Designing Neuromorphic Processor with On-Chip Learning

Jeong Woo Park[1], Dong Suk Jeon[a]
Graduate School of Convergence Science and Technology, Seoul National University
E-mail : [1]jeffjw@snu.ac.kr

Abstract - **In this paper, we present a neuromorphic processor that could learn to classify images through spiking information between neurons. Data is only computed locally, and there is no need for energy-hungry data transmissions that is required for any other machine learning algorithms. Careful algorithmic adaptations, along with novel hardware implementation of softmax functions are introduced to deliver maximum performance on image classification tasks while minimizing energy consumption. The design was fabricated in TSMC 65nm technology and consumes mere 23.6mW at a scaled voltage of 0.8V with 20MHz clock frequency, while showing a throughput of 73.9M pixels/second for supervised training on handwritten images**.

***Keywords*—Edge computing, Neuromorphic computing, Image classification, Learning systems, Multi-layer perceptrons**

I. INTRODUCTION

Artificial neural networks (ANN) have shown great success in computer vision tasks such as MNIST, CIFAR, ImageNet, etc. A special type of artificial neural network is spiking neural networks (SNN). Instead of using continuous-valued activation values, SNN uses binarized activations just like biological neurons. Apart from the fact that SNNs are more biologically plausible than standard neural networks, SNNs could be advantageous in terms of energy efficiency over standard ANNs in hardware implementations. This is mostly because of reduced data transmissions between processing elements, since only one bit is required to represent activations of neurons [1].

However, this reduction in data transmission is no longer possible to build a back-propagation based training processor. Back-propagation requires that a continuous value to be propagated between neurons: this breaks our assumption that only spikes communicate between processing elements. Because of this, most neuromorphic chips either focused on implementing only inference, such as the TrueNorth chip developed by IBM [2], or used a spike-timing-dependent-plasticity (STDP) rule [3], which often shows poorer performance compared to back-propagation based learning, to avoid this constraint.

In this paper, we introduce a neuromorphic processor, extending our prior work [16] with the focus on the design process and efficient hardware implementation of the softmax function. In section II A-B, we summarize the general system with the work from [16]. The adopted algorithm, a variant of difference target propagation [5], has the property that there is no high bit-width error path required for training, while maintaining a relatively high performance in classifying images. Moreover, the spiking nature of the feedbacks allow hardware designers to exploit sparsity without modifying the original algorithm, enabling energy-efficient training.

a. Corresponding author; djeon1@snu.ac.kr

Manuscript Received Jan. 29, 2020, Revised Mar. 26, 2020, Accepted Mar. 27, 2020

This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/bync/3.0) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

II. SYSTEM DESIGN

A. Algorithm Adaptation

The original algorithm described in [4] has some properties which are not desirable in digital hardware implementation. The main drawback is that it uses a double exponential post-synaptic potential for transition of spike data into internal neuron voltage.

$$\kappa(t) = \frac{e^{-\frac{t}{\tau_L}} - e^{-\frac{t}{\tau_S}}}{\tau_L - \tau_S}$$

$$PSP_j{}^B(t) = \sum_{s \in X_j} \kappa(t-s)$$

Since exponential functions are hard to implement in digital circuits efficiently, we have developed a different kernel that would significant reduce hardware overhead. The computational saving by implementing our own kernel is shown by calculating the theoretical number of FLOPs required in each time step. Using double exponential kernels, 409,840 FLOPs are required per timestep, while only 4,570 is required on average using implemented kernel.

B. Processor Architecture

The processor has an architecture of 1 input layer with 784 neurons, 2 hidden layers with 200 neurons each, and an output layer with 10 neurons. As shown in Fig. 2, each hidden neuron has RAMs that contains synaptic weights, and tracks how many times they have been called.

Although having one RAM per neuron is more desirable in terms of energy savings and biological similarity, it is often area-inefficient as smaller RAMs result in more area overhead to implement the same amount of memory. Thus, we have decided on 5 neurons sharing a single RAM by increasing the width of the RAM. This was possible since the implemented algorithm requires the same column of the RAM to be called at each time step.

Fig. 1 shows the overall architecture of implemented design. All communication between neurons are transmitted

through spike decoders, which translates spiking information to addresses. This architecture is expected to increase area efficiency, as connecting neurons to every neuron in the next layer will cause connecting wires to dominate area usage [6].

Inside the hidden layers in Fig.1, 200 neurons in a neuron clusters, along with two buffer memories totaling 2.2 or 4.8Kb and an update-value calculation unit with a sigmoid

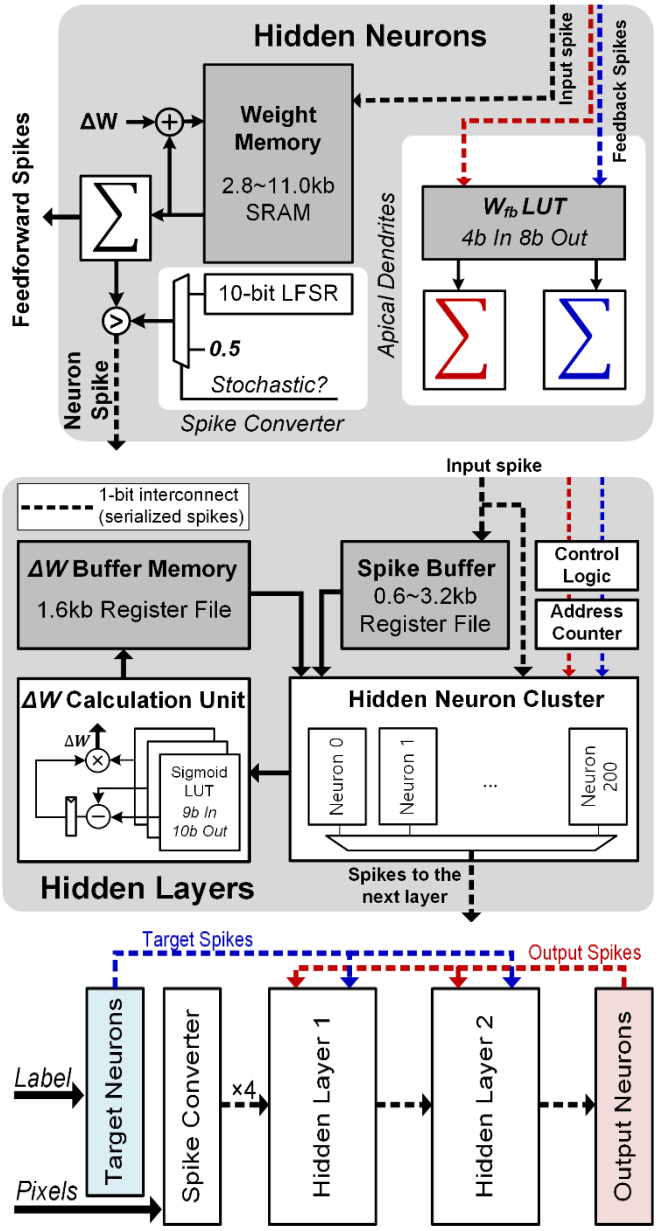

Fig. 1. Block diagram of implemented design, from hidden neuron, hidden layer, and top-level. Figure reprinted from [16].

function implemented as LUT. Inside the hidden neuron clusters, 5 neurons share a weight memory for higher area efficiency.

The hidden neuron in Fig. 1 shows that each hidden neuron accumulates three values: previous layer spikes, output layer spikes, and target spikes. The accumulated value of previous layer spikes is used to determine the firing probability of that neuron, and accumulated value from output layer spikes and target spikes are used to calculate weight update values.

C. Pipelining

To achieve efficient training, we propose to pipeline steps required for updating on an image by breaking down training procedures to layer processing and update value processing. Unlike conventional deep learning algorithms, this SNN algorithm relies on very small number of global spiking information to update its weights. This enables efficient pipelining, as other algorithms will require unrealistic

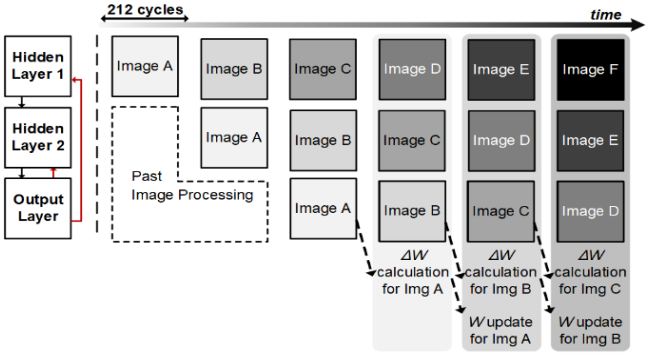

Fig. 2. The pipelining scheme and out-of-order updates. Figure reprinted from [16].

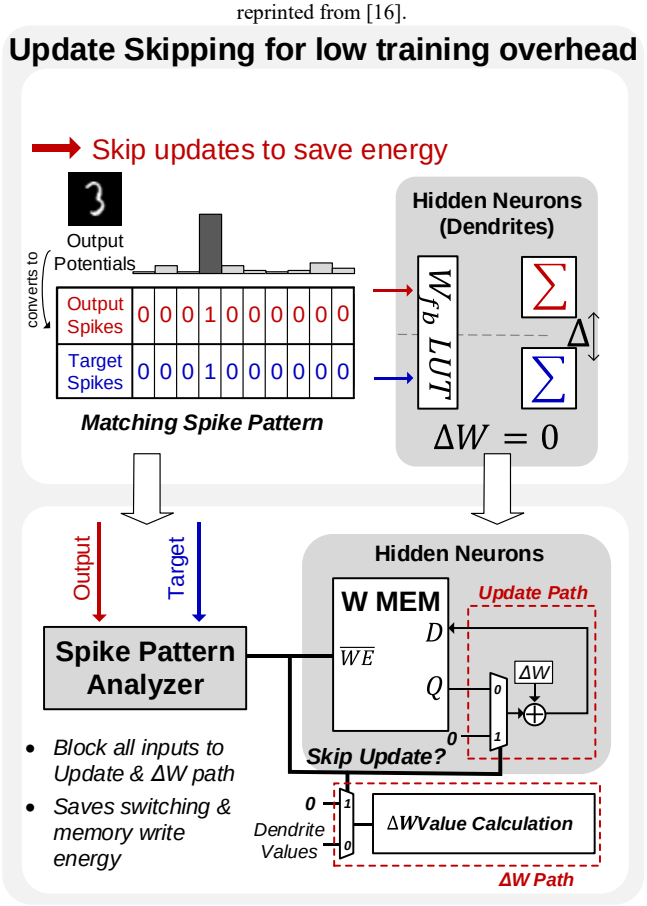

Fig.3. The update skipping mechanism implemented for our system. Figure reprinted from [16].

number of buffer registers for pipelining update value processing step, which is disastrous for both area and power of digital system. Moreover, update value in each column is identical for processing on a single image, which further reduces buffer register overhead for pipelined training.

Our implemented pipelining scheme is shown in Fig. 2. This pipelining is expected to increase training speed

drastically, while introducing relatively small overhead of buffer registers for storing spiking information of each neurons. This overhead is around 3k registers, which is around 2.8% of total registers used.

D. Update Skipping

As the ΔW values rely on the difference between the output neuron spikes and the target neuron spikes shown as one-hot-

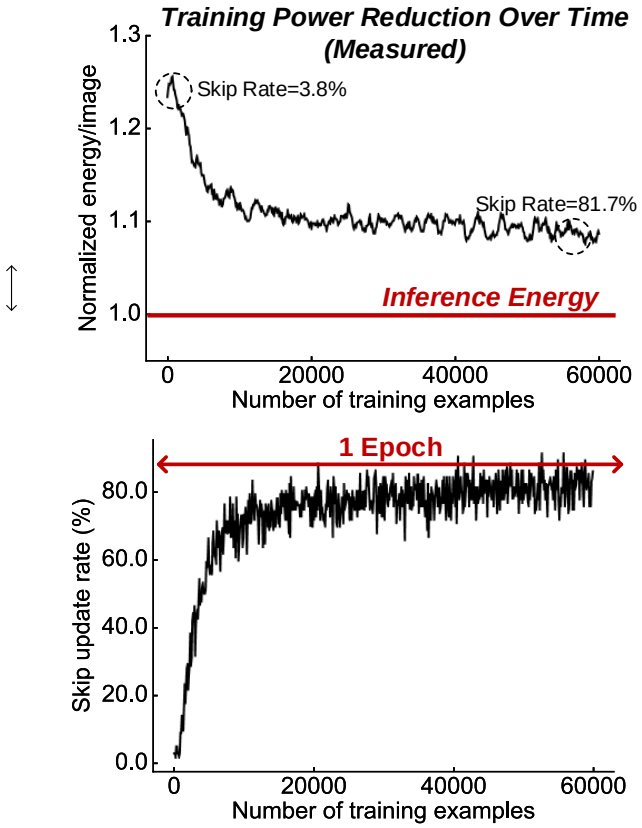

Fig.4. Analysis of the effect of update skipping mechanism in our system. Figure reprinted from [16].

encoded vector of the target label, the weight updates for the entire network is zero if the output neuron spikes and the target neuron spikes are matching exactly. To exploit this observation, we implemented an update skipping mechanism, as shown in Fig. 3. A spike pattern analyzer deserializes the output and target spikes, and asserts a skip update signal when they match exactly. When this occurs, the write ports to the SRAMs are deasserted, and the inputs to the combinational-logic based weight update value calculation unit is blocked to prevent unnecessary switching of logics. The effect of this update-skipping mechanism is shown in Fig.4. The skip update grows to 94.5% after 100 epochs, and the training energy compared to inference energy reduces to only 7.5%, compared to 25.6% in the beginning of the training.

E. Efficient Softmax Implementation

Moreover, we propose an efficient implementation for a softmax function, which is used during the inference phase in recent attention mechanisms [14] or for the last layer in the training phase in classification tasks. The mathematical definition of softmax is shown in the following equation.

$$\text{Softmax}(\bar{Z}) = \frac{e^{z_k}}{\sum_{z_k \in Z} e^{z_k}}$$

Notice that the softmax value of vector $\bar{z}$ remains the same even if some constant value is subtracted element-wise. This property is used in our implementation to restrict the maximum element value to 0 for better numerical properties.

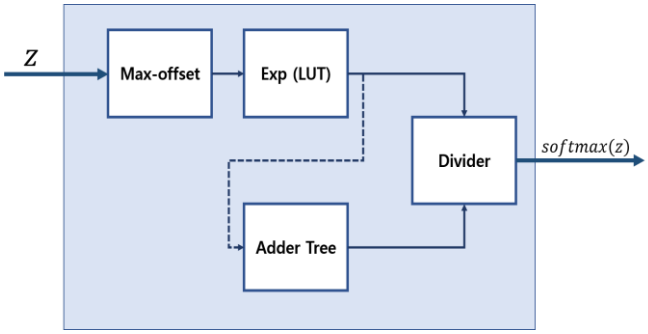

Fig.5. A naïve softmax implementation

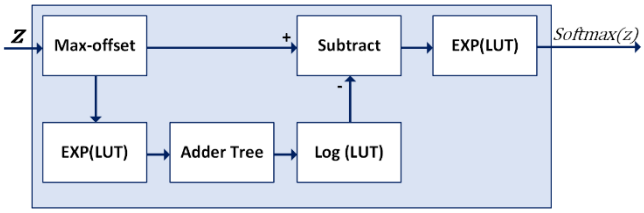

Fig.6. A log-space softmax implementation.

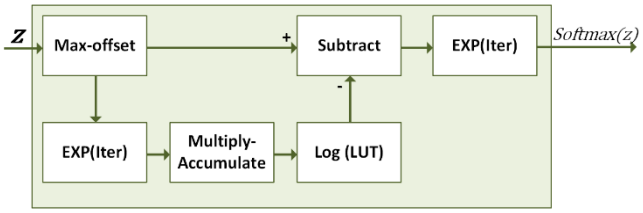

Fig.7. Proposed iterative softmax implementation.

TABLE I. Implementation of LUT-based and Iterative Method

	LUT-BASED[15]	PROPOSED
Area	37835μm²	1623.6μm²
Energy	240.36pJ	197.09pJ
Cycles	24	321
Power	2.003mW	0.307mW

*Synthesized results in 65nm process
** Our implementation

We first show two existing methods for implementing a softmax function in hardware: a naïve implementation using dividers, and a log-space implementation which avoids dividers [15]. Fig. 5 shows the block diagram for naïve implementation, and Fig.6 shows the block diagram for the log-space implementation. As it can be seen from the figure, both implementations require large look-up tables that exponentially require more resources as bit-precision grows higher.

We propose a new implementation method for softmax function that frees the need for expensive look-up tables using iterative calculation with a multiplier and a 16-entry register file. We show the block diagram of our implementation in Fig. 7. Instead of storing all data for every possible case 2^N for fixed point N bit numbers, only N cases are stored. This is possible through the algorithms described by equation 1 and equation 2. Equation 1 shows the fixed-point two's complement representation of X_Q in N-bit fixed point format $\{x_0, x_1, \dots, x_{N-1}\}, x_n \in \{0,1\}$ with fraction point at M. By using the property of exponentials, it could be extended to using iterative multiplication on each of the bits that are non-zero as shown in equation 2. The stored values are therefore N cases; $e^{2^{M-n}}$ for each of n cases (with exception of the most significant bit, which should be stored as $e^{2^{-M}}$)

$$Eq.1) \quad X_Q = 2^M \sum_{n=0}^{N-1} 2^{-n} x_n (-1)^{(n==0)}$$

$$Eq.2) \; e^{X_Q} = e^{2^M \sum_{n=0}^{N-1} 2^{-n} x_n (-1)^{(n==0)}} = e^{x_n 2^{-M}} \prod_{n=1}^{N-1} e^{x_n 2^{M-n}}$$

By using this iterative approach, bulky look-up tables could be freed, leading to a lower area and energy per softmax as shown in Table I. Moreover, our implementation shows better robustness for the signal-to-noise ratio for a softmax function with large input dimensions, as shown in Fig. 8. This may be due to higher precision in exponential values, whereas the LUT-base implementations have precision bounded by the lookup table.

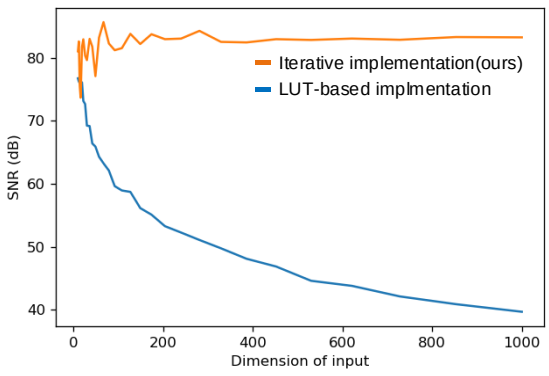

Fig.8. SNR for softmax on input vectors with large dimensions.

III. Simulation and Measurement

We developed and implemented our digital design using Vivado on Virtex-7 FPGA. A test environment was provided using a separate FPGA (XEM7310), as shown in Fig. 9. Our Verilog implemented model was tested in RTL simulation as well as on FPGA, and showed error rate of 2.2% on MNIST test dataset after training for 200 epochs.

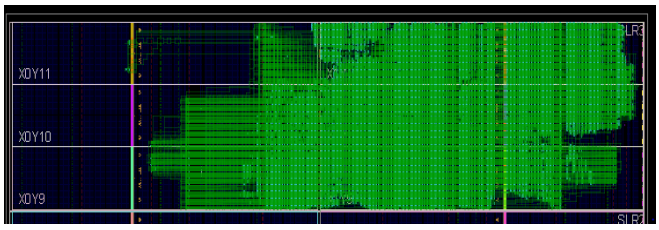

Fig.9. Implementation of synthesized design on Virtex-7 FPGA board.

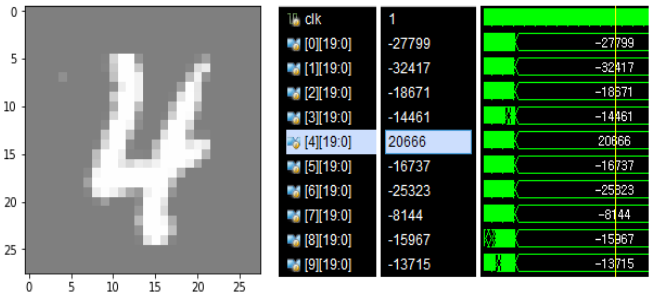

Fig.10. Part of our RTL simulation on classifying multiple MNIST test images. The image in right is fed to implemented neural network, and the simulation result is show in the left. It correctly classifies the image, giving '4' the highest score.

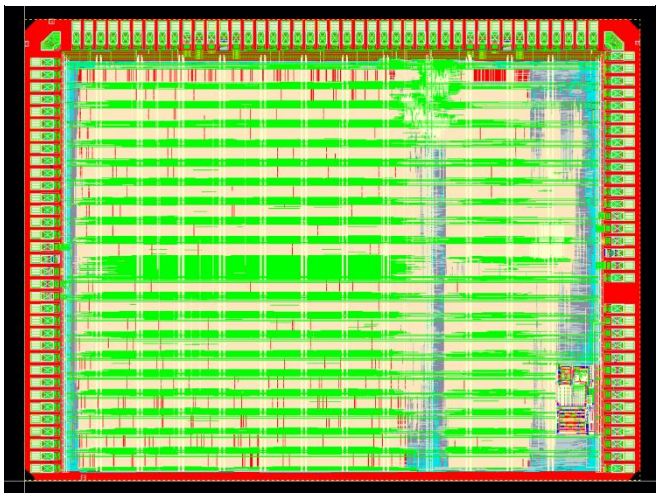

Fig.11. Layout of our implemented design.

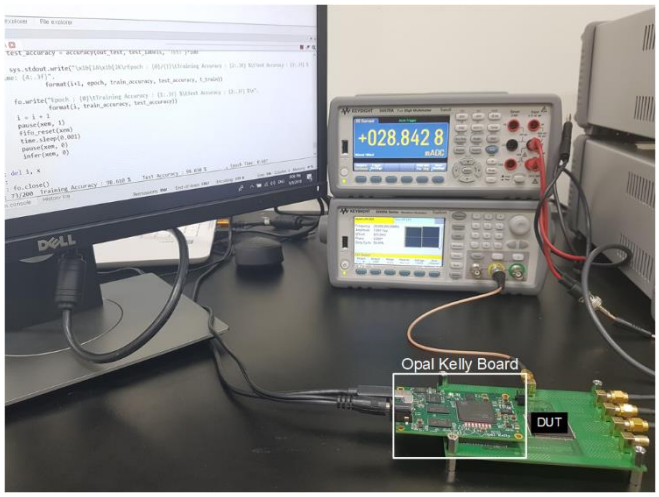

Fig.12. The verification setup. The opal Kelly FPGA board only processes data transfers between the PC and our system. Figure reprinted from [16].

TABLE II. Details of our implemented system

	OUR WORK
Technology	65nm
On-Chip Training	Yes
MNIST Accuracy	97.83%
Algorithm	SNN
Prediction Energy	236.nJ/Image
Training Overhead	7.5%*
Throughput (Inference)	100K-250K FPS
Throughput (Training)	94.3K-235.8K FPS
Power	23.6mW-96.2mW
Supply Voltage	0.8V-1.2V
Frequency	20-50MHz
Efficiency	3.40 TOPS/W

*Average of 100 training epochs

Design was verified using Opal-kelly board (XEM 7310), which is a Xilinx FPGA integrated module that provides application programmable interface (API) such as python, C++, etc. MNIST images were transmitted to the design through USB3.0 for high throughput (100.8MHz). Images are entered to FIFO in the XEM 7310, and our designed processor calls the FIFO to read images sequentially. 4 pixels of images are called to the processor in a single time step, meaning that 32-bits are entered to the processor per clock cycle. This image data is only read and entered to the design when the output 'image-request' signal is asserted. After finishing processing on the input image, the processor writes classification scores of each label, through indicating that classification scores are ready with 'output-valid' signal and acting as 'write' signal to FIFO on Opal-Kelly board. The RTL simulation result of this testing environment is shown in fig. 10.

Scores are 14-bit numbers, while first 4 bits only act to indicate which label it belongs to. By writing python modules to automatically send image data to the input FIFO in Opal Kelly board, and decode 14-bit numbers written in output FIFO to classification scores, we could automate the process of evaluating performance on classifying the MNIST dataset. The layout of our implemented system is shown in fig. 11.

Using these verification methods, we also measured our TSMC 65nm digital chip and verified the functionality. The verification setup is shown in fig.16. The chip operates at 20~50MHz with a voltage range of 0.8~1.2V. The details of our chip is summarized in Table II. Our implementation shows a training throughput of 235.8k FPS on MNIST dataset, requires 51.3 seconds of real-time training to finish 200 epochs of training. Compared to using a back-propagation algorithm (BP) deployed in commercial GPU(Titan-X), GPUs take more than 1104.8 seconds to finish 200 epochs of training. Accounting for the quicker convergence of BP-algorithms, it takes 63 epochs to reach the same target accuracy of 97.83%, making our implementation x5 faster in terms of effective training speed.

V. CONCLUSION

We have adapted an existing SNN algorithm to be hardware-efficient. This reduced the number of FLOPs to only 1.1% of the original algorithm, while maintaining performance on classification. The Verilog design is proved to show classification functionality, and several rigorous steps were used to verify that the synthesized and physical layout design is equivalent to the Verilog design. This work shows that training neural networks only using spike communication is feasible and energy efficient.

Moreover, the implemented pipelining scheme, which is enabled through significantly less buffer registers for pipelined training, increases training speed drastically while increasing energy efficiency per image. Our target speed outruns GPU-accelerated codes run on NVIDIA Titan-X by 5 times on a BP algorithm with a batch size of 64. Finally, our newly proposed hardware implementation for softmax function consumes less energy and area compared to prior works while showing better SNR in large input dimension sizes.

REFERENCES

[1] 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).

[2] P. A. Merolla et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface." Science 345.6197 (2014): 668-673.

[3] P. Knag et al. "A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding." IEEE Journal of Solid-State Circuits 50.4 (2015): 1070-1079.

[4] Guerguiev, Jordan, Timothy P. Lillicrap, and Blake A. Richards. "Towards deep learning with segregated dendrites." eLife 6 (2017).

[5] D.-H. Lee et al. "Difference target propagation." Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Cham, 2015.

[6] J. K. Kim et al. "Efficient hardware architecture for sparse coding." IEEE Transactions on Signal Processing 62.16 (2014): 4173-4186.

[7] S. Gonugondla et al. "A 42pJ/Decision 3.12TOPS/W Roust in-memory machine learning classifier with on-chip training." in IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers, Jan. 2018, pp. 490-491

[8] A. Amravati et al. "A 55nm time-domain mixed-signal neuromorphic accelerator with stochastic synapses and embedded reinforcement learning for autonomous micro-robots." in IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers, Jan. 2018, pp. 124-125.

[9] C. Tsai et al. "A 41.3/26,7pJ per neuron weight RBM processor supporting on-chip learning/inference for IoT applications." IEEE Journal on Solid-State Circuits (JSSC), vol. 52, no. 10, pp.2601-2612, Oct. 2017.

[10] P. Whatmough et al. “A 28nm SoC with a 1.2GHz 568nJ/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for IoT applications.” in IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers, Jan. 2017, pp. 242-243.

[11] F. Buhler, et al., “A 3.43TOPS/W 48.9pJ/Pixel 50.1nJ/Classification 512 analog neuron sparse coding neural network with on-chip learning and classification in 40nm CMOS.” in IEEE Int. Symp. VLSI Circuits Dig. Tech. Papers, June 2017, pp. 30-31

[12] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, “A 640M pixel/s 3.65mW sparse event-driven neuromorphic object recognition processor with on-chip Learning” in IEEE Int. Symp. VLSI Circuits Dig. Tech. Papers, June 2015, pp. 61–62

[13] S. Esser et al. “Back propagation for energy-efficient neuromorphic computing.” Advances in Neural Information Processing Systems (NIPS), 2015.

[14] A. Vaswani et. al, “Attention is all you need,” arXiv:1706.03762 [cs.CL], Jun. 2017.

[15] B. Yuan, “Efficient hardware architecture of softmax layer in deep neural network,” in Proc. IEEE International System-on-Chip Conference (SOCC), Sep. 2016.

[16] J. Park, J. Lee, and D. Jeon. "A 65-nm Neuromorphic Image Classification Processor with Energy-Efficient Training Through Direct Spike-Only Feedback." IEEE Journal of Solid-State Circuits 55.1 (2019): 108-119.

Jeongwoo Park received the B.S. degree from the department of electrical and computer engineering, Seoul National University, Seoul, South Korea, in 2017, where he is currently pursuing the Ph.D. degree. His research interests include neuromorphic algorithms and systems, quantized neural network training, and efficient ASIC design for deep learning inference/training.

Dong Suk Jeon received the B.S. degree in electrical engineering from Seoul National University, Korea, in 2009, and Ph.D degree from University of Michigan, Ann Arbor, MI, in 2014. He is currently an assistant professor in Seoul National University, Korea.

His main interests are digital signal processing, low power integrated circuits, System-on-Chip (SoC) architecture.