Hardware Efficient Weight-Binarized Spiking Neural Networks

Chengcheng Tang
Dept. of Electrical and Computer Engineering
University of Alberta
Edmonton, Alberta, Canada
ctang8@ualberta.ca

Jie Han
Dept. of Electrical and Computer Engineering
University of Alberta
Edmonton, Alberta, Canada
jhan8@ualberta.ca

***Abstract*—The advancement in spiking neural networks (SNNs) provides a promising and alternative approach to conventional artificial neural networks (ANNs) with higher energy efficiency. However, the significant requirements on memory usage presents a performance bottleneck on resource constrained devices. Inspired by the notion of binarized neural networks (BNNs), we incorporate the design principles in BNNs into that of SNNs to reduce the stringent resource requirements. Specifically, the weights are binarized to 1 and -1 for implementing the functions of excitatory and inhibitory synapses. Hence, the proposed design is referred to as a weight-binarized spiking neural network (WB-SNN). In the WB-SNN, only one bit is used for the weight or a spike; for the latter, 1 and 0 indicate a spike and no spike, respectively. A priority encoder is used to identify the index of an active neuron as a basic unit to construct the WB-SNN. We further design a fully connected neural network that consists of an input layer, an output layer, and fully connected layers of different sizes. A counter is utilized in each neuron to complete the accumulation of weights. The WB-SNN design is validated by using a multi-layer perceptron on the MNIST dataset. Hardware implementations on FPGAs show that the WB-SNN attains a significant saving of memory with only a limited accuracy loss compared with its SNN and BNN counterparts.**

***Index Terms*—Spiking neural networks, priority encoder, binarized weights, field programmable gate arrays (FPGAs)**

I. Introduction

To emulate the human brain, spiking neural networks (SNNs) have shown a significant potential to save energy and hardware usage compared with conventional artificial neural networks (ANNs) [1]. While understanding how the brain processes complex information is still an ongoing topic in neural science [2], [3], either the firing rate (or frequency) [4], [5] or the firing latency (or order) is used to encode the signal intensity in a neuron. Such encoding schemes enable the operation of SNNs to be event driven, that is, neurons only have to update their states upon the arrival of spikes, thus saving energy.

Another merit of spike-based computation is its potential to decompose costly arithmetic operations into repetitive simple tasks. For instance, the weighted sum can be transformed into a series of additions over a period of time, which leads to a multiplier-less and highly parallel system. One example is the neuromorphic chip, TrueNorth [6], which contains one million neurons working in parallel and each neuron uses a random number generator, an integrator and a threshold unit to instantiate the augmented integrate-and-fire (IF) model without using multipliers.

As the size of SNNs grows, however, memory becomes the bottleneck for further improving the performance. The increasing number of parameters has brought challenges to hardware design to effectively access the distributed memories. This inefficacy has become the major limitation of implementing SNNs on hardware-constrained platforms.

As another efficient model, binarized neural networks (BNNs) employ binary values ($+1$ and -1) for both weights and activations, so the key arithmetic operation, multiply-and-accumulate, can be replaced by an XNOR-count operation [7]. Such simplifications enable considerable memory savings without incurring a significant loss in accuracy.

In this paper, the design principle of binarization in BNNs is introduced into SNNs by leveraging the binary weights ($+1$ and -1) while keeping neuron activations to be $+1$ (for firing a spike) or 0 (for no spike). This design is referred to as a weight binarized spiking neural network (WB-SNN). It is different from the so-called binary weight SNN (BW-SNN) [8] in that the weights are -1, 0 and 1 in the BW-SNNs. Therefore, a sign extension is required for the weights, so incurring additional hardware. However, the weights in the proposed WB-SNNs are truly binary in $+1$ and -1 that can be stored as 1 and 0 in the memory, respectively. Then, the binary weights can be accumulated by using counters during the IF process.

The main contributions of this paper are: 1) The memory is substantially reduced in the WB-SNNs because of the binary weights. Additionally, the XNOR-count operation in BNNs is replaced by the priority encoders (PEs) shared by all neurons in one layer for a higher hardware efficiency. 2) This design is extended into linear (fully connected) layers so that one of the basic neural network models, the multi-layer perceptron (MLP), is supported. 3) The design is verified on the Modified National Institute of Standards and Technology (MNIST) dataset implemented on field programmable gate arrays (FPGAs). Substantial hardware savings are obtained when compared with some baseline designs.

II. PRELIMINARIES

A. *Spiking Neural Networks (SNNs)*

SNNs imitate the bioelectric activities observed in biological systems, as illustrated in Fig. 1. A neuron in SNNs receives several channels of input spike trains from other neurons in the former layer through its synapses. Each incoming spike will change the internal membrane potential of the neuron by an amount determined by the amplitude of the synapse weight. When this potential exceeds a predefined threshold V_{th}, an output spike is generated and the potential value is reset. The structure of SNNs is composed of several layers of such neurons that rely on these spikes to communicate and transmit information.

Several neuron models are applicable in SNNs, each of which uses a different function to describe the dynamics of neuron potential. Among these various representations, the IF model [1] is the simplest and most commonly used one. Its behavior can be expressed as

$$v_i = \int \sum_j W_{i,j} S_j(t) dt, \tag{1}$$

where v_i is the internal membrane potential of neuron i in one layer and $W_{i,j}$ is the weight for the synaptic connection between neuron i in the current layer and neuron j in the previous layer. $S_j(t)$ is an impulse function given by a spike train of '0's and '1's.

The computation in SNNs is rather simple because the neuron is activated by event-based spike trains so that the multiplication is transformed into integration. Therefore, no multiplier is needed in the hardware implementation. The variation in the neuron potential can be realized by an integrator that accumulates the synaptic weights on an event-triggered basis. SNNs can also be viewed as one form of binarization in the neuron activation while the weight values stay numeric. Moving one step forward toward binarizing both the weights and activations of neurons will lead to (fully) binarized SNNs. The computation is anticipated to be even simpler.

B. Binarized Neural Networks (BNNs)

The idea of BNNs was proposed in [7] as an effort to replace arithmetic operations with bit-wise operations. In a BNN, the weights and activations are constrained to binary values, such as -1 and $+1$. Therefore, a single bit can be used to quantize both synaptic weights and neuron activations. For example: the bit value 1 is used to represent $+1$ and 0 indicates -1. Thus, the multiplication is equivalent to an XNOR operation.

In order to convert the real values in ANNs into this binary representation, several functions have been utilized, including the deterministic binarization function and stochastic binarization function [7]. The first one is known as $Sign(x)$ with the binarized weight or activation, x_b, given by,

$$x_b = Sign(x) = \begin{cases} -1 & x < 0, \\ +1 & otherwise, \end{cases} \tag{2}$$

where x is a real-valued variable in BNNs as either a weight or an activation. In stochastic binarization, the variables get -1 or $+1$ depending on a probability determined by the hard sigmoid function $\sigma(x)$,

$$\sigma(x) = min(1, max(0, \frac{x+1}{2})). \tag{3}$$

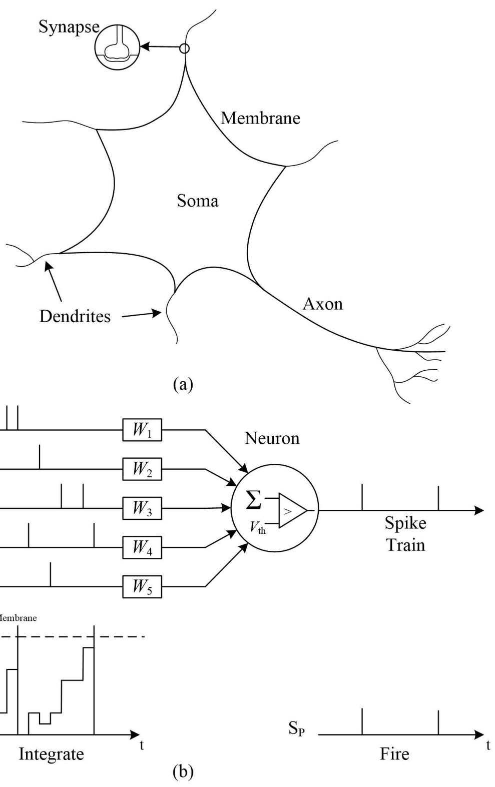


Fig. 1. Analogue between a biological neuron and a neuron in stochastic SNNs. (a) A biological neuron. (b) A neuron in SNNs and the spike train generation mechanism in the Integrate-and-Fire process.

The stochastic binarization needs to generate random bits to simulate probabilistic events, which would incur additional hardware. Therefore, we use the deterministic function to implement the binarization. However, the stochastic binarization function is used in the training process to approximate the derivative of binarization during back propagation.

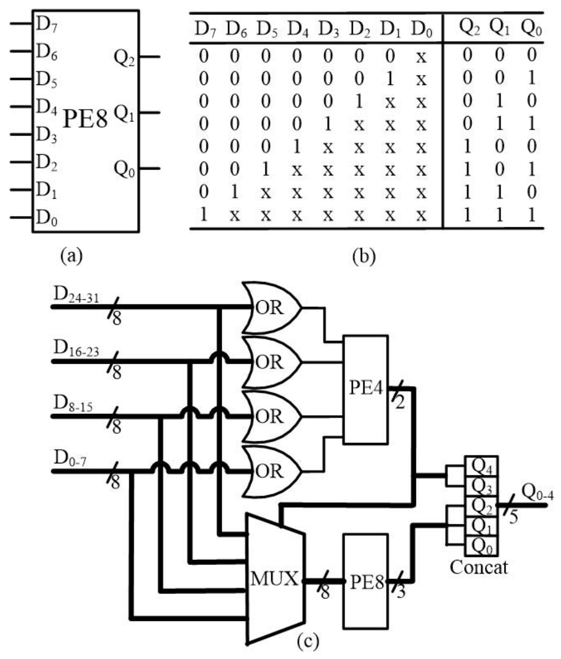
III. A Design Framework for WB-SNNs

This section describes how different types of neuron layers are designed and connected in the WB-SNN. They can be used to build various types of neural networks as reusable units.

A. A Priority Encoder (PE)

A PE is a basic unit widely used in the WB-SNN design. It encodes the index of the activated bit with the highest priority among all inputs. We rely on it to identify the index of an active neuron (i.e., a neuron with an output spike) in a group of neurons. The schematic and truth table of an 8-input priority encoder is outlined in Fig. 2. It should be noted that the least significant input D_0 has no impact on the output. It is reserved for the case when there is no input spike. Thus, it is not instantiated in hardware. This is true for all PEs in this design.

When working with a priority resolving (PRI) circuit [9], all the channels with logic-high inputs will be located and their indexes are sent to the output one by one. In collaboration with the PE, the PRI circuit is used to avoid repeated encoding of the same bit. When the PE finishes encoding the bit with



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	x	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

Fig. 2. (a) An 8 to 3 priority encoder (PE8). (b) The truth table. (c) PE32 constructed by a PE4 and a PE8 [9].

the highest priority, the PRI clears this bit so that the PE can move on to encode the bit with the second highest priority. Since the output of a PE is in the binary format, the width of the output signal is reduced to $\log_2 N$ for an N-channel input. When the PE is placed between two layers of neurons in a network, its output can be utilized to find the memory location and the signal width is also reduced from N to $\log_2 N$.

One useful feature of a PE is its ability to construct a large PE using smaller ones. Fig. 2(c) shows how a 32-input PE is obtained using 4-input and 8-input PEs. This feature provides considerable flexibility to make it fit into neuron layers of different sizes.

B. The Input Layer

While the WB-SNN uses spikes as a medium in all its inner layers, it is not the case for the input layer. The system receives real pixel values in images as inputs. Nonetheless, this does not cause any further alteration to the design except that the counter is replaced with an accumulator to support the accumulation of real values in this layer. Then, depending on if the input values are sparse or not, we can choose whether to omit the PE&PRI block or not. If the input image contains many zero pixel values, we can keep using the PE&PRI block to select only non-zero values and skip accumulating the zeros. If most of the pixel values in the input image are non-zeros, we can remove the PE&PRI block and accumulate every value in the input image. In the former case, the inference latency is reduced while hardware is saved in the latter case.

As can be seen in Fig. 3, the pixel values are sent to the input layer one by one. Although causing extra latency, this operation makes this design hardware efficient. The sign bit of each value is XNORed with the binary weight stored in the memory. The output of the XNOR gate is then concatenated with the magnitude bits and sent to an accumulator for integration. The accumulation is then compared with the predetermined threshold for firing a spike. After this IF process, the neurons in the next layer will receive the generated spike train.

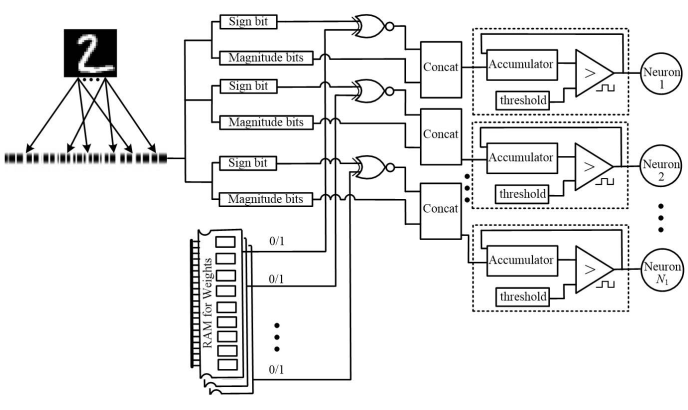


Fig. 3. Input layer design of the WB-SNNs.

C. Design of the Fully Connected Layer

Fully-connected (FC) layers can be found in various types of ANN instantiations such as the MLP. In an FC layer, each neuron is connected via a synapse with every neuron in the previous layer. Therefore, the number of synapses grows quadratically with the layer size, i.e., $O(N^2)$, which is costly when implementing large neural networks.

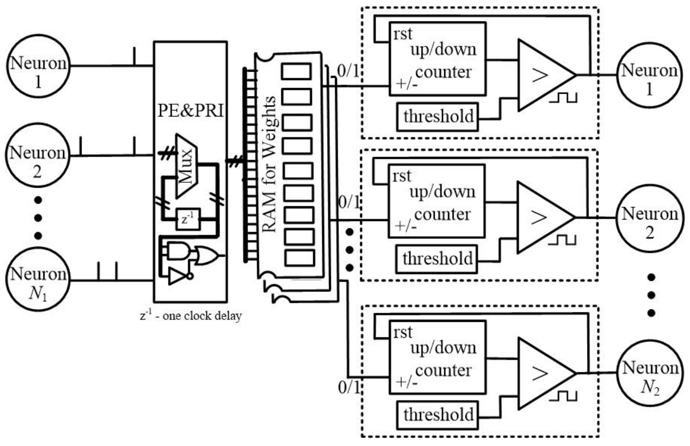


Fig. 4. Fully connected layer design in the WB-SNNs.

The design of the FC layer in the WB-SNN is illustrated in Fig. 4. Since the neuron output is either 0 or 1, all neurons from the previous layer are first connected to a PE&PRI block. This block will transfer the active or inactive status of each neuron into a binary number that indicates the neuron for which the input is 1. This binary number is then sent to the random access memory (RAM) where the weight is stored. The RAMs have multiple layers, each of which corresponds to one output neuron. Note that all RAMs share the same input address signal because each output neuron is connected with all input neurons. The only difference lies in the weight signal which is stored separately in the RAM. This feature also facilitates the parallel processing of the neurons. The output from the RAM is then sent to the processing unit for integration and spike generation.

One may wonder how -1 is stored and realized in the WB-SNN. It is achieved by using an up/down counter, as shown in Fig. 4. When the input is 1, the counter counts up; when the input is 0, the counter counts down. In that way, we only need one bit to store the weight and at the same time, to achieve excitatory or inhibitory synapse behavior. Compared with conventional FC layers, only neurons with active outputs are selected and processed because of the use of PE&PRI. This feature does not only save energy but also reduces the inference latency.

IV. Hardware Implementation and Performance Evaluation

To verify the WB-SNN design, we implemented it on the FPGA as a configurable unit. A widely used NN model, the MLP, was tested and its performance on inference accuracy and hardware utilization were evaluated on the MNIST dataset [10]. The FPGA platform adopted in the experiments is the Xilinx Virtex7 xc7vx485t board. The results were compared with the state-of-the-art designs from the literature.

The MLP is one type of classical feedforward ANNs. It usually consists of several fully connected layers and is widely benchmarked for small image classification datasets such as the MNIST. The number of layers and the number of neurons in each layer can be different and can affect the recognition accuracy and hardware utilization. To fully assess the performance of the WB-SNN design and compare it with previous work, MLP models of different sizes were constructed and evaluated. These models were first trained on a graphic processing unit (GPU) platform and then implemented on the FPGA.

In order to attain a high accuracy while keeping the weights binarized, some common training tactics in BNNs are applied in the training of the WB-SNN models. For example, an L-2 norm regularization $R_2(w) = (1 - |w|)^2$ is added to the total loss function so that the weight value, w, that diverges from 1 or -1 will be penalized [11]. The augmented loss function, $J(\mathbf{W}, \mathbf{b})$, is constructed from the original loss function, $L(\mathbf{W}, \mathbf{b})$, as

$$J(\mathbf{W}, \mathbf{b}) = L(\mathbf{W}, \mathbf{b}) + \lambda \sum_i R_2(\mathbf{W_i}), \tag{4}$$

where $\mathbf{W}$ and $\mathbf{b}$ represent weights and bias terms, respectively. $\mathbf{W_i}$ contains the weights at the ith layer. λ is a parameter that is used to adjust the ratio of regularization applied to the total loss. It is selected to be 0.001 in our experiments.

Simulations of hardware implementations of the WB-SNNs were conducted for several MLP models, including 784-N-N-10, i.e., one input layer with 784 nodes, two hidden layers with N nodes each and one output layer with 10 nodes. Five MLP neural network models of different sizes, i.e., with a hidden layer of 63, 127, 255, 511 and 1023 nodes, were developed and their accuracies during training are shown in Fig. 5. Note that the size of hidden layers we choose is always equal to 2^N-1 with an integer N. That is to reserve an empty space in the PE for the all-zero input. This is one significant feature of

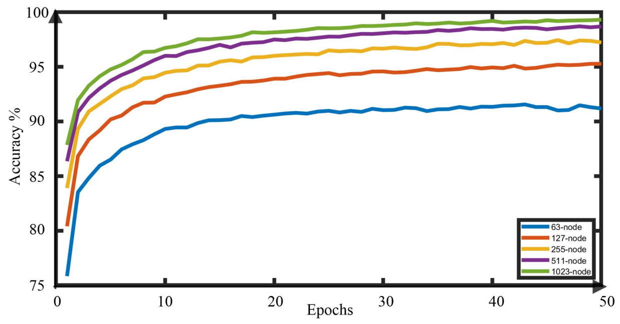


Fig. 5. Training accuracy of MLPs with hidden layers of different sizes.

the WB-SNN. The activations and the weights of neurons are trained to be 0, 1 and -1, 1, respectively. The hard sigmoid function is used during the backward propagation to binarize the weight. The detailed explanation of this function is given in Section II.B. BNNs.

It is not surprising that the training accuracy is positively associated with the number of nodes in the hidden layer, i.e., a larger number of nodes produces a higher accuracy. On the other hand, it is also found that the accuracy can be quite low when the number of neurons in the hidden layer is small because of the binarized weights. For example, the 63-node model achieves an accuracy of only around 91%. However, the accuracy improves when the size of the hidden layer increases. When there are 1023 nodes, the accuracy increases to around 99%. Therefore, the inadequacy due to the binarized weight is largely offset by the number of neurons in the network.

The inference test on the 10,000 images in the MNIST dataset was conducted on FPGAs. The accuracy and execution process were also recorded. Fig. 6 shows the snapshot of the inference process for the 784-1023-1023-10 model. The membrane potential of a neuron and the generated spike stream are presented in Fig. 6(a). It also showcases the IF process of the neuron. However, a neuron is not always active during inference. It remains silent for a certain amount of time and does not elicit spikes. It is a common behavior of the neurons in the WB-SNN. To better illustrate this, we also recorded the number of spikes generated by the neurons in the second hidden layer, as shown in Fig. 6(b) and (c). As can be seen, most of the neurons are located between a sparsity from 30% to 40% and the overall sparsity is 33.01%, which mean that the WB-SNN uses only about one third of all neurons during the inference. The sparsity here represents the percentage of activated neurons. This sparsity helps save energy and execution time in the overall architecture.

The hardware costs of the MLP models with selected sizes and their inference accuracy are summarized in Table I. To help better illustrate the cost effectiveness of the WB-SNN, the hardware utilization of a previous design [12] is listed under the same conditions. As can be seen from Table I, the storage of those weights in the size of Block Random-Access Memories (BRAMs) is inevitably larger than the proposed WB-SNN design. The required usage of BRAMs by the 784-255-255-10 MLP structure is only 11.0, whereas it is 395.5 for a

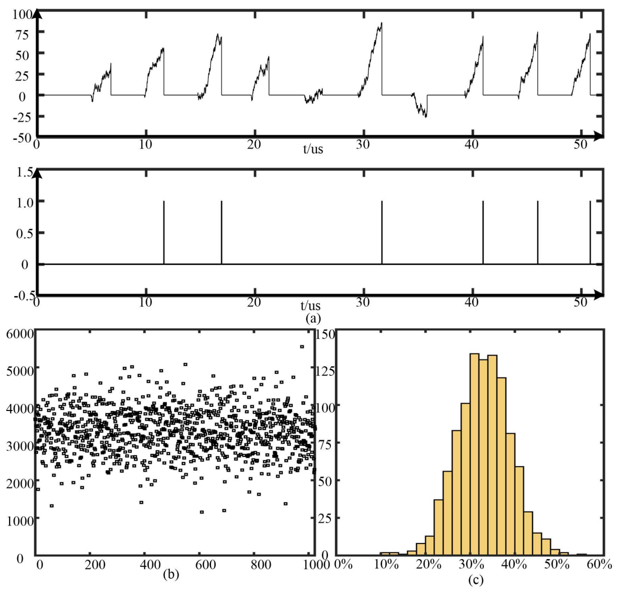


Fig. 6. A snapshot of the inference process in the second hidden layer for the 784-1023-1023-10 network on FPGAs. (a) The membrane potential value of a neuron and the generated spike stream. (b) Number of spikes elicited by the nodes. (c) The histogram of node sparsity.

TABLE I
HARDWARE UTILIZATION AND INFERENCE ACCURACY OF MLP MODELS WITH DIFFERENT SIZES

Structure Size	LUTs	FFs	BRAMs	Inference Accuracy	
784-63-63-10	16555 (5.45%)	20347 (3.36%)	101.5 (9.85%)	97.37%	[12]
	1653 (0.54%)	1043 (0.17%)	3.5 (0.34%)	90.61%	this work
784-127-127-10	24089 (7.93%)	24982 (4.11%)	199.5 (19.37%)	97.88%	[12]
	3189 (1.05%)	1939 (0.32%)	6.5 (0.63%)	93.31%	this work
784-255-255-10	39015 (12.85%)	34217 (5.64%)	395.5 (38.4%)	98.24%	[12]
	6999 (2.31%)	3741 (0.62%)	11.0 (1.07%)	95.89%	this work
784-1023-1023-10	24784 (8.16%)	14603 (0.81%)	56.5 (5.49%)	97.97%	this work

LUT: look-up table; FF: flip-flop;
BRAMs: block random-access memories (18 Kbits each)

4-layer MLP structure with the same hidden layer size in the stochastic SNN in [12] (the unit of BRAMs is 18 Kbits each). That merely takes 1.07% of the total amount of BRAMs on the FPGA and 2.78% BRAM utilization for the same structure in the stochastic SNN. From the comparison between the two networks with the smallest and largest hidden layers, as shown in Table I, the storage saving is around 30 times. These results indicate that the WB-SNN is much more hardware efficient than its previous counterpart. Generally, it uses less than 20% of look-up tables (LUTs) and Flip-Flops (FFs) for hidden layers of the same size compared to [12]. The saving in memory, i.e., the BRAM, is more significant as the WB-SNN uses only 1 bit to store a weight while the design in [12] requires 32-bit floating-point numbers. Therefore, the memory in the WB-SNN is reduced by about 30 times, which is consistent with what we expect from the reduction in the binary representation of the weights.

On the other hand, the inference accuracy of the WB-SNN is lower than its full-precision counterpart if the same NN structure is considered for comparison. Although there is an accuracy loss for the WB-SNNs, as shown in Table I, the numbers of LUTs and FFs are reduced to only 17.94% and 10.93%, respectively, of those used by the previous design with 255 neurons in the hidden layers. The reduction in memory or BRAM usage is even more significant. Furthermore, this accuracy drop tends to diminish as the number of neurons in the hidden layer increases. The accuracy loss declines from 6.76% for the network with 63-node hidden layers to 2.35% for the 255-node hidden layer model. Moreover, a larger number of neurons can be used in the WB-SNN to achieve an even higher accuracy. As shown in the last row of Table I, the WB-SNN with 1023-node hidden layers achieves a comparable accuracy as the 127-node full-precision model, but it requires significantly smaller hardware, especially for the BRAM. Therefore, there is a trade-off between the hardware utilization and inference accuracy. As the number of neurons in the hidden layer increases, the hardware becomes only moderately larger, however it leads to a large increase in recognition accuracy. For the 784-1023-1023-10 WB-SNN, this accuracy is 97.97% while the used BRAMs are 56.5 $\times$ 18 Kbits, much smaller than those required in the stochastic SNNs.

As an effort to assess the overall performance of the WB-SNN, we also compared our work with a large set of prior designs. Considering that the WB-SNN inherits features from both SNNs and BNNs, we selected some representative designs from these two categories for comparison. As can be seen from the results in Table II, the WB-SNN achieves a high accuracy at a relatively low hardware cost. For example, the WB-SNN uses only 53% of the LUTs, 48% of the FFs and 38% of the BRAMs required in the SNN model in [14] and 76% of the LUTs, 22% of the FFs and 47% of the BRAMs in the BNN model in [19] to achieve the same accuracy. Some other models return slightly higher accuracy, but their hardware costs are much larger than the WB-SNN. The model in [16] uses less resources but its accuracy is quite low and the implementation is not as efficient as the 255-node WB-SNN model (in Table I). Therefore, the WB-SNN design achieves both hardware efficiency and high accuracy at the same time. Although the FPGA implementation does not produce an accurate energy profile, the power dissipation is low considering that only one bit is used to encode information and that the spikes are sparse in the WB-SNN.

TABLE II
COMPARISON OF THE WB-SNN WITH OTHER SNN AND BNN DESIGNS

Design Source	Framework	Platform	Structure	Bit Width	LUTs	FFs	BRAMs	Accuracy
[14]	SNN	Kintex-7 XC7K325T	784-1024-1024-10	16-bit	46,371	30,417	150	97.7%
[16]	SNN	Spartan-6 XC6SLX45	784-500-500-10	Hybrid (5, 8 or 32-bit)	11,489	4,705	110	93.8%
[17]	SNN	Virtex-7 xc7vx485t	784(Conv.)-512-384-10	8-bit	44,000	–	217.5	98.1%
[18]	BNN	Kintex-7 XC7K325	784-1024-1024-1024-10	1-bit	88,000	115,000	124	98.4%
[19]	BNN	Spartan XC7S50	784-256-256-256-10	1-bit	32,600	65,200	120	97.87%
[20]	BNN	Zynq UltraScale+ ZU19EG	784-256-256-10	1-bit	91,131	–	4.5	95.83%
[20]	BNN	Zynq UltraScale+ ZU19EG	784-1024-1024-10	1-bit	82,988	–	396	98.4%
this work	WB-SNN	Virtex-7 xc7vx485t	784-1023-1023-10	1-bit	24,784	14,603	56.6	97.97%

V. CONCLUSION

A hardware-efficient WB-SNN design is proposed in this paper by introducing binarized weights into SNNs. This binarization significantly reduces the memory requirement for storing the weights in SNNs. A priority encoder (PE) is utilized as the basic unit to construct different layers in neural networks. Since PEs are shared among all neurons in one layer and its output is used to find the weight stored in the memory, the WB-SNN achieves a high hardware efficiency. An MLP model is instantiated and evaluated on the MNIST dataset to verify the proposed design. Hardware implementations on an FPGA show that the WB-SNN attains comparable accuracy while utilizing far less resources than other designs in the literature. The classification accuracy is high in a larger MLP structure with only slightly increased hardware.

In summary, the WB-SNN design significantly reduces memory utilization and achieves a high hardware efficiency while keeping a high inference accuracy. Future work will investigate more complex models such as the convolutional neural network for applications using a larger dataset such as the CIFAR-10.

REFERENCES

[1] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: a survey." *ACM Journal on Emerging Technologies in Computing Systems*, vol. 15, no. 2, pp.1-35, 2019.

[2] A. Kumar, S. Rotter, and A. Aertsen, "Spiking activity propagation in neuronal networks: reconciling different perspectives on neural coding," *Nature reviews. Neuroscience,* vol. 11. pp. 615-27. 2010.

[3] R. V. Rullen and S. J. Thorpe, "Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex," *Neural Computation,* vol. 13, no. 6, pp. 1255-1283, June 2001

[4] H. Tang, H. Kim, H. Kim and J. Park, "Spike counts based low complexity SNN architecture with binary synapse," *IEEE Trans. Biomed. Circuits Syst.* , vol. 13, no. 6, pp. 1664-1677, Dec. 2019.

[5] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Front. Comput. Neurosci.,* vol. 9, p. 99, Aug. 2015.

[6] F. Akopyan et al., "TrueNorth: design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.,* vol. 34, no. 10, pp. 1537-1557, Oct. 2015.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016, [online] Available: https://arxiv.org/abs/1602.02830.

[8] P. -Y. Chuang, P. -Y. Tan, C. -W. Wu and J. -M. Lu, "A 90nm 103.14 TOPS/W Binary-Weight Spiking Neural Network CMOS ASIC for Real-Time Object Classification," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1-6.

[9] X. Nguyen, H. Nguyen and C. Pham, "A Scalable High-Performance Priority Encoder Using 1D-Array to 2D-Array Conversion," *IEEE Trans. Circuits Syst. II, Exp. Briefs,* vol. 64, no. 9, pp. 1102–1106, Sep. 2017.

[10] LeCun, Yann and Cortes, Corinna, "MNIST handwritten digit database," 2010

[11] W. Tang, G. Hua, L. Wang, "How to Train a Compact Binary Neural Network with High Accuracy?" in *Proc. the Thirty-First AAAI Conf. on Artificial Intelligence*, San Francisco, CA, USA, Feb. 2017, pp. 2625-2631.

[12] C. Tang and J. Han, "Design and Implementation of a Highly Accurate Stochastic Spiking Neural Network," in *Proc. 2021 IEEE Workshop on Signal Processing Systems (SiPS)*, Coimbra, Portugal, 2021, pp. 1-6.

[13] S. Liu, W. J. Gross and J. Han, "Introduction to Dynamic Stochastic Computing," *IEEE Circuits Syst. Mag.,* vol. 20, no. 3, pp. 19-33, 2020.

[14] Y. Liu, Y. Chen, W. Ye and Y. Gui, "FPGA-NHAP: A general FPGA-based neuromorphic hardware acceleration platform with high speed and low power," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 6, pp. 2553-2566, June 2022.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.,* vol. 15, no. 1, pp. 1929-1958, 2014.

[16] D. Ma et al., "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *J. Syst. Archit.*, vol. 77, pp. 43-51, 2017.

[17] M. T. L. Aung, C. Qu, L. Yang, T. Luo, R. S. M. Goh and W. -F. Wong, "DeepFire: acceleration of convolutional spiking neural network on modern field programmable gate arrays," in *Proc. 2021 31st Int. Conf. on Field-Programmable Logic & Appl. (FPL)*, 2021, pp. 28-32.

[18] P. Jokic, S. Emery and L. Benini, "BinaryEye: A 20 kfps streaming camera system on FPGA with real-time on-device image recognition using binary neural networks," in *2018 IEEE 13th Int. Symp. on Ind. Embedded Syst. (SIES)*, 2018, pp. 1-7.

[19] M. Ghasemzadeh, M. Samragh and F. Koushanfar, "ReBNet: residual binarized neural network," in *Proc. Annu. IEEE Symp. Field-Program. Cust. Comput. Mach.*, 2018, pp. 57-64.

[20] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, K. Vissers, "Finn: a framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. on Field- Programm. Gate Arrays*, 2017, pp. 1-10.