Neurocomputing 382 (2020) 106–115

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A low-cost and high-speed hardware implementation of spiking neural network

Guohe Zhang[a], Bing Li[a], Jianxing Wu[a], Ran Wang[a], Yazhu Lan[b], Li Sun[a], Shaochong Lei[a], Hai Li[b], Yiran Chen[b,*]

[a] School of Microelectronics, Xi'an Jiaotong University, Xi'an, China
[b] Duke University, Durham, NC, USA

ARTICLE INFO

Article history:
Received 8 August 2019
Revised 14 November 2019
Accepted 22 November 2019
Available online 4 December 2019

Communicated by Dr. Wen Wujie

Keywords:
Spiking neural network
Neurons
Hardware implementation
Speed-up
Leaky-Integrate–Fire
Tempotron supervised learning rules

ABSTRACT

Spiking neural network (SNN) is a neuromorphic system based on the information process and store procedure of biological neurons. In this paper, a low-cost and high-speed implementation for a spiking neural network based on FPGA is proposed. The LIF (Leaky-Integrate–Fire) neuron model and tempotron supervised learning rules are used to construct the SNN which can be applied to the classification of pictures. A combined circuit instead of lookup table implementation method is proposed to realize the complex computing of kernel function in LIF neuron model. In addition, this work replaces the multiplication operations in the weights training with the arithmetic shift, which can speed up the training efficiency and reduce the consumption of computing resources. Experimental results based on Vertix-7 FPGA shows that the classification accuracy is approximately 96% and the average time for classifying a sample is 0.576*us* at the maximum frequency 178 MHz which achieves approximately 908,578 speedup compared with the software implementation on Matlab.

1. Introduction

In the past few decades, spiking neural networks have gained increasing attention due to their application in simulating complex neural circuits and exploring various neural phenomena, such as neural plasticity, visual systems, auditory systems, neural oscillations, and many other important neurological functions [1]. Currently, typical artificial intelligence applications use the learning functions of SNNs for classifier and autonomous control. SNNs represent a powerful tool in neuroscience that allows the simulation of living neuronal function and attempts to gather clues about the characteristics of living tissue from the fitted neural network features.

SNNs are known as the third generation of artificial intelligence neural networks, simulating the way biological neurons store and process information in parallel, overcoming the shortcomings of traditional neural networks based on the von Neumann architecture system [2], whose efficiency is very impressive in the task of complex operations or processing huge information. Pulsed neurons as a basic unit of SNNs processing information are highly biomimetic. This spiking-driven computer system is close to the processing of biological neurons, and the study of spiking neural networks helps to explore the working mode of the biological brains. Since the spiking neurons take into account the time information of the input signal, part of the information lost in the binary coding process can be compensated to a certain extent by the information of the time dimension. Therefore, the SNNs have better recognition ability and robustness in theory.

Although SNNs have proven to be the closest network to the biological nervous system, the learning mechanism of the biological cranial nervous system is still unclear, so the study of learning methods is not mature [3]. Spiking neurons transmit signals in the form of discrete spiking. The non-differentiable nature of the response function makes it difficult for SNNs to directly apply gradient-backward based backpropagation algorithms for effective training [4]. Existing learning algorithms are generally inefficient and not fully compatible with the biomimetic features of SNNs. Therefore, the search for a new algorithm to play the powerful computing power of spiking neurons has become one of the hotspots in the field of artificial intelligence. Since the neurodynamics model of SNNs uses time-coded spiking information as input, powerful parallel computing capability is required. The processor of traditional computing architecture has very low computational efficiency when running SNNs. In addition, the size

* Corresponding author.
E-mail addresses: zhangguohe@xjtu.edu.cn (G. Zhang), libing888@stu.xjtu.edu.cn (B. Li), wjx18829040915@stu.xjtu.edu.cn (J. Wu), ranwang234@163.com (R. Wang), yazhu.lan@duke.edu (Y. Lan), sl_lxa@mail.nwpu.edu.cn (L. Sun), leisc@xjtu.edu.cn (S. Lei), hai.li@duke.edu (H. Li), yiran.chen@duke.edu (Y. Chen).

https://doi.org/10.1016/j.neucom.2019.11.045
0925-2312/

of SNNs is generally very large, and may contain more than 10^5 neurons. Each neuron has a synaptic connection with as many as 10^4 other neurons. The large network scale cannot be handled by traditional computing architectures. Therefore, high speed low cost hardware implementation of spiking neural network computing is essential for application in practical.

SNNs are more realistic than traditional neural networks simulated by neuroscience, considering not only neuronal and synaptic states, but also the concept of time into their operational models. The discharge activity of spiking neurons is determined by the evolution of their membrane potential, which follows a certain model equation. Not only do different models produce different fire behaviors, but in the same model, parameters can often be adjusted so that artificial neurons can reproduce different fire patterns of specific cells [5]. Methods for simulating SNNs on hardware are divided into analog circuits and digital circuits designs. The analog circuits approach provides a compact solution that enables small areas and low power consumption, allowing for large parallel neuron architectures and accelerated information processing. However, analog solutions are less flexible, less accurate, and lack of effective analog circuitry for storage components. The implementation of SNNs using digital circuits can solve the shortcomings of the implementation method using analog circuits. It is suitable for exploring neural network topology by reconfigurable digital logic platforms to save resources and power consumption.

A novel and hierarchical network-on-chip architecture for SNN hardware was presented by Carrillo et al. [6]. Aiming to address the scalability issue, a modular array of clusters of neurons using a hierarchical structure of low and high-level routers were introduced into the SNN [6]. The hardware implementation on 65 nm CMOS technology demonstrates high-throughput, low-cost area, and power consumption of the hierarchical network-on-chip architecture. A new hardware architecture of self-repairing spiking neural network was proposed to mimic the self-repairing capability in the human brain [7]. It was demonstrated that the hardware circuits can self-detect and self-repair synaptic faults by maintaining the system performance with fault densities of up to 40%. Yang et al. presented a real-time digital neuromorphic system for the simulation of large-scale conductance based spiking neural networks, which has the advantages of high biological realism and large network scale [8]. The cost-efficient conductance-based neuron models and the novel router architecture proposed in their paper were quite suitable for the implementation of the large-scale and real-time computational spiking neural network. A hardware efficient, scalable, and real-time computing strategy was proposed for the implementation of large-scale biologically neural networks with one million multi-compartment neurons [9]. Memory and multiplier resources can be saved with a set of efficient neuromorphic techniques for single-CMN implementation. The hardware performance of computational speed was enhanced by 56.59% with in comparison with the classical digital implementation method. A hardware implementation of a bio-plausible online-learning spiking neural network model focusing on reducing hardware cost and power consumption was proposed in [10]. The introduction of fixed-point operations and bit-shift operations reduces the hardware resources and power consumption significantly. A neural computing hardware unit and a neuromorphic system architecture based on a modified leaky integrate and fire neuron model in a spiking neural network was designed by Farsa et al. [11]. Their hardware implementation on FPGA showed that the maximum frequency of the neuron model was 412.371 MHz and has an excellent performance. Heidarpur et al. [12] realized the Izhikevich neuron model by using the CORDIC algorithm on FPGA and constructed online learning spiking neural network using the implemented Izhikevich neuron model. The results in the paper [12] show that the accuracy, effectiveness, and higher speed of the system has great improvement compared with the original model.

According to the previous works, it is obviously that high speed and low cost hardware implementation is important to simulate SNNs [13–17]. Actually, the main difficulties of the hardware implemented SNNs are to design the efficient exponent arithmetic circuit and multiplying unit. The goal of this paper is to describe and validate the scalable modular hardware architecture to simulate the neurodynamics of biological neurons in real time, providing a basis for exploring the development of bidirectional interfaces between SNNs hardware and biology. The SNN architecture in this paper is mainly referred to [18], which can be applied to the classification of pictures. The difference is that the LIF (Leaky-Integrate–Fire) neuron model and tempotron supervised learning rules are used in this paper. The SNN is fully coded and optimized in Verilog hardware description language for development on a single field-programmable gate array (FPGA) chip, so it can be used to build small and medium-sized SNNs in any lab. A combined circuit instead of lookup table implementation method is proposed to realize the complex computing of kernel function in LIF neuron model. In addition, this work replaces the multiplication operations in the weights training with the arithmetic shift, which can speed up the training efficiency and reduce the consumption of computing resources. As the basic unit of SNNs processing information, it can realize large-scale SNNs through VLSI design, which provides convenience for the design of new SNNs architecture in the future.

The organizational structure of this paper is as follows. Section 2 mainly describes the methods used to construct the spiking neural network applied to the classification of visual color features and software simulation results. Section 3 mainly describes the method of building the proposed spiking neural network based on FPGA hardware and the analysis of circuit performance index. Section 4 discusses the advantage of the hardware implementation in terms of system performance and circuit area. Finally, Section 5 concludes the paper.

2. Structure of SNN

In the context of spiking neural networks, the current activation level is normally modeled as some differential equation and thus can be considered to be a neuron's state, with incoming spikes pushing this value higher, and then either firing or decaying over time. Various coding methods exist for interpreting the outgoing spike train as a real-value number, either relying on the frequency of spikes, or the timing between spikes, to encode information. LIF neuron model and population coding are discussed in this section.

2.1. LIF neuron model

According to different biomimetic effects, neuron models can be generally divided into the compartment model and threshold fire model. The compartment model mimics the microscopic level of neurons by modeling the activity of the ion channel to simulate the state of the neurons and the transmission of the signal, whose computational complexity is high, so it is not easy to be truly applied to experimental engineering [19]. The LIF neuron model which can reproduce multiple pulse modes by simply adjusting its parameters (membrane capacitance and membrane resistance) is suitable for hardware calculations because of its good compromise between computational complexity and biomimetic accuracy.

Fig. 1 shows the equivalent physical model of LIF neurons, which is a resistive-capacitor (RC) circuit consisting of a membrane capacitance C_m and a membrane resistance R_m. The external input current I is used as the driving current to simulate the

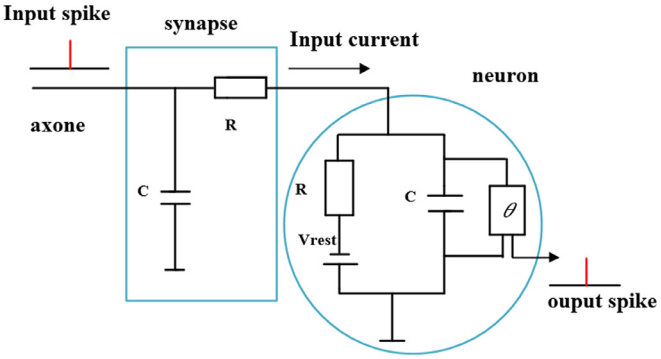

Fig. 1. Equivalent physical circuit of LIF neurons.

process of kinetic changes of the membrane potential of LIF neurons. The membrane potential V remains the same as the battery voltage V_{rest} in a resting state without any input current. When the current enters the neuron circuit, the branch current $I_c(t)$ charges the capacitor, and the branch current $I_R(t)$ flows through the resistor to discharge the capacitor.

The process in which the membrane potential of LIF neurons changes with input current is described in the form of a differential equation as follows:

$$\tau_m \frac{dV}{dt} = -(V - V_{rest}) + R_m I(t) \tag{1}$$

where $\tau_m = C_m R_m$ is called the membrane potential time constant, and membrane potential V, Synaptic current I, resting potential V_{rest}.

2.2. Population coding

Neurocoding is one of the important processes in memory cognitive systems by processing perceptual input information so that advanced brain centers can discern different input patterns, such as memory storage and retrieval. Benefiting from the similarity between spiking neurons and biological neurons in processing discrete spike signals, time-encoding is generally used to process image information and transform it into a series of spike trains that are easy to process by spiking neurons to simulate the pattern of biological neurons receiving signals.

Time coding uses different fire times of neurons to encode different data, the purpose of which is to represent the input information as an accurate spike-issue time train. In this paper, feature information is encoded by a population coding method suitable for image pixel processing. Population coding uses a set of Gaussian curves with different mean and the same variance to encode the input data through a mathematical probability distribution model, and maps the input data to the spikes time of multiple neurons, which can be obtained by equations following:

$$\begin{aligned} \mathrm{f}(x_i) &= A \exp\left(-\frac{(x_i - c_i)^2}{2\sigma^2}\right) \\ \sigma &= \frac{1}{\gamma(m+1)} \\ c_i &= \frac{i-1}{m-1} \end{aligned} \tag{2}$$

where A is expressed as the maximum spike time, and m is expressed as the number of gaussian reception domains, γ is expressed as a control parameter.

2.3. Synaptic plasticity

It is biologically consistent that the human brain continuously adjusts the connection strength between brain neurons to achieve the purpose of learning and memory by receiving external stimuli, and the storage and retrieval of information occur in the synapses of these neurons [20]. Spiking neurons simulate the learning mode of biological neurons, and dynamically learns the synaptic weight according to different spike distribution modes. The training algorithms are generally divided into unsupervised learning rules and supervised learning rules [21,22]. As a typical representative of unsupervised learning [23], STDP learning rules, without the constraints of the target, spontaneously learn the intrinsic characteristics of the input information, and require less sample information. However, the iteration accuracy is slower, and the network training speed and accuracy are not high. The tempotron supervised learning rules are derived from STDP, which can adjust the weight of the neural network in a direction through the supervised process, so that the network can achieve the effect of improving the iterative accuracy efficiently.

The LIF neuron membrane potential is the weighted sum of post-synaptic voltages (PSP) from all afferent neurons:

$$V(t) = \sum_j w_j \sum_i K(t - t_i) + V_{rest} \tag{3}$$

Where and t_i are the ith incoming synaptic weight and spike issuance time, respectively. V_{rest} is the resting potential of neurons. K represents the normalized PSP kernel function:

$$K(t - t_i) = V_o\left(e^{-\frac{(t-t_i)}{\tau_m}} - e^{-\frac{(t-t_i)}{\tau_s}}\right) \tag{4}$$

Where τ_m and τ_s denote the decay time of membrane potential and synaptic current with time, respectively. The formula for the tempotron learning rule weight update is:

$$\Delta w_1 = \lambda \sum_{t_i < t_{\max}} K(t_{\max} - t_i) \tag{5}$$

Where $t_{\max}$ represents the time when the neuron membrane potential V peaks, and λ is the learning rate.

According to the fire state of output neurons, we can encode the classification results by binary coding or one-to-one coding. When there are many categories to be divided, it is recommended to use binary coding, which can greatly reduce network parameters, reduce computational complexity, and speed up training. But it also limits the generalization ability of the network and its accuracy. For fewer categories, one-to-one coding can make the network more robust and leave room for improvement in network performance. After comprehensively considering the performance of network accuracy and training speed, this paper uses one-to-one coding to encode all classes. From a biological point of view, the training algorithm should adjust the synaptic weights according to the time difference between the presynaptic and post-synaptic neuron release spikes. Larger time differences result in less change in weight. Conversely, a shorter time difference will result in a dramatic change in weight. Fig. 2 is a schematic diagram of weight update, where the LTP window is used to increase synaptic weights and the LTD window is used to reduce synaptic weights [24].

Based on the original tempotron algorithm, this paper proposes a training method combining weight momentum. It is inspired by the laws of physical motion in nature, which simulates the inertia of an object during motion. When calculating the current weight update amount, a certain degree of last weight update amount is reserved to make fine adjustments to determine the final update direction. It does not depend entirely on the current computational gradient, which can increase the stability of the algorithm to some extent and speed up the convergence. In addition, due to the existence of weight momentum, the training algorithm has the ability to jump out of the local optimal trap to a certain extent, so that the network achieves optimal performance. This paper proposes a

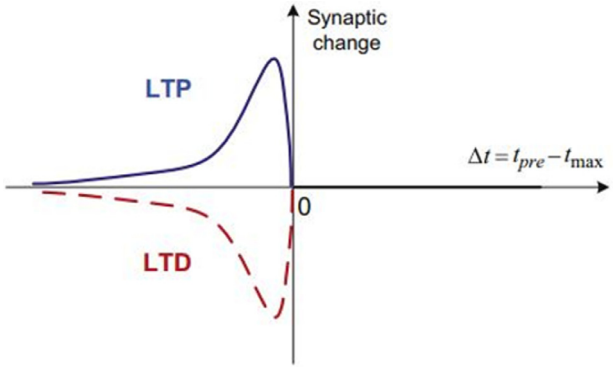

Fig. 2. Tempotron supervised learning rules.

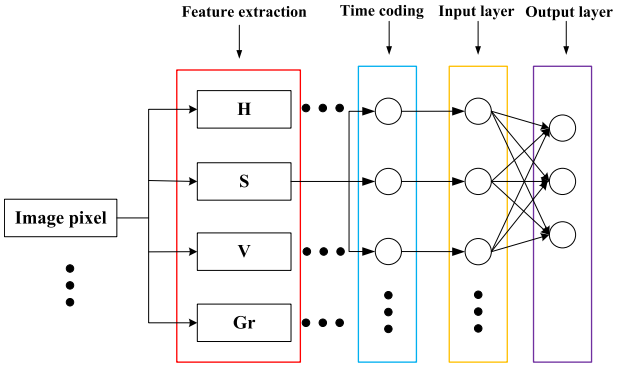

Fig. 3. SNNs topology applied to visual color features classification.

weight update rule that combines momentum:

$$\Delta w_i = \lambda \sum_{t_i < t_{\max}} K(t_{\max} - t_i) + \Delta w_i^d \tag{6}$$

Where $\Delta w_{\mathrm{i}}^{\mathrm{d}}$ is the current weight momentum term, and Δw_{i} is the current weight update amount.

The weight momentum is equal to the momentum learning rate multiplied by the last weight change amount, where the momentum learning rate m is similar to the definition of the weight learning rate and is a constant. It characterizes the extent to which the last weight change contributes to the current gradient, with values between 0 and 1. The formula for calculating the weight momentum is:

$$\Delta w_{\mathrm{i}}^{\mathrm{d}} = \lambda_{\mathrm{m}} \Delta w_{\mathrm{i}}^{l} \tag{7}$$

Where Δw_i^l is the last weight change amount. As the network converges, the update of the weight parameters will enter the fine-tuning phase. If the same momentum learning rate as the initial state is still used, the network accuracy will be slightly oscillated in the final stage of training. It can be solved by introducing a linearly attenuated momentum. The specific implementation method is as follows: multiply the momentum learning rate by a momentum learning attenuation factor. As the training progresses, the weight momentum will gradually decay and disappear. In the final stage of training, the network mainly determines the update amount of the weight from the current gradient, which is beneficial to the network to reach a stable convergence state. The formula for calculating the linear attenuation weight momentum is:

$$\Delta w_i^{\mathrm{d}} = d_\lambda \lambda_{\mathrm{m}} \Delta w_i^l \tag{8}$$

Where d_λ is the attenuation factor of the momentum learning rate. The research discussion on the weight momentum can be further explored. For example, different weight parameters have different amounts of change in the same training process, so adaptively assigning different momentum learning rates to each weight parameter is a solution. The specific implementation method is to establish certain rules, and dynamically adjust the respective momentum learning rates according to the magnitude of the respective parameter update amounts. But it also increases the computational complexity and needs to be considered based on our design needs.

2.4. SNN prototype

In this paper, a population of 12 Gaussian curve clusters is used to encode the input characteristics of four channels, and a time series consisting of 48 discrete events is obtained. Therefore, 48 LIF neurons are designed to accept the input spike information, and three output neurons are designed to correspond to the three categories that need to be divided.

Fig. 3 is a schematic diagram of the proposed spiking neural network topology applied to the classification applications. It consists of an image feature extraction layer, a temporal coding layer, an input layer, and an output layer.

The image feature extraction layer extracts color features generate four-channel input information with high color correlation. The time coding layer encodes the feature information into a series of spike trains using a population coding. The kinetic model of the LIF neuron accepts the input of the spike trains and converts them to the membrane potential of the neuron. The weight matrix is multiplied by the membrane potential and summed to obtain the output neuron membrane potential, and compared with the set threshold to determine whether the output neuron fire. In the training phase, the tempotron learning rules are used to train the network, update the weight matrix, and the trained network is obtained after the iteration is completed. During the testing phase, synaptic weights no longer change. The network accepts the input spike trains and directly outputs the classification result after calculation. The network parameter settings in this paper are shown in Table 1.

3. Hardware implementation

Based on the SNN structure mentioned in Section 2, the high-speed and low-cost design of hardware implementation will be discussed in detail. Five necessary parts such as a SNN module, a recognition module, a weight updating module, a controller block and a memory module should be designed for full functional hardware implementation of a SNN, as shown in Fig. 4.

In this hardware design, all the computations are performed using 16-bit fixed-point arithmetic to satisfy the accuracy threshold of the SNN prototype and create a balance between performance and circuit scale. Composed by a series of registers, memory module stores all the weights of synapses. Driven by clock signal "*Clock*", a counter module of the controller starts counting from "0" when a new sample is entered into the SNN. The counter value is one of the crucial control signals to the whole SNN hardware. The counting range is from "0" to "100". So, the worst case execution time of one input sample is 100 system clock cycles. However, the average execution time of one input sample is no more than 20 system clock cycles.

The proposed SNN hardware implementation adopts on-chip training pattern to update the stored weights in real time. Splitting into training mode and recognition mode, the operating modes change with the control signal "*Train_en*" in Fig. 4. The signal "*label*" represents the labels of input samples and the signal "*Result*" carries the target recognition results. SNN module, recognition module and weight updating module complete the main arithmetic operations. To reduce the size of the hardware circuit, SNN module shares the arithmetic units of kernel function with weight

Table 1
The parameters set in simulations about neuron, population coding and training.

Parameter type	Symbol	Description	Value
Neuron	τ_m	Membrane potential decay time constants	1.5 ms
	τ_s	Synaptic current decay time constants	0.375 ms
	V_{thresh}	Neuron threshold potential	1 mV
	V_{rest}	Neuron resting potential	0 mV
Population coding	A	Gaussian curve amplitude	1
	γ	Gaussian curve control parameters	1.15
	m	Number of Gaussian curves	12
Training	λ	Weight learning rate	0.004
	λ_m	Momentum learning rate	0.99
	d_λ	attenuation factor	0.5
	T	Time interval of a simulation	100 ms

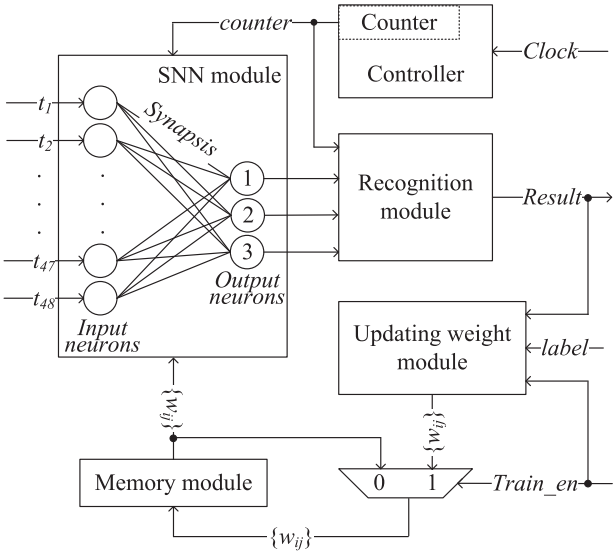

Fig. 4. The structure of hardware implementation.

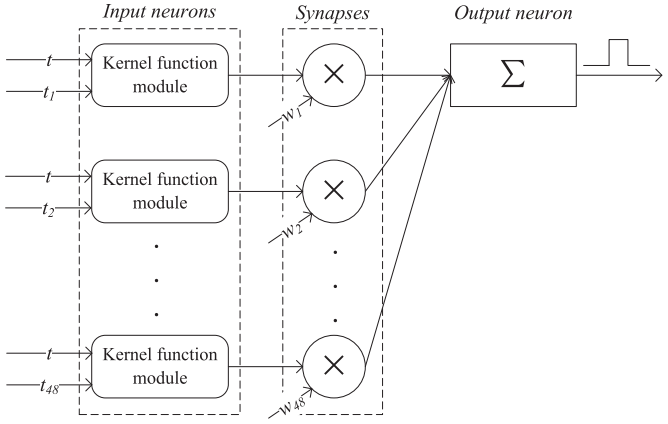

Fig. 5. The basic data path of SNN.

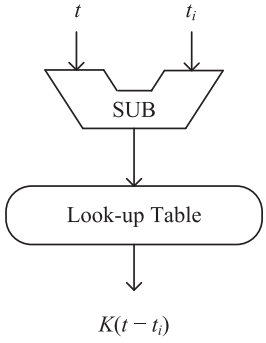

Fig. 6. The hardware implementation of kernel function (an input neuron).

updating module. The main hardware modules are described in the rest of this section.

3.1. SNN module

SNN module consists of 48 input neurons, 3 output neurons, and 144 synapses. All neurons and synapses run in parallel to achieve high performance. The main arithmetical operations are the kernel function in each input neuron, the accumulation operation in each output neuron and the multiplication operation in each synapsis. The basic data path is exhibited in Fig. 5. The input neurons receive an input sample and transmit the output results of the kernel function to the synapses. After multiplied by the weights in the synapses, the data enter the accumulator of the output neuron. Finally, the output neuron outputs a pulse signal only if the result of the accumulator exceeds the threshold value. From the perspective of hardware design, the kernel function is the most complicated but most significant, which directly relates to the performance of SNN hardware. In the parallel input layer, every input neuron has a kernel function module, which may result in a huge hardware scale.

Traditionally, the kernel function hardware circuit is implemented through Coordinate Rotation Digital Computer (CORDIC), which is inefficient and costly. A more efficient and lower cost implementation of kernel function can be achieved using simple precomputations and look-up tables. The hardware circuit of the kernel function designed in this paper is shown in Fig. 6. The combined circuit implemented look-up table module stores precomputed values of kernel function and outputs the function value based on the value of $(t - t_i)$. In the proposed SNN model, the values of t and t_i are quantified as the integers between "0" and "100" according to the coding of image samples. As a result, all the values of $(t - t_i)$ are the integers between "−100" and "100". Since the kernel function outputs "0" when the input value is smaller than "0" or bigger than "100", the valid values of $(t - t_i)$ are the integers between "0" and "100". Compared with the software implementation based on Matlab, the proposed hardware of kernel function achieves an acceptable accuracy. As shown in Fig. 7, the maximum absolute error is less than 0.6×10^{-4}.

Compared with CORDIC, the proposed implementation of kernel function based on combined circuit implemented look-up table has prominent advantages in terms of circuit performance and hardware resources as shown in Table 2. The proposed implementation based on Vertix-7 FPGA only needs 57LUTs and can output

Table 2
The hardware implementations of kernel function.

	Platform	Working frequency	Area	Cycles	Elapsed time
Our work	Vertix-7	480 MHz	57 LUTs	1	2.1 ns
CORDIC	Vertix-7	250 MHz	831 LUTs	16	64 ns

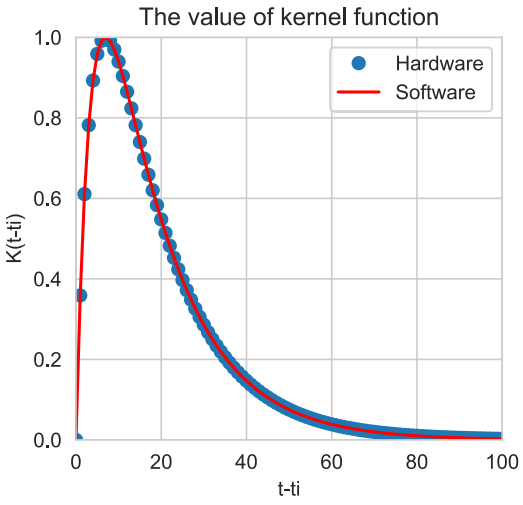

Fig. 7. The precision of hardware implemented kernel function.

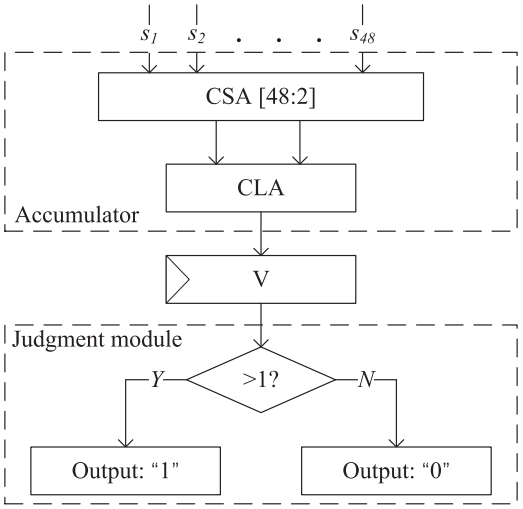

Fig. 8. The hardware structure of an output neuron.

the right result in 2.1 ns. However, The CORDIC implementation needs 831 LUTs and 64 ns to accomplish the same operations. Because of the small circuit scale, the proposed implementation of kernel function can be used for devising parallel input neurons and is easy to expand the network scale. The most important thing is that the input neurons can achieve high performance with small hardware size.

There are 48 parallel input neurons in the SNN module of Fig. 4, and each input neuron consists of a hardware circuit of kernel function to output the desired values. As shown in Fig. 6, t_i receives the data of input sample and t connects to the output of the counter of the controller.

The output values of input neurons are transmitted to the connected synapses for the subsequent multiplication operations. Since each synapsis is a multiplier which is a rather costly hardware block in terms of circuit size and power consumption, the abundant DSP blocks of FPGA are highly preferred for high-performance and low-cost. In this work, the DSP blocks are used for designing the synapses.

One output neuron is composed by an accumulator, a register array and a judgment module, as shown in Fig. 8. The accumulator has a compressor and a carry lookahead adder (CLA). The compressor composed by several carry save adders (CSA) compresses 48 data given by synapses into 2 data, and then the CLA generates the final result "V" which is stored in the register array. If the judgment module outputs "1" in certain clock cycle, the output neuron and its connective synapses stop working the judgment module starts outputting "0" in the next clock cycle. Until a new sample coming, the output neuron and its connective synapses resume their operations.

3.2. Recognition module

Recognition module judges which class the sample belongs in by the arrival orders of the output pulses of output neurons shown in Fig. 4. The first arriving pulse triggers the output of the recognition module. The output results {'1', '2' and '3'} represent related classifications (red, green or blue) respectively. If no pulse arrives within the time required, the recognition module will output a result '0' which means that the SNN fails to identify the input sample. It is convenient to implement the recognition module by using the behavior level coding of verilog HDL.

3.3. Weight updating module

To reduce circuit area and speed-up the hardware performance, the weight updating module is designed by using shareable modules and approximate operations. According to the weights updating strategy before, the weight updating module needs the kernel function modules and the multiplying units to accomplish the operation of $\lambda \times K(t)$ (λ is the learning rate). Each input neuron contains a kernel function module which can be easily reused in the weight updating module. To update all the weights of 144 synapses, the weight updating module requires three steps because of only 48 kernel function modules existing. In every step, the weight updating module enhances or inhibits the relevant weights. In this work, the learning rate is 0.004 which can be approximated by $1/2^8 = 0.00390625$. So, the computation of $\lambda \times K(t)$ can be simplified through the following approximation relation.

$$\lambda \times K(t) \approx \frac{K(t)}{2^8} \tag{9}$$

Similarly, the value of Δw_i^d in Eq. (8) can be calculated through the arithmetic right shift as well.

$$\Delta w_i^d \approx \frac{\Delta w_i^l}{2} \tag{10}$$

Using the arithmetic right shift to replace the multiplication, the weight updating module becomes more efficient and smaller. The weight updating module consists of 48 identical submodules and the structure of one submodule is shown in Fig. 9. The kernel function module is shared with the input neuron. The value of the kernel function is right shift 8 bits to approximate $0.004 \times K(t)$. The two "ADD" modules are adders used for computing the Eq. (6). The register "Δw_i^l" stores the last change of the weight "w_i". The module "ADD/SUB" can be used as an adder-subtractor

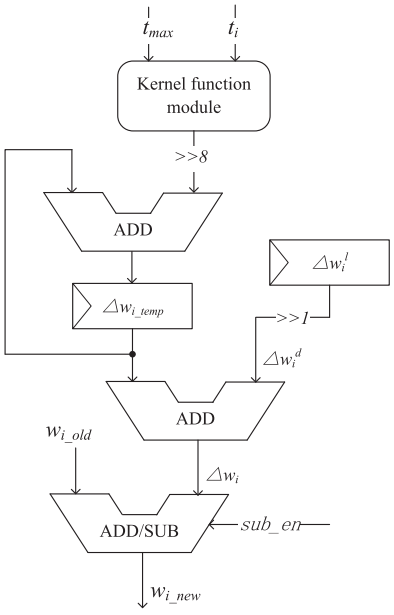

Fig. 9. The hardware structure of one submodule of the weight updating module.

which is controlled by the signal "*sub_en*". If "*sub_en*" equals "1", the weights will be inhibited. If "*sub_en*" equals "0", the weights will be enhanced. The details of the weights updating process are as following:

- **STEP 1**: If the "*label*" of the input sample is "1", the control signal "*sub_en*" will equal to "0" and the weights of the synapses connected to output neuron 1 will be enhanced. If the "*label*" is not "1", "*sub_en*" will equal to "1" and the weights of the synapses connected to output neuron 1 will be inhibited.
- **STEP 2**: If the "*label*" of the input sample is "2", the control signal "*sub_en*" will equal to "0" and the weights of the synapses connected to output neuron 2 will be enhanced. If the "*label*" is not "2", "*sub_en*" will equal to "1" and the weights of the synapses connected to output neuron 2 will be inhibited.
- **STEP 3**: If the "*label*" of the input sample is "3", the control signal "*add_en*" will equal to "0" and the weights of the synapses connected to output neuron 3 will be enhanced. If the "*label*" is not "3", "*sub_en*" will equal to "1" and the weights of the synapses connected to output neuron 3 will be inhibited.

3.4. Controller

The controller is a finite state machine containing seven states as shown in Fig. 10. The main control signals are "*start_snn*" and "*train_en*". "*start_snn*" is the flag of starting to work and will be set to "1" when a sample data enter. "*train_en*" is the flag of training.

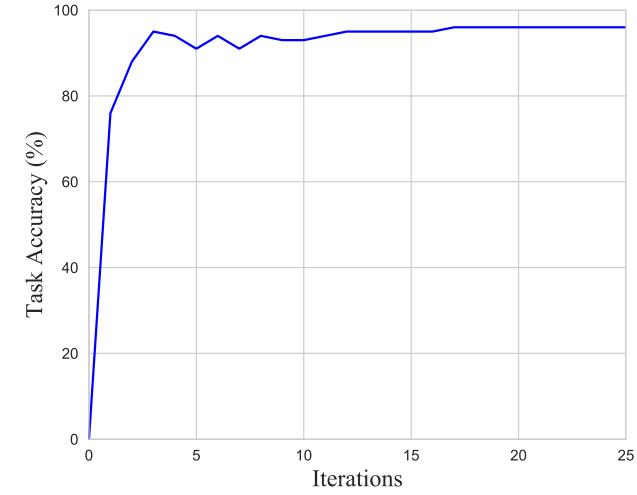

Fig. 11. The network accuracy of the proposed SNN hardware implementation.

- **IDLE**: In this idle state, the whole hardware keeps off work to reduce energy consumption until the users start inputting a sample.
- **SAMPLE_IN**: The hardware receives the input sample and transmits the data to the SNN module.
- **RECOGNITION**: The SNN circuit starts classifying the input sample and the result will be outputted in one clock cycle.
- **FINISH**: The hardware outputs the recognition result and then enters the idle state in next cycle.
- **UPDATA_W1**: The weight updating module starts working to enhance or inhibit the synapses connected with the output neuron 1.
- **UPDATA_W2**: The weight updating module starts working to enhance or inhibit the synapses connected with the output neuron 2.
- **UPDATA_W3**: The weight updating module starts working to enhance or inhibit the synapses connected with the output neuron 3.

4. Experimental results and discussions

This section presents the results of experiments with the hardware implementation of the SNN on a Xilinx Vertix-7 FPGA Development board. In order to compare the performance of hardware system with software simulations, the SNN is also implemented in Matlab on a PC with an Intel core i7-6700 CPU. The data set used in the experiment contains 1000 samples(pixels) which are encoded using Gaussian curve clusters. Imitating the function of the human eyes, the SNN recognizes the dominant tone (red, green or blue) of each sample. The training set contains 800 samples and the testing set contains 200 samples. The total iterations in the experiment is set to 25. To test the performance of the SNN, the

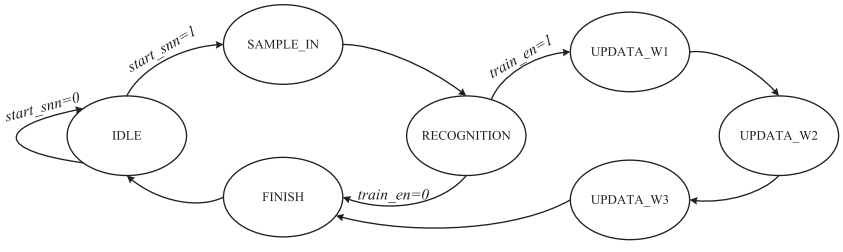

Fig. 10. The state machine diagram of the controller.

Table 3
The hardware implementations of the proposed SNN hardware design.

Platform	Total time	Samples	Iterations	Speed-up
Hardware	14.397 ms	1000	25	908,578
Software	13080.806 s	1000	25	–

Table 4
The hardware implementations of the proposed SNN hardware design.

Platform	Max Speed	Slice LUTs	Slice Registers	DSPs
Vertix-7	178 MHz	13,862	5487	144

Table 5
The resources occupations of different parts in SNN hardware.

Module	Slice LUTs	Slice Registers	DSPs
SNN module	6959	2656	144
Recognition module	30	0	0
Weight updating module	3528	768	0
Controller	3345	1963	0

Table 6
The comparison of between the proposed design and other works in terms of the hardware implementation of the neuron model.

Reference	Platform	LUTs	Registers	DSPs	Max speed	Error(%)
This work	Vertix-7	112	53	0	480 MHz	0.006
[12]	Spartan-6	469	280	0	212.8 MHz	0.003
[25]	Spartan-6	1221	829	0	134.3 MHz	0.040
[26]	Spartan-3	543	250	1	78.4 MHz	Not report
[27]	Kintex-7	416	364	1	663.1 MHz	0.910
[28]	Virtex-2	593	286	0	146 MHz	0.790

whole test time for 25 iterations and the network accuracy in each iteration are recorded concisely.

The proposed SNN hardware implementation achieves a good network accuracy after 17 training iterations. As shown in Fig. 11, the network accuracy converges rapidly and the final correct rate approximately stabilizes at 96%. Compared with the software implementation, the proposed SNN hardware implementation has a prominent advantage in performance. As shown in Table 3, the proposed hardware implementation only needs 14.397 ms to complete all the 25 iterations and gets 908,578 times faster than the software implementation.

For hardware experiment results, Table 4 shows the details of the proposed SNN hardware design in terms of performance and resources. Table 5 shows the resources occupations of different parts in SNN hardware. Table 6 shows comparison between the proposed design and other works in terms of the hardware implementation of the neuron model. Since the FPGA devices and synthesizer used are different in these works, this table results should be considered relatively. However, it can be seen that the proposed implementation of the LIF neuron model has the minimum circuit area and the impressive performance. The normalized root mean square deviation error of the proposed design is smaller than most works. The accuracy of the neuron model is mainly related to the bit length of the fixed point number used in the designs. In order to balance the computation accuracy and circuit speed, the 16 bits fixed point number is used in our SNN hardware implementation. The tiny errors of our design further confirm that the proposed neuron implementation has a similar behavior to the original model. Overall, the proposed hardware design is quite suitable for the large scale SNN implementation.

5. Conclusions

This work focuses on the hardware design of SNNs and proposes a low-cost and high-performance implementation of SNN based on FPGA. A combinational logic circuit including a precomputation module is designed to achieve the function of the LIF neuron model. Compared with the traditional CORDIC implementation, the proposed structure consumes smaller area with higher speed. The multiplication operations during the weights training process can be saved by a well-designed arithmetic shift according to a reasonable calculated weight learning rate, which greatly speeds up the training efficiency and reduces the consumption of hardware resources. Based on the Vertix-7 FPGA platform, the total resources with on-chip learning function only consume 13,862 LUTs, 5487 registers and 144 DSPs working at a maximum frequency 178 MHz. The correct rate of color classification is approximate to 96%. Compared with the software implementation, the proposed SNN hardware implementation gets 908,578 times faster in terms of performance.

Declaration of Competing Interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled.

CRediT authorship contribution statement

Guohe Zhang: Project administration, Methodology, Writing - review & editing. **Bing Li:** Conceptualization, Writing - original draft, Software. **Jianxing Wu:** Investigation, Writing - original draft, Software. **Ran Wang:** Formal analysis, Writing - review & editing. **Yazhu Lan:** Data curation, Visualization. **Li Sun:** Software, Visualization. **Shaochong Lei:** Resources, Supervision. **Hai Li:** Writing - review & editing, Validation. **Yiran Chen:** Methodology, Supervision.

Acknowledgments

This research was funded by the National Natural Science Foundation of China under Grant No.61701531.

References

[1] K.D. Carlson, J.M. Nageswaran, N. Dutt, J.L. Krichmar, An efficient automated parameter tuning framework for spiking neural networks, Front. Neurosci. 8 (2014) 10.
[2] H. Fang, A. Shrestha, D. Ma, Q. Qiu, Scalable NoC-Based Neuromorphic Hardware Learning and Inference, 2018, pp. 1–8. 10.1109/IJCNN.2018.8489619
[3] A. Taherkhani, A. Belatreche, Y. Li, L.P. Maguire, A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks, IEEE Trans. Neural Netw. Learn. Syst. 29 (11) (2018) 5394–5407, doi:10.1109/TNNLS.2018.2797801.
[4] J. Haeng Lee, T. Delbruck, M. Pfeiffer, Training deep spiking neural networks using backpropagation, Front. Neurosci. 10 (2016), doi:10.3389/fnins.2016.00508.
[5] D. Pani, P. Meloni, G. Tuveri, F. Palumbo, P. Massobrio, L. Raffo, An FPGA platform for real-time simulation of spiking neuronal networks, Front. Neurosci. 11 (2017), doi:10.3389/fnins.2017.00090.
[6] S. Carrillo, J. Harkin, L.J. McDaid, F. Morgan, S. Pande, S. Cawley, B. McGinley, Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations, IEEE Trans. Parallel Distrib. Syst. 24 (12) (2013) 2451–2461, doi:10.1109/TPDS.2012.289.
[7] J. Liu, J. Harkin, L.P. Maguire, L.J. McDaid, J.J. Wade, Spanner: a self-repairing spiking neural network hardware architecture, IEEE Trans. Neural Netw. Learn. Syst. 29 (4) (2018) 1287–1300, doi:10.1109/TNNLS.2017.2673021.
[8] S. Yang, J. Wang, B. Deng, C. Liu, H. Li, C. Fietkiewicz, K.A. Loparo, Real-time neuromorphic system for large-scale conductance-based spiking neural networks, IEEE Trans. Cybern. 49 (7) (2019) 2490–2503, doi:10.1109/TCYB.2018.2823730.

[9] S. Yang, B. Deng, J. Wang, H. Li, M. Lu, Y. Che, X. Wei, K.A. Loparo, Scalable digital neuromorphic architecture for large-scale biophysically meaningful neural network with multi-compartment neurons, IEEE Trans. Neural Netw. Learn. Syst. (2019) 1–15, doi:10.1109/TNNLS.2019.2899936.
[10] G.C. Qiao, S.G. Hu, J.J. Wang, C.M. Zhang, T.P. Chen, N. Ning, Q. Yu, Y. Liu, A neuromorphic-hardware oriented bio-plausible online-learning spiking neural network model, IEEE Access 7 (2019) 71730–71740, doi:10.1109/ACCESS.2019.2919163.
[11] E.Z. Farsa, A. Ahmadi, M.A. Maleki, M. Gholami, H.N. Rad, A low-cost high-speed neuromorphic hardware based on spiking neural network, IEEE Trans. Circuits Syst. II: Exp. Briefs (2019) 1, doi:10.1109/TCSII.2019.2890846.
[12] M. Heidarpur, A. Ahmadi, M. Ahmadi, M. Rahimi Azghadi, Cordic-SNN: on-FPGA STDP learning with Izhikevich neurons, IEEE Trans. Circuits Syst. I: Regul. Pap. 66 (7) (2019) 2651–2661, doi:10.1109/TCSI.2019.2899356.
[13] A.M. Abdelsalam, F. Boulet, G. Demers, J.M. Pierre Langlois, F. Cheriet, An efficient FPGA-based overlay inference architecture for fully connected DNNs, in: Proceedings of the 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2018, pp. 1–6, doi:10.1109/RECONFIG.2018.8641735.
[14] Y. Fan, C. Zou, K. Liu, Y. Kuang, X. Cui, A digital neuromorphic hardware for spiking neural network, in: Proceedings of the 2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), 2019, pp. 1–2, doi:10.1109/EDSSC.2019.8754093.
[15] A. Ghani, T.M. McGinnity, L.P. Maguire, J. Harkin, Area efficient architecture for large scale implementation of biologically plausible spiking neural networks on reconfigurable hardware, in: Proceedings of the 2006 International Conference on Field Programmable Logic and Applications, 2006, pp. 1–2, doi:10.1109/FPL.2006.311352.
[16] M.A. Nuno-Maganda, M. Arias-Estrada, C. Torres-Huitzil, H.H. Aviles-Arriaga, Y. Hern'ndez-Mier, M. Morales-Sandoval, A hardware architecture for image clustering using spiking neural networks, in: Proceedings of the 2012 IEEE Computer Society Annual Symposium on VLSI, 2012, pp. 261–266, doi:10.1109/ISVLSI.2012.46.
[17] T.H. Vu, A.B. Abdallah, Low-latency k-means based multicast routing algorithm and architecture for three dimensional spiking neuromorphic chips, in: Proceedings of the 2019 IEEE International Conference on Big Data and Smart Computing (BigComp), 2019, pp. 1–8, doi:10.1109/BIGCOMP.2019.8679363.
[18] Q.Y. Sun, Q.X. Wu, X. Wang, L. Hou, A spiking neural network for extraction of features in colour opponent visual pathways and FPGA implementation, Neurocomputing 228 (2017) 119–132, doi:10.1016/j.neucom.2016.09.093. Advanced Intelligent Computing: Theory and Applications.
[19] H. Soleimani, A. Ahmadi, M. Bavandpour, Biologically inspired spiking neurons: piecewise linear models and digital implementation, IEEE Trans. Circuits Syst. I: Regul. Pap. (IEEE T CIRCUITS-I) 59 (2012) 2991–3004, doi:10.1109/TCSI.2012.2206463.
[20] A. Shukla, U. Ganguly, An on-chip trainable and the clock-less spiking neural network with 1r memristive synapses, IEEE Trans. Biomed. Circuits Syst. 12 (4) (2018) 884–893, doi:10.1109/TBCAS.2018.2831618.
[21] S.J. Jarvis, R. Caz, C. Clopath, Extending the tempotron with hierarchical dendrites allows faster learning, BMC Neurosci. 16 (2015) P37, doi:10.1186/1471-2202-16-S1-P37.
[22] X. Xie, G. Liu, Q. Cai, H. Qu, M. Zhang, The maximum points-based supervised learning rule for spiking neural networks, Soft Comput. (2018), doi:10.1007/s00500-018-3576-0.
[23] Y. Hao, X. Huang, M. Dong, B. Xu, A Biologically Plausible Supervised Learning Method for Spiking Neural Networks Using the Symmetric STDP Rule 121 (2020) 387–395, doi:10.1016/j.neunet.2019.09.007.
[24] E. Roggenhofer, P. Fidzinski, O. Shor, J. Behr, Reduced threshold for induction of LTP by activation of dopamine D1/D5 receptors at hippocampal CA1 subiculum synapses, PloS One 8 (2013) e62520, doi:10.1371/journal.pone.0062520.
[25] M. Heidarpour, A. Ahmadi, R. Rashidzadeh, A cordic based digital hardware for adaptive exponential integrate and fire neuron, IEEE Trans. Circuits Syst. I: Regul. Pap. 63 (11) (2016) 1986–1996, doi:10.1109/TCSI.2016.2598161.
[26] T. Iakymchuk, A. Rosado, J.V. Frances, M. Batallre, Fast spiking neural network architecture for low-cost FPGA devices, in: Proceedings of the 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2012, pp. 1–6, doi:10.1109/ReCoSoC.2012.6322906.
[27] H. Soleimani, E.M. Drakakise, An efficient and reconfigurable synchronous neuron model, IEEE Trans. Circuits Syst. II: Exp. Briefs 65 (1) (2018) 91–95, doi:10.1109/TCSII.2017.2697826.
[28] S. Gomar, A. Ahmadi, Digital multiplierless implementation of biological adaptive-exponential neuron model, IEEE Trans. Circuits Syst. I: Regul. Pap. 61 (4) (2014) 1206–1219, doi:10.1109/TCSI.2013.2286030.

Guohe Zhang was born in Hubei, China in 1981. He received his B.S. and Ph.D. degrees in 2003 and 2008 respectively from Xi'an Jiaotong University, China. He is currently an Associated Professor in School of Microelectronics, Xi'an Jiaotong University, Xi'an, China. His research interests include the semiconductor device physics and integrated circuits design, image processing and intelligent system, algorithm and hardware co-design and implementation for deep learning and signal processing systems, error-resilient low-cost computing techniques for embedded systems.

Bing Li is currently a Ph.D. in School of Microelectronics, Xi'an Jiaotong University, Xi'an, China. He received the B.S. degree and M.S degree from School of Electronic and Information Engineering, Henan University of Science and Technology, China, in 2011 and 2015, respectively. His current research interests include Elliptic curve cryptosystem, machine learning, and hardware implementation of neural networks.

Jianxing Wu is currently a Master Degree Candidate at the school of microelectronics, Xi'an Jiaotong University in China. He received a bachelor's degree in electronic science and technology from Chang'an University in June 2017 in China. His research is focused on spiking neural networks for image recognition, digital integrated circuit design.

Ran Wang is currently a Master Degree Candidate in the Xi'an Jiaotong University, in China. He received B.S. degree from Nanchang University, in China, in 2018. His research interests are machine learning, visual image processing algorithms and FPGA technology.

Yazhu Lan received his Ph.D. degree from Chinese Academy of Sciences University, Beijing, in 2015. He is now an assistant professor in Institute of Computing Technology, Chinese Academy of Sciences. Starting from December 2017, he has done postdoctoral research at Duke University, Durham, NC, USA. His current research interests include architecture for deep learning and the FPGA-based architecture for hardware accelerators. In addition, it has deep research on data center networks and core backbone architecture.

Li Sun studied computer science at the Northwestern Polytechnic University, China, where she received the Ph.D. degree in 2010. In 2016, she became an associate professor for information and communication engineering at the Airforce Engineering University. She is the currently working for Machine Learning in the microelectronics Academy at the Xi'an Jiaotong University. Her research interests include image processing, machine learning, and probabilistic graphical model.

Shaochong Lei received the M.S.E. and Ph.D. degrees from Xi'an Jiaotong University (XJU), Xi'an, China, in 1990 and 2007, respectively. He is currently a Professor in School of Microelectronics, Xi'an Jiaotong University, Xi'an, China. His current research interests include VLSI computer aided designs, system-on-chip testing, and design-for-test, VLSI design, testing and design for testable, ECC algorithm and hardware implementation.

Hai (Helen) Li is Clare Boothe Luce Associate Professor of Electrical and Computer Engineering Department at Duke University, USA. She works on hardware/software co-design for accelerating machine learning, brain-inspired computing systems, and memory architecture and optimization. She has authored or co-authored more than 300 technical papers in these areas and has authored a book entitled Nonvolatile Memory Design: Magnetic, Resistive, and Phase Changing (CRC Press, 2011). She is serving as the associate director of NSF Industry University Cooperative Research Center (IUCRC) for Alternative Sustainable and Intelligent Computing (ASIC) and co-director of Duke Center for Evolutionary Intelligence (CEI). Dr. Li also serves as an Associate Editor of IEEE TVLSI, IEEE TCAD, ACM TODAES, ACM TACO, IEEE TMSCS, IEEE TECS, IEEE CEM, and IET-CPS. She has served as organization committee and technical program committee members for over 30 international conference series. Sheis a recipient of the NSF CAREER Award (2012), the DARPA Young Faculty Award (2013), TUM-IAS Hans Fisher Fellowship from Germany (2017) and seven best paper awards. She is the Fellow of IEEE and Distinguished Member of ACM, a distinguished speaker of ACM (2017–2020), and a distinguished lecturer of IEEE CAS society (2018–2019).

Yiran Chen received B.S and M.S. from Tsinghua University and Ph.D. from Purdue University in 2005. After five years in industry, he joined University of Pittsburgh in 2010 as Assistant Professor and then promoted to Associate Professor with tenure in 2014, held Bicentennial Alumni Faculty Fellow. He now is the Professor of the Department of Electrical and Computer Engineering at Duke University and serving as the director of NSF Industry University Cooperative Research Center (IUCRC) for Alternative Sustainable and Intelligent Computing (ASIC) and co-director of Duke University Center for Computational Evolutionary Intelligence (CEI), focusing on the research of new memory and storage systems, machine learning and neuromorphic computing, and mobile computing systems. Dr. Chen has published one book and more than 350 technical publications and has been granted 94 US patents. He serves or served the associate editor of several IEEE and ACM transactions/journals and served on the technical and organization committees of more than 50 international conferences. He received 6 best paper awards and 13 best paper nominations from international conferences. He is the recipient of NSF CAREER award and ACM SIGDA outstanding new faculty award. He is the Fellow of IEEE and Distinguished Member of ACM, a distinguished lecturer of IEEE CEDA, and the recipient of the Humboldt Research Fellowship for Experienced Researchers.