26], with a companion paper [27] giving a neural network design as an example. Recently SC has been applied in designing Deep Belief Networks [8], [28] and other non-DCNN-type fully-connected neural networks [9], [29]. Previous work on SC-based acceleration of deep neural networks can be classified into the category: fully-parallel. In the fully parallel approach, all neurons are implemented spatially and they operate in parallel such that the neural network circuit will produce output as the wave of input data sweeps through the circuit. It can have very high energy efficiency owing to the fact that it does not involve external memory access to store intermediate result, but has limited applicability because it cannot support arbitrarily large networks.

We target DCNNs, which are more widely used today but much more complicated than the aforementioned fully-connected types of neural networks. The StoRM approach [30] is similar to ours in that both focuses on a MAC array. However, our proposed techniques are tied to DCNNs, and designed to improve scalability and accuracy of SC-DCNN accelerators whereas [30] is not.

2.3. Dynamic Precision and Logarithmic Quantization

Many techniques have been proposed to address the high computational complexity of deep neural networks DCNNs. At the algorithm level, researchers have proposed model reduction such as low-rank approximation, which can reduce complexity with little impact on accuracy. It is also shown that storage requirement due to weight parameters can be reduced to its 5 to 10% level [31], which may alleviate memory bottleneck problem. However, these techniques are effective mainly for fully-connected layers, with much less success with convolution layers, which account for the majority of computational complexity. At the implementation level, reducing the precision of arithmetic operations can reduce hardware cost, without significantly affecting accuracy. Approximate computing has also been applied to some of the neurons [32], which may be exploited to improve energy efficiency of neural network accelerators. All these results strongly suggest that current DCNN models are highly redundant, and essentially the same recognition performance can be achieved through retraining with much less weight parameters and less precise computation. That is why we may expect higher energy efficiency through SC, which is known to be more efficient for low-precision computation.

Recent DCNN hardware implementations [23], [33]-[34] all have their precision fixed at design time. This has obvious disadvantages when the precision requirement is different, such as low accuracy (when the needed precision is higher) and lower efficiency than achievable (when lower). Also, since these DCNN accelerators have tile-based architectures, all layers must use the same precision. However, the precision requirement of layers can be quite different [35], which causes some inefficiency even when the accelerator is running the DCNN for which it is designed. One solution to these problems is to use bit-serial hardware such as bit-serial multiplier [35] which can calculate the inputs of dynamic precisions at different runtime. While this approach can solve the abovementioned problems, bit-serial multipliers are inherently inefficient compared with a bit-parallel version except in low precision, which limits the effectiveness of the approach. The dissertation proposes to exploit dynamic precision for SC. The effect could be higher in SC than in conventional digital, since 1-bit reduction in SC may reduce delay by up to 50%.

There is an emerging format of quantization, logarithmic quantization. There are many applied logarithmic quantization researches on deep neural networks [12]-[14], [36]-[39]. [12] introduces logarithmic quantization for activation and weight, showing their acceptable network accuracy. [13]-

[14], [38] include hardware implementation results. [37] proposes an automation methodology of quantization. [14] shifts logarithmically quantized network accuracy really close to the level of floating-point. [39] applies logarithmic quantization to the biomedical image segmentation. Their logarithmic base is 2 by default to calculate the multiplication of linear (fixed-point) input activation and logarithmic weight by a shifter which is usually much smaller than a fixed-point multiplier. When, extreme cases, recognition rate is not acceptable, choosing more fine-grained base (e.g., $\sqrt{2}$) can improve it [12]. We explain how weights are represented in detail. Suppose maximal data precision is 16-bit (including sign bit). w is a real number of weight and ranges $(-1, 1)$ because weights are distributed near 0 and the magnitude is mostly less than 1. Logarithmic representation \tilde{w} is sign-magnitude formatted so that 1-bit is used for a sign. The magnitude is $|\tilde{w}| = \text{round}(-\log(|w|))$. Because \tilde{w} is saved as an integer, the real logarithmic value is rounded. How many bits would be required for the magnitude? Because $|w| < 1$, $|\tilde{w}|$ is larger than 0. The least unit that can be represented by 16-bit (including sign bit) is $1/2^{15}$ resulting $|\tilde{w}| = 15$. $|\tilde{w}|$ ranges $[1, 15]$ except when $w=0$. Covering this range, 4-bit unsigned integer ranges $[0, 15]$. 0, which is not assigned yet, can represent the case where $w=0$. The complete equation is

$$|\tilde{w}| = \begin{cases} \text{round}(-\log(|w|)). & \text{if } w \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

In conclusion, 16→5 data size reduction happens. To help understanding, Table 1 gives representation examples.

Table 1: Logarithmic data representation example.

w	\tilde{w}		
Real value	Sign bit	$ \tilde{w} $	Binary format
-0.5	1	1	1 0001
0	0 or 1	0	0 0000 or 1 0000
0.25	0	2	0 0010

III. Scalable, Accurate, and Efficient SC-DCNNs

3.1. Binary-Interfaced Stochastic Computing

3.1.1. Tile-Parallel Architecture for Scalability

Previous work on SC-neural networks [8]-[9], [28] has assumed that every layer and neuron can have dedicated resources. At the same time, since the networks they use are all fully-connected, it means that the application throughput is one (e.g., 1 image per cycle), and there is no need to buffer any intermediate result for any extended length of time. This is an ideal situation in terms of energy efficiency, since no memory and no resource sharing mean almost no power dissipation for anything but computation. Such a fully-parallel architecture may be realizable for special applications where the area budget is large and the size of the neural network is small. It may also be how biological neural networks operate. However, for low-cost embedded applications and/or if targeting real-life DCNNs, the fully-parallel architecture quickly becomes very irrelevant. We need an architecture that can support large DCNNs as well. A typical solution is to partition computation into multiple equal-sized blocks (or tiles) in a manner analogous to loop tiling, and to design hardware for one computation tile. Such hardware requires on-chip buffers for fast processing, but on-chip memory is invariably too small to hold entire data even for one layer. Double buffering is commonly employed, where on-chip buffers hold only the portion of the data currently needed while the entire layer data are stored in the off-chip memory. The on-chip buffers should be large enough to hide the data transfer latency. In this paper we call such an architecture tile-parallel architecture.

3.1.2. Rationale for Binary-Interfaced Stochastic Computing Design

Compared with conventional binary, SC uses much less resources for computation but requires far more cycles to achieve an equivalent resolution. The simplicity of SC circuits can help achieve higher clock speed. Let us assume that SC needs 2^k bits to get a similar resolution of k -bit fixed-point implementation in the conventional binary. One can see that from the throughput point of view, SC can beat conventional binary when k is low. When it comes to handling data, such as storing them on-chip or sending them to off-chip, conventional binary is more efficient for any value of k , although a conventional binary representation may also be more susceptible to errors. Thus fully-stochastic design such as in [8], [9] is largely incompatible with the idea of tile-parallel architecture. Otherwise the overhead in on-chip memory capacity and off-chip bandwidth can be many-fold: as much as 32X for $k=8$, and 4X even when $k=4$. Further, given that a large portion of the die area is dedicated for on-chip memory rather than the MAC array itself [23], the many-times increase in overhead in the case of fully-stochastic design seems simply too high. Thus, we conclude that for scalable architecture it is necessary

to have binary-interfaced design. We assume that the conversion from stochastic bitstreams to binary numbers happens before they are saved to on-chip memory.

3.1.3. Our Proposed Architecture: Doubly Hybrid

1) Binary-Interfaced SC for Scalability: Our DCNNs accelerator uses SC for the MAC array only, converting the input and weights from conventional binary to SC bitstreams, and converting back the output bitstreams to binary. Coincidentally this helps us avoid implementing other layers in SC, such as max-pooling layer, ReLU layer, and LRN layer, which can be tricky to implement in SC. We perform even the accumulation (of a MAC operation) using conventional binary arithmetic, utilizing SC for multiplication only. Multiplication is quite important, accounting for 76% of the area in our synthesis result for a conventional-binary MAC design (see Figure 9-(a)). Thus, even with this hybrid approach we can expect some efficiency improvement.

Perhaps one of the biggest downsides of this scheme is the conversion overhead between SC and conventional binary. It is not that there are unnecessarily many converters all around the accelerator; physically, there is only one array of binary-to-stochastic (B-to-S) converters and one array of stochastic-to-binary (S-to-B) converter, as shown in red/blue blocks in Figure 1-(b). But these converters are used for every layer of a DCNN, which may seem redundant and unnecessary from the viewpoint of a fully-parallel, fully-stochastic-computing accelerator. However, we see this as a fair price to pay, necessary to support large DCNNs on a small accelerator. Also, the B-to-S overhead can be amortized by sharing resources such as LFSRs (Linear Feedback Shift Registers) across the MAC array, or reduced significantly by using emerging devices such as magnetic tunnel junction (MTJ) transistors [40]. We address the S-to-B overhead by using approximation and bit-parallel SC design (see Section 3.1.4).

2) Hybrid Layer Composition for Accuracy: The other feature of our architecture is that SC is applied to convolution layers only whereas fully-connected layers use conventional binary, which is one form of what we term hybrid layer composition. This is the rationale. The number of MACs in the fully-connected layers of a DCNN is usually very small, but just because they are very close to the output, they can have a direct impact on the final outcome. Using conventional binary for the fully-connected layers can therefore ensure high accuracy, while using SC for convolutional layers can increase cost-effectiveness, thus striking a good balance between accuracy and efficiency. Though simple, it does not cost any more hardware and works surprisingly well.

3.1.4. SC-MAC Unit: Minimizing Overhead with Parallelism and Approximation

Figure 4 illustrates our SC-MAC design in comparison with a conventional binary one. In SC, multiplication and addition can be performed very cheaply by an XNOR gate and a MUX, respectively [26]. The problem is that they are too small, compared with the ensuing S-to-B converter, which is typically done by a counter. This counter can also serve as the accumulator, leading to a very simple 1-bit SC-MAC design as shown in Figure 4-(b). But in this design the overhead of the accumulator (also serving as a counter) can be nearly 100 times (see Figure 9-(b)), in terms of area according to our evaluation. We address this problem by employing bit-parallel SC design and approximation. An N -bit parallel SC design allows us to share the accumulator among N 1-bit SC-MACs, essentially reducing the accumulator overhead by N times. However, in a bit-parallel version we need an explicit counter, such as an N -bit parallel counter, as illustrated in Figure 4-(c). Since the size of the accelerator is largely unchanged regardless of N , we can reduce the accumulator overhead very effectively with bit-parallelism. The parallel counter overhead, however, is a different story, which we tackle using approximation. Nonetheless by putting together a parallel counter and an accumulator, which are together known as accumulative parallel counter, we can exploit known circuit optimization techniques such as [41] to achieve higher operating frequency and smaller area. The parallel counter overhead cannot be reduced by going parallel, because it increases in proportion to N . Instead we use an approximate S-to-B method [42]. The idea is to reduce N bits into $N/2$ bits with approximately half the number of “one”s. While this can nearly halve the parallel counter overhead, it has a side effect that lowering the recognition accuracy, as a result of which we may end up using a longer bitstream, defeating the purpose of using approximation. Thus, we need a careful evaluation to assess its effectiveness (see Section 3.3).

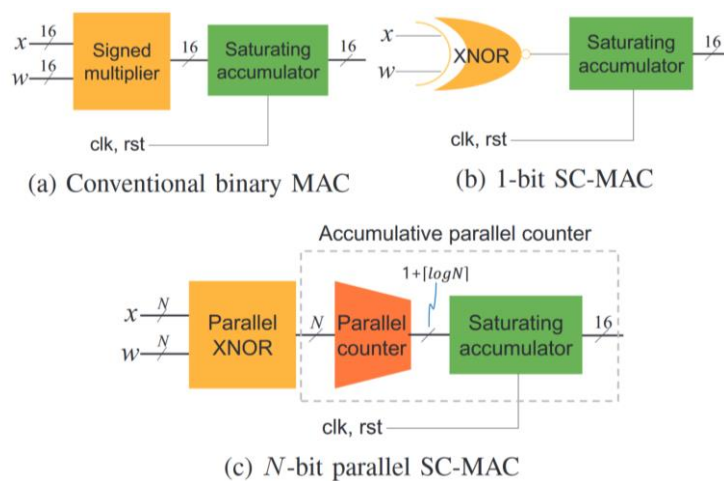


Figure 4: Structure of a MAC unit. Our SC-MAC integrates a stochastic-to-binary (S-to-B) converter (i.e., parallel counter).

3.2. New Stochastic Computing Multiplication Algorithm

3.2.1. Conventional Stochastic Computing Multiplication

Figure 5-(a) illustrates conventional SC multiplication which is conducted in Section 3.1. In SC, a number 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, which is synonymous to B-to-S converter, takes an N -bit binary number and generates a stochastic bitstream, and it typically consists of a random number generator such as an N -bit LFSR and an N -bit comparator, which generates 1 if the random number is less than the input binary number, and 0 otherwise [43]. 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 stochastic number to a binary number. An up-down counter does the same for bipolar.

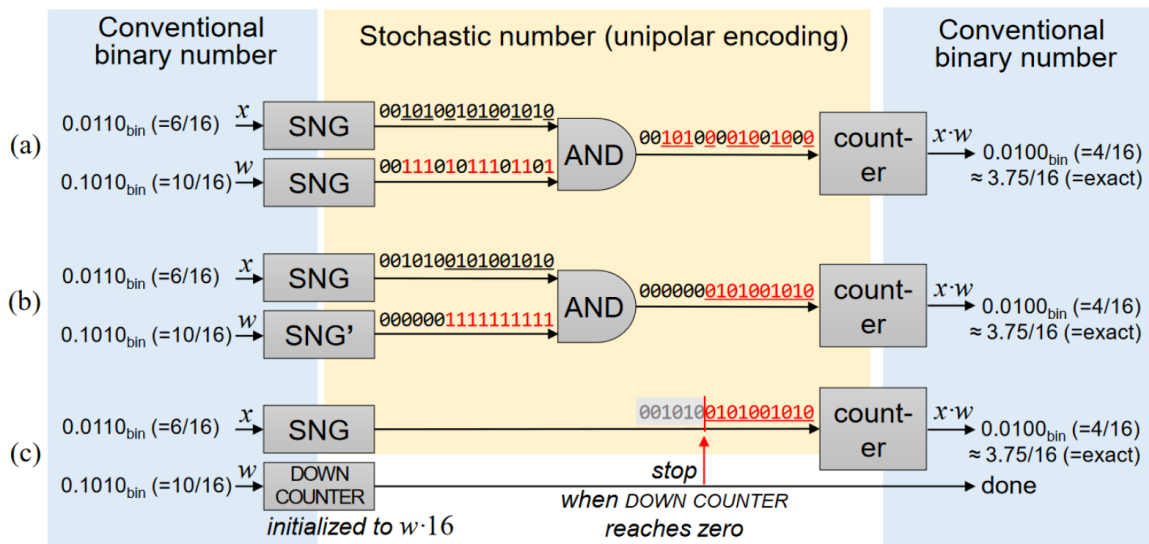


Figure 5: How new SC multiplication works.

3.2.2. Our Proposed SC-MAC

Suppose we reorder the bits for one input w such that all the 1s appear first as illustrated in Figure 5-(b). Certainly, it does not affect the value of stochastic number for w or that of the resulting stochastic number after the AND operation, if the two stochastic numbers are still statistically uncorrelated. Therefore, the binary number outcome in Figure 5-(b) is expected to be the same as that of Figure 5-(a). Note that the order of stochastic number 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 Figure-5(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 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 3.2.6. Third, it has a shorter average latency, again without losing any accuracy, as compared to the conventional method. Our SC multiplier illustrated in Figure 5-(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.

3.2.3. Enhancing Accuracy via Low-Discrepancy Code

The accuracy of the binary number output in Figure 5-(c) depends on how uniformly the 1s are distributed in the stochastic number bitstream, suggesting a good use of low-discrepancy code such as Halton sequences [5]. Though low-discrepancy code has already been used in SC [5], [44], 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 a stochastic number 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 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 [5] (see Section 3.3.3).

Given an N -bit fractional number $w \in [0,1)$, let $k = 2^N w$. The accuracy objective for our SC multiplication in Figure 5-(c) dictates that the partial sum, P_k , of the stochastic number 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 binary number, $x_{N-1} \cdots x_0$, 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 Figure 6-(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. It is proved that with this pattern, the number of

times x_{N-i} appears within the first k cycles equals $\text{round}(k/2^i)$. The theoretical maximum error of our SC multiplication is $\sum_{i=1}^N 1/2^i = N/2$ for xk , or $N/2^{N+1}$ for wx . But this error bound is not tight; instead, we show maximum error empirically in Section 3.3.3.

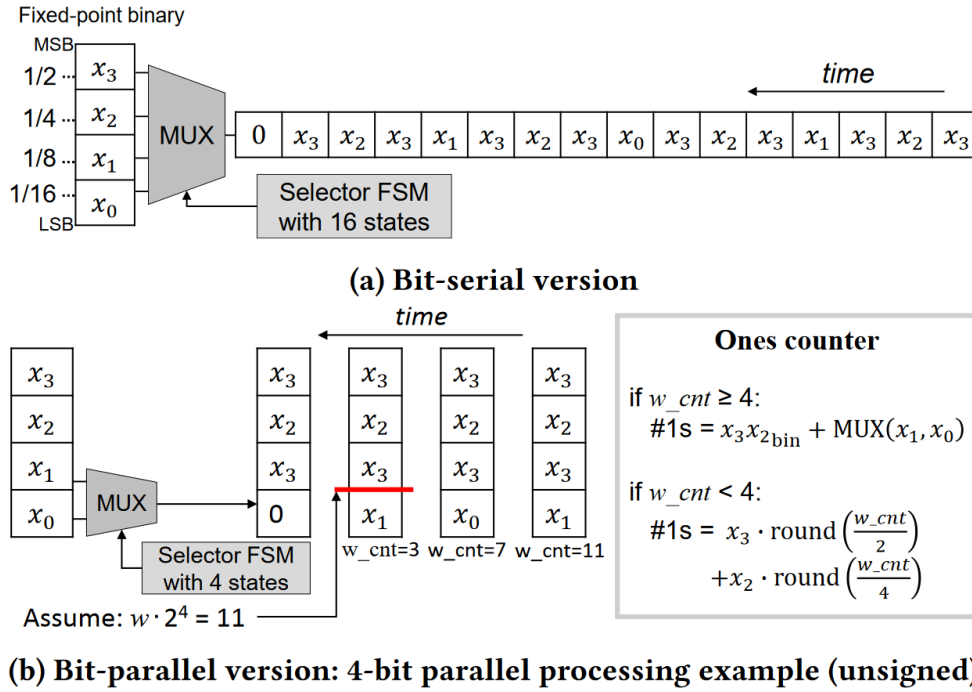


Figure 6: Simple and accurate low-discrepancy code generation using FSM.

3.2.4. 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 2. 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 2: Signed multiplication example.

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

3.2.5. Bit-parallel Processing Optimization

To further reduce the latency of our SC multiplier, we propose bit-parallel processing. Let us consider the example in Figure 6-(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 3.2.3. 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.

3.2.6. Proposed BISC-MVM and SC-DCNN Accelerator

1) BISC-MVM, Vectorization of Our SC-MAC: Figure 7-(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} and w_i ; no additional hardware is necessary. Then the accumulation result can be read from the array of up-down counters at cycle $\sum_{i=1}^d |2^{N-1}w_i|$.

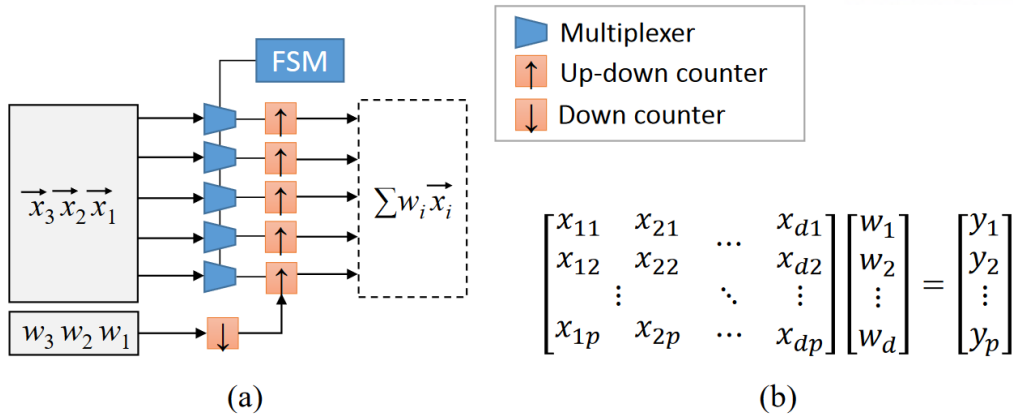


Figure 7: (a) Our SC matrix-vector multiplier (simplified) and (b) the operation it performs, where

$$y_j = \sum_i w_i x_{ij}.$$

Mathematically this is a matrix-vector multiplication in the form of Figure 7-(b). Our BISC-MVM has the following features. First, all SC multipliers share both the down counter and the FSM, but this causes no accuracy degradation, which is quite contrary to conventional SC. In addition, 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 Figure 7-(b) may not be how a neural network layer is typically described mathematically. We next discuss how this BISC-MVM can be applied to accelerate DCNNs.

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 [45] 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 Figure 8, where the 3 innermost loops are executed by a hardware accelerator as fully unrolled (i.e., simultaneously).

```

for ( $m_1 = 0; m_1 < M; m_1 += T_M$ )
  for ( $r_1 = 0; r_1 < R; r_1 += T_R$ )
    for ( $c_1 = 0; c_1 < C; c_1 += T_C$ )
      for ( $z = 0; z < Z; z++$ )
        for ( $i = 0; i < K; i++$ )
          for ( $j = 0; j < K; j++$ )
            for ( $m = m_1; m < \min(M, m_1 + T_M); m++$ )
              for ( $r = r_1; r < \min(R, r_1 + T_R); r++$ )
                for ( $c = c_1; c < \min(C, c_1 + T_C); c++$ )
                   $B[m][r][c] += W[m][z][i][j] \times A[z][Sr + i][Sc + j];$ 

```

Figure 8: 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.

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) Our SC-DCNN Accelerator Architecture: Our SC-DCNN accelerator architecture is by design very similar to previous DCNN accelerators [45]-[46] based on conventional binary. In fact, there should be no difference in the top-level architecture from that of [45] in particular, since we use the same parallelization 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 [15] 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.

3.3. Evaluation

3.3.1. Experimental Setup and Implementation Detail

An evaluation of any SC design must consider both functional accuracy and the efficiency of hardware implementation. In the case of SC-DCNN, we also need to consider (re) training of weight parameters, which is really crucial as we will show. Fortunately for us, we were able to easily extend an existing DCNN model, Caffe [47], to generate a model of our SC-DCNN and use it for training and functional simulation, because our architecture retains the same interface as the conventional binary version.

To see the efficiency of hardware implementation we also designed and implemented our SC-MAC unit. We designed not just one SC-MAC unit, but an array of SC-MAC units, which we call compute tile or simply a tile, as needed in a DCNN accelerator. Again, because our SC-MAC compute tile has the same interface as its binary version, we can easily and accurately evaluate our proposal without necessarily building the entire system in hardware.

We use a DCNN designed for the MNIST digit recognition benchmark, which has been frequently used in previous work [8]-[9], [28]. Ours is based on a DCNN model distributed with Caffe, but modified by adding tanh-activation layers, in line with previous work using the same dataset.

1) MAC Design: We have implemented different MAC designs in Figure 4 using Verilog RTL. For bit-parallelism N , we use 64 and 128 (The approximate version is 128-bit only). In the binary MAC, we use fixed-point consisting of a 1-bit sign bit and a 15-bit fractional part. No integer part is necessary as values are between -1 and 1. The multiplier generates 16-bit signed number. The accumulator, which is shared with SC-MACs, uses 18 bits internally due to a 2-bit integer part, but the final output is given by the most significant 16 bits. The parallel counter is an up/down counter as our stochastic bitstream follows a bipolar representation.

For accurate area evaluation we have implemented the entire compute tile, including LFSR-based B-to-S, referred to as SNG (Stochastic Number Generator), and a MAC array, whose size is varied from 4x4x4 to 8x8x8. For synthesis we use Synopsys Design Compiler with TSMC 45nm technology. The target clock period is set to 0.65ns from the best achievable value by the binary MAC array. No pipelining is used inside a MAC unit.

2) Simulation and Training: We have extended Caffe with a new convolution layer definition and a new fully-connected layer definition based on our SC-MAC array. The binary input and weight parameters are first converted into stochastic bitstreams and go through the SC-MAC array, after which the binary-converted output is generated. The binary input values are within -1 and 1 due to tanh function, but weight values may not be. Thus, we scale them before convolution and re-scale back after

convolution. We use 4 and 2 for scale parameters for the first and second convolution layers, respectively.

Due to the way the convolution operation is defined mathematically, the input data are reused for computing different output feature maps. Our modeling, as well as our hardware architecture, assumes that these input data are reused at the binary level as opposed to the stochastic bitstream level, and new stochastic bitstreams are generated from the same binary input data whenever they are needed. Retraining of SC-DCNN is done by applying SC to the forward pass only. The B-to-S conversion is done by comparing a random number with the binary input. For random number generation we tested both LFSR and CUDA random function, but the latter, which is used in all our results, turned out to be orders-of-magnitude faster than LFSR without much difference in accuracy.

3.3.2. Binary-Interfaced Stochastic Computing

1) Hardware Implementation Efficiency: Figure 9 shows the area breakdown of a MAC unit. The area of a MAC (without the gray portion in the figure) is the average value obtained by dividing the tile area by the number of MAC units. The figure shows that the accumulator area is almost the same for all the cases, and the parallel counter area is proportional to the bit-parallelism. The approximate version is very effective in reducing the parallel counter overhead.

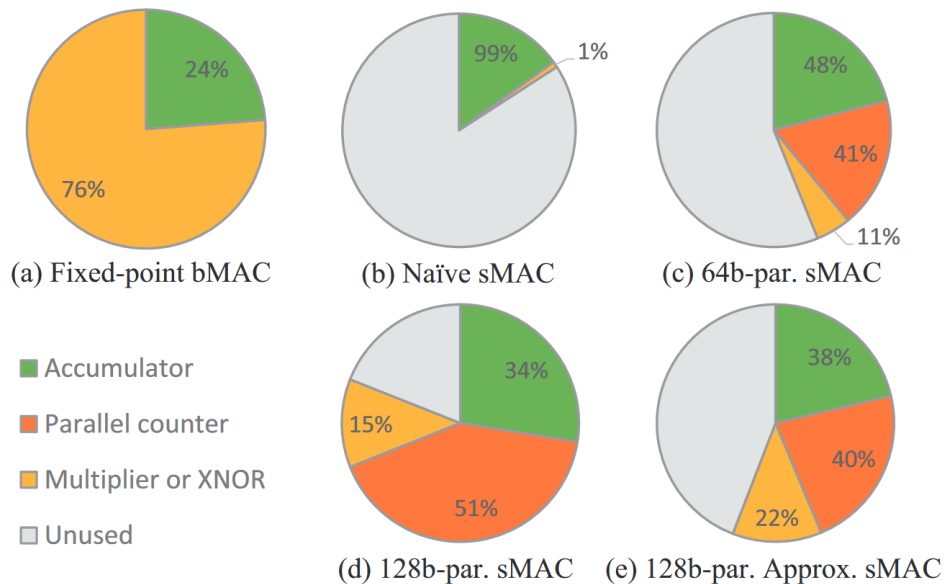


Figure 9: Area breakdown of a MAC, where bMAC and sMAC stand for a conventional binary MAC and an SC-MAC, respectively. Each whole pie, including the *unused* portion shown in gray, represents the same area, which is the area of the bMAC; sMAC uses far less. A number in each pie represents the portion of a component in the *used area*.

Under the same area-delay product, naïve SC-MAC (1-bitserial version) can process 8 bits while the binary version processes 1 word. Our 128-bit approximate version can process nearly 256 bits, which is about 30X improvement. Increasing bit-parallelism increases parallel counter area and latency, but does not increase the overall MAC latency because the parallel counter and the accumulator can be optimized together as an accumulative parallel counter. The larger area of the accumulator in Figure 9-(d) is due to the increased critical paths, which increase the areas of both the parallel counter and accumulator due to timing optimization.

Table 3 summarizes tile synthesis result, where the data for tile size of 12x12x12 are extrapolated (could not be synthesized due to size). Extrapolation is possible because the area is linearly proportional to the size of the array and increasing tile size does not affect the critical path (affected most by bit-parallelism). The peak performance is the theoretical performance assuming that the MAC units are utilized 100%. In practice, utilization varies depending on the layer parameters and whether there are other bottlenecks in the system such as data transfer. For the SC versions we use sMAC/s, which is the number of 1-bit SC-MAC operations performed per second, since the application throughput depends on the stochastic bitstream length. The table shows that the SNG overhead, which includes that of both input and weight parameters, is quite high when the tile is small, but reduced quickly as the tile size increases.

Table 3: Compute tile synthesis results.

Tile size	Case	Area (mm ²)		Peak Performance
		SNG	MAC array	
4x4x4	Binary	—	0.094	98.5G MAC/s
	64b-SC	0.074	0.042	6.3T sMAC/s
	128b-SC approx.	0.151	0.053	12.6T sMAC/s
8x8x8	Binary	—	0.765	787.7G MAC/s
	64b-SC	0.220	0.336	50.4T sMAC/s
	128b-SC approx.	0.448	0.426	100.8T sMAC/s
12x12x12 (est.)	Binary	—	1.494	2.7T MAC/s
	64b-SC	0.442	1.134	170.1T sMAC/s
	128b-SC approx.	0.904	1.438	340.3T sMAC/s

2) Overall Performance: Figure 10 shows the recognition accuracy of different schemes for different stochastic bitstream length (SBL) values. The original DCNN using floating-point arithmetic is first trained using the default training script provided in Caffe, which runs 10,000 iterations. It is followed by retraining of 5,000 iterations using different versions of SC-DCNN. We use the same parameters except that the base learning rate is decreased by 10 times, which is a common practice.

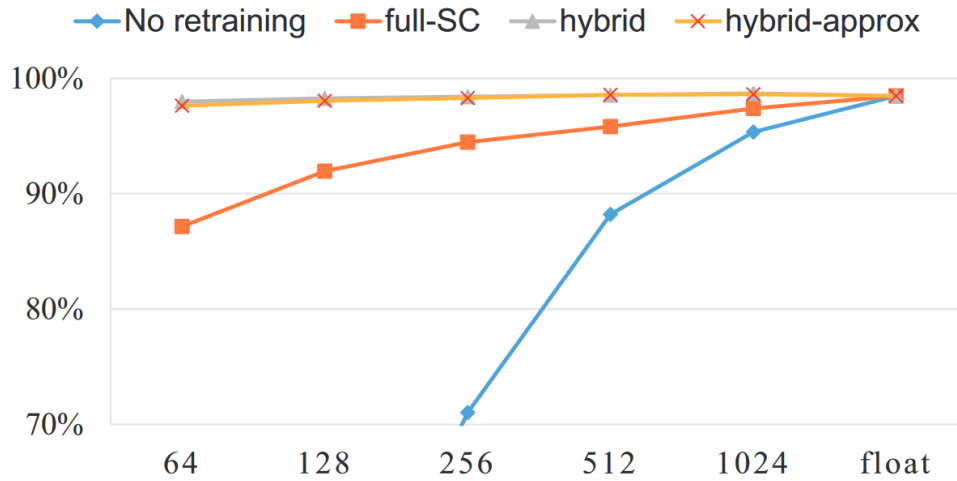


Figure 10: Recognition accuracy vs. stochastic bitstream length.

These graphs are obtained after retraining the network, except for the floating-point case. Without retraining the recognition accuracy is extremely poor, being around 88% even for SBL of 512 bits (the full-SC DCNN case), with steep decline for lower values of SBL.

The graph clearly shows that there is a significant gap in recognition performance between our hybrid DCNN and full-SC DCNN (i.e., fully-connect layers are in SC as well), suggesting the superiority of the hybrid design. Between the two versions of hybrid design, approximate vs. non-approximate versions, we see that the difference in recognition performance is negligible. However, the MNIST dataset is relatively easy, and we may find larger gap for more complex dataset, which is left for future work.

Figure 11 shows the overall performance as measured in area-delay product, which is the inverse of throughput per area, for different cases. For this comparison we use SBL of 128 bits, for both approximate and non-approximate SC-MAC cases, and the tile size is 8x8x8. Tile size being the same, utilization must also be the same for all the cases considered here, regardless of layer parameters.

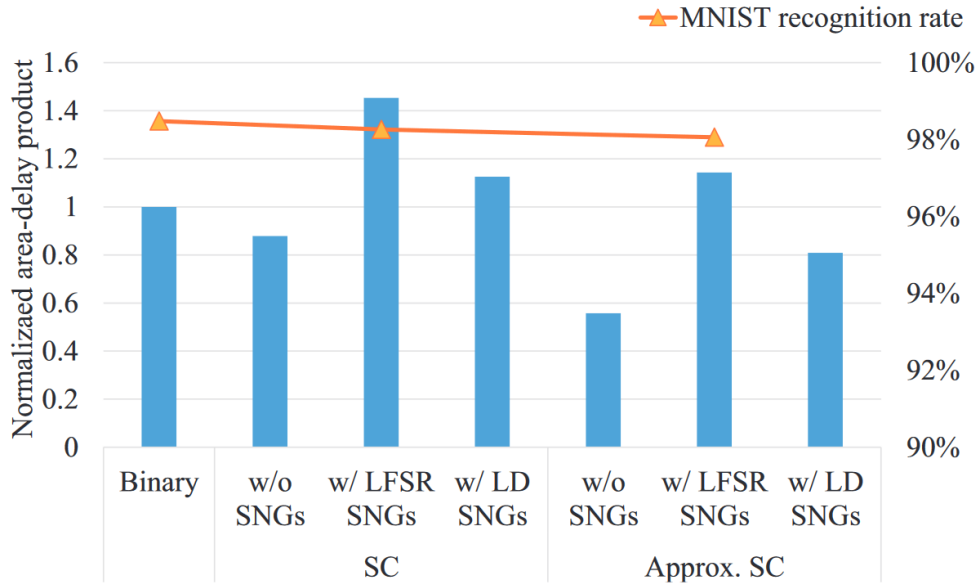


Figure 11: Area-delay product comparison (SBL: 128-bit).

The graph shows that in terms of the recognition accuracy, there is only less than 1%-point difference among them. Yet in terms of throughput the approximate version can achieve about 35% higher performance over the non-approximate version, without considering SNGs. If SNGs are not considered, the best SC version can be nearly twice as area-efficient as the binary version. With recent MTJ devices the cost of SNG is getting much cheaper, but here due to the lack of accurate area information of MTJ-based random number generators, we instead use the area of the recent low-discrepancy (LD) SNGs [44], which are designed for bit-parallel SC circuits and therefore well-suited for our application. This can reduce the area overhead of SNGs down to about 50% of the MAC array. Overall this graph suggests that SC-DCNN can be both accurate and area-efficient, even with SNG overhead, compared with a conventional binary version. Note that this comparison is without considering the inherent advantages of SC such as dynamic energy-quality trade-off and better error tolerance. For future technologies in which variability and noise are expected to grow, the advantages of SC will be greater.

3.3.3. New Stochastic Computing Multiplication Algorithm

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

1) Accuracy Analysis of Our Proposed SC Multiplication Algorithm: The conventional SC has different flavors depending on the SNG: (1) LFSR and comparator, (2) Halton [5], and (3) even-distribution-based low-discrepancy code abbreviated as ED [44]. 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. Figure 12 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. For our proposed method, at cycle $|w|/2^{5-x}$ or $|w|/2^{10-x}$. Thus, it shows not only the statistics at the end of the bitstream, but also how fast the output converges. ED [44] 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.

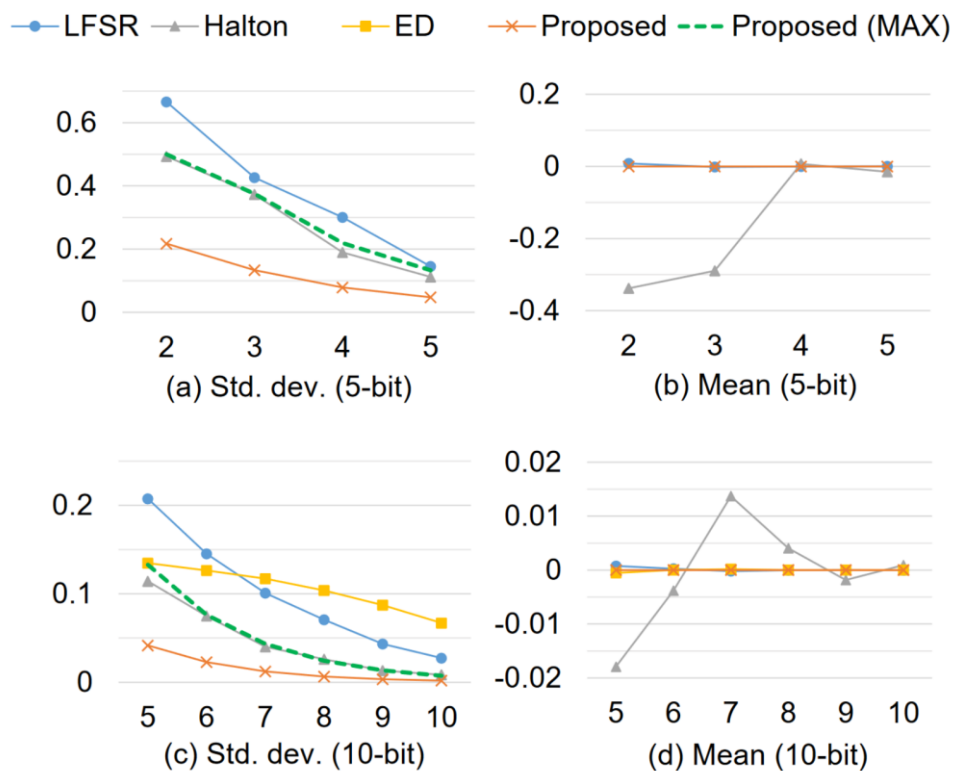


Figure 12: Error statistics of various SC multipliers.

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. Actually, the result for the Halton case depends on the prime number base. We use 2 and 3 for x and w , respectively.

2) Recognition Accuracy of Our Proposed SC-DCNN: To evaluate the recognition performance of our SC-based DCNNs we use the Caffe framework [47], in which the convolution layer is extended for fixed-point and SC. We use two DCNNs 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 network 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 Figure 13 show the test accuracy of the MNIST network. 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.

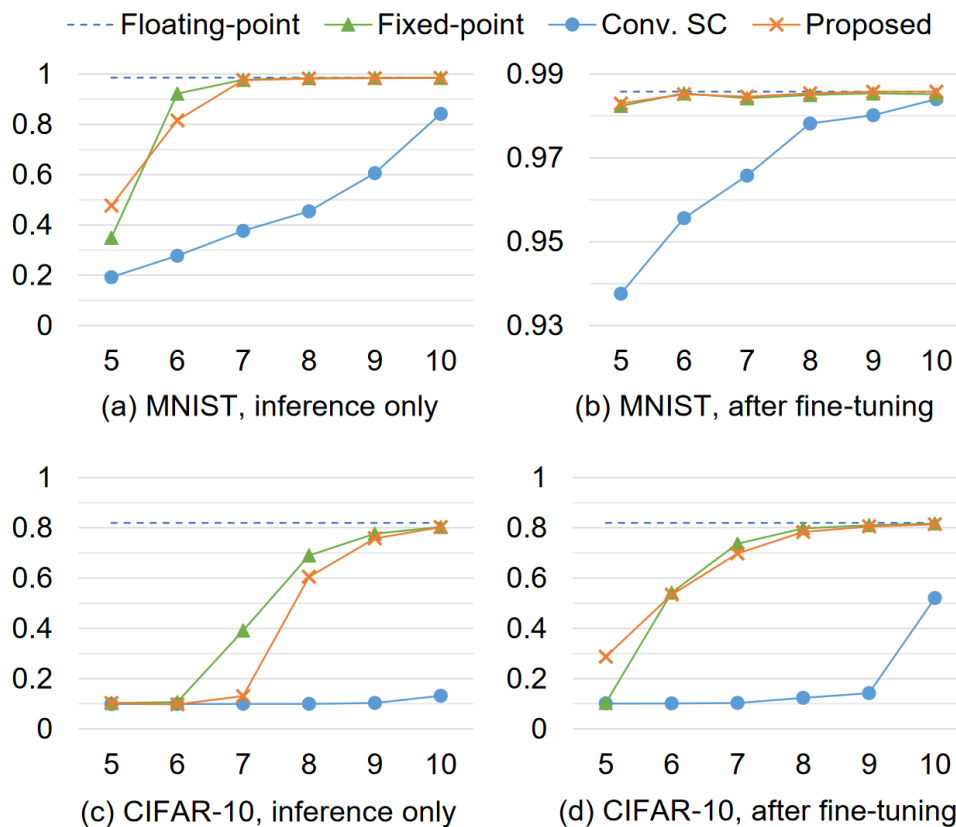


Figure 13: MNIST, CIFAR-10 recognition accuracy (the first 5,000 test images), where x -axis is multiplier precision including the sign bit.

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-DCNN 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 [9].

We run a similar experiment with CIFAR-10, the result of which is summarized in Figure 13-(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-DCNN can achieve near-fixed-point performance 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; nonetheless, it is successfully filled via retraining, making our SC-DCNN virtually indistinguishable from the fixed-point version in terms of accuracy.

3) Implementation Efficiency of Our Proposed SC-DCNN: 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 using TSMC 45nm technology as did in Section 3.3.2. 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, an FSM and a down counter are shared across all SC-MACs. Results are summarized in Figure 14, where we include the bit-parallel processing option for the CIFAR-10 experiment, with the parallelism of 8 bits.

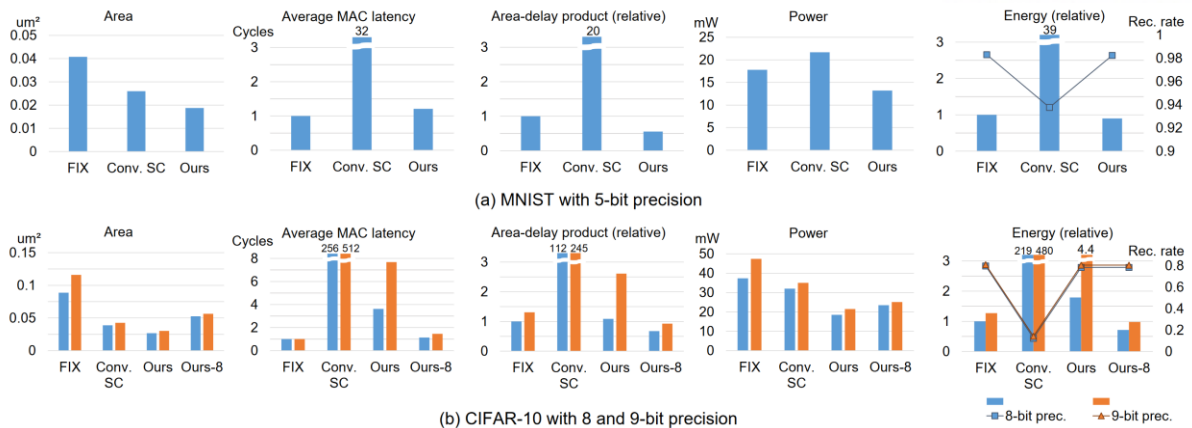


Figure 14: Comparison of MAC arrays: fixed-point binary version (“FIX”), LFSR-based conventional SC version (“Conv. SC”), and our proposed BISC-MVM (Ours and Ours-8 are our bit-serial and 8-bit-parallel versions, respectively).

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 binary number to stochastic number and back, as confirmed by area breakdown in Table 4.

Table 4: 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	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	203.7	33.9	1.8	–	108.0	347.3
		ED	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).

Table 4 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 DCNN-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 [44], which is evaluated for the 9-bit precision setting only, uses a bit-parallel SNG that generates 32 bits per cycle, requiring 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 Figure 14).

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 Figure 12-(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 Figure 14. 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 Figure 14 than in Table 4.

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.

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. 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 stochastic number and binary number, 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-DCNN 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 trade-off and better error tolerance. For future technologies in which variability and noise are expected to grow, the advantages of SC may be greater.

Table 5 provides a brief comparison with previous neural network 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 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.

Table 5: Comparison with previous neural network accelerators.

		Frequency	Area*	GOPS	GOPS/mm ²	Tech.	Scaled GOPS/mm ² ***	Scope for area
Binary	MWSCAS’12	400 MHz	12.50	160.00	12.80	45nm	65	Total chip
	ISSCC’15	200 MHz	10.00	411.30	41.13	65nm	869	Total chip
	ASPLOS’14	980 MHz	0.85	501.96	592.94	65nm	2,556	NFU** only
	GLSVLSI’15	700 MHz	0.98	274.00	278.85	65nm	1,683	SoP (\approx MAC) units only
SC	ArXiv’15	400 MHz	0.09	1.01	11.91	65nm	126	One neuron
	DAC’16	1000 MHz	0.06	75.74	1262.33	45nm	2,556	One neuron with 200 inputs
	Proposed (9b-precision)	1000 MHz	0.06	351.55	6242.37	45nm	12,641	MAC array (size: 256)

*Note 1: See the rightmost column for the scope of area (in mm²)

**Note 2: NFU can also perform pooling and activation functions.

***Note 3: GOPS/mm² normalized by frequency and (1/technology)²

Compared to a recent SC design [9], ours has much higher area efficiency. Note that the previous work is a fully-parallel architecture, but ours has scalability, which cannot be provided by the previous work. Compared to the others, our architecture has the highest area-efficiency.

In addition to the differences in terms of the scope of area and the level of implementation, the performance-per-area (GOPS/mm²) metric in Table 5 uses different frequencies and technologies. Thus, we provide another comparison which normalizes frequency and technology effects, shown in the 8th column. The technology is normalized in square because area scales at the square of technology node (“Scaled GOPS/mm²” = GOPS/mm² * Tech.² / frequency). In this comparison, DAC’16 and ALPLOS’14 tie for the second position while our proposed SC-MVM shows the best compute density.

There are other recent SC-based DCNNs [48]-[49], which however do not provide area/power numbers (focusing on accuracy or targeting FPGAs), as well as DCNNs based on concepts similar to SC [50]. In particular XNOR-Net [50] 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 [1]. Such training methods are orthogonal to our contributions, and can bolster the case for SC-DCNNs in general.

3.3.4. Small FPGA Prototype

We have also prototyped a DCNN accelerator on an FPGA board. The goal of this is two-fold: to validate our BISC MVM-based DCNN accelerator as well as to assess our new SC multiplication algorithm’s suitability for FPGA implementations. We have used a Xilinx VC707 board, which includes a Virtex-7 FPGA and a DDR3 memory. Figure 15 illustrates the system architecture, where the MicroBlaze softcore is used to run the software part, and an AXI bus and an AXI memory controller are used to connect hardware modules to the DDR3 memory. The main MVM module (which contains 8x8 MAC units) is designed in Verilog HDL, for the conventional binary case as well as various proposed SC cases. For the prototyping we have used LeNet-5 and CIFAR-10 DCNN, with precisions as listed in Table 6. Only the convolution layers are implemented on the MVM while the other layers are implemented in software on the MicroBlaze. We have verified the correct operation, matching with the software simulation results produced by Caffe on a GPU.

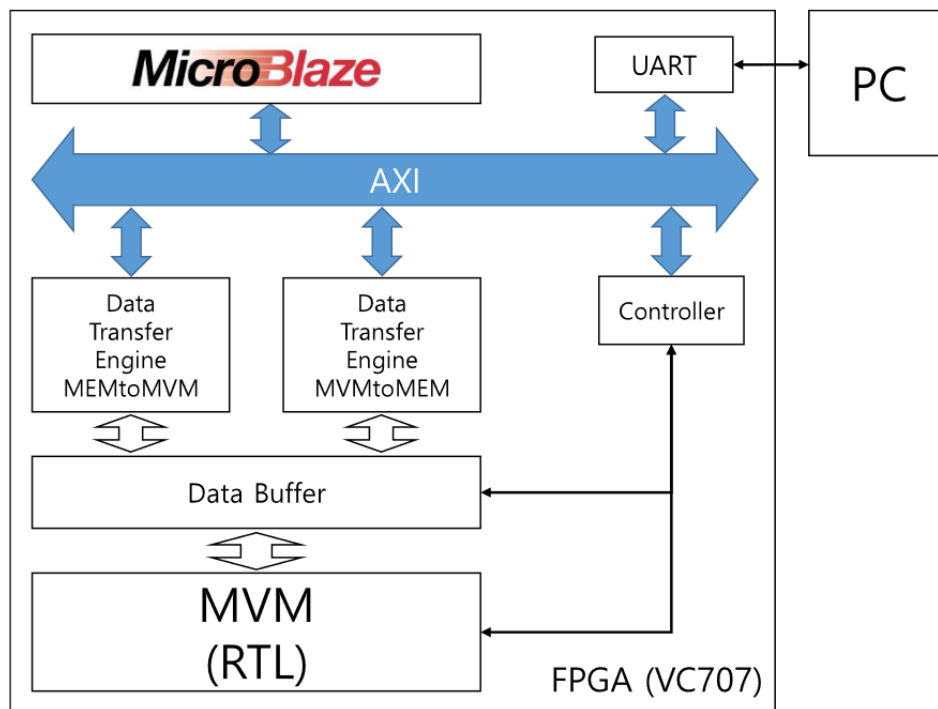


Figure 15: Block diagram of MVM-based accelerator implementation on VC707.

Table 6: Comparison of various MVM implementations on FPGA (#MAC=8x8).

Target DNN	Case	Frequency	LUT	FF	DSP
LeNet-5 (5-bit precision)	FIX		2494	448	
	LFSR-SC		1161	469	
	Ours		1035	457	
CIFAR-10 (9-bit precision)	FIX	100 MHz	6976	704	None
	LFSR-SC		1742	729	
	Ours		1668	722	
	Ours-8		3585	727	

Table 6 shows our FPGA synthesis result, after technology mapping and place and route. For fair comparison, LUT-based MACs are used for the binary case, as DSP blocks are hardly used in SC-based designs. No manual optimization is attempted for any design; all optimization is that of the Xilinx RTL synthesis tool. Even then, we can see from the table that SC-based designs use much less resources compared with the binary designs, thanks to the simpler computational units. Thus SC-based designs can be useful for cost-sensitive applications. On the other hand, conventional SC designs take much longer latency as shown in Figure 14. Our proposed SC designs including the 8-bit parallel version for CIFAR-10, achieves near 1 cycle latency for MNIST and less than 1.5 cycles for CIFAR-10, and thus consistently achieve the highest efficiency in terms of area-delay product. Since our SC-MAC does not utilize DSP blocks, a hybrid approach that combines both binary MVMs and SC-MVMs on a single FPGA could provide the best solution in terms of FPGA resource utilization, which remains for future work.

IV. Stochastic Computing in Synergy with Quantization

The previous section proposes a scalable, efficient, and accurate SC-MVM at static precision. However, it is well known that there is a wide range of precision requirements through DCNNs and even layers [35]. In case of SC, 1-bit reduction provides about half latency reduction. (see Figure 18) However, using low-precision is not always a solution because of significant accuracy loss [58]. Those facts backup the necessity of supporting dynamic precision.

On the other hand, log-quantization has successfully reduced the binary multiplier area in addition to saving memory bandwidth. In this section, we will demonstrate that log-quantization also benefits a bit-parallel SC multiplier.

Log-quantization has its own limitation that errors at larger values cannot be restored by increasing bitwidth. In other words, usual dynamic precision cannot help accuracy saturation of log-quantization. We suggest a new quantization method which gives another dynamicity to overcome the accuracy limitation.

4.1. Dynamic Precision Scaling

4.1.1. Analysis of Baseline SC-MAC

Here we present an analysis of the SC-MAC proposed in [16]. The SC-MAC takes two operands labeled x and w , and generates output y which should approximate xw . All the inputs/output are represented as conventional digital, as illustrated in Figure 16 (shown in red is for dynamic precision discussed in the next section).

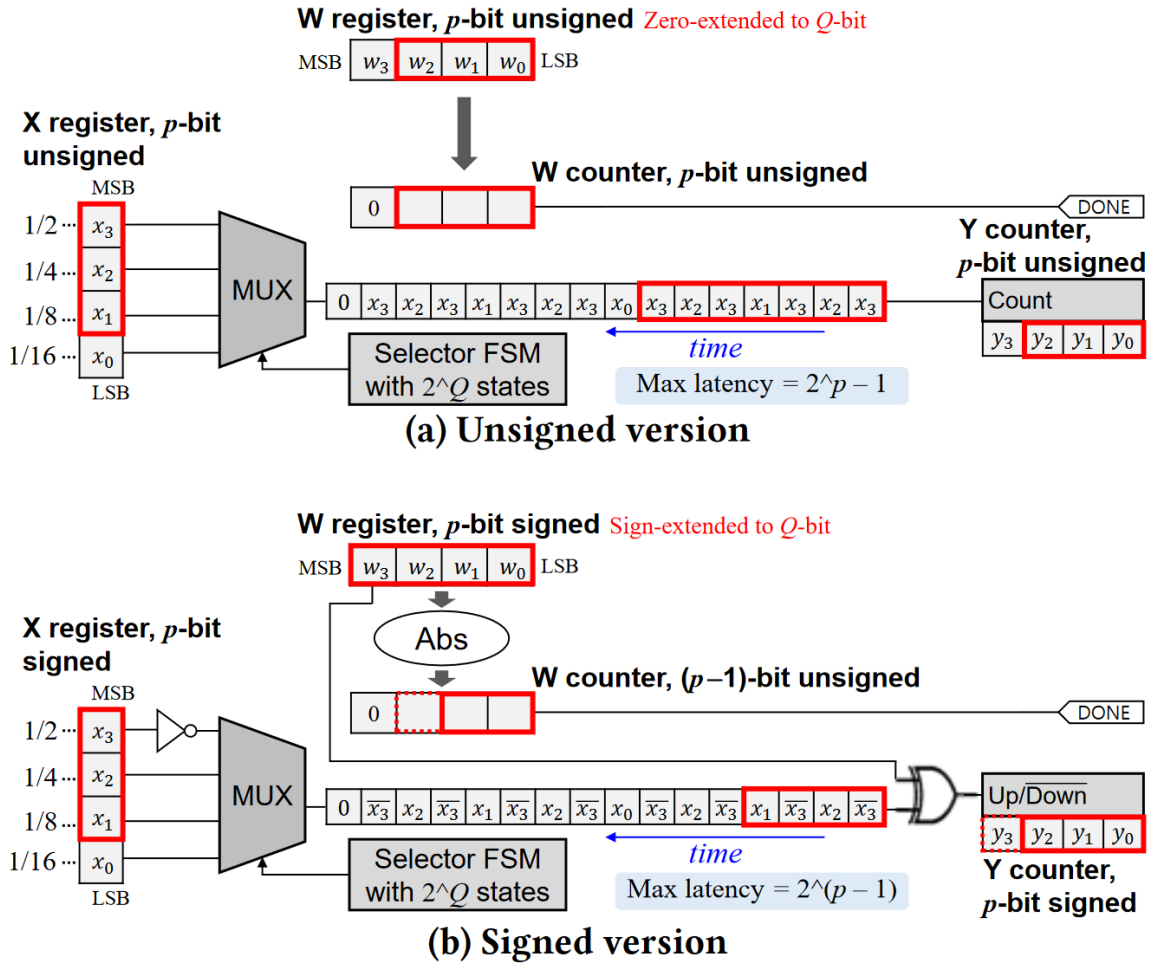


Figure 16: SC-MAC from [16] extended for dynamic precision. Datapath remains the same; only control is changes. (p : dynamic precision, Q : the maximum supported precision)

First consider the unsigned version, where the inputs/output are interpreted as fractional numbers between 0 and 1. Let $X = x2^Q$ and $W = w2^Q$, where Q is the width of the X and W registers. The MUX-FSM circuit is designed to generate a bitstream whose signal probability is close to x [16]. Thus, counting bits from the x -bitstream for W cycles gives approximately $xW = xw2^Q$. Therefore, $y \approx xw$ with Q -bit precision.

In the signed version, the inputs/output are 2's complement numbers between -1 and 1. Since MSB is used as the sign bit, $X = x2^{(Q-1)}$ and $W = w2^{(Q-1)}$. The W counter is initialized to the absolute value of W . If W is positive, feeding the x -bitstream to the Y counter (which is now an up/down counter) for W cycles will give approximately $xW = xw2^{(Q-1)}$. To understand why: (i) Unsigned interpretation of X register after inverting the MSB is $(X+2^{(Q-1)})/2^Q$. (ii) The expected contribution of a single bit z to an up/down counter is $(2z-1)$. Thus the expected change of Y per cycle is $2(X+2^{(Q-1)})/2^Q-1 = x$. If W is

negative, the x -bitstream is inverted, resulting in the negated value of $x(-W)$, or xW in the Y counter. In either case, $y \approx xw$ with Q -bit precision (including the sign bit).

4.1.2. Extension to DPS SC-MAC

The extension to support dynamic precision scaling (DPS) on the SC-MAC level costs very little hardware. In fact, the datapath remains almost the same, the major difference being in the control logic. Here, we explain the operation of DPS SC-MAC.

The DPS SC-MAC has an additional input p , which is the precision of the input/output values. Figure 16 illustrates an example with $p = 3$ where the maximum precision Q is 4-bit.

Let us first consider the unsigned version. The X register holds the integer version of x , or $x2^p$, aligned at the MSB. The remaining bits, if any, are not used. The W register is initialized to the integer version of w , or $w2^p$, zero-extended to fill the Q -bit register, i.e., $W=w2^p$. The selector FSM is unchanged regardless of p .

It is easy to see that the latency of multiplication is $W = w2^p$ cycles, which is at most 2^{p-1} . During this period, the LSB part of X register that is not initialized from x , is not used (x_0 in the example). Thus, counting the x -bitstream still approximates xW , with less accuracy due to reduced precision of W . Since $Y \approx xW = xw2^p$, $y \approx xw$ with p -bit precision.

In the signed version, the X register holds $x2^{(p-1)}$ aligned at MSB. After inverting the MSB, the unsigned interpretation of X register is $0.5+x/2$ with p -bit precision, assuming the decimal point right before MSB. The W register is initialized to $w2^{(p-1)}$, sign-extended to Q -bit, i.e., $W = w2^{(p-1)}$. If W is positive, the Y counter holds $xW = xw2^{(p-1)}$ after W cycles. If W is negative, it holds $-x(-W) = xW$ due to the XOR gate. In either case, $y \approx xw$ with p -bit precision including the sign bit.

The latency of signed multiplication is $|w|2^{(p-1)}$ cycles. The maximum latency is $2^{(p-1)}$ when $w = -1$, in which case the W counter requires p -bit, as indicated by the dotted box in Figure 16-(b) (but it never needs more than Q bits). Similarly, the Y counter needs $p+1$ bits, which may exceed Q bits; however, the Y counter has extra bits already in order to serve as an accumulator.

Since the p -bit precision of y always starts from LSB, it means that the decimal point will have to move depending on the precision. Fixing the decimal point can be done with a single shifter.

4.1.3. Half-Range Specialization for DPS SC-MAC

Since the effect of precision in SC is exponential, saving even 1 bit is worthwhile. Half-range specialization (HRS) is based on the observation that the range of certain variables, namely, input activations, are guaranteed to be non-negative due to the particular shape of activation function (ReLU) used in the preceding layer.

One way to exploit the limited range of input is through a data scaling framework as in Section 4.1.4. But data scaling works best for symmetrical ranges, and making asymmetrical ranges symmetrical incurs additional overhead.

Alternatively, we can make a full use of input precision in our DPS SC-MAC of Figure 16-(b) by treating x as unsigned. This effectively increases x 's precision top-bit while the precision of w remains the same (i.e., $(p-1)$ -bit due to 1-bit sign). Since w is unaffected, latency is also the same. The main effect of this scheme is accuracy improvement: in our evaluation, $(p-1)$ -bit multiplication with HRS shows a similar accuracy as p -bit multiplication without HRS (see Figure 33). Conversely, HRS can achieve a similar accuracy with 1-bit less precision, or at half the latency.

It is important to note that HRS cannot guarantee the same accuracy as that of 1-bit lower precision, since w 's precision is not increased. However, in most layers and most DCNNs, input activations turn out to require a higher precision than weight parameters [51], which explains why increasing x 's precision through HRS is very effective in practice.

Also, important to note is that in order to support layers whose input is not necessarily one-sided (e.g., the first layer), the hardware must retain the original behavior of Figure 16-(b). Thus, we make HRS runtime-programmable through an extra input XIS (meaning “ x is signed”), as shown in Figure 17.

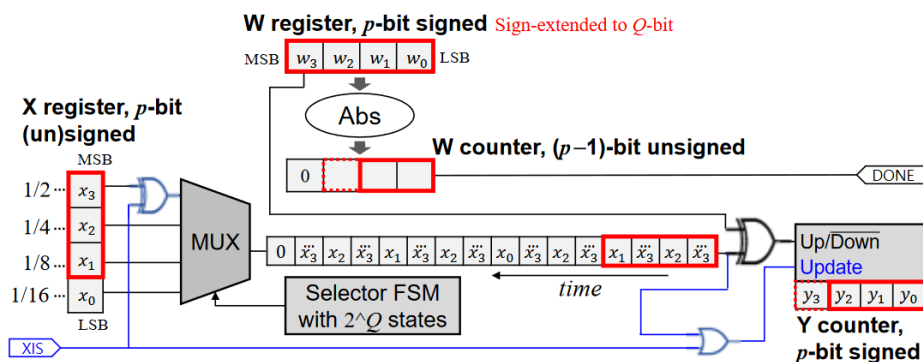


Figure 17: Supporting HRS (Half-Range Specialization) mode for x along with the normal mode.

Extension is shown in blue. The new input XIS indicates whether x is signed.

When XIS is 1, the hardware degenerates into the signed version. When XIS is 0, the x part becomes like the unsigned version, and the up/down operation of the Y counter is suppressed if the x -bitstream's output is 0, essentially making it perform either up or down, depending on the sign of w , which ensures a correct operation.

4.1.4. Design Optimization for DPS SC-DCNN

1) Hardware Precision vs. Software Precision: So far, our notion of precision has been the width of a variable in the application program as represented in conventional digital, which typically ranges up to 32-bit. Quantization is to reduce the precision in the application code. Thus, this kind of precision maybe called *software precision*. When converted into SC, a variable of n -bit software precision requires about a 2^n -bit bitstream.

While a bit-serial multiplier as in [35] computes only one bit at a time, it is quite common in SC to employ bit-parallel hardware. To generate a variable of n -bit software precision, a bit-serial SC multiplier needs 2^n cycles, but a k -bit parallel SC multiplier can do it in $2^n/k$ cycles. We define *hardware precision* as the base-2 logarithm of k .

We have extended our DPS SC-MAC to a bit-parallel version, which supports integer hardware precisions for efficiency reasons. It is based on the bit-parallel version of the baseline SC-MAC [16].

2) Design Flow: Our design objective is to minimize ADP while meeting accuracy constraint, which we set to be 1% point below the reference accuracy achieved by an unquantized version (i.e., floating-point implementation). We consider the following design parameters: (i) data scaling parameters, (ii) software precision of each layer, and (iii) hardware precision of SC-MAC. Next, we discuss each of these.

3) Determining Data Scaling Parameters: Previous work [52] has pointed out the importance of scaling input data to better utilize the limited range of SC or fixed-point representations. The idea is to scale more than covering the worst-case input data, such that some of the input values go out of range. It may introduce errors to some input, but those in the range can be represented more precisely. More-than-worst-case scaling is particularly effective when the out-of-range input data get saturated. To avoid the overhead due to scaling, scaling parameters are typically restricted to powers of 2.

The issue here is how to determine data scaling parameters, the effect of which seems highly unpredictable. We use the following scheme. (1) Determine the scaling factor so that all values are within range (i.e., worst-case design). (2) Double the scaling factor and check whether the recognition accuracy improves. (3) Repeat the above while there is improvement.

The above procedure is repeated for each layer, starting from the first layer. We do not retrain the DCNN during this procedure. We find this scheme robust as it does not rely on any arbitrary design parameter, which is a major advantage of the scheme. While this algorithm is greedy and not able to address the possible inter-dependence issue among layers, doing so would run into a combinatorial problem which can require prohibitive amount of resource for large DCNNs.

4) Determining Software Precision of Each Layer: Similar to data scaling parameter exploration, here we optimize one layer at a time in order to avoid combinatorial problems. There are also differences.

First, precision optimization uses retraining, which is crucial to get meaningful accuracy at low precisions. On the other hand, retraining takes much longer than inference, and can take hours and days for SC even when using GP-GPUs for simulation. Second, higher precision is more detrimental than a lower precision can save. Thus, we first find the uniform precision for the SC version that satisfies the accuracy constraint with retraining. This can be solved in linear time, since all layers have the same precision. The uniform precision is used as the precision upper-bound for each layer. Third, knowing the uniform precision also helps determine hardware precision (see the next section). Fourth, to speed up the search we use the result of conventional digital implementation’s optimized precision. However, since there is usually a gap between the precisions of the two, we use a concept called *precision slack*.

To illustrate precision slack, suppose that a conventional digital implementation is optimized to have the following precisions across 5 layers: 10-9-5-6-8. Precision slack is the difference in precision between the highest and the current layer, e.g., 0-1-5-4-2 in this ex-ample. Then we subtract precision slack from the uniform precision value to get the precision lower-bound. The rationale is that while SC gives higher reward for lower precision, its accuracy is also more sensitive to it. Also using very low precisions gives diminishing return (see Figure 18) while often hurting accuracy too much. Having a lower-bound enables a binary search instead of a linear search, saving retraining time.

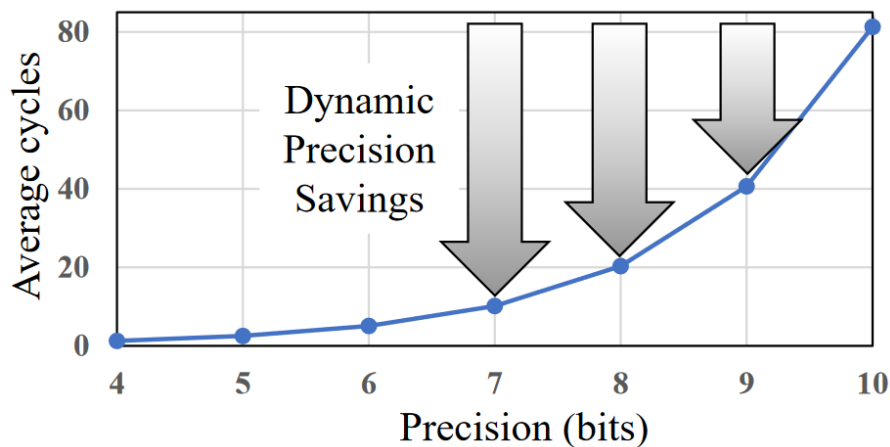


Figure 18: Dynamic precision, or using lower precision whenever possible, can give huge boost in efficiency for SC.

5) Determining Hardware Precision: Hardware precision affects both delay and area of our DPS SC-DCNN (see Section 4.4.2). Therefore, the decision in this step could affect the optimality of the precision setting found in the previous step as ADP can change as we use a different hardware precision. To avoid this problem, we run this step twice, first for the uniform precision value, then after non-uniform precision setting is found. Finding the best hardware precision is straightforward, and can be

done quickly as it has only a linear complexity and does not require retraining. (Changing hardware precision does not affect recognition accuracy.)

4.2. Log-Quantized Stochastic Computing

Here, we present logarithmically quantized SC-DCNN accelerator and explain how it is improved compared to the state-of-the-art SC.

4.2.1. Application of Log-Quantized Weight to SC-MAC

The starting point of log-quantized SC is storing weights as logarithmic representation while using the same SC-MAC. Figure 19 is a block diagram of proposed SC-MAC when the logarithmic weight is given. Because the baseline SC-MAC only takes linearly quantized fixed-point binary number, log-to-linear converter should be added before the weight down-counter, where

$$|w| = \begin{cases} 2^{-|\tilde{w}|} & \text{if } |\tilde{w}| \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

from Section 2.3. This converter takes 4-bit of weight’s magnitude and forwards 15-bit linearly quantized number to the down-counter. MAC operation is working identically same with the baseline unipolar (unsigned) SC which is explained in Section 3.2, except that sign bits are XORed to determine whether 1s will be added or subtracted as we can see red lines in Figure 19. For now, weight data size which affects bandwidth and model size is just reduced (16→5-bit per word) without loss of MAC precision.

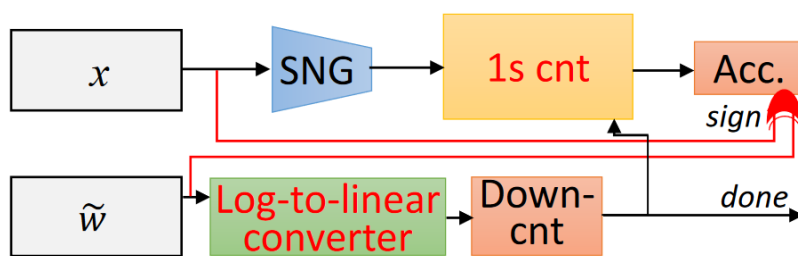


Figure 19: Proposed log-quantized SC-MAC (red color means where it is modified).

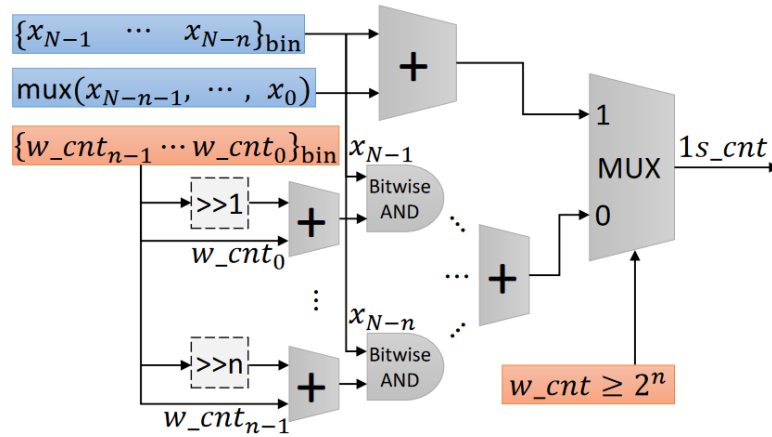
4.2.2. Improvement of 1s Counter

The possible input set is reduced ($2^{16} \rightarrow 2^5$) due to the logarithmic quantization. Utilizing this fact, we mathematically analyze 1s counter which is main area bottleneck and simplify it.

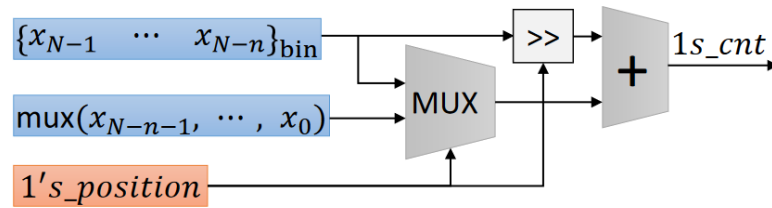
Suppose input magnitude precision is N -bit and the magnitude is represented as binary number, $\{x_{N-1} \cdots x_0\}_{\text{bin}}$. We parallelize 2^n cycles. In the baseline [16], 1s counter value is determined by the case, where

$$1s_cnt = \begin{cases} \{x_{N-1} \cdots x_{N-n}\}_{\text{bin}} + \text{mux}(x_{N-n-1}, \cdots, x_0) & \text{if } w_cnt \geq 2^n \\ \sum_{i=1}^n x_{N-i} \times \text{round}\left(\frac{w_cnt}{2^i}\right) & \text{otherwise.} \end{cases}$$

Let's look at the digital logic implementation of it, Figure 20-(a). When the weight down-counter value is greater than or equal to 2^n , it is relatively simple. However, otherwise, there are many number of small logics including rounding operations which are actually adding 1 when the last truncated bit is positive. Because it is hard-wired, shifter takes no logic area (dotted line boxes). A multiplier consists of n bitwise AND gates. There are $2n$ adders with different bitwidth for each. For $n \leq 8$, their area is still much less than fixed-point binary multiplier [17].



(a) Baseline 1s counter



(b) Proposed 1s counter for the reduced input set

Figure 20: Logic implementation comparison of 1s counter.

Because weights are logarithmically quantized, the number of 1s in the initial weight down-counter value is no more than 1. Furthermore, by the fact that the counter decrements by 2^n , the number of 1s in

n LSBs of the weight counter should be one or zero always. When the weight down-counter is 0, we can just stop the update of an accumulator. Let's handle the case where there is a 1 in LSBs. Let's rephrase $\sum_{i=1}^n x_{N-i} \times \text{round}(\frac{w_cnt}{2^i})$ as

$$\sum_{i=1}^n x_{N-i} \times (w_cnt \gg i) + \sum_{i=1}^n x_{N-i} \wedge w_cnt_{i-1}.$$

Suppose 1's position of LSBs is p (0 means LSB) which can be implemented in the shared weight converter. The equation will be

$$\sum_{i=1}^n x_{N-i} \times (2^p \gg i) + x_{N-p-1}.$$

The first term is equal to $\{x_{N-1} \cdots x_{N-p}\}_{\text{bin}}$. That can be implemented as a small shifter in Figure 20-(b). Latter term can be implemented with MUX. One more merit is that shifter+MUX structure can be used when $w_cnt \geq 2^n$. Compared to the baseline, we make huge and novel design reduction.

4.2.3. Log-Quantized SC-MVM

Figure 21 is a block diagram of a proposed SC-MVM architecture with 4 SC-MACs. \vec{x} includes 4 elements. Because SC-MACs share a weight, just one log-to-linear converter before the weight down-counter is enough. An FSM for SNG also can be shared which is not shown in the figure. Sign bits are treated separately as we do in a Section 4.2.1. Indeed, sharing a weight is effective in many aspects for SC-DCNN accelerators.

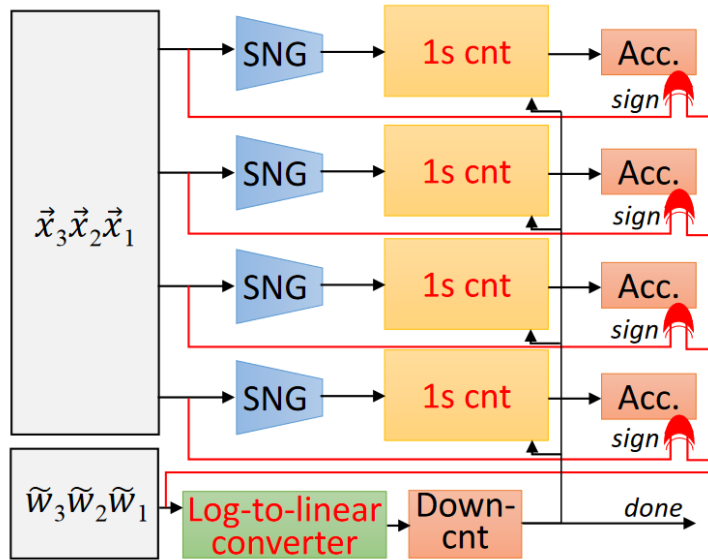


Figure 21: Proposed log-quantized SC-MVM (red color means where it is modified).

4.2.4. Log-Quantized SC Multiplication Accuracy Analysis

1) Quantization Error vs. Multiplication Error: It is interesting to see how SC multiplication accuracy affected when the input set is reduced to the power of 2. Figure 22 shows multiplication error that comes from SC in terms of mean square error. Uniform-distributed 16-bit (including sign bit) fractional numbers are multiplied to weights of AlexNet’s first layer. There are two types of error we can measure. For **quantized multiplication error** (quant+mult. error), true multiplication outputs are computed by multiplying floating-point weight. In contrast, linearly or logarithmically quantized weights are multiplied as true outputs to calculate **multiplication error**. In linearly quantized SC, quantized multiplication error is similar with multiplication error. It means that linearly quantized SC multiplication error is mainly from the faulty multiplication itself. However, in logarithmically quantized SC, there is high gap between quantized multiplication error and multiplication error. That indicates that the error is mainly originated from logarithmic **quantization error**. Multiplication error is lower than linearly quantized SC and achieves almost the same level of shifter-based multiplication accuracy which is fixed-point multiplication for a logarithmically represented multiplicand at one side. Quantized multiplication error of logarithmically quantized SC decreases at first, but begins to saturate because quantization error couldn’t be improved with high data precision. For example, 0.7 is always quantized to 0.5 even though data precision increases.

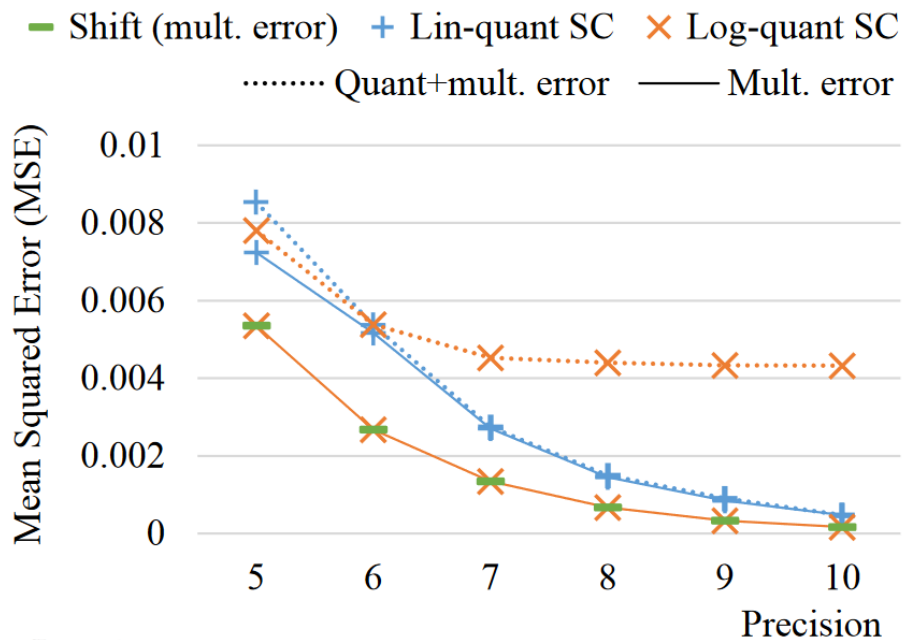


Figure 22: Multiplication error against floating-point weights for linearly and log-quantized SC.

2) Fine-tuning to Recover DCNN Accuracy from Quantization Error: Small bitwidth quantization methods usually requires re-training or fine-tuning to restore original network accuracy. Approximate and faulty accelerators such as SC also can achieve higher network accuracy with fine-tuning. It is worth to remark that the network accuracy is more resilient to logarithmic quantization error than multiplication error. Therefore, we argue that precision requirements can remain almost same for logarithmically quantized SC compared to the state-of-the-art SC. Experimental results in Section 4.3.3 support this claim.

4.3. Successive Log-Quantization and Its Application to SC

4.3.1. Analysis and Motivation

1) Analysis of Quantization Error: Consider an N -bit number q that is quantized from a real-valued number x in $[-1,1]$. We can see q as an integer in $[-M, M-1]$, where $M = 2^{N-1}$. Table 7 shows how x may be reconstructed from a quantized number q , for linear and log-quantization. Mathematically, the reconstructed value is $\tilde{x} = q/M$ in the case of linear quantization, and in the case of log-quantization (assuming the log-base of 2 for brevity):

$$\tilde{x} = \begin{cases} \text{sign}(q) 2^{-|q|}, & \text{if } q \neq 0 \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Table 7: Comparison of linear vs. logarithmic quantization.

q	$-M$	$-(M-1)$	\dots	-1	0	1	\dots	$(M-1)$
LinQ	-1	$-(1 - \frac{1}{M})$	\dots	$-\frac{1}{M}$	0	$\frac{1}{M}$	\dots	$1 - \frac{1}{M}$
LogQ	-2^{-M}	$-2^{-(M-1)}$	\dots	-2^{-1}	0	2^{-1}	\dots	$2^{-(M-1)}$

From the table, one can see that linear quantization distributes quantized values uniformly across the dynamic range $[-1,1]$, whereas log-quantization places most of them near zero. As a result, while linear quantization has constant precision as measured in ULP (unit of least precision; the spacing between two consecutive numbers), which is $1/M = 2^{-(N-1)}$, log-quantization's ULP varies greatly from $2^{-(M-1)}$ to 2^{-2} . Consequently, they have drastically different quantization errors as shown in Figure 23-(a). Since the maximum quantization error is half of ULP, for small values of $|x|$ (the threshold is around $|x|=2^{-(N-2)}$) log-quantization's error is less than or equal to that of linear quantization, but for larger values log-quantization's error tends to be much greater.

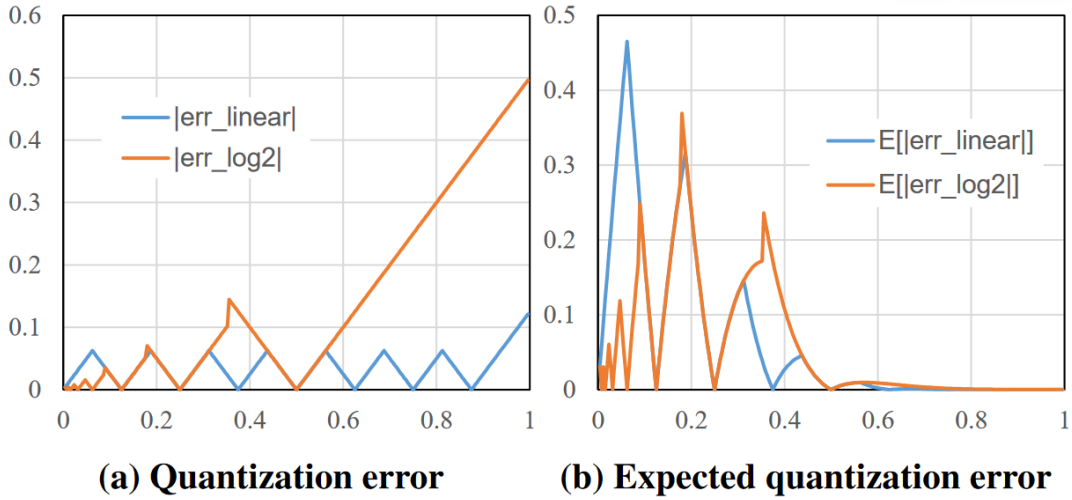


Figure 23: Quantization error comparison for $N = 4$, assuming $x \sim N(0, 0.2^2)$ in (b).

There are two implications. First, log-quantization can be more accurate than linear quantization if most of the input occurs near zero, such as weight parameters of some DCNNs. Figure 23-(b) shows the expected error for Gaussian input. In this particular example, log-quantization has lower total quantization error (the area under curve) than linear quantization. Second, however, at higher resolution (i.e., when N is high), the range of input where log quantization is more accurate than, or as accurate as, linear quantization, decreases exponentially. In fact, using higher resolution does not reduce log-quantization’s error at all, except for $|x| \leq 2^{-(M-1)}$, which approaches zero very rapidly.

Using a smaller base of logarithm (e.g., $\sqrt{2}$) [12] can help reduce quantization error. However, this is not as effective as in linear quantization due to the uneven precision distribution, and can jeopardize the simple multiplication advantage of log quantization

In summary, while log quantization has certain advantages especially at low resolution, there is no effective way to increase the precision of log-quantization, which severely limits its usability for large DCNNs. To address this problem, we next present our novel extension to log-quantization.

2) Our Approach: Successive log-quantization (SLQ) is to represent a value with a series of quantized words, where the length of the series is dependent on the input value. The idea is that for the most part just one quantized word should suffice, but when quantization error is large, we can use additional quantized words to reduce quantization error.

Successive quantization can be defined for linear quantization as well, but due to the uniform precision across the full dynamic range, there is very little incentive to do so. Log-quantization, on the contrary, have greatly varying precision, and consequently varying quantization error. Also, these large quantization errors are a statistically rare event, thus it makes sense to make additional quantizations optional, as opposed to increasing the resolution of quantization.

Another important appeal of SLQ is that it has very little hardware overhead, since additional quantizations can be performed using the same datapath over multiple cycles. While combining the reconstructed values from multiple quantizations will require addition operation, this addition can be implemented using the accumulation circuitry that is already present in DCNN inference hardware.

3) Stochastic Computing: In stochastic computing a number $x \in [0,1]$ is represented by a bitstream $X = X_1X_2\cdots$, whose signal probability, or the probability of X being 1, matches x , i.e., $P(X=1) = x$. Over the years many SC circuit elements have been developed, but the minimum we need to implement a convolution/fully-connected layer is: stochastic number generator (SNG), SC multiplier, and SC-to-binary converter (i.e., counter). An SC adder can also be used, but due to its limited dynamic range we use a conventional-binary accumulator, which does addition as well.

Figure 24 illustrates a highly efficient SC-MAC architecture [16], in which the whole block including two SNGs, an SC multiplier, a counter (for SC-to-binary conversion), and an accumulator is optimized into what is shown in the figure. The idea is that by counting the number of 1s in X (SC bitstream of x) for w cycles, one can calculate $x \cdot w$ fairly accurately. While the bit-parallel version is more cost-efficient, it requires a rather complex 1s counter, because it must count n bits of X simultaneously, where n is the smaller of the degree of bit-parallelism and the down counter value.

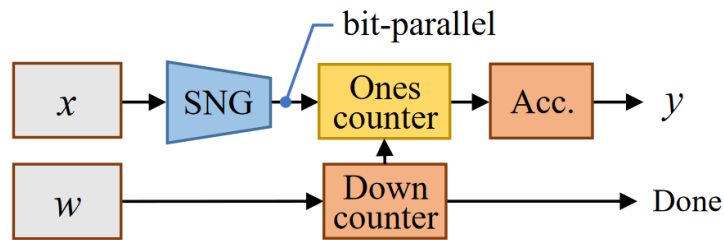


Figure 24: SC-MAC [16]. Ones (1s) counter is absent from the bit-serial version.

4.3.2. Successive Log-Quantization

1) Definition: SLQ approximates a real number x with a series of log-quantized numbers $\{q_i | 1 \leq i \leq k\}$ such that

$$x \simeq \tilde{x}_k \equiv \sum_{i=1}^k \text{LogDequant}(q_i) \quad (2)$$

Where LogDequant is any log-to-linear conversion function such as (1).

Table 8 shows examples. Note that by allowing negative q_i , SLQ can better represent numbers on both sides of 2^{-i} such as 0.4375, which can be represented with two words as opposed to three, which would have been the case if negative numbers were allowed only in the first word.

Table 8: Quantization examples.

Real	Log Quantization		SLQ	
	FP5	Recovered	FP5+	Recovered
0.5	00001	0.5	00001	0.5
-0.25	11110	-0.25	11110	-0.25
0.4375	00001	0.5	00001 11100	0.4375
0.4375	00001	0.5	00010 00011 00100	0.4375

We refer to the resolution used in the underlying log-quantization as the *base resolution* of SLQ. The effective resolution is determined by the base resolution and the average length of SLQ series. SLQ series may have different lengths, leading to the encoding problem (see next section).

We perform linear-to-SLQ conversion by successively applying log-quantization to the current residue of SLQ, controlled by two parameters: threshold θ and max length L . Each quantization step is given as

$$\tilde{x}_0 \equiv 0 \quad (3)$$

$$q_i = \text{LogQuant}(x - \tilde{x}_{i-1}), \quad i \geq 1 \quad (4)$$

which is repeated as long as the residue $(x - \tilde{x}_i)$ is greater than θ , up to $k \leq L$. In our experiment, SLQ with $L = 2$ turns out to be good enough for $\theta = 1/128$, as shown in Figure 25.

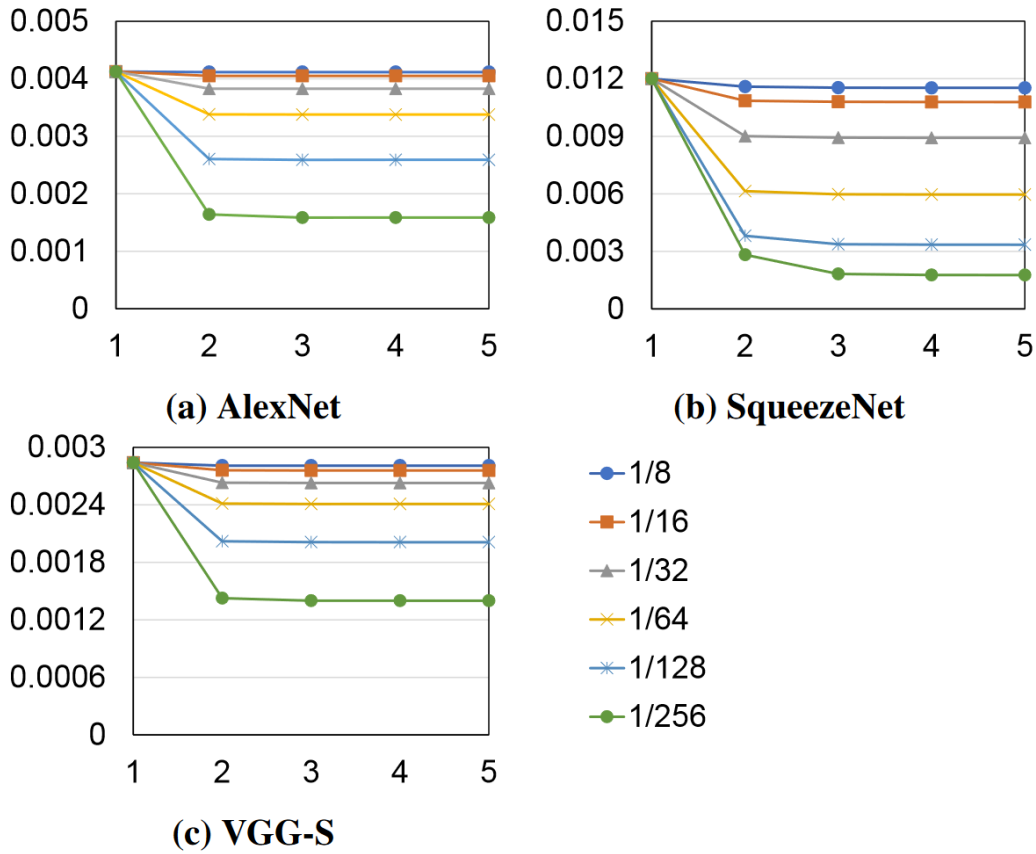
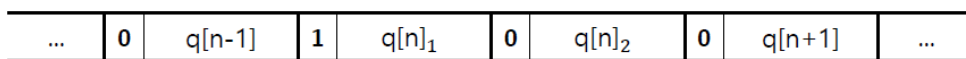
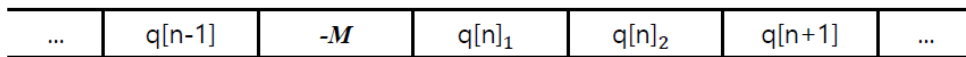


Figure 25: RMS error vs. L (see Section 4.4.4 for methodology).

2) Encoding: We propose two schemes to specify the length of SLQ series as illustrated in Figure 26.



(a) 1-bit tagging



(b) Using a special code ($-M$)

Figure 26: Encoding schemes.

One method is *Tagging*. Tagging adds a 1-bit field to each quantized word q_i to specify whether it is followed by the next quantized word q_{i+1} of the same series. The advantage of this scheme is that hardware design is relatively simple, since every word that is loaded can be used for computation in the

current cycle and prefetching the next word is not necessary (see Section 4.3.3). However, 1-bit overhead can be quite significant due to the small word size (typically 4~5-bit) especially if the majority has length one.

The other is using *Special Code*. If most series have only one quantized word, it can be more economical to use a special code to signal the existence of multiple quantized words. Assume that $L = 2$. This scheme uses one of the $2M$ possible quantization symbols (e.g., $-M$) as the special code, and when the special code is encountered, the next two words are interpreted as the quantized words. While not using the special code for single-word quantization may diminish the accuracy of quantization, this loss of accuracy is small, and can be recovered by using double-word quantization. The special code scheme has less memory overhead than tagging if the double word ratio is less than 33% (or 25%) with 4-bit (or 5-bit) word size.

4.3.3 SLQ HARDWARE

We present hardware modification to incorporate SLQ in DCNN inference engines. Following [13] we assume that weights are log- or SLQ-quantized and activations are linear-quantized, which is appealing because weight parameters are more concentrated around zero and inaccurate log-domain addition can be avoided. While SLQ is also applicable to conventional-binary MAC, in this paper we focus on SC-DCNNs.

1) SLQ for Conventional-Binary MAC: In addition to superior low-resolution performance, log-quantization allows for simpler multiplication using shift. As illustrated in Figure 27, our SLQ can reuse the datapath of log-quantization with one exception: when $k > 1$, the input activation must be held until sub-sequent log-quantized words, q_2, q_3, \dots , are all processed, which then correctly realizes (2).

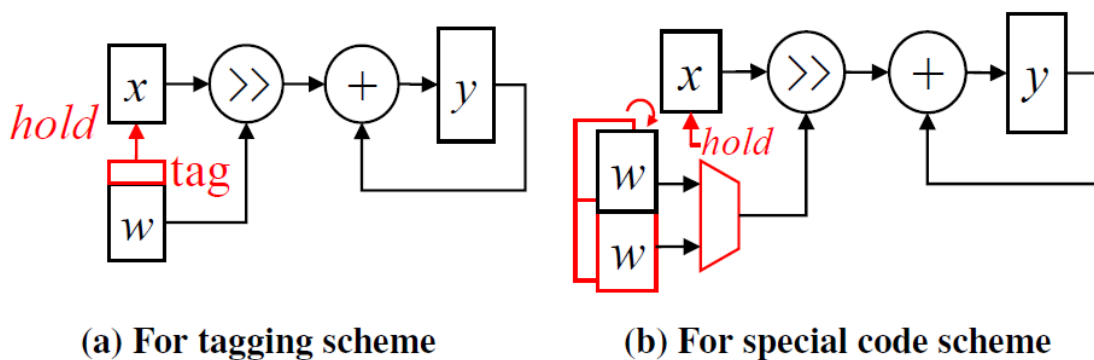


Figure 27: Conventional-binary MAC with SLQ (red: extension).

2) SLQ for SC-MAC: For a straightforward implementation, we first convert the log-scale weight parameter to linear scale using log-to-linear converter as [18], which is essentially a shifter, and then

the linear weight is fed to the SC-MAC, as illustrated in Figure 28. The difference between SLQ weight vs. log-weight SC-MAC is that in the former we use each of $\{q_i\}$ values, which requires holding the input activation when $k > 1$. Thus SLQ-weight SC-MAC reuses the datapath of log-weight SC-MAC, but with more complex control due to the detection of multi-word quantization and input/weight register management.

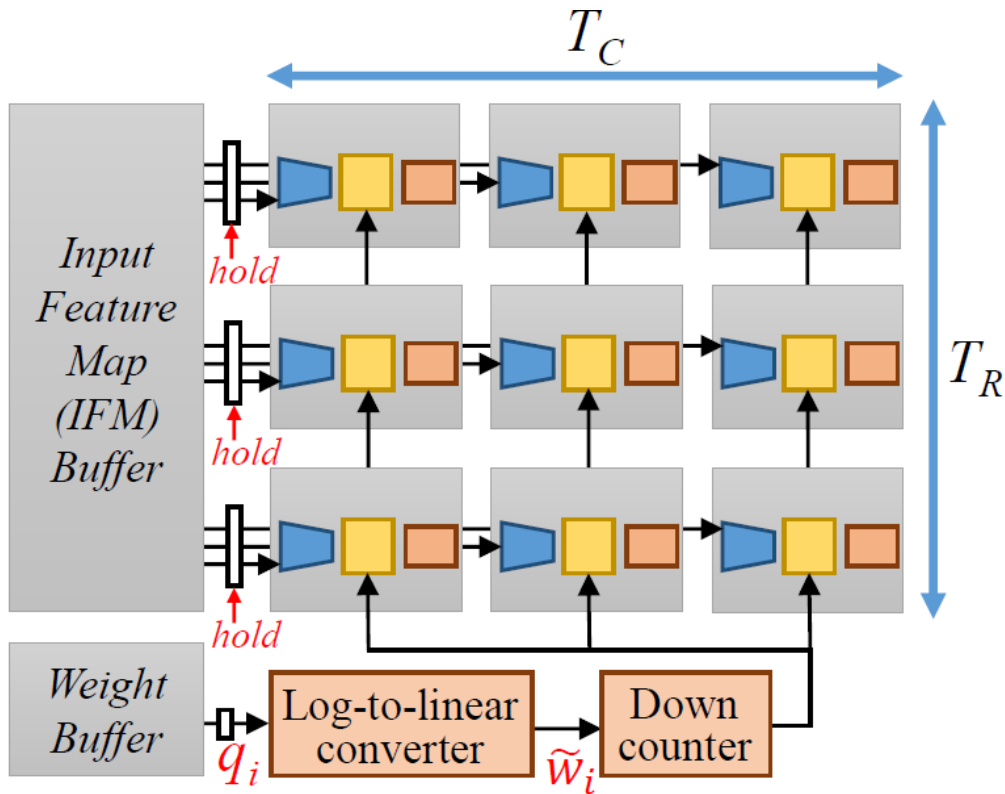


Figure 28: SC-MVM using SLQ for weight parameters.

The hardware advantage of using log-scale weights is that we can simplify the 1s counter, (which is the largest component in a bit-parallel SC-MAC) exploiting the fact that a weight converted from a log-scale value can have at most one 1-bit. In particular when the down counter value (DC) is less than 2^p (degree of bit-parallelism), it must count DC bits of X , which now can be implemented efficiently using a MUX. This is in part due to the way SNG in Figure 24 [16] is constructed. Log-weight SC-MAC hardware is analogous to shifter-based MAC, and achieves similar reduction in hardware complexity.

3) SLQ-based SC-MVM: One challenge in applying SLQ to DCNN inference is how to handle variable latency of our SLQ scheme. Interestingly the baseline SC-MAC in Figure 24 also has variable latency, determined by the weight value. Thus, by reusing the same SC-MVM architecture of the baseline [16] that uses linear weights, we can expect to achieve accuracy improvement and hardware

cost/power reduction without significant degradation in execution speed. The runtime of our SC-DCNN using SLQ will be higher than that of using linear/log weights. Assuming special code scheme and $L = 2$, the runtime increase is proportional to the ratio of double-word quantizations.

The SC-MVM architecture targets convolution layers, parallelizing along the image row (T_R) and column (T_C) dimensions (see Figure 28). The architecture can exploit weight reuse very naturally, and input reuse can be exploited by using a known technique such as [45].

Our SLQ increases model size slightly compared with linear/log quantization. Assuming special code scheme, $L = 2$, and the same (base) resolution ($= N$), the increase is $2rN + 10M$ bits, where r is the number of double-word quantizations and M is a DCNN hyperparameter for the length of the output channel (i.e., the number of output features) of a layer. The last term is to facilitate prefetching of the next weight parameter *chunk* from the off-chip memory (a weight parameter tensor is partitioned into M chunks, and the size (or offset) of each chunk is represented by a 10-bit integer).

4.4. Evaluation

4.4.1. Experimental Setup and Implementation Detail

To evaluate our approach, we use ImageNet-targeting DCNNs (AlexNet, VGG, GoogLeNet) in addition to smaller ones for comparison with previous work. For training and accuracy evaluation we use Caffe [47] extended for SC. Recognition accuracy is reported for the first 10,000 images of the ImageNet validation set. SC architecture is modeled cycle-accurately to generate exact cycle count in a data-dependent manner.

We have extended the SC-MVM (Matrix-Vector Multiplier) in [16] to use our DPS SC-MACs, which is referred to as DPS SC-MVM. The previous SC-MVM, our DPS SC-MVM, and the conventional digital baseline MVM are all implemented in Verilog and synthesized using Synopsys Design Compiler. All syntheses are done for the same target frequency, 1GHz, although SC is more likely to meet higher frequency. The conventional digital baseline uses fixed-point binary multipliers with rounding accumulators. The area for Stripe [35] is estimated to be 207% of the digital baseline as per their paper. Only convolution layers are accelerated in all the approaches compared, permitting us to use ideal convolution layer speedup for delay comparison. Accuracy degradation is set to 1% point.

Our main figure of merit is area-delay product (ADP), which is the product of MVM area and average MAC cycles, or its inverse representing operations-per-area.

4.4.2. Dynamic Precision Scaling

1) Area Overhead of Our DPS SC-DCNN: Figure 29 compares the area of our proposed DPS SC-MVM against the previous SC-MVM [16]. The DPS SC-MVM includes our optimizations such as HRS,

which have very small extra logic (see Figure 17). Not surprisingly, the graph shows that the area overhead of ours is mostly small, typically at around 5%, though varied depending on the hardware precision shown on the x -axis. The graph also shows that the area is linearly proportional to the hardware precision, or logarithmically to bit-parallelism. This is due to the optimization exploiting the structure of the bitstream ordering. Overall the average area overhead is 6%, which is small.

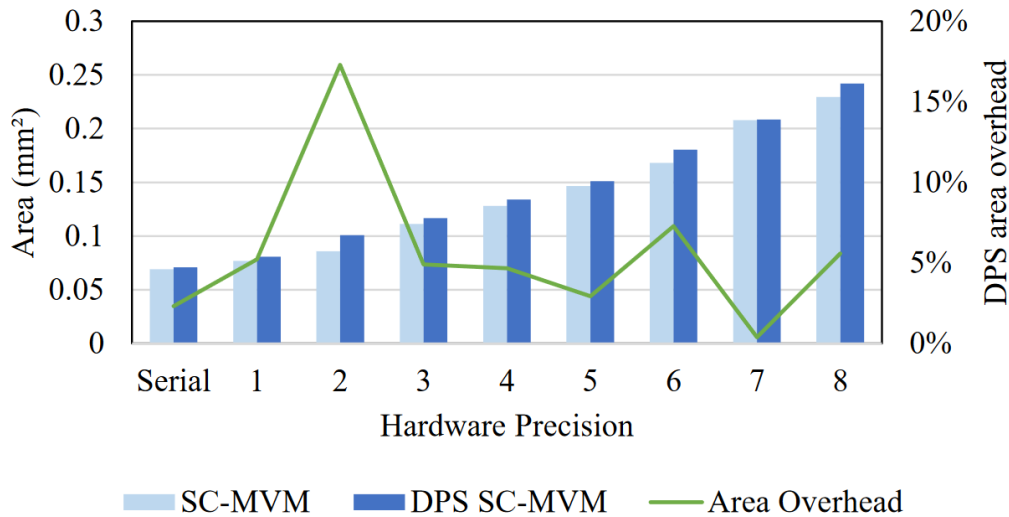


Figure 29: Area overhead of DPS-MVM.

2) ADP vs. Software and Hardware Precisions: Figure 30 shows the ADP trend as we vary software precision. For our ADP result we use AlexNet parameters as our SC-MAC has data-dependent variable latency. The digital baseline does not support dynamic precision, thus constant ADP. For stripe, delay is proportional to the precision, resulting in linear ADP. The graph shows that Stripe becomes inefficient over the conventional digital baseline beyond 7- or 8-bit (①). DPS-2⁴, which is our DPS SC-MAC with hardware precision of 4, confirms the exponential increase of ADP as software precision increases. But the range of software precision for which DPS-2⁴ is more efficient than conventional digital is wider than that of Stripe.

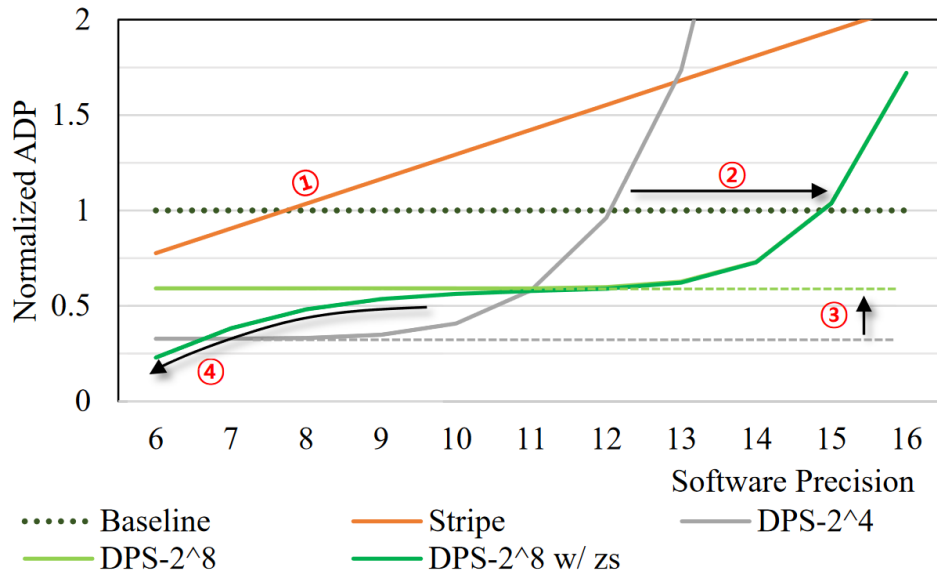


Figure 30: ADP vs. software precision.

DPS-2⁸, which is our DPS SC-MAC with hardware precision of 8, can widen the efficient operating range even further (②). At the same time, it has higher ADP than DPS-2⁴ when software precision is lower (③), as it is more optimized for higher precision workload. Some of the efficiency loss can be reclaimed by zero skipping (④), which is to skip computation of multiplication whose w operand is 0 (after quantization) as shown in the graph.

As demonstrated, ADP depends on hardware precision. As hardware precision increases, (average) delay decreases monotonically until it saturates, whereas area increases more or less linearly to hardware precision. This leads to an optimization issue for hardware precision. Figure 31 shows ADP vs. hardware precision (a) for a single DCNN and (b) for multiple DCNNs.

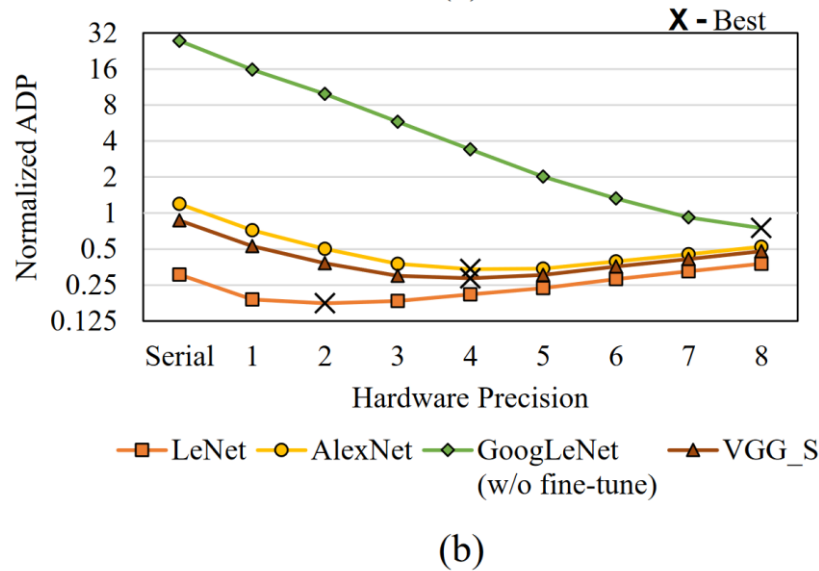
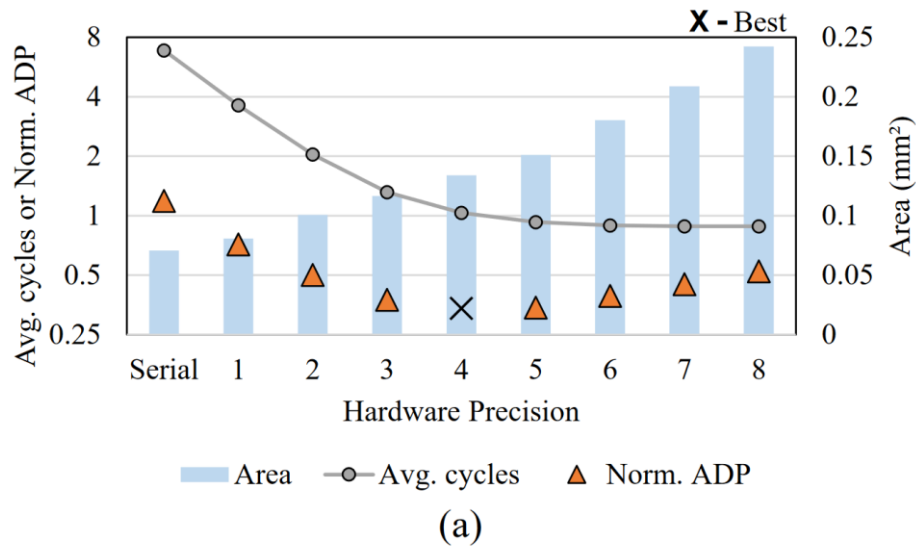


Figure 31: ADP vs. hardware precision.

The first graph is for AlexNet. As hardware precision increases, delay is reduced at first but eventually saturates. In this example DPS-2⁴ is the optimal. But in other DCNNs, different points can be optimal as there are different weight distributions and precision requirements depending on the DCNN. Figure 31-(b) shows how ADP as well as optimal hardware precision changes depending on application. Understandably large and high precision DCNNs seem to be better off with higher hardware precision. Our hardware precision for the multi-application scenario (see the next section) is chosen based on this profile.

3) Multi-application Scenario: Figure 32-(a) compares our DPS SC-DCNN and previous DCNNs. First ours is highly area efficient which is not surprising given the area efficiency of SC. Stripe has the

largest tile area because the number of MACs is 16 times greater than that of others, as shown in Figure 32-(b); the others have 256 MACs only.

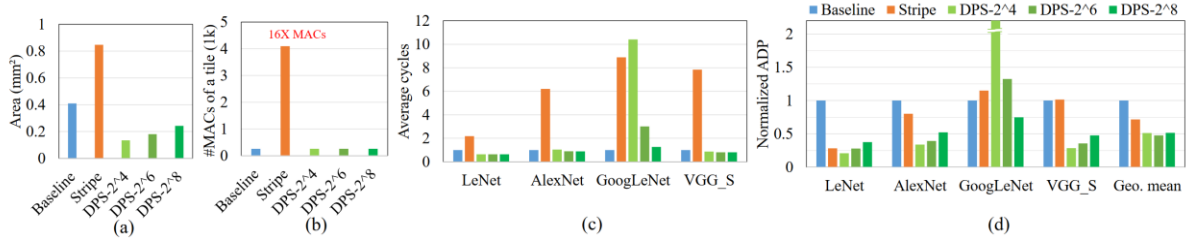


Figure 32: Comparison with the digital baseline and Stripe [35].

Figure 32-(d) shows the ADP result, which is normalized to the digital baseline. First, all these results are from implementations that achieve less than 1%-point accuracy drop from the reference floating-point implementations (see Table 9). That SC-DCNNs can achieve this high accuracy for large DCNNs is very significant. Also, this is why this graph has no comparison with previous SC-DCNNs. Second, at the same time the efficiency of ours as measured in ADP is actually higher than conventional digital, often significantly. For instance, DPS-2⁸, which is optimized for large DCNNs, shows consistently better results than previous digital designs. It also demonstrates the flexibility as well as efficiency of our DPS SC-DCNN. Third, the optimal design as measured in geometric mean of ADP is DPS-2⁶ for this mix of DCNNs, which is obviously influenced by the existence of a small network. But our scheme can flexibly support different workloads through the hardware precision, while simultaneously being able to support dynamic software precision at runtime. Over-all, our DPS-2⁶ achieves more than 2X and 1.5X improvements over the baseline and Strip, respectively, in terms of operations-per-area.

Table 9: Recognition accuracy (for 10K images) and DPS precision setting.

CNN	float	DPS	DPS precisions found
MNIST	0.9904	0.9826	5 (uniform)
AlexNet (top-5)	0.807	0.7999	10-9-8-9-9
GoogLeNet (top-5)	0.8926	0.8844	13 (uniform, w/o fine-tune)
VGG_S (top-5)	0.8341	0.8247	9-9-10-9-10

4) Single Application Comparison: We also compare different DCNNs including the previous state-of-the-art SC-DCNN [16], when they are designed and used for just a single DCNN, with AlexNet as the example. Figure 33 shows area, average delay, and ADP results in one graph, all normalized to that

of the digital baseline. For SC designs, hardware precision is set to 4. Maximum software precision supported (Q) is determined to be the minimum value that meets the recognition accuracy constraint, which is largely dependent on how accurate the MAC is. The digital baseline requires 9-bit while the previous SC-DCNN requires 11-bit. Our DPS SC-DCNN requires 10-bit with uniform precision; dynamic precision setting is listed in Table 9.

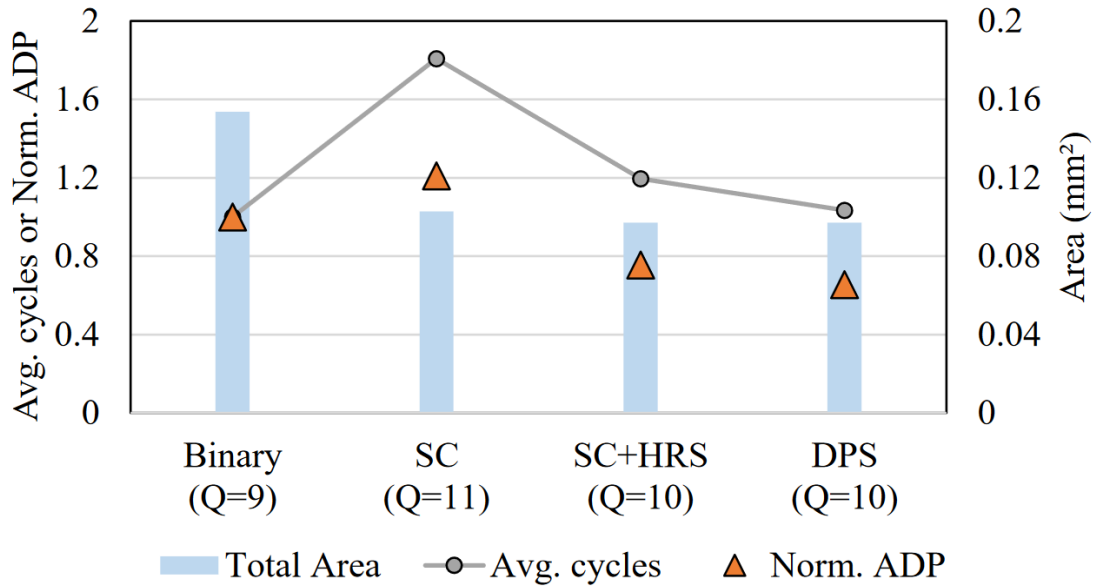


Figure 33: Comparison with previous SC-DCNN on AlexNet.

The graph shows that previous SC-DCNN has smaller area than the digital baseline but its average delay is much higher, which is due to the high precision requirement. Applying HRS to it (but not DPS) can reduce precision requirement by 1-bit with significant saving in delay, but its average delay is still higher than that of conventional digital. Applying DPS further gives 14% more reduction in ADP, achieving the best efficiency. The relatively weak impact of DPS is due to the small number of layers in AlexNet and limited precision exploration. For deeper networks and if optimal precision combinations can be used, the impact is likely higher. Even with these limitations our proposed design achieves 34% and 46% reduction in ADP (or 52% and 85% increase in operations-per-area) over the digital baseline and the previous SC-DCNN, respectively.

4) Fault Tolerance: To evaluate the fault tolerance of our proposed schemes, we have performed an error injection experiment. For the fault model, we assume that random bit flip can occur at the input, as is done in a previous study on SC [56]. The SRAM memories are assumed to be protected such as using hardened logic or ECC (error correcting code). For a given fault rate f , we flip the bits of input registers, whose size varies depending on the scheme, with the same probability f . This fault model is

integrated into the Caffe framework. No retraining is performed, but only inference, in the presence of faults.

Figure 34 shows accuracy degradation for AlexNet as we vary fault rate. First, we observe that SC-based implementations show significantly higher fault tolerance than the conventional digital implementation, which agrees with previous studies [56]-[57]. Second, there is quite a variance among the SC-based implementations. The neuromorphic implementation, which is based on bit-serial SC, shows the highest fault tolerance whereas the bit-parallel version, DPS-2⁴, is less error resilient. There are a number of differences between the two architectures. One relevant fact is that the bit-parallel version performs a weighted bitcount operation to process multiple bits in parallel, which is more like digital logic than SC, and thus may contribute to its lower fault tolerance.

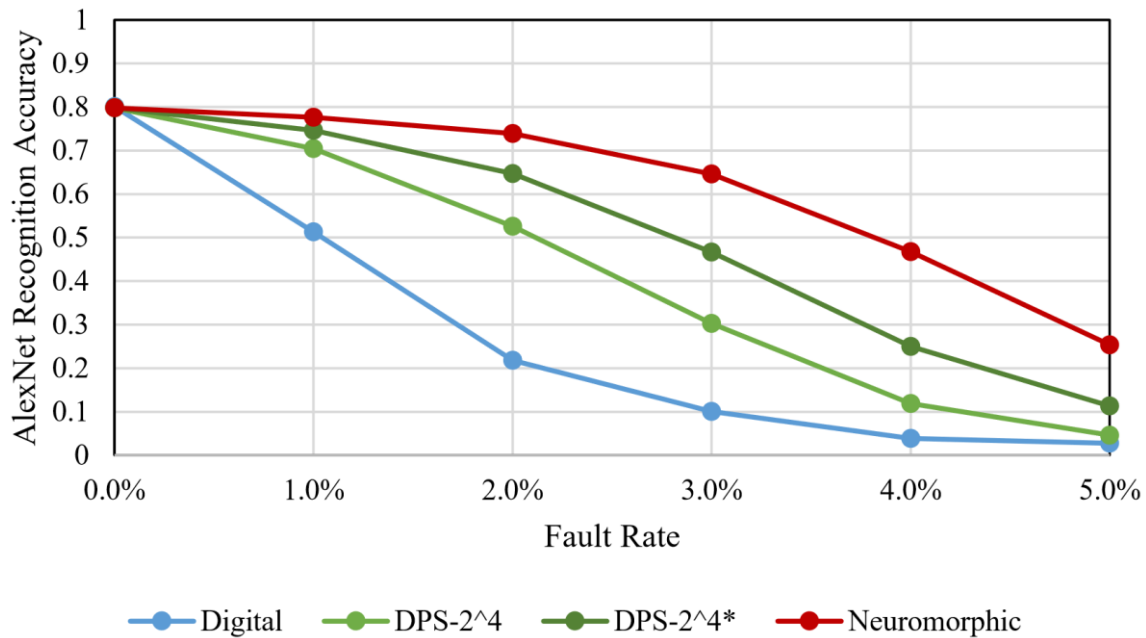


Figure 34: Fault tolerance comparison among different schemes, for AlexNet.

Another difference is that the neuromorphic architecture reloads input every cycle due to the tight SRAM integration. (The neuromorphic architecture has input registers too like the other architectures.) To test if this contributes to the higher fault tolerance, we test a variant of the DPS-2⁴ scheme, denoted by DPS-2⁴*, which is to reload input every cycle even when it is not necessary to do so. Our experimental result in Figure 34 clearly shows that input reloading helps. This may come as a surprise, but while input reloading does not lower the average number of faults in the circuit, it does lower the chance of having *correlated faults*, faults that occur at the same bit position of the input and therefore

are more detrimental to the correctness of computation. The neuromorphic architecture has the least correlated faults, which helps achieve the highest fault tolerance.

4.4.3. Log-Quantized Stochastic Computing

We evaluate area and power of our log-quantized SC-MVM (*Log-SC*) compared to state-of-the-art SC-MVM (*SotA-SC*). To show our design efficiency further, comparison includes fixed-point binary multiplier based MVM (*Fixed*) and shifter based MVM (*Shift*) which is binary version of log-quantized MVM. We also measure energy consumption as well as DCNN recognition rate. Finally, we show off the area cost and latency curve change.

1). Area and Power Efficiency Evaluation: For the situation that logarithmically quantized weights are given, we designed a remarkably lightened 1s counter which was main area bottleneck for the parallelized SC-MAC. Here, we evaluate MVMs consisting of 256 MACs for all comparison cases. We implemented Verilog RTL codes and synthesized those using Synopsys Design Compiler. The target standard cell library is TSMC 45nm and the target frequency is fixed to 1 GHz. Common data precision is 16-bit. There are bit-parallel designs for SC and 2⁴ cycles are parallelized for this evaluation. In other words, hardware precision is 4.

As we can see in Figure 35-(a), fixed-point binary design has large multipliers, taking over 80% of total area. Accumulator area (counter in serial SC) is similar through all cases. Shifter based MVM for logarithmically quantized weights is multiple times smaller. Serial state-of-the-art SC-MVM area is dominated by counters, almost 90%. Bit-parallel state-of-the-art SC-MVM area becomes almost double with the large 1s counters but it is still smaller than *Shift*. 1s counter area is about 1/5 in *Log-SC* and the total area is dominated again by accumulators. Consequentially, area is reduced by 40% compared to the bit-parallel *SotA-SC*.

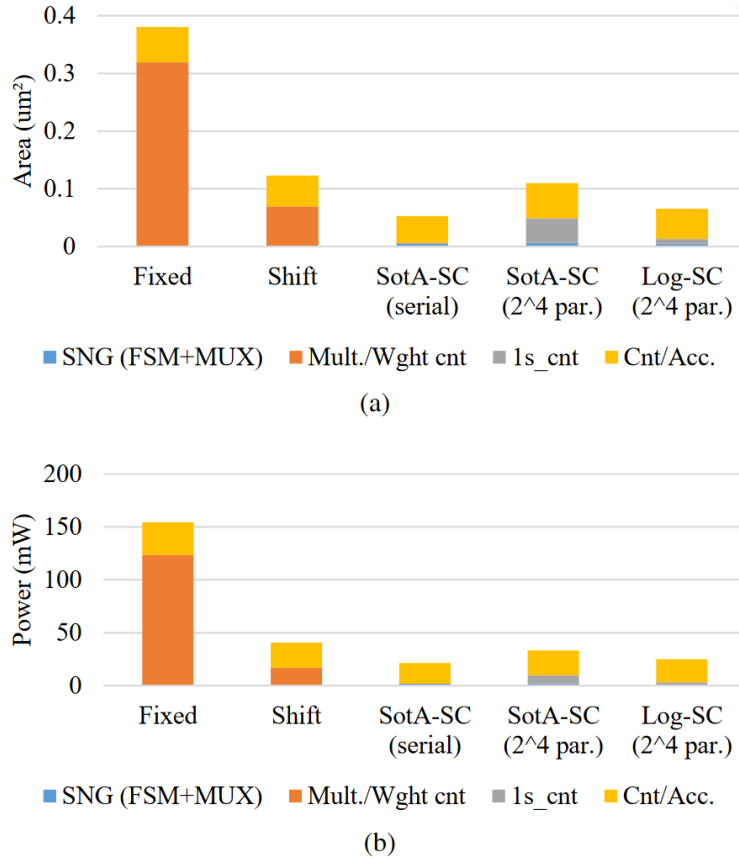


Figure 35: (a) Area and (b) power breakdown of MVMs.

Power consumption has a similar trend to the area usage shown in Figure 35-(b). *Log-SC* consumes 84% and 39% less power compared to *Fixed* and *Shift*. Compared to the bit-parallel *SotA-SC*, 24% smaller power is consumed by *Log-SC*, which is just 17% larger than serial SC-MVM.

2) Effectiveness of Log-Quantized SC Application to DCNNs: Because logarithmic quantization naturally involves error against the original value, acceptable neural network performance should be guaranteed explicitly. We tested it by the simulation extending Caffe framework [47]. Only convolutional layers which are taking most computations of target image classification DCNNs are accelerated. Target DCNNs are LeNet and AlexNet for MNIST and CIFAR10 which are 10-class dataset. The larger scale AlexNet is also experimented for LSVRC-2010 ImageNet dataset with 1,000 classes. The baseline floating-point trained weights are fine-tuned for 5,000 iterations in common.

Figure 36-(a) shows 10-classes image classification results. Surprisingly, *Log-SC* shows better recognition rates in overall while 5 and 9-bit are common requirements for the strict error constraint (1~2% loss). It means that even though quantization error is larger for logarithmically quantized weights, recognition rate can be recovered by fine-tuning on weights. Furthermore, in fine-tuning, SC multiplication error is a stronger obstacle than quantization error. This phenomenon is more enhanced

for the larger and difficult DCNN shown in Figure 36-(b). *SotA-SC* starts to learn from precision of 10-bit while *Log-SC* starts at 7-bit. Accuracy saturation starts at 8-bit similarly to the linearly quantized fixed-point binary. *Log-SC* looks more accurate than the linear quantized SC even with the smaller parameter size and the larger quantization error.

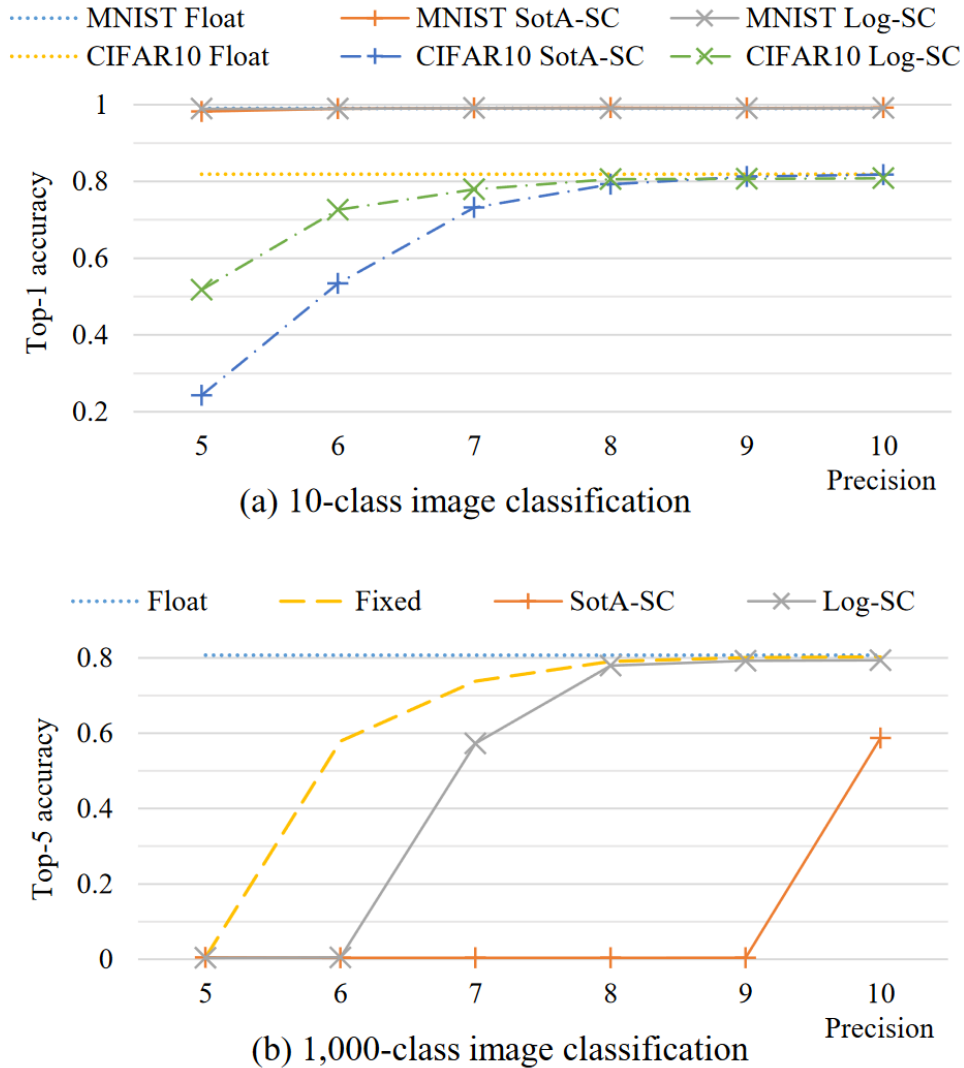


Figure 36: Recognition test on target DCNNs fine-tuned for 5,000 iterations.

However, because *SotA-SC* operates as same as *Log-SC* for the same input except processing of sign bit, a sign-magnitude design of *SotA-SC* can achieve the same accuracy with the *Log-SC* by using its fine-tuned weights. Therefore, even though we cannot say that *Log-SC* outperforms *SotA-SC* in terms of precision requirements at the last, *Log-SC* outperforms in fine-tuning and has no recognition rate degradation compared to *SotA-SC*.

All MVMs have the same number of MACs. Assuming that MAC utilization is always 100%, binary designs have static throughput, 256 GMACs. However, throughputs of SC designs are not static and the latency is depending on both precision and weight values which are acquired from the Caffe simulation. Unlike *Fixed* and *Shift*, SC designs can operate on different precision between DCNNs and even layers, called dynamic precision scaling (DPS) [17]. To satisfy strict error constraint (1~2% loss), precision requirements of dynamic precision scaling SC are 5, 9 and 10-bit for MNIST, CIFAR10 and ISVRC-2010 ImageNet dataset each, acquired from Figure 35. We assume that precision does not change through layers for the simplicity. The latency is geometrically averaged for 3 target DCNNs.

Let's look at Table 10. For SC cases, dynamic precision scaling is essential to reduce the average latency. Always working on full supported precision (16-bit) is uncompetitive in terms of latency and energy. Serial *SotA-SC* consumes less energy than *Fixed* with dynamic precision scaling. However, the latency is 6 times slower and the energy consumption is 3 times higher than *Shift*. Bit-parallelism of *SotA* effectively reduces the average latency and results a little smaller energy consumption than *Shift*. With the almost same latency, *Log-SC* reduces the power consumption of heavy 1s counter and outperforms all other designs in terms of energy.

Table 10: Geo-mean latency and energy consumption comparison for target DCNNs.

Cases	Avg. MAC latency (cycles)	Power (mW)	Energy per MAC (pJ)	Norm. energy
Fixed (16-bit)	1.00	154.43	0.60	544%
Shift (16-bit)	1.00	40.81	0.16	144%
SotA-SC (16-bit, serial)	1,071.22	21.40	89.54	80,703%
SotA-SC (DPS, serial)	5.69	21.40	0.48	428%
SotA-SC (DPS, 2 ⁴ par.)	1.13	33.14	0.15	132%
Log-SC (DPS, 2 ⁴ par.)	1.13	25.05	0.11	100%

Multiple designs can be generated for SC depending on how many stochastic cycles ($=2^n$) are parallelized where n is called *hardware precision*. Let's see how Pareto-optimal points change with *Log-SC* in terms of area and latency. Because the clock period is commonly set, the latency is proportional to the MAC cycles. The left and lower design is better in Figure 37. *Fixed* and *Shift* have the best latency which is always 1. However, their area is usually larger than SC designs resulting area-delay product sub-optimal. SC design area increases with the larger parallelism while the average latency is reduced. *SotA-SC* exceeds *Shift* area starting from 2⁵. However, *Log-SC* area is much smaller than *SotA-SC* and the difference increases with the more parallelism. *Log-SC* has always more than 38% less area compared to *Shift*. SC latency is depending on the target DCNN precision requirements and

weights distribution. Nevertheless, considering that 2^8 is relatively large [17], *Log-SC* is on the very advantageous position.

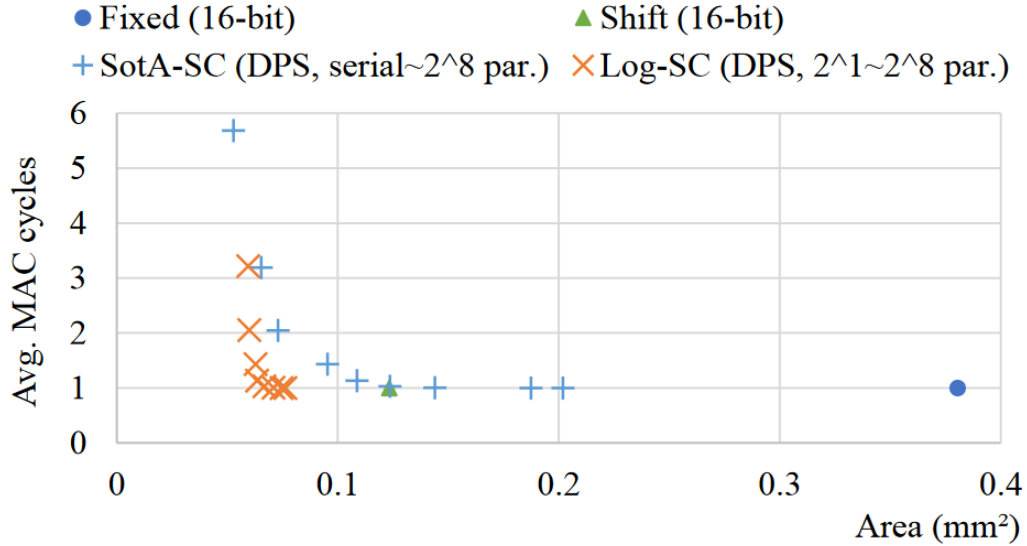


Figure 37: Area and latency Pareto-optimal points of MVMs.

4.4.4. Successive Log-Quantization and Its Application to SC

To evaluate our SLQ and SLQ-based SC-DCNN, we use three large scale DCNNs: AlexNet [1], SqueezeNet [53], and VGG-S [3], along with their published pre-trained weights.

1) Comparison of Quantization Performance: To evaluate the performance of different quantization methods, we compare the RMS (Root Mean Squared) error as shown in Figure 38. Error is defined as the difference between a quantized-and-restored value and the original weight ($= \tilde{w} - w$). We use the pre-trained weights of all convolution layers (no retraining tailored for quantization methods). The x -axis shows effective resolution, or the bitwidth of quantized words. For SLQ, it is the total number of bits per weight parameter, needed to represent all quantized words taking into account encoding overhead. Both encoding methods are tried, and the one giving the lower effective resolution is chosen (which happened to be the special code scheme most of the time). We vary the (base) resolution from 4 to 8-bit. In addition, for SLQ we vary θ from 2^{-3} to 2^{-8} and L is set to 2.

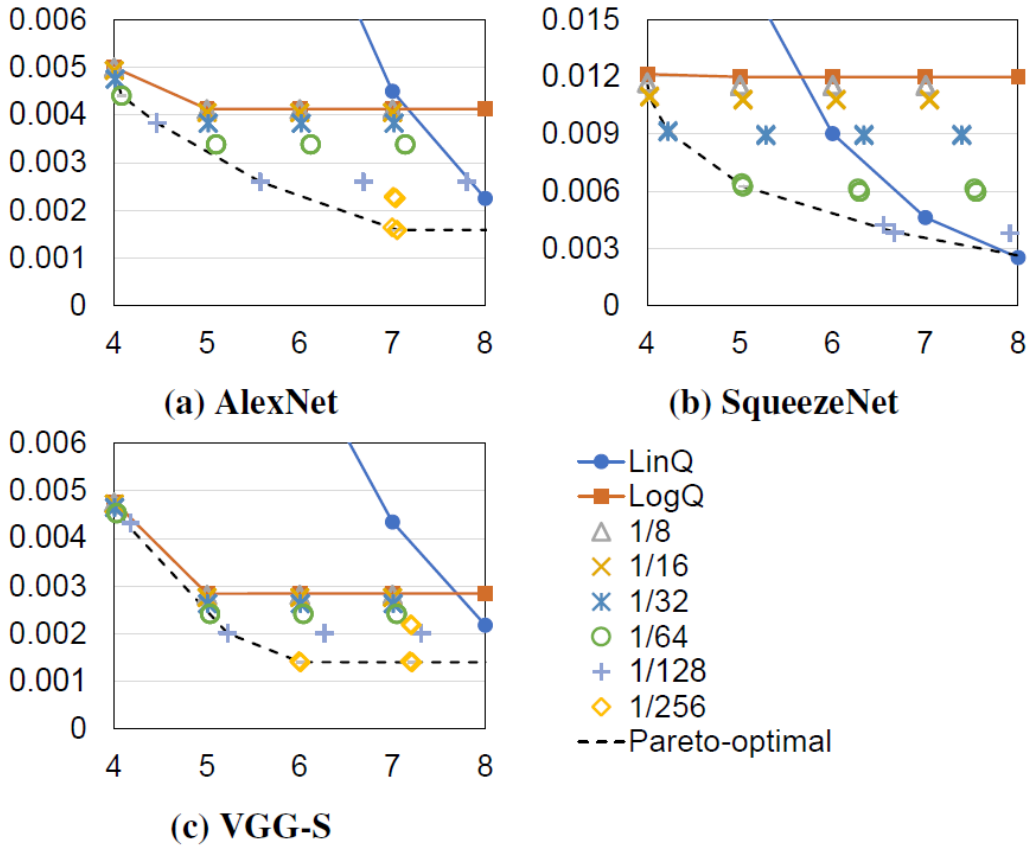


Figure 38: RMSE vs. effective resolution.

The results in Figure 38 suggest that log quantization is generally better than linear quantization at low resolution, which however is reversed at 6~8-bit depending on the network. SLQ’s performance depends on θ . As expected, lower θ gives better accuracy while raising effective resolution. Importantly, while log-quantization’s performance saturates at around 5-bit, SLQ’s performance continues to improve with higher resolution. Thus, SLQ is better than log quantization at any resolution often with a large margin, which is attributed to the richer configuration options of SLQ (L, θ) as well as its sophistication. Lastly the Pareto-optimal points of SLQ indicate that our SLQ can give the best performance for any resolution up to 8-bit, which is more than enough to reach sufficient classification accuracy with most DCNNs.

2) DCNN Classification Accuracy: To evaluate the effectiveness of SLQ on SC-DCNNs, we apply SLQ to weight parameters. Each model is fine-tuned for 5,000 additional iterations with the learning rate of 10^{-5} , which is repeated for each quantization method and resolution. For SLQ we set $L = 2$ and choose θ based on the best cases in Figure 25 considering the target resolution. All the selected cases use the special code encoding scheme. SC-MVM is applied to convolution layers only, but since its

interface is in conventional binary, other layers (e.g., max pooling) implemented in conventional binary can be mixed in. All activations and weights of fully-connected layers are quantized to 16-bit linear scale. We implemented all quantization methods as well as fixed-point and SC arithmetic in the Caffe framework [47], allowing us to use state-of-the-art training methods.

Table 11 summarizes the classification accuracy for the ImageNet validation set (the first 10,000 images are used). For SLQ, the effective resolution is shown in parentheses, which also suggests the ratio of double-word quantizations. Note that $\text{effective_resolution} = \text{base_resolution} \cdot (2 \cdot \text{double_ratio} + 1)$. As expected, log-quantization outperforms linear quantization at low resolution (around 4~6-bit). However, there is a significant gap between the low resolution performance and floating-point performance, which is not closed by increasing resolution. This contrasts with linear quantization performance, which achieves floating-point level performance eventually. Thus, it is interesting to see if SLQ can indeed achieve floating-point level performance.

Table 11: SC-DCNN classification accuracy (top-5, unit: %).

AlexNet	LinQ	LogQ	SLQ 4b	SLQ 5b
4b	1.23	75.82	78.57 (4.08b)	-
			79.32 (4.46b)	
5b	3.94	76.24	-	78.90 (5.10b)
6b	74.15	76.30	-	79.41 (5.57b)
float 32b	80.70			
SqueezeNet	LinQ	LogQ	SLQ 4b	SLQ 5b
4b	0.58	77.94	78.98 (4.03b)	-
			80.16 (4.22b)	
5b	58.08	78.14	80.82 (5.02b)	80.14 (5.28b)
6b	80.44	78.14	-	80.90 (6.28b)
float 32b	81.06			
VGG-S	LinQ	LogQ	SLQ 5b	SLQ 6b
5b	0.46	81.45	83.19 (5.03b)	-
			83.70 (5.22b)	
6b	51.53	81.72	83.72 (5.99b)	-
7b	82.70	81.52	-	83.79 (7.19b)
float 32b	83.41			

The result shows that not only does our SLQ consistently beat all the other quantization methods, but it can also achieve floating-point level performance using quite low resolution. Compared with log-quantization, our SLQ achieves significantly higher accuracy (by 2 to 4%p), which shows the superiority of our proposed scheme. In terms of the number of bits saved, our SLQ can save 1.5 to 2 bits compared with linear quantization for the same accuracy (within 1 to 1.5%p from floating-point accuracy).

3) Effect on Model Size: Table 12 compares the model sizes of using different quantization methods. For fair comparison, we use 6~7 bits for linear quantization and 4~5 bits for log quantization and SLQ, as they are found to give similar accuracy. Only VGG-S uses 7-bit and 5-bit; the others use 6-bit and 4-bit. While SLQ requires slightly larger model sizes compared to log quantization, this overhead is small (about 10% or less), and easily justifiable given SLQ's superior performance. Also, SLQ's model size compares very favorably with that of linear quantization, which is again due to SLQ's superior performance.

Table 12: Model size comparison (convolution layers only).

	float	LinQ	LogQ	SLQ
AlexNet	8.90MB	1.67MB	1.11MB	1.24MB
SqueezeNet	4.75MB	0.89MB	0.59MB	0.63MB
VGG-S	24.90MB	5.45MB	3.89MB	4.06MB

4) Effect on Hardware Efficiency: To evaluate the hardware overhead of our SLQ, we have designed in Verilog, SC-MAC MVMs consisting of 256 SC-MACs using different quantization methods. For comparison we have also designed conventional binary MVMs, using fixed-point multipliers (for linear weights) and shifter-based multipliers (for log-scale weights). Those designs are synthesized using Synopsys Design Compiler with TSMC 45nm standard cell library, targeting 1GHz frequency. For easier comparison, we use 16-bit datapath for all cases. For SC-MACs, we use 2^4 -bit parallelism.

Figure. 39 shows area breakdown (power breakdown is very similar to area breakdown, and thus omitted). Not surprisingly, SC circuits show significant area saving as compared with conventional binary circuits, which is mostly due to the elimination of multipliers. Among them, the bit-serial version has the smallest area, which however also has the largest latency. The bit-parallel version reduces latency significantly, but at the expense of large area overhead due to 1s counter, which again is effectively reduced by using log-scale weights. Finally, while *SLQ-SC* has more complicated control

and additional registers/MUXes due to weight prefetching as compared to *Log-SC*, its area overhead is very small (around 0.1% of SC-MVM area) and is effectively amortized over many SC-MACs.

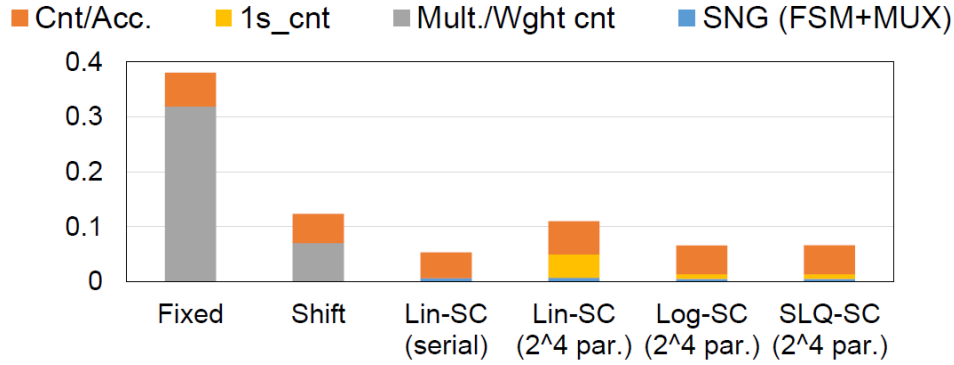


Figure 39: Area breakdown (um²).

Figure 40 shows the latency and energy results. Due to the variable latency SC-MAC we use, the average latency depends on the weight parameter values, their usage, quantization resolution, and bit-parallelism of the SC-MAC, as well as the ratio of double-quantized words in the case of our SLQ. For quantization resolution, we again choose the resolutions used for Table 12. In terms of latency, the *SLQ-SC* case has about 5~10% overhead over the linear or log case, which is also the energy overhead of *SLQ-SC* over *Log-SC*. Again, this energy difference is for unequal classification accuracy, as *Log-SC* could never reach the accuracy of *SLQ-SC*, and therefore is justifiable. On the other hand, compared with *Linear-SC*, our *SLQ-SC* shows superior energy efficiency thanks to simpler 1s counter, reducing energy by about 32.6% on average while simultaneously achieving higher classification accuracy. These results confirm that our SLQ-based SC-DCNN improves both accuracy and efficiency of inference significantly over the previous state-of-the-art SC-DCNNs.

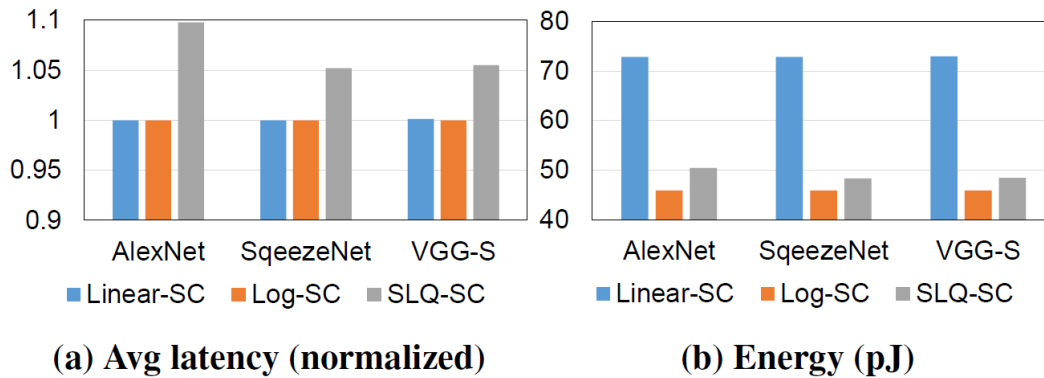


Figure 40: Latency and energy comparison of SC-DCNNs.

5) Discussion: Table 13 compares the classification accuracy of using our SLQ on conventional binary vs. SC-based DCNN. First, this result confirms that on both SC and conventional binary, our SLQ can make significant improvement in accuracy over log quantization. Second, the result shows that our SLQ-SC is highly competitive against conventional binary DCNNs in terms of network performance, thanks to the use of log-weights. This is in contrast with linear-SC, which requires 1-bit higher resolution (than conventional binary) to match the performance of linear-weight conventional binary DCNN [17].

Table 13: SLQ on SC vs. conventional binary DCNN.

	float	Log-SC	SLQ-SC	Log-Bin	SLQ-Bin
AlexNet 4b	80.70	75.82	79.32	75.75	79.27
SqueezeNet 4b	81.06	77.94	80.16	77.78	80.34
VGG-S 5b	83.41	81.45	83.70	81.75	83.44

Our SC-DCNN is based on a previously known SC-MAC [16]. However, we extend it significantly in terms of both accuracy, efficiency, and scalability while retaining its adaptive-precision advantage. [54] proposes value-aware quantization, which bears some similarity with our technique; however, our technique aims to extend log-quantization by endowing it with a precision dimension without increasing hardware complexity. Recently, a new data format called Flexpoint [55] was proposed, which also aims at low-cost and high-accuracy; however, unlike ours, it offers little advantage over fixed-point when it comes to inference.

In summary, our proposed method improves the performance-per-area of MVM datapath by about twice. Let us calculate how much it will affect the end-to-end performance of a DCNN processing unit, which is a function of how other computations that are not in our scope are implemented. We consider three scenarios. The first scenario is that our proposed SC-MVM also performs almost all the other computations (e.g., MACs in fully-connected layers). In this case, performance-per-area gain is the same. However, a few last layers, which are mainly fully-connected, can be less error tolerant because they directly affect the output quality or decision. Therefore, it may not be easy to accelerate them with our proposed SC-MVM. The second scenario is that only convolutional layers are enhanced by the SC-MVM. Let us assume that computations in convolutional layers account for 90% which is conservative for latest DCNNs. Latency of the rest is maintained and 90% latency is reduced by half when the area is fixed. The total relative latency is 55%. Still the scenario is not realistic enough. Unlike convolutional layers, fully-connected layers usually take longer latency than its computation ratio often because of memory bottleneck. In addition, they require higher precision than convolutional layers as stated before

which means that their cost or latency per calculation is larger than that of convolutional layers. The last scenario is that the rest 10% computations takes 30% of the total latency before our proposed SC-MVM is applied. After applying the SC-MVM, the total latency becomes 65%, 35% reduction. For all cases, our cost efficiency improvement is significant because of a dominant computation amount of convolutional layers.

V. Conclusion

Low-cost SC has many merits when it is applied to DCNNs. Even though SC nature comes from uncertainty, its deterministic interpretation works also very well with many advantages. Recent optimization technologies, such as DPS and log-quantization, strengthen SC-DCNN's efficiency.

First, to improve the accuracy of SC-DCNN, we proposed a hybrid SC-binary layer architecture. To minimize the overhead of SC-binary conversion, we proposed a parallel and approximate SC-MAC. We call it BISC. We presented a highly accurate, low-latency, and cost-efficient SC multiplication algorithm and its vector version, BISC-MVM.

Related to quantization of dynamic precision, we presented a set of optimizations to enable highly accurate and efficient SC-based DCNN implementations up to ImageNet-targeting DCNNs. Our key ingredient is the optimal use of precision.

Non-linear quantization is promising to cover highly biased weight distribution with the small bitwidth. The mainstream of non-linear quantization is logarithmic formatted where numbers are represented as power of 2 (or some other base). We proposed logarithmic quantization to the state-of-the-art SC-DCNN and demonstrated that it is also beneficial.

In addition, we presented SLQ, which generalizes logarithmic quantization with significant improvements in precision and accuracy. SLQ achieves it while reusing the existing datapath of log-quantization, thus retaining the simple-multiplication advantage of log-quantization. Our experimental results show that SLQ can represent weight parameters of a DCNN with less average error than both linear and log-quantizations across the full range of resolution relevant to DCNN inference

While SC-DNNs have often been limited to small DCNNs or DCNNs with low precision requirements, our SLQ can significantly extend both the accuracy and efficiency of SC-DCNNs over the state-of-the-art solutions, achieving less than 1~1.5%p accuracy drop for AlexNet, SqueezeNet, and VGG-S at mere 4~5-bit weight resolution.

SC is competitive with additional benefits but not popular as conventional binary computing. The dissertation would bring about much spotlight on SC hopefully.

REFERENCES

- [1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Communications of the ACM* 60.6 (2017): 84-90.
- [2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
- [3] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).
- [4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
- [5] Alaghi, Armin, and John P. Hayes. "Fast and accurate computation using stochastic circuits." *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014.
- [6] Gross, Warren J., Vincent C. Gaudet, and Aaron Milner. "Stochastic implementation of LDPC decoders." *Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, 2005..* IEEE, 2005.
- [7] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rosselló. "A new stochastic computing methodology for efficient neural network implementation." *IEEE transactions on neural networks and learning systems* 27.3 (2015): 551-564.
- [8] K. Sanni, G. Garreau, J. L. Molin, and A. G. Andreou. "Fpga implementation of a deep belief network architecture for character recognition using stochastic computation." *2015 49th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2015.

- [9] Kyoungsoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoungh Choi. "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks." *Proceedings of the 53rd Annual Design Automation Conference*. 2016.
- [10] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Quantized neural networks: Training neural networks with low precision weights and activations." *The Journal of Machine Learning Research* 18.1 (2017): 6869-6898.
- [11] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. "Quantization and training of neural networks for efficient integer-arithmetic-only inference." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.
- [12] Miyashita, Daisuke, Edward H. Lee, and Boris Murmann. "Convolutional neural networks using logarithmic data representation." *arXiv preprint arXiv:1603.01025* (2016).
- [13] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong. "Lognet: Energy-efficient neural networks using logarithmic computation." *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017.
- [14] Aojun Zhou, Anbang Yao, Yiwon Guo, Lin Xu, and Yurong Chen. "Incremental network quantization: Towards lossless cnns with low-precision weights." *arXiv preprint arXiv:1702.03044* (2017).
- [15] H. Sim, D. Nguyen, J. Lee, and K. Choi. "Scalable stochastic-computing accelerator for convolutional neural networks." *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017.
- [16] Sim, Hyeonuk, and Jongeun Lee. "A new stochastic computing multiplier with application to deep convolutional neural networks." *Proceedings of the 54th Annual Design Automation Conference 2017*. 2017.

- [17] Sim, Hyeonuk, Saken Kenzhegulov, and Jongeun Lee. "Dps: Dynamic precision scaling for stochastic computing-based deep neural networks." *Proceedings of the 55th Annual Design Automation Conference*. 2018.
- [18] Sim, Hyeonuk, and Jongeun Lee. "Log-quantized stochastic computing for memory and computation efficient DNNs." *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 2019.
- [19] S. Lee, H. Sim, J. Choi, and J. Lee. "Successive Log Quantization for Cost-Efficient Neural Networks Using Stochastic Computing." *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019.
- [20] Sim, Hyeonuk, and Jongeun Lee. "Cost-effective stochastic MAC circuits for deep neural networks." *Neural Networks* 117 (2019): 152-162.
- [21] P. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello. "NeuFlow: Dataflow vision processing system-on-a-chip." *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2012.
- [22] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H. Yoo. "4.6 A1. 93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications." *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*. IEEE, 2015.
- [23] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning." *ACM SIGARCH Computer Architecture News* 42.1 (2014): 269-284.
- [24] Lukas Cavigelli, David Gschwend, Christoph Mayer, Samuel Willi, Beat Muheim, and Luca Benini. "Origami: A convolutional network accelerator." *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. 2015.

- [25] Bade, Stephen L., and Brad L. Hutchings. "FPGA-based stochastic neural networks-implementation." *Proceedings of IEEE Workshop on FPGA's for Custom Computing Machines*. IEEE, 1994.
- [26] Brown, Bradley D., and Howard C. Card. "Stochastic neural computation. I. Computational elements." *IEEE Transactions on computers* 50.9 (2001): 891-905.
- [27] Brown, Bradley D., and Howard C. Card. "Stochastic neural computation. II. Soft competitive learning." *IEEE Transactions on Computers* 50.9 (2001): 906-920.
- [28] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross. "VLSI implementation of deep neural network using integral stochastic computing." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.10 (2017): 2688-2699.
- [29] Y. Ji, F. Ran, C. Ma, and D. J. Lilja. "A hardware implementation of a radial basis function neural network using stochastic logic." *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015.
- [30] Vinay K. Chippa, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. "StoRM: A stochastic recognition and mining processor." *Proceedings of the 2014 international symposium on Low power electronics and design*. 2014.
- [31] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." *arXiv preprint arXiv:1510.00149* (2015).
- [32] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. "ApproxANN: An approximate computing framework for artificial neural network." *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015.
- [33] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks." *IEEE journal of solid-state circuits* 52.1 (2016): 127-138.

- [34] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. "EIE: efficient inference engine on compressed deep neural network." *ACM SIGARCH Computer Architecture News* 44.3 (2016): 243-254.
- [35] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos. "Stripes: Bit-serial deep neural network computing." *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016.
- [36] Gudovskiy, Denis A., and Luca Rigazio. "Shiftcnn: Generalized low-precision architecture for inference of convolutional neural networks." *arXiv preprint arXiv:1706.02393* (2017).
- [37] Park, Eunhyeok, Junwhan Ahn, and Sungjoo Yoo. "Weighted-entropy-based quantization for deep neural networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.
- [38] Kim, Dongyoung, Junwhan Ahn, and Sungjoo Yoo. "Zena: Zero-aware neural network accelerator." *IEEE Design & Test* 35.1 (2017): 39-46.
- [39] Xiaowei Xu, Qing Lu, Lin Yang, Sharon Hu, Danny Chen, Yu Hu, and Yiyu Shi. "Quantization of fully convolutional networks for accurate biomedical image segmentation." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.
- [40] R. Venkatesan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan. "Spintastic: Spin-based stochastic logic for energy-efficient computing." *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015.
- [41] Parhami, Behraoz, and Chi-Hsiang Yeh. "Accumulative parallel counters." *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*. Vol. 2. IEEE, 1995.
- [42] Kim, Kyoungheon, Jongeun Lee, and Kiyong Choi. "Approximate de-randomizer for stochastic circuits." *2015 International SoC Design Conference (ISOCC)*. IEEE, 2015.

- [43] Alaghi, Armin, and John P. Hayes. "Survey of stochastic computing." *ACM Transactions on Embedded computing systems (TECS)* 12.2s (2013): 1-19.
- [44] Kim, Kyoungsoon, Jongeun Lee, and Kiyoun Choi. "An energy-efficient random number generator for stochastic circuits." *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016.
- [45] Rahman, Atul, Jongeun Lee, and Kiyoun Choi. "Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array." *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016.
- [46] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. "Optimizing fpga-based accelerator design for deep convolutional neural networks." *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*. 2015.
- [47] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. "Caffe: Convolutional architecture for fast feature embedding." *Proceedings of the 22nd ACM international conference on Multimedia*. 2014.
- [48] Li, Bingzhe, M. Hassan Najafi, and David J. Lilja. "Using stochastic computing to reduce the hardware requirements for a restricted Boltzmann machine classifier." *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2016.
- [49] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan. "Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks." *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 2016.
- [50] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks." *European conference on computer vision*. Springer, Cham, 2016.

- [51] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, Raquel Urtasun, and Andreas Moshovos. "Reduced-precision strategies for bounded memory in deep neural nets." *arXiv preprint arXiv:1511.05236* (2015).
- [52] Lin, Darryl, Sachin Talathi, and Sreekanth Annapureddy. "Fixed point quantization of deep convolutional networks." *International conference on machine learning*. 2016.
- [53] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size." *arXiv preprint arXiv:1602.07360* (2016).
- [54] Park, Eunhyeok, Sungjoo Yoo, and Peter Vajda. "Value-aware quantization for training and inference of neural networks." *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.
- [55] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K. Bansal, William Constable, Oguz Elibol, Scott Gray, Stewart Hall, Luke Hornof, Amir Khosrowshahi, Carey Kloss, Ruby J. Pai, and Naveen Rao. "Flexpoint: An adaptive numerical format for efficient training of deep neural networks." *Advances in neural information processing systems*. 2017.
- [56] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja. "An architecture for fault-tolerant computation with stochastic logic." *IEEE transactions on computers* 60.1 (2010): 93-105.
- [57] A. Zhakatayev, S. Lee, H. Sim, and J. Lee. "Sign-magnitude SC: Getting 10X accuracy for free in stochastic computing for deep neural networks." *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018.
- [58] V. Sze, Y. Chen, T. Yang, and J. S. Emer. "Efficient processing of deep neural networks: A tutorial and survey." *Proceedings of the IEEE* 105.12 (2017): 2295-2329.

