Energy-Efficient Convolutional Neural Networks with Deterministic Bit-Stream Processing

S. Rasoul Faraji*,1, M. Hassan Najafi†,1, Bingzhe Li*, David J. Lilja*, and Kia Bazargan*
*Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, USA
†School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, USA
faraj008@umn.edu, najafi@louisiana.edu, lixx1743@umn.edu, lilja@umn.edu, kia@umn.edu

***Abstract*—Stochastic computing (SC) has been used for low-cost and low power implementation of neural networks. Inherent inaccuracy and long latency of processing random bit-streams have made prior SC-based implementations inefficient compared to conventional fixed-point designs. Random or pseudo-random bitstreams often need to be processed for a very long time to produce acceptable results. This long latency leads to a significantly higher energy consumption than binary design counterparts. Low-discrepancy sequences have been recently used for fast-converging deterministic computation with stochastic constructs. In this work, we propose a low-cost, low-latency, and energy-efficient implementation of convolutional neural networks based on low-discrepancy deterministic bit-streams. Experimental results show a significant reduction in the energy consumption compared to previous random bitstream-based implementations and to the optimized fixed-point design with no quality degradation.**

***Keywords*-Convolutional neural networks, bitstream processing, stochastic computing, energy-efficient design, low-cost design.**

I. Introduction

Stochastic computing (SC) [5][1][18], an unconventional paradigm processing random bit-streams, has been recently used for low-cost implementations of convolutional neural networks (NNs) [9][7][11][21][20][12][19][3][10]. Multiplication operations, as essential and costly operations in conventional fixed-point and floating-point hardware implementations of NNs, can be implemented with simple standard AND gates in the stochastic domain. This results in significant savings in hardware area and power costs. Redundant representation of data with long bit-streams also makes SC-based implementations of NNs inherently noise tolerant.

Quality degradation, and high processing time and energy consumption, however, are still the main barriers in wide adoption of SC-based NNs. Random fluctuation in generating bitstreams and correlation between bitstreams have resulted in approximate computations and therefore quality loss in different layers of NNs. Often a long processing time is inevitable to achieve acceptable results. The long processing time leads to a high energy consumption, which in most cases is higher than that of the corresponding conventional fixed-point design. Designing high accuracy energy-efficient SC-based NNs is still an open problem.

Recently, hybrid stochastic-binary implementations of NNs have been proposed to improve the accuracy and reduce the energy consumption of SC-based NNs. The authors in [9] used SC to implement the first convolutional layer of the NN. The remaining layers were all implemented in the binary domain. The result was a significant improvement in the energy-consumption compared to conventional all-binary design and a better accuracy compared to prior SC designs. Hybrid stochastic-binary designs proposed in [7], [11], and [21] also use approximate parallel counter, accumulative parallel counter, and binary arithmetic to improve the accuracy and energy-efficiency of NN designs. None of these prior hybrid designs, however, is able to achieve the same classification rate as the conventional fixed-point binary design.

Deterministic computation on stochastic bit-streams [6][15] has been recently introduced as an evolution in the idea of SC. By properly structuring the bit-streams, computation with stochastic construct can be performed deterministically. The results are completely accurate and the same as the results produced by the conventional binary designs. The operations must continue for an exact number of cycles (i.e., the product of the length of input bit-streams) to guarantee producing completely accurate results. These methods were initially developed using unary bit-streams (streams with a sequence of 1s followed by a sequence of 0s). The nature of unary bit-streams, however, led to a very long processing time and a very high energy consumption to produce acceptable results. The authors in [4] developed a hybrid binary-unary method to mitigate the energy consumption issue. However, their method is not applicable to multi-variate functions such as the multiplication operation.

A modified version of the early deterministic methods was proposed in [16] by replacing unary bit-streams with pseudo-random bit-streams. For a fixed accuracy level, by generating pseudo-random but deterministic bit-streams, a lower processing time and energy consumption were achieved compared to the unary stream-based deterministic methods. More recently, low discrepancy (LD) deterministic methods have been proposed in [17] for fast-converging energy-efficient processing of bit-streams with stochastic constructs.

In this work, we propose a hybrid bitstream-binary design for low-cost, energy-efficient and yet accurate implementation of convolutional NNs. We use LD deterministic bit-streams to implement the first convolution layer of the NN, as convolution layers account for more than 90 percent of the overall hardware cost in some convolutional NNs [21]. We explore the performance of different combinations of LD sequences in generating deterministic bit-streams for the proposed design. Experimental results show a significant reduction in the hardware cost and energy consumption compared to the

[1]These authors contributed equally.

conventional fixed-point design. In contrast to prior stochastic and hybrid stochastic-binary designs, the proposed design achieves the same classification rate as the fixed-point design.

This paper is structured as follows: Section II presents background information on SC and the recently developed LD bit-stream-based deterministic methods. In Section III, we describe our proposed method for energy-efficient high-quality design of convolutional NNs. In Section IV, we evaluate the efficiency and performance of the proposed design by hardware implementation of a LeNet5 NN architecture. Finally, in Section V, we present conclusions.

II. Background

A. Stochastic Computing

Stochastic computing (SC) is an unconventional computing paradigm operating on random bit-streams. Independent of the length, the ratio of the number of ones to the length of the stream determines the value of the bit-stream in the [0,1] interval. For example, 1101011101 is a representation of 0.7 in the stochastic domain. Implementing complex operations with simple hardware and the ability of tolerating high rates of noise are the primary advantages of SC. Multiplication, as a costly common operation in convolutional NNs, can be implemented with a standard AND gate in SC. As shown in Fig. 1, an AND gate works as a stochastic multiplier if independent (uncorrelated) bit-streams are connected to its inputs. To convert input data from binary to stream representation, a random or pseudo-random number is compared to a constant number (based on the input data) and the output of the comparison produces one bit of the stochastic bit-stream in each cycle. Fig. 2 shows the structure of a stochastic bit-stream generator responsible for converting data from conventional binary to bit-stream representation. The generated bit-streams are processed using stochastic constructs and an output bit-stream is produced. The output bit-stream is converted back to binary by simply counting the number of ones in the bit-stream using a binary counter.

B. Deterministic LD Bit-Stream-Based Computing

Low discrepancy (LD) sequences (e.g., Sobol, Halton) have been previously used in improving the speed of computation on stochastic bit-streams [2], [13], [14]. With LD sequences, 1s and 0s in the bit-streams are uniformly spaced and so the streams do not suffer from random fluctuations. The result is

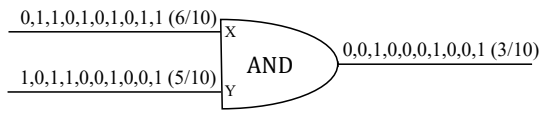

Fig. 1. Example of stochastic multiplication using AND gate.

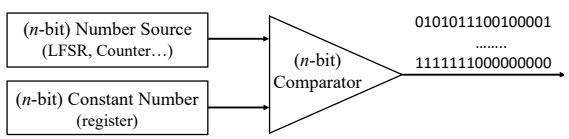

Fig. 2. Structure of a stochastic stream generator.

a faster convergence to the expected correct value and so a lower processing time to achieve a fixed accuracy level.

The authors in [17] showed that Sobol sequences can be used for fast-converging deterministic multiplication of bit-streams. A different Sobol sequence must be used in converting each independent input data to bit-stream representation. When multiplying two N-bit precision input data, the inputs are converted to two 2^{2N}-bit length bit-streams by comparing each to the first 2^{2N} numbers of a different Sobol sequence. Alternatively, the inputs can first be converted to bit-streams of 2^N-bit length by comparing each to the first 2^N numbers of a different Sobol sequence. The bit-stream corresponding to the first input is then repeated until it becomes a stream of 2^{2N}-bit length. The bit-stream corresponding to the second input is stalled after every 2^N cycles and repeated until the total length of the bit-stream becomes 2^{2N}. Both of these methods produce completely accurate results when running the operation for 2^{2N} cycles. They also produce a similar result and give the same truncation error when operating on short bit-streams ($\leq 2^N$).

Fig. 3 shows the first 16 numbers of the first four Sobol sequences from the MATLAB built-in Sobol sequence generator. An important property of Sobol sequences, including these four sequences, is that the first 2^N numbers of any Sobol sequence can precisely present all possible N-bit precision numbers in the [0,1] interval. Hence, the only error in converting an input data to a 2^N-bit length stream using the first 2^N numbers of a Sobol sequence is the quantization error (there is no random fluctuation error with a 2^N-bit length Sobol-based bit-stream). In what follows, we see an example of multiplying two 2-bit precision numbers, $1/4$ and $3/4$, by converting them to bit-stream representation using the first 16 numbers of Sobol Seq. 1 and Sobol Seq. 2, and ANDing the generated bit-streams. Note that a one in the bit-stream is generated if the Sobol number from the Sobol sequence is less than the input target data.

$$\begin{array}{rl} 1/4 & = 1000\ 1000\ 1000\ 1000 \\ 3/4 & = 1101\ 1110\ 0111\ 1011 \\ \hline 3/16 & = 1000\ 1000\ 0000\ 1000 \end{array}$$

As can be seen, the deterministic accurate output of multiplying the two 2-bit precision input values is obtained by processing the generated 16-bit long bit-streams. In the conventional random-stream based SC, we needed to perform the multiplication operation multiple times (each time generating and ANDing a different pair of bit-streams) and average the outputs to produce statistically significant results. Due to converting input data to deterministic bit-streams, the output of processing LD bit-streams is deterministic with a standard deviation of zero. Once performing the operation is therefore sufficient to have a result free of variation.

We evaluated the performance of four different combinations of Sobol sequences for all possible cases of multiplying two 8-bit precision input data in the [0,1] interval. The results are reported in Table I. We compare the performance using the mean absolute error (MAE) metric. As can be seen, all four selected combinations of Sobol sequences produce

Sobol Seq 1	**0**	**1/2**	**1/4**	**3/4**	**1/8**	**5/8**	**3/8**	**7/8**	**1/16**	**9/16**	**5/16**	**13/16**	**3/16**	**11/16**	**7/16**	**15/16**
Sobol Seq 2	**0**	**1/2**	**3/4**	**1/4**	**5/8**	**1/8**	**3/8**	**7/8**	**15/16**	**7/16**	**3/16**	**11/16**	**5/16**	**13/16**	**9/16**	**1/16**
Sobol Seq 3	**0**	**1/2**	**1/4**	**3/4**	**7/8**	**3/8**	**5/8**	**1/8**	**11/16**	**3/16**	**15/16**	**7/16**	**5/16**	**13/16**	**1/16**	**9/16**
Sobol Seq 4	**0**	**1/2**	**3/4**	**1/4**	**7/8**	**3/8**	**1/8**	**5/8**	**7/16**	**15/16**	**11/16**	**3/16**	**9/16**	**1/16**	**5/16**	**13/16**

Fig. 3. MATLAB built-in first four Sobol sequences

TABLE I
MEAN ABSOLUTE ERROR (%) COMPARISON OF USING DIFFERENT COMBINATIONS OF SOBOL SEQUENCES IN LD BIT-STREAM-BASED MULTIPLICATION OF 8-BIT PRECISION INPUT VALUES

Operation Cycles	Sobol 1&2	Sobol 3&4	Sobol 1&4	Sobol 2&3
4	15.8	15.8	15.8	15.8
5	14.7	9.5	11.1	10.0
6	13.5	9.3	9.5	12.1
7	13.2	9.3	11.2	10.6
8	8.9	8.9	7.8	7.8
9	6.3	7.9	10.4	5.7
10	6.1	6.7	7.9	5.7
16	3.7	4.4	4.3	3.9
32	1.8	2.3	2.4	1.9
2^{16}	0.0	0.0	0.0	0.0

completely accurate results after 2^{16} cycles (processing 2^{16}-bit streams). When operating for a shorter number of cycles, the outputs include some truncation error due to inaccurate representation of data. The rate of this error is different for different combinations because of the change in the order of numbers in the Sobol sequences which leads to different interactions between bit-streams. In this work, we use Sobol-based bit-streams in accurate implementation of convolutional NNs. We show that processing these bit-streams for only a few cycles is enough to produce the same results as the results from a baseline fixed-point binary design.

C. Convolutional Neural Networks

A general convolutional NN consists of convolutional layers followed by pooling layers and fully connected layers. A convolutional layer extracts a feature map from its input by applying a set of filters (kernels) and by activating them when specific types of features are found. Each pixel in a feature map is a convolution neuron which is obtained by the convolution of a filter-sized moving window and the inputs in the moving window. The activation functions which are used to extract specific types of features from the convolution neurons are non-linear transformation functions, such as hyperbolic tangent (tanh) or Rectified Linear Units (ReLU). Choosing each of these has a trade-off between computational complexity and overall performance. To reduce the dimensions of feature maps and mitigate over-fitting issues, a subsampling technique is applied to data by a pooling layer. There are various subsampling techniques such as max pooling, L2-norm pooling, and average pooling. The fully connected layer fully connects its input to all activated neurons of its previous layer. Each neuron in this layer computes dot-product (inner-product) of its inputs and corresponding weights. To specify the deviation between the predicted and real labels in the network training process, a loss function as a loss layer is used. The loss function can be sigmoid cross-entropy loss, softmax loss, or Euclidean loss.

In general, there are three types of arithmetic operations in a convolutional NN: dot-product, subsampling, and a non-linear operation as the activation function. In this work, we focus on the dot-product operation as the basic and essential operation in the convolutional and fully connected layers. We implement the first convolutional layer using a hybrid deterministic bit-stream-binary dot-product.

III. PROPOSED HYBRID DESIGN

Convolutional layers, including the first layer, account for the most hardware area and power cost in a class of convolutional NNs [21]. The basic operation in these layers is dot-product, multiplying and accumulating the input data and the weights (obtained from the training step), followed by an activation function. In this work, we use the LD bit-stream processing method of [17] in low-cost and energy-efficient implementation of the first convolutional layer. We limit our work to the first layer of NN to avoid the issue of compounding errors over multiple layers.

Multiplications and Accumulation. We perform the multiplication operations in the bit-stream domain. Two different Sobol sequences are used to convert the two inputs of each multiplication operation to corresponding bit-stream representation. Multiplications are implemented using standard AND gates instead of costly fixed-point binary multipliers. The deterministic method discussed in Section II-B guarantees independence between bit-streams and so producing accurate results. Deterministic computation is performed on the generated bit-streams and the output bit-streams are converted back to binary format implicitly by accumulating them in the binary domain using conventional binary adders.

Since accumulating the outputs of multiplications is performed in the binary domain, the correlation between the produced output bit-streams does not affect the accuracy of the accumulation. As a result, only two different Sobol sequence generators are sufficient to convert all input data to bit-stream representation. The generated bit-streams will be re-used by a large number of multiplication units, minimizing the overhead of bit-stream generators in the overall cost of the NN design.

Handling Negative Weights. The weight inputs of the multiplication operations involve both positive and negative data. The common approach of handling negative data in the stochastic domain is through extending the range of numbers from [0,1] to [-1,1] using a linear transformation and processing bit-streams in a so-called stochastic bipolar domain [1]. This method, however, requires a longer processing time for the same accuracy. We divide the weights into positive and negative subsets and convert them to bit-stream representation

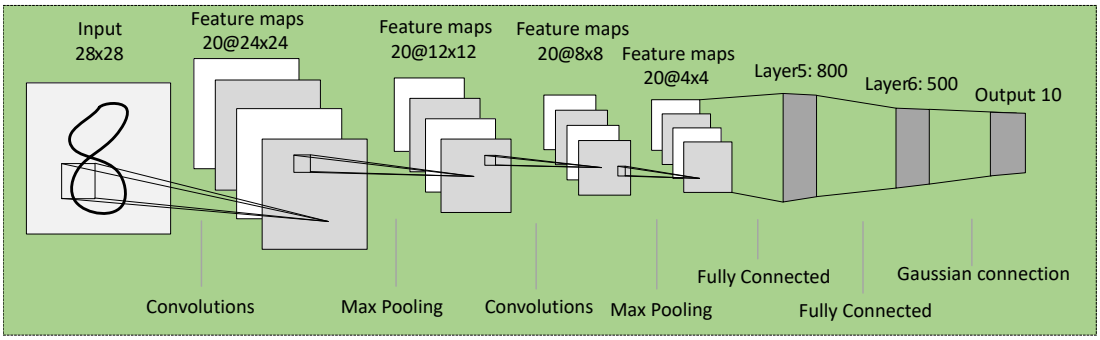

Fig. 4. Configuration of the implemented NN based on LeNet5 topology.

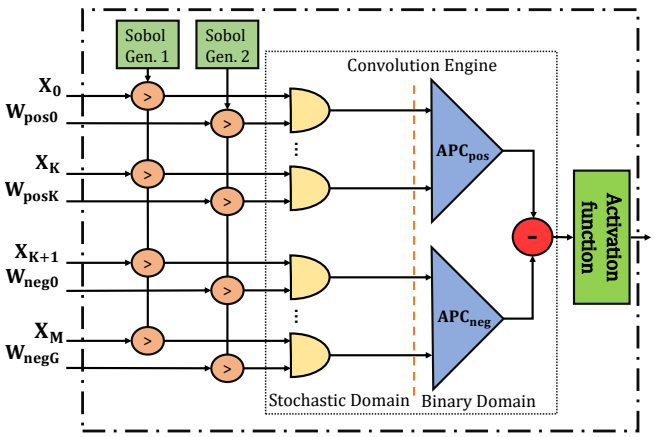

Fig. 5. Proposed Hybrid Design for the Convolutional Layer.

assuming that all are positive values (taking the absolute value of the negative values). In the accumulation step (APC units in Fig. 5), the multiplication outputs of the "positive" subset and the "negative" subset are first summed separately and then subtracted from each other to produce the final output value.

Producing completely accurate results with the deterministic bit-stream methods requires running the operation for 2^{2N} cycles when multiplying two N-bit precision numbers. As we showed in Table I, running the operation for fewer cycles introduces some inaccuracy in the produced results. However, as we will show in Section IV, due to the inaccuracy tolerance of NNs, the same or even in some cases a lower misclassification rate than the rate of the fixed-point design can be achieved using the proposed design.

Fig. 5 shows our proposed hybrid design for the first convolutional layer of the NN. We use this design to implement the first layer of the LeNet-5 NN topology [8] as a well-studied and common convolutional NN. We implement a 784-11520-2880-1280-320-800-500-10 configuration of this topology as illustrated in Fig. 4. The selected convolutional NN consists of two convolution layers, two max-pooling layers, two fully connected layers, and one softmax layer. The first convolution layer processes each pixel of the input image with 20 filters of 5×5 size. We perform the multiplication operations in the bit-stream domain. The output bit-streams are accumulated in each cycle using binary adders, implicitly converting from bit-stream to binary representation. The produced binary output is passed to a ReLU as the activation function. The remaining layers of the NN are all implemented in the binary domain. Since parallel processing of all pixels in the first layer requires $24 \times 24 \times 5 \times 5$ multiplication operations for each filter, exploiting bit-stream based computing can significantly decrease the hardware area and power cost compared to the conventional fixed-point binary design.

IV. Experimental Results

We use MNIST, a standard image database for handwritten digit recognition to train and test the NN. The database includes 60,000 training and 10,000 testing gray-scale images, all in 28×28 size. We evaluate the efficiency of the proposed design by comparing it to an 8-bit fixed-point implementation of the NN as the baseline design. Both the fixed-point baseline and the proposed bitstream-based designs were implemented in MATLAB for accuracy evaluation and in Verilog hardware description language for hardware cost comparisons. Misclassification rate (one minus the classification accuracy) is multiplied by 100 and reported as a percentage for accuracy evaluation. The Synopsys Design Compiler vH2013.12 is used to synthesize the implemented designs with a 45nm gate library. We train the NN over the 60,000 training images using the 8-bit fixed-point binary design. In the testing step, we evaluate the performance of the NN with the 10,000 test images with both the fixed-point binary and the proposed bitstream-based designs.

For the bitstream-based approach we implemented five design structures: 1) the conventional stochastic random bitstream-based method, 2-5) the proposed deterministic LD bitstream-based method with different pairs of Sobol sequences to generate the bit-streams. The implemented structures only differ in the bit-stream generation part (converting input pixel intensities and weights from binary to bit-stream representation) and the core logic to perform the multiplications and accumulations is the same in all structures.

For the previous random bitstream-based approach we use the MATLAB built-in random number generator to generate the required random numbers in the bit-stream generation process. For the proposed deterministic design we use four different combinations of the first four Sobol sequences from the MATLAB built-in Sobol sequence generator (Fig. 3). Input test images and weights are converted to their corresponding bit-stream representation by comparing them to these Sobol sequences.

TABLE II
MISS CLASSIFICATION RATES (# OF INCORRECT RECOGNITIONS / 10,000 * 100) OF THE IMPLEMENTED BITSTREAM-BASED DESIGNS FOR DIFFERENT NUMBER OF OPERATION CYCLES.

Design Approach \Operation Cycles	2^{16}	2^8	2^6	2^5	2^4	9	8	7	6	5	4
Conventional Random SC	0.80%	0.81%	0.88%	0.93%	1.08%	1.33%	1.41%	1.48%	1.70%	1.91%	2.46%
Proposed Design - Sobol 1 and 2	0.80%	0.79%	0.79%	0.82%	0.84%	0.86%	**0.84%**	0.87%	0.92%	1.15%	1.12%
Proposed Design - Sobol 3 and 4	0.80%	0.79%	0.77%	0.85%	0.86%	0.89%	**0.84%**	0.91%	1.05%	1.12%	1.12%
Proposed Design - Sobol 1 and 4	0.80%	0.79%	0.79%	0.83%	0.83%	0.95%	**0.81%**	0.87%	0.98%	1.12%	1.12%
Proposed Design - Sobol 2 and 3	0.80%	0.79%	0.80%	0.78%	0.85%	0.89%	**0.88%**	1.12%	1.17%	1.12%	1.12%

A. Performance comparison

Classification of the 10,000 test images with the 8-bit fixed-point binary design showed a misclassification rate of 0.80%. Table II compares the misclassification rates of the implemented bitstream-based designs for different number of operation cycles. For the conventional random stream-based approach, we report the mean of 20 trials running the simulation for statistical significance. As reported, the proposed deterministic bitstream-based designs achieve a better classification rate than the conventional random-stream based one for all different operation cycles (bit-stream lengths).

As can be seen in Table II, the optimum number of operation cycles to achieve high-quality results with the proposed designs is *only* eight cycles. Choosing Sobol Seq. 1 and Sobol Seq. 4, for example, as the two required sources of numbers in the NN gives a misclassification rate of 0.81% after 8 cycles, which means only one additional error (out of 10,000 test images) than the number of errors with the 8-bit fixed-point binary design. With other combinations of Sobol sequences also a similar level of error rate is achieved.

Due to an imprecise representation of input data, increasing the operation time for one more cycle (operating for a total of 9 cycles) has increased the misclassification rates in the proposed design. The general trend, however, is a decreasing behavior in the error rate when the number of operation cycles increases. After 2^6 cycles, exactly the same or even a lower misclassification rate than the 8-bit fixed-point design is obtained. The reason behind observing a lower rate compared to the baseline fixed-point design is that the imprecise computation with truncated bit-streams has turned a couple of misclassifications into correct classification. Comparing the misclassification rates of the eight cycle case and the 2^6 or longer cases, with longer bit-streams only a fewer number of misclassifications (out of 10,000 test images) can be achieved. Considering the fact that energy is the product of power and processing time, eight is the optimum number of cycles for the proposed design. If the application accepts higher misclassification rates, even four cycles might satisfy the quality expectations (with a rate of around 1.1%) and become an efficient termination point for the operations.

An advantage of using the proposed design is that, in contrast to the prior random bitstream-based designs, the results are deterministic (standard deviation of zero) and hence reproducible. We can guarantee the same error rate every time processing the same test inputs. Due to its inherent randomness, the conventional random SC design has a higher standard deviation and also a higher worst misclassification error for different bit-stream lengths.

TABLE III
AREA (μm^2), CRITICAL PATH (ns), POWER (mW) (@ MAXIMUM WORKING FREQUENCY), AND ENERGY CONSUMPTION (pJ) OF THE CONVOLUTION ENGINE (5X5 FILTER)

Design Approach	Area (μm^2)	CP (ns)	Power (mW)	Energy/cycle (pJ)
Fixed-Point Binary (Non-Pipelined)	19,902	1.60	15.15	24.24
Fixed-Point Binary (Fully-Pipelined)	31,736	0.95	26.52	25.19
Proposed LD Bitstream-Binary	851	0.88	1.28	1.13

TABLE IV
SYNTHESIS RESULTS OF THE CONVOLUTION LAYER (24×24 CONVOLUTON ENGINES + BITSTREAM GENERATORS)

Design Approach	Area (μm^2)	CP (ns)	Power (W)	Energy/cycle (nJ)
Fixed-Point Binary (Non-Pipelined)	10,966,339	1.94	7.47	14.50
Fixed-Point Binary (Fully-Pipelined)	17,212,059	1.28	12.56	16.07
Proposed LD Bitstream-Binary	581,752	1.12	0.46	0.52

B. Cost comparison

The synthesis results of the convolution engine which is based on a 5×5 filter is reported in Table III. A total of $20 \times 24 \times 24$ convolution units is required in the first layer of the NN if performing all convolutions in parallel. Table IV shows the synthesis results of the first convolution layer for the case of parallel processing all pixels of an input image with one filter (implementing 24×24 convolution units in parallel). For the proposed design, we used the Sobol sequence generator of [13] to generate bit-streams. We did not consider the cost of the bit-stream generators in the numbers reported in Table III because the two Sobol sequence generators are re-used in converting all the input image and weight data, and the effective cost of the comparators will be insignificant considering the fact that each generated bit-stream is being used by a large number of convolution units. We included the bit-stream generators costs in the numbers reported in Table IV.

For the fixed-point binary design, we report the synthesis results for two cases of non-pipelined and fully-pipelined implementations. Due to replacing the costly 8-bit fixed-point multipliers with simple AND gates, more than 23$\times$ (37$\times$) hardware area saving is observed from the non-pipelined (fully-pipelined) fixed-point design to the proposed design for implementation of the convolution engine. The critical path

and power consumption are also significantly decreased with the proposed design. As reported in Table IV, for the entire convolution layer in which we consider the overhead cost of bit-stream generators, the hardware area cost is reduced by $19\times$ ($30\times$).

The important metric to evaluate the efficiency of the proposed design is still energy consumption. Energy is evaluated as the product of processing time and power consumption. The output of the convolution unit in the non-pipelined fixed-point design is ready in only one clock cycle. This results in a total energy consumption of 24 pJ for the convolution unit (Table III) and 14.5 nJ for the entire convolution layer (Table IV). The fully-pipelined design is faster but costs significantly higher area and power, and consumes 16 nJ energy per cycle for the entire convolution layer.

For the proposed design the energy consumption depends on the number of cycles processing bit-streams. Multiplying the number of cycles by the energy per cycle from Table IV, an energy consumption of 4.1 nJ is measured when operating for eight cycles ($8 \times 0.52nJ$). So the proposed design achieves more than 70% energy saving compared to the non-pipelined fixed-point design while preserving the quality of the results. Although pipelining the fixed-point design increases the maximum working frequency (reduces the critical path) and the throughput, it costs higher hardware area and power consumption, longer total end-to-end latency, and higher energy for processing each input data.

With the conventional random stream-based design a significantly longer processing time (e.g., 64 to 256 cycles) is required to achieve the same classification rate as the fixed-point design. This longer processing time makes the prior bit-stream-based designs energy inefficient compared to the fixed-point and the proposed design.

Note that we implemented 24×24 parallel 5×5 convolution engines to parallel process a 28×28 input test image with one filter. To parallel process the input image with 20 filters we need to implement 20 copies of the implemented design. The bit-stream generators converting the input image data and also the Sobol number generators will be shared between more convolution engines and so the overhead cost of bit-stream generation will be further reduced.

V. Conclusion

This work proposes a low-cost and energy-efficient design for hardware implementation of convolutional neural networks (NN). Low-discrepancy deterministic bit-streams and simple standard AND gates are used to perform fast and accurate multiplication operations in the first layer of NN. Compared to prior random bit-stream-based designs, the proposed design achieves a lower misclassification rate for the same processing time. Evaluating LeNet5 NN, the proposed design achieved the same classification rate as the conventional fixed-point binary design with 70% savings in the energy consumption of the first convolutional layer. If accepting slight inaccuracies, higher energy savings is also possible by processing shorter bit-streams. In future work, we will extend our evaluation to other NN architectures.

Acknowledgment

This work was supported in part by National Science Foundation grant no. CCF-1438286. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] A. Alaghi and J. P. Hayes. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):92:1–92:19, 2013.

[2] A. Alaghi and J. P. Hayes. Fast and accurate computation using stochastic circuits. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, Belgium, 2014.

[3] 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 VLSI Systems*, 2017.

[4] S. R. Faraji and K. Bazargan. Hybrid binary-unary hardware accelerators. In *2019 24th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2019.

[5] B. Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, pages 37–172. Springer US, 1969.

[6] D. Jenson and M. Riedel. A Deterministic Approach to Stochastic Computation. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, New York, NY, USA, 2016.

[7] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi. Dynamic Energy-accuracy Trade-off Using Stochastic Computing in Deep Neural Networks. In *DAC '16*, New York, NY, USA, 2016. ACM.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[9] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 13–18, March 2017.

[10] B. Li, M. H. Najafi, and D. J. Lilja. Using Stochastic Computing to Reduce the Hardware Requirements for a Restricted Boltzmann Machine Classifier. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 36–41, New York, NY, USA, 2016. ACM.

[11] J. Li, A. Ren, Z. Li, C. Ding, B. Yuan, Q. Qiu, and Y. Wang. Towards Acceleration of Deep Convolutional Neural Networks using Stochastic Computing. In *2017 22nd ASP-DAC*, pages 115–120, Jan 2017.

[12] Z. Li, A. Ren, J. Li, Q. Qiu, B. Yuan, J. Draper, and Y. Wang. Structural Design Optimization for Deep Convolutional Neural Networks using Stochastic Computing. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 250–253, March 2017.

[13] S. Liu and J. Han. Energy Efficient Stochastic Computing with Sobol Sequences. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 650–653, March 2017.

[14] S. Liu and J. Han. Toward Energy-Efficient Stochastic Circuits Using Parallel Sobol Sequences. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(7):1326–1339, July 2018.

[15] M. H. Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani. Time-Encoded Values for Highly Efficient Stochastic Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(5):1644–1657, May 2017.

[16] M. H. Najafi and D. Lilja. High Quality Down-Sampling for Deterministic Approaches to Stochastic Computing. *IEEE Transactions on Emerging Topics in Computing*, 2018.

[17] M. H. Najafi, D. J. Lilja, and M. Riedel. Deterministic Methods for Stochastic Computing Using Low-discrepancy Sequences. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '18, pages 51:1–51:8, New York, NY, USA, 2018. ACM.

[18] W. Qian, X. Li, M. Riedel, K. Bazargan, and D. Lilja. An Architecture for Fault-Tolerant Computation with Stochastic Logic. *Computers, IEEE Trans. on*, 60(1):93–105, Jan 2011.

[19] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan. SC-DCNN: Highly-Scalable Deep Convolutional Neural Network Using Stochastic Computing. In *ASPLOS'17*, New York, USA, 2017.

[20] H. Sim and J. Lee. A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017.

[21] H. Sim, D. Nguyen, J. Lee, and K. Choi. Scalable Stochastic Computing Accelerator for Convolutional Neural Networks. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017.