TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 04/10 pp479–486
DOI: 10.26599/TST.2019.9010019
Volume 25, Number 4, August 2020

Hardware Implementation of Spiking Neural Networks on FPGA

Jianhui Han, Zhaolin Li*, Weimin Zheng, and Youhui Zhang*

Abstract: Inspired by real biological neural models, Spiking Neural Networks (SNNs) process information with discrete spikes and show great potential for building low-power neural network systems. This paper proposes a hardware implementation of SNN based on Field-Programmable Gate Arrays (FPGA). It features a hybrid updating algorithm, which combines the advantages of existing algorithms to simplify hardware design and improve performance. The proposed design supports up to 16 384 neurons and 16.8 million synapses but requires minimal hardware resources and archieves a very low power consumption of 0.477 W. A test platform is built based on the proposed design using a Xilinx FPGA evaluation board, upon which we deploy a classification task on the MNIST dataset. The evaluation results show an accuracy of 97.06% and a frame rate of 161 frames per second.

Key words: Spiking Neural Network (SNN); Field-Programmable Gate Arrays (FPGA); digital circuit; low-power; MNIST

1 Introduction

Over recent years, Neural Networks (NNs) have been successfully deployed in a wide range of applications. Compared to conventional Artificial Neural Networks (ANNs) which use analog values to represent activations inside the networks, Spiking Neural Networks (SNNs) imitate real biological neurons and encode the activation with the timing information of neuron spikes. As opposed to ANNs, where the major operations are the matrix multiplication of weights and activation of network layers, the spiking nature of SNNs avoids complex matrix multiplications and thus they require lower computation resources and are more energy efficient.

SNNs are built with neurons, which are connected with weighted synapses to form layers and networks. As the basic building block of SNNs, each spiking neuron maintains a value to represent its current state. Inputs to these neurons are spikes transmitted through synapses. The neurons collect these input spikes and integrate the corresponding weights to update their internal states. Whenever the state reaches a certain threshold value, the neuron is activated and outputs a spike to its successor neurons. After a pre-configured propagation delay, the successors receive the spike and update their states accordingly. Besides these common operating principles, different neuron models feature different neuron behaviors. One of the most commonly used neuron models is the Leaky Integrate-and-Fire (LIF) model. LIF neurons are inspired by the membrane voltage leaking in biological neurons, with their states decreasing over time if no input spikes present.

However, current computer architectures are not ideally suited to executing SNNs. The massive parallelism inherent to an SNN, in which a large number of neurons work in a similar but simple manner, requires a more parallel architecture than that provided by current Central Processing Units (CPUs). Although Graphics Processing Units (GPUs) can capitalize on

• Jianhui Han is with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China. E-mail: hanjh16@mails.tsinghua.edu.cn.
• Zhaolin Li is with the Research Institute of Information Technology, Tsinghua University, Beijing 100084, China. E-mail: lzl73@mail.tsinghua.edu.cn.
• Weimin Zheng and Youhui Zhang are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: zwm-dcs@mail.tsinghua.edu.cn; zyh02@tsinghua.edu.cn.
∗ To whom correspondence should be addressed.
Manuscript received: 2019-03-26; revised: 2019-04-25; accepted: 2019-05-05

the parallelism of SNN, the kernel-launch programming paradigm of current GPUs makes them ill-suited to event-driven computation, which is an important class of updating algorithms for SNNs (as described below in Section 4.1). However, Field-Programmable Gate Arrays (FPGA) can address this issue since they provide parallel processing capability and are flexibly reconfigurable to cater to different computing models. Furthermore, FPGA are more energy efficient than current CPUs and GPUs.

This paper therefore proposes an FPGA-based SNN module implementation. By utilizing an enhanced hybrid updating algorithm, the proposed module supports up to 16 384 neurons and 16.8 million synapses with minimal on-chip resources and 0.477 W power consumption. We build a test platform to evaluate the proposed accelerator, on which we map an SNN model for the classification of handwritten digits in the MNIST[1] dataset. The evaluation results show that the classification accuracy is 97.06% and the performance for processing neuron firing events is 6.72×10^5 events per second, resulting in 161 frames per second for the MNIST dataset.

2 Background

A wide variety of neuron models have been used in hardware implementations of SNNs, including the Izhikevich model[2] and variations of the Integrate-and-Fire (IF) model. Our approach adopts the LIF neuron model, which simulates the membrane voltage leaking of biological neurons with an exponential process:

$$V(t_2) = V(t_1) \cdot \mathrm{e}^{-(t_2-t_1)/\tau} \tag{1}$$

where $V(t)$ are the neuron states at time t (the membrane voltage in biological neurons), and τ is the leaky constant. For the time-stepped updating algorithm, the values of t are discrete and the term $t_2 - t_1$ will be a constant Δt. For the event-driven algorithm, t_1 is the timestamp of the previous event that updated this neuron, and t_2 is the timestamp of the current event.

The neuron state is also changed by input current from other neurons. A weight $W_{i,j}$ is assigned to the synapse from neuron i to neuron j. When neuron i generates an output spike, $W_{i,j}$ is added to the state of neuron j after a fixed delay. Therefore, for the time-stepped updating algorithm, the updating equation of neuron j from time step t_n to t_{n+1} is

$$V(t_{n+1}) = V(t_n) \cdot \mathrm{e}^{-\Delta t/\tau} + \sum_{i=0}^{m-1} s_i(t_n) \cdot W_{i,j} \tag{2}$$

where m is the fan-in of neuron j, and $s_i(t_n)$ is the spiking status of neuron i at time step t_n. The spiking status s_i is 1 if neuron i outputs a spike, otherwise it is 0. For the event-driven updating algorithm, an update is triggered by the activation event of neuron i. For each neuron j, where neuron j is a successor of neuron i, the update is carried out by

$$V(t') = V(t) \cdot \mathrm{e}^{-(t'-t)/\tau} + W_{i,j} \tag{3}$$

where the parameters have the same meaning as in Eq. (1).

After updating the neuron states, a threshold value V_{th} is used to decide whether a neuron is activated and outputs a spike. After a neuron activates, its state is reset to V_{reset}.

3 Related Work

3.1 Hardware implementation

Significant effort has been invested into the hardware implementation of SNNs and some designs have been presented in the form of digital[3–7], analog[8, 9], and mixed analog/digital circuits[10, 11]. For digital implementations, both FPGA-based systems and Application-Specific Integrated Circuit (ASIC) systems have been widely studied.

The TrueNorth chip[3] is one of the most well-known ASIC designs. A core in the TrueNorth system contains a 256×256 crossbar that implements the function of synapses and is configured to map incoming spikes to neurons. By integrating 4096 such processing cores, the TrueNorth chip carries 1 million neurons and 256 million synapses. The scale can be further extended by connecting multiple chips together. SpiNNaker[4] is another fully custom digital system and is composed of many small ARM processors. It features a custom interconnect communication scheme that is designed to be suitable for a large number of small spike-like messages and thus optimized for the communication behavior of a spike-based network architecture. Like TrueNorth, SpiNNaker supports the cascading of multiple chips to form large-scale systems.

Previous works[5–7] have also proposed several FPGA-based SNN accelerator designs. BlueHive[5] supports up to 65 536 neurons and 67.1 million synapses with a multi-FPGA architecture. However, it implements the neurons with the complex Izhikevich model[2] and a low firing rate assumption, which targets biological neural network simulations and does not support real-world applications effectively. Another representative design is Minituar[6], which treats the activation of neurons as events and utilizes an event-driven algorithm to update them. It implements up to 65 536 LIF neurons and 16.8 million synapses. Minituar faithfully models the exponential leaky process of neurons and employs on-chip Digital Signal Processors (DSPs) to carry out the fixed-point computation. It also maintains a hardware event queue that requires a sorting operation for each incoming event to support spikes with delays; this increases design complexity and run-time latency.

3.2 Network model

In recent years, Deep Belief Networks (DBNs)[12] have been proven to be effective in a variety of domains, such as machine vision[13] and machine audition[14]. DBN is a multilayered probabilistic generative model that uses a stacked structure of multiple Restricted Boltzmann Machines (RBMs). Previous work[15] has proposed methods to convert DBNs to LIF-based spiking DBNs and explored the processing of spiking DBNs with the event-driven algorithm.

Another study[16] tried to solve the loss of accuracy arising in the conversion from Fully-Connected Networks (FCNs) and Convolutional Neural Networks (CNNs) to SNNs. The proposed optimization techniques include using Rectified Linear Units (ReLUs) with zero bias during training to suit spiking encoding, a weight normalization method to help regulate firing rates, and a threshold balancing scheme to enable low-latency processing.

In this paper, we use the techniques proposed in Ref. [16] to train our SNN model and explore the efficient hardware implementation of such SNN models.

4 System Design

4.1 Hybrid updating algorithm

We use an updating algorithm that is a hybrid of the conventional time-stepped updating algorithm and the event-driven updating algorithm. In this subsection, we first briefly describe these two existing algorithms and then present our hybrid.

The time-stepped algorithm processes all of the neurons based on discrete time steps. Within each time step, the state of each neuron is updated and checked to decide whether it outputs a spike. Information about these spikes is stored for use in future time steps according to their transmission delay. This algorithm can waste computing resources since it schedules unnecessary operations for neurons that do not receive any input spikes. The event-driven algorithm, on the other hand, processes only the activation events of neurons. An event queue is used as storage for the events, and is sorted by the event timestamps. After each event dequeues from the event queue, only the states of successive neurons are updated, thereby generating new events. In this way, unnecessary operations are avoided. Although the event-driven algorithm can be efficient, the hardware implementation of the event queue is complicated since it requires sorting the events whenever a new event enqueues.

Therefore, we combine the time-stepped and event-driven algorithm, as described in Algorithm 1. We use multiple event queues, each of which is tagged with timestamp Q_n (where n ranges from 0 to $D-1$ where D is the maximum delay allowed), to store the events to be processed after n time steps from the current time. In this way, events with the same timestamp can be stored in the same queue with no sorting operation required. To manage these event queues, time steps are maintained globally. At each time step, the event queue with tag Q_0 is set to be the active queue and its events are processed. Once an event queue is empty, the current time step finishes. Before the next time step, the tags of all event queues decrease by one, such that Q_1–Q_{D-1} becomes Q_0–Q_{D-2} and Q_0 is reused in a circular manner as Q_{D-1}. Sorting operations are avoided in this hybrid updating algorithm, which reduces the system's run-time latency.

4.2 System architecture

The architecture of the proposed module is shown in

Algorithm 1 Hybrid updating algorithm

```
Input: Event queues $Q_0, Q_1, ..., Q_{D-1}$
1:  for $t \Leftarrow 0 : \Delta t : T$ do
2:      while not $Q_0$.is_empty() do
3:          $event \Leftarrow Q_0$.dequeue()
4:          for $neuron$ in $event$.successors() do
5:              $neuron$.update_state()
6:              $neuron$.check_activation()
7:              if $neuron$.is_activated() then
8:                  $new\_event \Leftarrow neuron$.form_new_event()
9:                  $delay \Leftarrow new\_event$.get_delay()
10:                 $Q_{delay}$.enqueue($new\_event$)
11:             end if
12:         end for
13:     end while
14:     for $i \Leftarrow 1 : D-1$ do
15:         $Q_{i-1} \Leftarrow Q_i$
16:     end for
17:     $Q_{D-1} \Leftarrow Q_0$
18: end for
```

Fig. 1. There are four main memory components, each of which has its own controller to manage its reading and writing operations. The event queues submodule is the hardware implementation of the multiple event queues described above in Section 4.1. The event controller submodule is in charge of managing these event queues by enqueuing generated events and dequeuing events for processing. The weight memory and state memory submodules are used to store weight and state data, respectively. The weight memory submodule is read-only, while the state controller also controls the writing back of updated states from the state updater. Another memory submodule is the delay memory, which is read-only and stores the delay values of different events. Details about the implementation of the memory components are discussed below in Section 4.3.

The state updater carries out the main body of computation. It first decays the neuron states and then sums the incoming weights to update the neuron states. Checks for neuron activation are then carried out to decide whether a new event is generated. If any neuron is activated, its neuron state is reset to a predetermined constant. The state updater can exploit the parallelism of SNN by updating multiple neuron states at the same time. The layered structure of SNN ensures that the successors of a neuron are independent of each other, which makes simultaneous updating possible.

The execution flow is as follows. The event controller receives controlling signals and values of the current time step from the system controller (which is omitted from Fig. 1 for clarity). It then sets the current event queue to be active and sequentially reads events from it. The event data is sent to the weight controller and the state controller. They access the weight and state memory with the event data and then send them to the

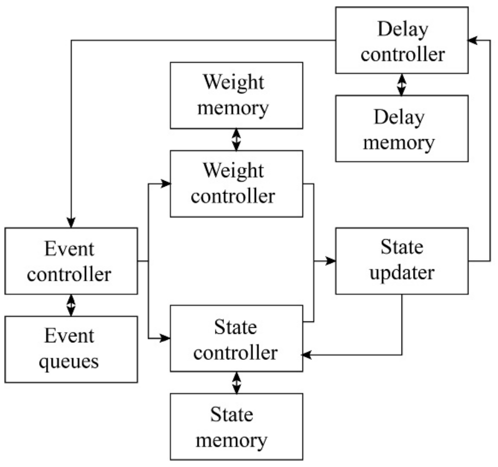


Fig. 1 Architecture of the proposed system.

state updater. If there are activation events after the state update, these events go through the delay controller to look up their delays. The event controller calculates the corresponding destination event queues according to the delay, and writes the events to them.

The system works in an asynchronous manner to improve throughput. For example, after the event controller sends event data to the weight and state controllers, it begins to read the event queue immediately. When it collects the data request signal from the weight and state controllers, event data are sent again. Communication between other submodules is similar, through requests and responses. Another example is the *source controller* (also omitted from Fig. 1 for clarity) for the state updater. It contains two First-In-First-Outs (FIFOs) to hold the operands of the state updater from the weight controller and the state controller. In this way, although the latter two controllers have different memory access times, waiting between them is avoided, provided that the FIFOs still have space for incoming data.

4.3 Implementation

We use signed 16-bit fixed-point numbers to represent the weights and neuron states. The maximum number of neurons is set to 16 384, which results in a 14-bit index for each neuron. The maximum number of neurons in one layer is 1024, with fully-connected synapses supported. This means that the maximum number of synapses is 16.8 million. The maximum delay is set to 16, which is adequate according to previous research[6].

The proposed module needs to store up to 32 MB of weight data. Since the amount of on-chip Block RAM (BRAM) of FPGA is often limited, it is impractical to store all of the weights on-chip. Therefore, we use external Double Data Rate (DDR) memory to store all of the weights, while implementing all of the other memory modules with BRAM. Considering that for each event the corresponding weights are always of the same group, we store these weights in consecutive spaces in the external memory. With this mapping, the burst read feature of the DDR memory can be exploited to optimize the latency of memory access.

For the event queues, we implement 16 FIFOs with BRAM that can be accessed separately with a 4-bit address. To implement the hybrid updating algorithm, the event controller always reads the FIFO with the lower four bits of the current time register. After a new event is generated by the state updater and the delay is obtained from the delay controller, the delay value is added to the current time and the lower four bits of the result are used to select the correct FIFO to which to write the data.

The operations of the weight updater are mainly the addition of weights and neuron states and the comparison of updated states with the threshold parameter. The decay of neurons is also carried out by the weight updater. For simplicity, the exponential computation is implemented with the subtraction of a constant. To fully utilize the memory access bandwidth, 32 adders and 32 comparators are instantiated in the weight updater. Further analysis in Section 5.4 below shows that neither the throughput of the weight updater nor its consumption of hardware resources is limiting factors of the proposed module.

5 Evaluation

5.1 Experimental setup

Benchmark. We apply a feed-forward SNN with two hidden layers to a classification task on the MNIST handwritten digit dataset[1] to evaluate the proposed module. The topology of the network is shown in Fig. 2. The input layer includes 784 neurons to process the input spikes converted from the 28×28 pixel digit figure. The two hidden layers both have 1024 neurons, while the output layer has 10 neurons corresponding to the classification results of digits 0–9. The layers are fully connected, which means the connections are all-to-all between two adjacent layers.

This paper focuses on the hardware implementation of SNN, and the benchmark is used for system performance measurement. Although the model has a limited depth (i.e., number of layers), it takes up the fan-in/fan-out of each layer as much as possible. The proposed module updates neuron states based on events and the updating operations and memory accesses of each event are dependent on the fan-in/fan-out. Therefore, as analyzed in Section 5.4 below, the system performance measured with this benchmark is close to the theoretical upper limit. The functional correctness of the proposed module is also testified by the classification accuracy reporeted in Section 5.3 below.

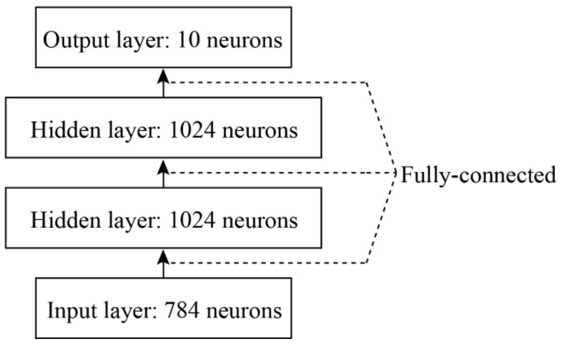


Fig. 2 Topology of the benchmark SNN model.

Test platform. We use a Xilinx ZC706 evaluation board[17] with a XC7Z045 SoC to build the test platform. The evaluation board provides 1 GB DDR3 memory and two ARM Cortex-A9 MPCores. The structure of the test platform is illustrated in Fig. 3. We implement the proposed module in the programmable logic on the evaluation board. The module runs at 200 MHz. One of the ARM processors is employed to configure the internal registers of the proposed module and manage the data movement. The weight data are stored in the DDR3 memory as described above. These three parts are connected through an on-chip Advanced eXtensible Interface (AXI) bus, as shown in Fig. 3.

5.2 Hardware utilization

The hardware utilization results are obtained based on the synthesis results of the proposed accelerator, as listed in Table 1. According to the results reported in Table 1, the most intensive on-chip resource is BRAM, with which the neuron states and activation event queues are implemented. This implies that managing BRAM is an important design consideration for FPGA-based SNN

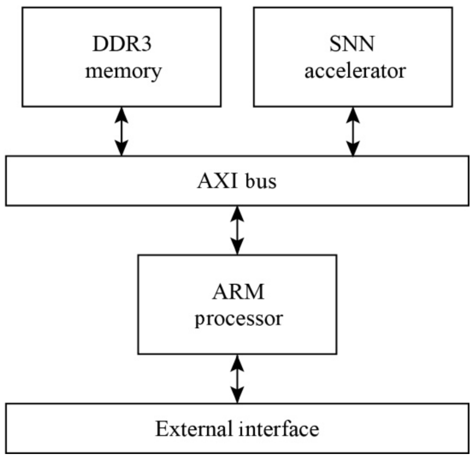


Fig. 3 Structure of the test platform.

Table 1 Hardware utilization for ZC706 board.

Component	Cells used	Utilization (%)
LUT	5381	2.46
FF	7309	1.67
BRAM	40.5	7.43
BUFG	1	3.13

acceleration systems. Potential optimization techniques, such as compressing the presentation of events or reducing the bit-width of states and/or events, could be beneficial.

The total power consumption is 0.477 W, with the detailed breakdown shown in Fig. 4. Static power is 0.246 W, which is nearly 52% of the total power consumption. BRAM accounts for 19%, and is thus also a major component of system power use.

5.3 Accuracy

The classification accuracy is tested on the MNIST test dataset, which consists of 10 000 frames of figures. The model is trained with the weight and threshold balancing scheme[16] with MATLAB, using 32-bit floating-point numbers. The training results in an accuracy of 98.48%. The trained weights are turned into 16-bit fixed-point numbers and deployed on the proposed module. The accuracy of the on-board test is 97.06%. There is therefore an accuracy loss of 1.42%, which is mainly caused by the conversion from floating-point to 16-bit fixed-point numbers.

5.4 System performance

We evaluate the performance of the proposed module with the overall throughput on the benchmark dataset (in frames per second) and the number of activation events processed per second. To measure the processing time, we insert a hardware counter into the test platform. It begins counting when the ARM processor finishes the initial configuration of the DDR3 memory and the proposed module, and stops counting when the final frame obtains its classification result. The processing time for the 10 000 frames is 62.1 s, which supplies a frame rate of 161 frames per second.

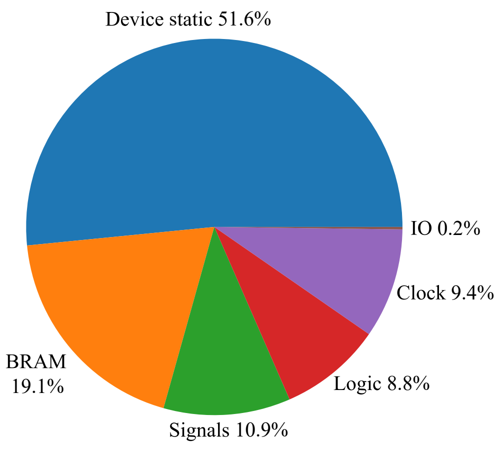


Fig. 4 Power consumption breakdown.

For the number of activation events, we use MATLAB to simulate the fixed-point computation of the proposed module. With this simulator, the number of activation events is 4.2×10^7. With the processing time acquired as mentioned above, the performance in processing activation events is 6.72×10^5 events per second.

For the proposed module to process each activation event of a neuron, the states of all its successive neurons need to be updated; this requires fetching 1024 weight values from DDR3 memory.

Given that the proposed module runs at 200 MHz and the width of the data bus is 64 bits, the bandwidth can be up to 1600 MB/s (the weights are read-only during the classification and only one direction is considered). To process each neuron activation event, the weights for all its successive neurons are required. Since we use 16 bits for each weight and the maximum fan-out of each neuron is 1024, the amount should be 2 KB. Combining these two constraints, the maximum performance of the system is 8×10^5 events per second.

Comparing the practical performance with this theoretical value, the bandwidth has been well exploited in the proposed design. These system performance results further identify that external memory bandwidth is the bottleneck of FPGA-based SNN implementations. Techniques like reducing the bit-width of weights and network pruning can alleviate this issue; in particular, network pruning has been demonstrated to be effective in ANNs[18].

5.5 Comparison with GPU

For comparison, the same SNN model is implemented with the PyTorch[18] framework to run on an NVIDIA Tesla P100 GPU[19]. Note that GPUs are not suitable for event-driven updating, so we implement the time-stepped updating algorithm instead. Again, we measure the execution time of the 10 000 frames of the MNIST test dataset. The runtime power during GPU execution is measured with the NVIDIA System Management Interface[20].

The processing time on a GPU is 7.96 s and the average power consumption is 29.6 W. Therefore, although the proposed module demands more execution time, it consumes much less power than a GPU. This equates to much greater power efficiency: 337.6 frames per second per watt compared to 42.2 frames per second per watt on a GPU. These results imply that the proposed design is suitable for application scenarios that have strict constraints on power consumption and a tolerance for execution latency.

6 Discussion

As mentioned above in Section 5, there are further optimization opportunities for the proposed SNN implementation. Since external memory bandwidth is crucial for the system's performance, we briefly discuss the impact of lowering the weight bit-widths and pruning network connections in this section. We apply these two techniques to the MNIST model evaluated in Section 5 and re-evaluate the classification accuracy with our software simulator.

Figure 5 shows the classification accuracy with lower bit-widths. The results shown in Fig. 5 demonstrate that the influence of bit-width on accuracy is negligible even with the weight bit-width lowered to 6. This gives a 62.5% reduction in the total size of external data requests. Although the ability to operate with such a low bit-width partially stems from the simplicity of the MNIST dataset, reducing bit-width is a promising method of improving system performance. Therefore, for the implementation and benchmarking of other systems, the choice of bit-width should be carefully considered.

Similarly, the impact of network pruning is illustrated in Fig. 6. Weights with small values are set to 0, with a sparsity parameter used for the threshold. With sparsity at 50%, half of the weights can be pruned with minimal loss of classification accuracy. By weight pruning, the weights that are needed to process an event are reduced, which benefits performance. We use a naive pruning that simply masks out small weights; with further optimization such as fine-tuning[21], the loss of accuracy can be reduced further.

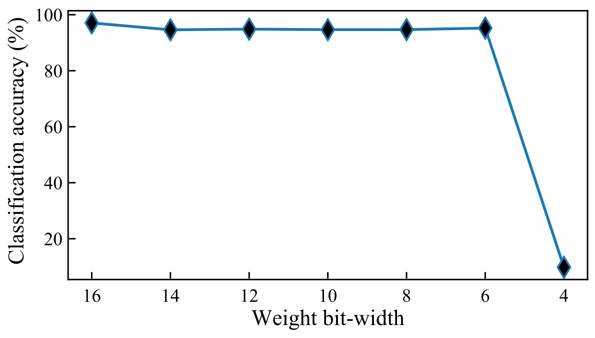


Fig. 5 Classification accuracy vs. bit-widths on MNIST.

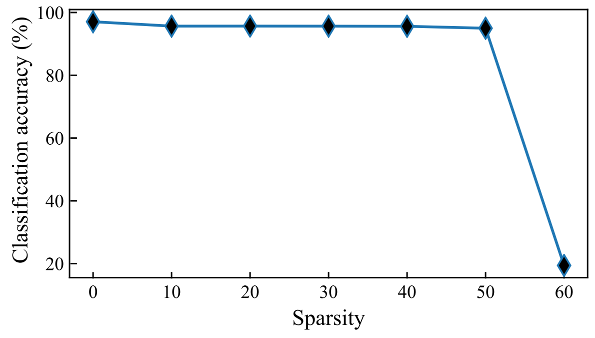


Fig. 6 Classification accuracy vs. sparsity on MNIST.

Furthermore, these two optimization techniques are orthogonal to each other and can be applied simultaneously. This requires further exploration of the design space and is left to a future study.

7 Conclusion

In this paper, an FPGA-based SNN hardware implementation is proposed. The proposed module is designed based on a hybrid of the time-stepped and event-driven updating algorithms. An evaluation of the proposed module is carried out using the MNIST dataset with the results showing a classification accuracy of 97.06%. With 0.477 W power consumption, the performance for processing neuron activation events is 6.72×10^5 events per second, which results in a frame rate of 161 frames per second on MNIST dataset. The proposed module further identifies that memory bandwidth is the bottleneck of the system. To address this issue, two potential optimization techniques are discussed for FPGA-based SNN implementations.

Acknowledgment

This work was supported in part by the Beijing Innovation Center for Future Chip, Tsinghua University, in part by the Science and Technology Innovation Special Zone project, China, and in part by the Tsinghua University Initiative Scientific Research Program (No. 2018Z05JDX005).

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] E. M. Izhikevich, Simple model of spiking neurons, *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, 2003.

[3] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al., A million spiking-neuron integrated circuit with a scalable communication network and interface, *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[4] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor, in *2008 IEEE Int. Joint Conf. on Neural Networks (IEEE World Congress on Computational Intelligence)*, Hong Kong, China, 2008, pp. 2849–2856.

[5] S. W. Moore, P. J. Fox, S. J. T. Marsh, A. T. Markettos, and

A. Mujumdar, Bluehive—A field-programable custom computing machine for extreme-scale real-time neural network simulation, in *2012 IEEE 20th Int. Symp. on Field-Programmable Custom Computing Machines*, Toronto, Canada, 2012, pp. 133–140.

[6] D. Neil and S. C. Liu, Minitaur, an event-driven FPGA-based spiking network accelerator, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 12, pp. 2621–2628, 2014.

[7] K. Cheung, S. R. Schultz, and W. Luk, A large-scale spiking neural network accelerator for FPGA systems, in *Artificial Neural Networks and Machine Learning – ICANN 2012*, A. E. P. Villa, W. Duch, P. Érdi, F. Masulli, and G. Palm, eds. Springer, 2012, pp. 113–120.

[8] E. Farquhar, C. Gordon, and P. Hasler, A field programmable neural array, in *2006 IEEE Int. Symp. on Circuits and Systems*, Island of Kos, Greece, 2006, pp. 4114–4117.

[9] M. Liu, H. Yu, and W. Wang, FPAA based on integration of CMOS and nanojunction devices for neuromorphic applications, in *Int. Conf. on Nano-Networks*, M. Cheng, ed. Springer, 2009, pp. 44–48.

[10] B. V. Benjamin, P. R. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations, *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[11] T. Pfeil, J. Jordan, T. Tetzlaff, A. Grübl, J. Schemmel, M. Diesmann, and K. Meier, Effect of heterogeneity on decorrelation mechanisms in spiking neural networks: A neuromorphic-hardware study, *Phys. Rev. X*, vol. 6, no. 2, p. 021023, 2016.

[12] G. E. Hinton and R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[13] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, Deep, big, simple neural nets for handwritten digit recognition, *Neural Comput.*, vol. 22, no. 12, pp. 3207–3220, 2010.

[14] A. R. Mohamed, G. E. Dahl, and G. Hinton, Acoustic modeling using deep belief networks, *IEEE Trans. Audio Speech Lang. Process.*, vol. 20, no. 1, pp. 14–22, 2012.

[15] P. O'Connor, D. Neil, S. C. Liu, T. Delbruck, and M. Pfeiffer, Real-time classification and sensor fusion with a spiking deep belief network, *Front. Neurosci.*, vol. 7, p. 178, 2013.

[16] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. C. Liu, and M. Pfeiffer, Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing, in *2015 Int. Joint Conf. on Neural Networks (IJCNN)*, Killarney, Ireland, 2015, pp. 1–8.

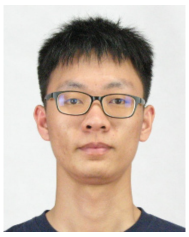
[17] Xilinx Inc., Xilinx Zynq-7000 SoC ZC706 evaluation kit, https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html, 2019.

[18] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. M. Lin, A. Desmaison, L. Antiga, and A. Lerer, Automatic differentiation in PyTorch, in *Proc. 31st Conf. on Neural Information Processing Systems*, Long Beach, CA, USA, 2017, pp. 1–4.

[19] NVIDIA Corporation, NVIDIA Tesla P100: The world's first AI supercomputing data center GPU, https://www.nvidia.com/en-us/data-center/tesla-p100/, 2019.

[20] NVIDIA Corporation, NVIDIA system management interface, https://developer.nvidia.com/nvidia-system-management-interface, 2019.

[21] S. Han, H. Z. Mao, and W. J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv preprint: 1510.00149, 2015.



Jianhui Han received the BS degree from Tsinghua University, Beijing, China, in 2016. He is currently working toward the PhD degree at the Institute of Microelectronics, Tsinghua University, Beijing, China. His main research interests include digital circuit/system design and emerging technology-based machine learning acceleration.

Weimin Zheng received the MS degree from Tsinghua University, Beijing, China. Currently he is an Academician of Chinese Academy of Engineering and a professor at the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include high performance computing, network storage, and parallel compiler.

Zhaolin Li received the BS and PhD degrees from Harbin Institute of Technology, Harbin, China, in 1994 and 2000, respectively. He is currently a professor with the Research Institute of Information Technology, Tsinghua University, Beijing, China. His current research interests include embedded systems, parallel computing, multicore design, and system-on-a-chip.

Youhui Zhang received the BS and PhD degrees from Tsinghua University, Beijing, China, in 1998 and 2002, respectively. He is currently a professor in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include computer architecture and neuromorphic computing. He is a member of CCF, ACM, and IEEE.