Wang SQ, Wang L, Deng Y *et al.* SIES: A novel implementation of spiking convolutional neural network inference engine on field-programmable gate array. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 35(2): 475–489 Mar. 2020. DOI 10.1007/s11390-020-9686-z

SIES: A Novel Implementation of Spiking Convolutional Neural Network Inference Engine on Field-Programmable Gate Array

Shu-Quan Wang, Lei Wang*, Yu Deng, Zhi-Jie Yang, Sha-Sha Guo, Zi-Yang Kang, Yu-Feng Guo, and Wei-Xia Xu

College of Computer Science and Technology, National University of Defense Technology, Changsha 430041, China

E-mail: wangshuquan3@163.com; arrowya@gmail.com
{dengyu, yangzhijie18, guoshasha17, kangziyang14, guoyufeng, xuweixia}@nudt.edu.cn

Received May 1, 2019; revised February 13, 2020.

Abstract Neuromorphic computing is considered to be the future of machine learning, and it provides a new way of cognitive computing. Inspired by the excellent performance of spiking neural networks (SNNs) on the fields of low-power consumption and parallel computing, many groups tried to simulate the SNN with the hardware platform. However, the efficiency of training SNNs with neuromorphic algorithms is not ideal enough. Facing this, Michael *et al.* proposed a method which can solve the problem with the help of DNN (deep neural network). With this method, we can easily convert a well-trained DNN into an SCNN (spiking convolutional neural network). So far, there is a little of work focusing on the hardware accelerating of SCNN. The motivation of this paper is to design an SNN processor to accelerate SNN inference for SNNs obtained by this DNN-to-SNN method. We propose SIES (Spiking Neural Network Inference Engine for SCNN Accelerating). It uses a systolic array to accomplish the task of membrane potential increments computation. It integrates an optional hardware module of max-pooling to reduce additional data moving between the host and the SIES. We also design a hardware data setup mechanism for the convolutional layer on the SIES with which we can minimize the time of input spikes preparing. We implement the SIES on FPGA XCVU440. The number of neurons it supports is up to 4 000 while the synapses are 256 000. The SIES can run with the working frequency of 200 MHz, and its peak performance is 1.562 5 TOPS.

Keywords spiking neural network (SNN), field-programmable gate array (FPGA), neuromorphic, systolic array, spiking convolutional neural network (SCNN), integrete and fire (I&F) model, hardware accelerator

1 Introduction

As we all know, the human brain has a good performance of power efficiency, and it computes in a parallel way. Inspired by its attractive performance, people want to simulate the behaviors of neurons so that we can make further research on the brain. Many researchers tried to imitate biological neurons on the hardware platform. However, most of them worked inefficiently because of the large number of data operations in the procedure of SNN (spiking neural network) simulation. The demand for hardware accelerator for SNN simulation is increasing quickly.

To support the massive scale of SNN simulation, Akopyan *et al.*[1] from IBM proposed the TrueNorth chip. They used an event-driven simulation approach to improve the efficiency of data processing. Geddes *et al.*[2] from Stanford University worked out the Neurogrid system, which improved the efficiency of Neurocore chip with mixed-analog-digital hardware design. Schemmel *et al.*[3] from Heidelberg proposed a scheme called BrainScaleS, and they used analog circuits to simulate neurons with a 10 000-time speedup compared with the real biological process. Furber *et al.*[4] from Manchester showed a scheme called SpiNNaker. They used 18 ARM968 and 96 Kb on-chip memory on each core chip to reduce additional off-chip data moving.

Regular Paper
Special Section of ChinaSys 2019
The work was supported by the HeGaoJi Program of China under Grant Nos. 2017ZX01028103-002 and 2017ZX01038104-002, and the National Natural Science Foundation of China under Grant No. 61472432.
*Corresponding Author

Davies *et al.*[5] from Intel Laboratory proposed a neuronal multicore processor called Loihi. They provided a spike-based computation platform which has an advantage of configurable synaptic learning rules.

These researches are for SNNs which are trained by neuromorphic algorithms. As we know, the classifying accuracy of this kind of SNN is lower than that of the corresponding DNN (deep neural network) which has the same topology[6, 7]. Fortunately, we have a replacement scheme[6–8], and we can convert a trained DNN into the corresponding SNN which we call SCNN (spiking convolutional neural network). With this method, we can speed up the procedure of SNN training with the help of the corresponding DNN. Since SNN has an advantage of low power consumption, we can use SCNN to accomplish the task of inference on the area of power restricted hardware. At the same time, the recognizing precision of SCNN is almost the same as that of DNN. So far, there have been a few studies which focus on the hardware acceleration of SCNN.

Aiming to accelerate the procedure of SCNN inference, we propose an SNN inference engine called SIES (Spiking Neural Network Inference Engine for SCNN Accelerating). We integrate all the computation units in a systolic array to accelerate the computation of membrane potential increment. To reduce unnecessary data moving, we implement an optional max-pooling module and design a hardware mechanism to accelerate the procedure of input spikes preparing. With this design, we can significantly improve the efficiency of data processing and increase the parallelism of the hardware SNN. Note, though this design is similar to the classic DNN accelerator, our inference engine is for SCNN. It can support the updating of neuron membrane potential while the DNN accelerators such as TPU cannot. SIES is for SNN.

The main contributions of this paper are as follows.

• We propose an SNN inference engine called SIES to accelerate SCNN. It uses a systolic array to accelerate the procedure of membrane potential increment computation.

• We implement an optional max-pooling module and design a hardware data setup mechanism for convolutional layer on SIES. With these modules, we can reduce additional data moving and make the SIES efficient.

• We implement SIES on the FPGA (field-programmable gate array) platform of XCVU440 and verify the functions with a VGG-16 SNN. SIES can run with a frequency of 200 MHz, and its peak performance is 1.562 5 TOPS.

The content of the paper is as follows. We introduce related work in Section 2 and talk about the preliminaries in Section 3. Section 4 provides the architecture of SIES, and we explain the design details of functional modules in Section 5. We design the hardware data setup mechanism of the convolutional layer on the SIES in Section 6. In Section 7, we show the procedure of SNN inference on SIES. Then, we give a top view of our experimental setup and have an in-depth discussion of the experimental results in Section 8. Finally, we conclude and introduce the future work of our team in Section 9.

2 Related Work

With the growing of DNN scale, many people turned to accelerate the procedure of data processing with hardware accelerators. Du *et al.*[9] proposed an idea of reusing the input data to make the CNN simulate efficiently in the design of ShiDianNao. Guan *et al.*[10] proposed an FPGA-based accelerator for LSTM-RNNs, and they accelerated the procedure of multiplication with fine-tuned pipelines. Zhou *et al.*[11] introduced a 5-layer accelerator for MNIST (Mixed National Institute of Standards and Technology Database) digit recognition task, which uses the scheme of fixed point presentation. These researches showed the detail of hardware accelerator designing and inspired the implementation of the SIES.

There is also some work focusing on the field of SNN hardware accelerating[1–5]. Neil[12] proposed an event-driven FPGA-based spiking network accelerator which makes a trade-off between accuracy and latency. Wang *et al.*[13] proposed a time multiplexing scheme for SNN simulating, which gets a significant advantage in the field of real-time pattern recognition. Glackin *et al.*[14] introduced a scheme for the implementation of large-scale spiking neural network which adopts a fully parallel implementation strategy. Cheung[15] proposed an accelerator which parallelizes the synaptic processing with run time proportional to the firing rate of the network. However, a little of the work focuses on the SCNN accelerating.

On the other hand, there are a series of DNN-to-SNN methods. Rueckauer *et al.*[7] proposed a series of spiking equivalents operations to replace the corresponding CNN operations. Diehl *et al.*[6] introduced a set of optimization techniques to minimize performance loss in the procedure of DNN-to-SNN transformation. Rueckauer *et al.*[8] proposed several powerful tools to

convert a larger and more powerful class of deep networks into SNNs. These studies proved the correctness of the DNN-to-SNN method and showed the advantage of SCNN on the area of SNN training. The research of SCNN hardware accelerating is becoming more and more pressing.

3 Background and Preliminaries

3.1 Neuron Model

As we know, the biological neuron of the human brain consists of three components (i.e., dendrites, soma, and axon) [16]. The dendrites are used to receive spikes from other neurons. The soma is responsible for membrane potential updating. It can generate spikes to give feedback to the membrane potential changing. The axon is used to pass the spikes to the post neurons. The biological neuron also has a period of silence time after one spike action. It will not give any feedback to the membrane changing in this special period of time which we call the refractory period.

Since it is too complex to express all the details of biological neuron behaviors directly, people expressed the procedure of the neuronal membrane potential updating with differential equations. Hodgkin *et al.* [17] proposed the Hodgkin-Huxley model, which can simulate the neuronal behavior in a precise way with four complex differential equations. Izhikevich *et al.* [18] proposed the Izhikevich model, and it can model the biological neuron with two simple differential equations. Brunel *et al.* [19] proposed the Integrate and Fire (I&F) model, which expresses the behavior of the neuron in a much more simplified way compared with others. There are also many variants [20–22] which made further improvements of the I&F model. There are many other models [23–25], but we typically focus on these three models. They can represent different levels of complexity in the field of the hardware implementation of biological neurons.

Concerning the model selection, the Hodgkin-Huxley model can express the procedure of neuronal membrane potential updating accurately, but it is hard to implement. It is too complex, especially on the hardware resources limited FPGA platform. The Izhikevich model can model the behavior of biological neuron, but it still needs to handle the complex differential equations. The I&F model is much more straightforward than the Izhikevich model, and we can transform the single differential equation into a simple form with the Euler integration method [26, 27]. It will significantly reduce the challenge of hardware implementation on FPGA. There are also many variants of I&F model, which can fulfill the need of different researching scenes. Concerning the potential of the hardware accelerator simulation ability, we choose the I&F model to be our simulation target of the neuron model. We implement the I&F model neuron with hardware.

3.2 I&F model

The working procedure of the I&F model can be considered as a procedure of membrane potential updating. It follows (1) [28], where τ_m is a time constant and V denotes the membrane potential. E_L is the initial membrane potential of the neuron, and R denotes abstract resistance. I_e denotes the action current of the neuron.

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = E_L - V + RI_e. \tag{1}$$

With the Euler integration method, (1) transforms into (2).

$$V(t) = E_L + RI_e + (V(t_0) - E_L + RI_e) \exp^{\frac{-(t-t_0)}{\tau_m}}. \tag{2}$$

Since I_e is a constant in a little time period of $\mathrm{d}t$, the membrane potential updating process can be expressed in (3).

$$V(t+\mathrm{d}t) = V(t) + (V(t_0) - E_L + RI_e) \exp^{\frac{-(t-t_0)}{\tau_m}} \times (\exp^{\frac{-\mathrm{d}t}{\tau_m}} - 1). \tag{3}$$

Using ΔV to replace the long string of symbols where ΔV symbolizes $(V(t_0) - E_L + RI_e) \exp^{\frac{-(t-t_0)}{\tau_m}} (\exp^{\frac{-\mathrm{d}t}{\tau_m}} - 1)$, (3) turns to (4).

$$V(t+\mathrm{d}t) = V(t) + \Delta V. \tag{4}$$

The updating procedure of the membrane potential can be treated as a simple adding and accumulating process. The membrane potential of the next cycle is the membrane potential of the current cycle, adding with the increment of membrane potential. With (4), the neuron can accomplish the procedure of membrane potential updating. In our scheme, the membrane potential increment equals the dot product of spikes and weights. Here, we use a fixpoint scheme of 32 bits to express the membrane potential and the weight and 1-bit scheme to express the spike. ΔV equals $\sum_{i=0}^{n} S_i^l W_i^l$ in value.

3.3 Systolic Array

The theory of systolic array was proposed to improve the throughput of data with the limited bandwidth of storage [29]. It typically uses a systolic array

to replace the simple PE (processing element). Since the systolic array consists of a series of PEs which are organized regularly, we can control the data flows in the systolic array accurately. We can also get an acceleration of data processing with the help of multi-PE. Unlike the technology of pipeline, the directions of data flow in the systolic array are different, and the PEs in the different rows or columns share the same data flows. In this way, the systolic array reuses the data flow so that it can reduce the unnecessary data moving. We can treat the systolic array as a combination of pipeline and SIMD (single instruction multiple data).

There are many variants of the systolic array such as the 1-D systolic array, the 2-D systolic array, and so on [30, 31]. Since the 2-D systolic array is a natural match to the CNN for the character of the convolution operation, we generally use the 2-D systolic array to accomplish the task of convolution. In our scheme, we introduce a 2-D systolic array to speed up the procedure of membrane potential increment computation, which equals $\sum_{i=0}^{n} S_i^l W_i^l$. We implement the dot product of spikes and weights with the help of PEs. The PEs in the same row share the same data flow of spikes, and the PEs in the same column share the same data flow of weights.

3.4 Spiking Convolutional Neural Network

So far, the task of training SNNs with pure STDP (Spiking Timing Dependent Plasticity) algorithm is inefficient. The classifying accuracy of STDP-trained SNN is lower than that of DNN, which has the same topology. To solve these problems, Diehl *et al.* [6] proposed a novel method which can quickly transform a trained DNN into the corresponding SNN (called SCNN). The classifying accuracy of the SNN is almost as same as the original DNN. With the development in DNNs, we can train DNNs in many ways and get a considerable classifying accuracy. That means we can make the task of training the SNN more efficient and take advantage of the SNN on the field of power consumption.

The conversion from the DNN to the SNN is an equivalent replacement. We typically implement SNN based on DNN, which means DNN shares the same topology with the corresponding SNN. Since the spikes cannot express the number between "0" and "1", the average pooling process of SNN cannot be realized. We replace pooling layers with max-pooling layers. We use the method of random sampling, which follows the Poisson distribution to make a data conversion. We convert all the input pixels into the corresponding sequence of spikes. All these are accomplished by a software program. Here, we use "1" to denote the spike and "0" to no spike. We use a fixpoint scheme of 32 bits to express the weight. All the spikes and weights are processed in the same way as the corresponding DNN operations.

4 SIES Architecture

Fig.1 shows all the nine modules of SIES. They are the spikes buffer (IB), the weights buffer (WB), the systolic controller (systolic_ctrl), the array of PEs (PEs), the membrane potential updating and spike generating module (MU&SG), the spikes accumulating and max-pooling module (SA&MP), the results buffer (RB), the bypass logic, and the post controller (post_ctrl). To save the time of spikes preparing, we design the module of data setup which can arrange the spikes in the right order to fulfill the timing requirement of the SIES.

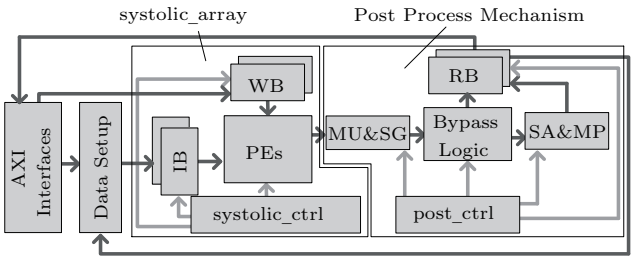

Fig.1. All the modules in the SIES. It also shows the directions where the internal data flows.

These modules are divided into two parts. The first part is used to compute the membrane potential increments which we call systolic_array. The second part is used to accomplish the tasks of membrane potential updating and max-pooling, which we call it the post process mechanism. The first part consists of four modules (i.e., the spikes buffer, the weights buffer, the systolic controller, and the array of PEs). The second part consists of the rest modules (i.e., the membrane potential updating and spike generating module, the spikes accumulating and the max-pooling module, the results buffer, the bypass logic, and the post controller). They work together to accomplish tasks such as membrane potential updating, spiking generating, and max-pooling. Here, the process of max-pooling is a free choice based on different SNN architectures (shown in Fig.2). We use the bypass logic to control the direction of the internal dataflow.

We use IB to store the input spikes and WB to store the weights. The array of PEs fetches the spikes and

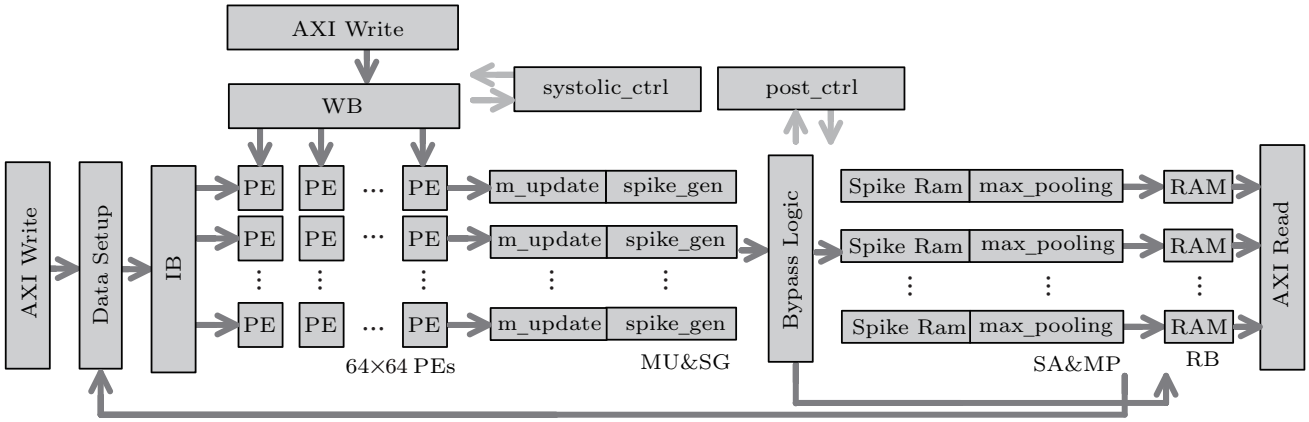

Fig.2. Architecture of the SIES.

the weights in pair and computes the intermediate results of membrane potential increments. All the intermediate results of membrane potential increment are accumulated in the corresponding PEs. In the systolic controller, we use a clock counter to record the clock cycles of computation. When all the PEs finish the computation, we pop the potential increments to the post process mechanism. We use MU&SG to accomplish the task of membrane updating and spike generating. Then, we accumulate the spikes belonging to the same neuron in the SA&MP, and process all the spikes in batch. Note, the process of max-pooling is optional based on the SNN. The final results are stored in the results buffer and then passed to the host.

For communicating with the host processor, we also implement an AXI (advanced extensible interface) for data writing and two for reading. Since one PE corresponds to one hardware neuron, we can simulate 4 096 biological neurons once with the scale of 64×64 PEs. Note, every 64 neurons share the 64 synapses and the bit width we use to express the membrane potential and weights is 32 bits in fixed point.

5 Modules Design

5.1 PE Array

The PE of SIES is responsible for the computation of membrane potential increment. One PE corresponds to one neuron in SNN (shown in Fig.3). It receives spikes and weights from the former PE and passes them to the next PE directly. It generally has four inputs, i.e., pre_spike, pre_res, pre_weight, and switch_in. It has three outputs (i.e., spike, weight, and res). We also implement a series of internal registers to buffer the intermediate result of membrane potential increment.

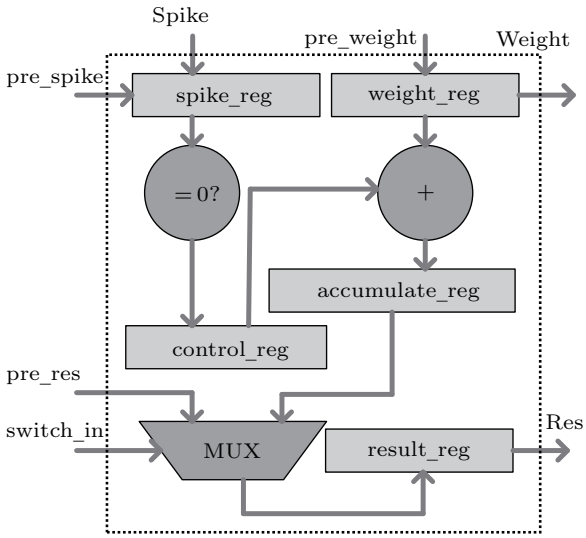

Fig.3. Architecture of single processing element. Res: Result.

The spikes and weights are used to participate in the operation of the dot product. Since the spike is logical "1" or "0", we implement the multiplication with a selector and an adder. As you see, if the spike denoting "1" is coming, the weight which the PE received is accumulated and added to the history membrane potential. If the spike denoting "0" is coming, the weight is dumped. Each step of the dot product needs one clock cycle. Here, though we dump the weight corresponding to the spike of "0", the PE still need to wait for one clock cycle for the concerning of timing alignment. The PE computes the membrane potential increment with the help of weights accumulating. When the PE collects all the weights of the corresponding pixel, it works out the result of the dot product (i.e., the value of $\sum_{i=0}^{n} S_i^l W_i^l$).

Since we need a series of spikes and weights to work out one dot product, we accumulate all the intermediate

results in the local regs of the corresponding PE. When the PE gets all the spikes and weights, it works out the final membrane increment. We use a control wire called switch_in to decide when to pass the final result value. The switch_in is generated by systolic_ctrl, where we count the clock cycles with an internal counter. If switch_in is set to "0", the PE passes the intermediate accumulated result to the next PE at each clock cycle, and this intermediate result is abandoned at the last PE of this row. If switch_in is set up to "1", the current accumulated result value is passed, and we store it at the last PE of this row.

In SIES, the PEs are organized in a systolic array. The PEs in the same row share the same spike trains, and these spikes are passed from the former PE to the next PE one by one. The PEs in the same column share the same weights, and these weights are passed in the same way as the spike trains. We implemented a time counter in the systolic_ctrl module to count the clock cycles. With this counter, the systolic_ctrl module can correctly set up the switch_in of each row. The intermediate and final computation results are passed in the same channel, and they are passed one by one. We count the clock cycles to decide when to store the final computation results and get them at the last PE of each row.

5.2 Membrane Potential Updating and Spike Generating Module

This module is responsible for membrane potential updating and spike behavior simulating (shown in Fig.4). The procedure of membrane potential updating can be expressed in two steps. One step is to work out the membrane potential increment of the current neuron, which is processed by the systolic_array part. It can be expressed in (5). The parameter of m denotes the membrane potential, and l denotes the neuron number of the current layer. We use N to denote the total neuron number and w to denote the weight. Each PE corresponds to one neuron. When all the PEs finish the computation, the systolic_array part pops the membrane potential increments to this module. Each row of PEs corresponds to one MU&SG module.

$$\Delta m_j^l = \sum_{i=1}^{N^{l-1}} (w_{ij}^l(t) spike_i^{l-1}(t)). \tag{5}$$

The other step is to update the membrane potential, which means adding the membrane potential increment to the accumulated membrane potential in history. We implement a series of on-chip ram banks to store the accumulated membrane potentials. Every time we get a new membrane at the last PE of each row, we add it to the corresponding accumulated membrane potential and work out a result called the added result. Then, we store the new membrane potential into the corresponding bank of ram. In each clock cycle, this module updates one membrane potential of the corresponding neuron. We update all the neuron one by one. Note, if the added result is over the threshold, the new membrane potential is reset to a fixed value called $RESET\_VALUE$. Otherwise, we store it into a bank of RAM directly. It can be expressed with Algorithm 1.

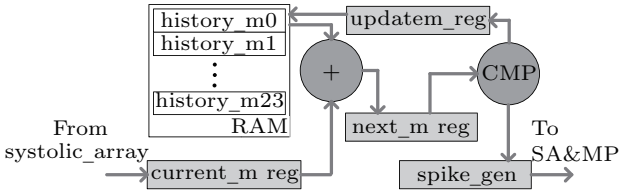

Fig.4. Directions of internal data flowed and the procedure of membrane potential updating where m denotes the membrane potential.

Algorithm 1. Membrane-Potential-Updating

Initialization: $m_j^l = 0; T = Thr;$
$\Delta m_j^l = 0; R = RESET\_VALUE;$
While $j \leftarrow 0$ to l
  do $SUM(m_j^l) \leftarrow m_{j-1}^l + \Delta m_j^l$
    ▷ Working out the added result.
    If $SUM(m_j^l) \geqslant T$
      $m_j^l \leftarrow m_{j-1}^l + \Delta m_j^l$
    If $SUM(m_j^l) < T$
      $m_j^l \leftarrow R$
    ▷ Storing the updated membrane potential to the corresponding bank of RAM.
  $j \leftarrow j + 1$

The procedure of neuronal spiking can be treated as a process of membrane comparison. If the added result is over the threshold, the module works out a spike denoting "1". Otherwise, it works out a spike of "0". It can be expressed with Algorithm 2, where Thr denotes the threshold.

Algorithm 2. Spike-Generating

Initialization: $spike_j^l = 0; T = Thr;$
While $j \leftarrow 0$ to l
  do ▷$SUM(m_j^l)$ comes from Algorithm 1.
    If $SUM(m_j^l) \geqslant T$
      $spike_j^l \leftarrow 1$
    If $SUM(m_j^l) < T$
      $spike_j^l \leftarrow 0$
    ▷ Passing the generated spike to the module of SA&MP.
  $j \leftarrow j + 1$

5.3 Spikes Accumulating and Max-Pooling Module

This module is designed for spikes accumulating and max-pooling (shown in Fig.5). In the procedure of SNN simulation, we often need to simulate the same neurons with multi-cycle, which means we need to repeat the procedure of membrane potential accumulating and updating with the same neurons. If we do not accumulate the spikes, we need to store the membrane potentials every time when we finish a simulation cycle. When we start a new simulation cycle of the same neurons, we should transform these pass membrane potentials to SIES. We need to repeatedly transport the membrane potentials between the host and SIES. For the reason of reducing additional data moving, we design this module to accumulate a certain number of spikes so that we can efficiently accomplish the max-pooling operation. We can accumulate all the spikes of the same neurons with multi cycles and process them in batch. We can simulate the same neurons in the same SNN layer with multi-cycle and store all the spikes in this module. When we finish the simulation of the same neurons, we release the banks of RAM so that the SIES can start a new simulation cycle. With this method, we can save both the time of membrane potentials transformation and the resources of RAM.

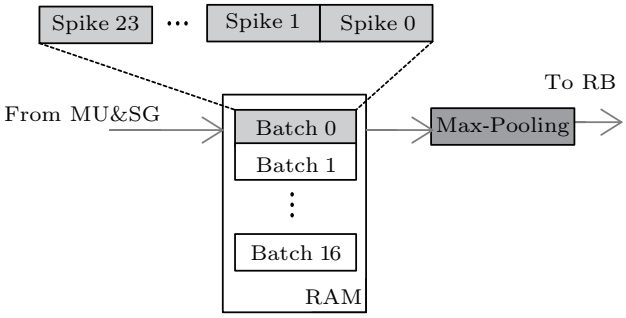

Fig.5. Procedure of spikes accumulating where the batch denotes that all the data of the same neurons at the current time step have been processed.

Since the spikes in SNN are a series of "0" or "1", we cannot express the result of the average-pooling operation. It means there is only the max-pooling operation in SNN. The scheme we adopt for the max-pooling operation is a logical operation of OR. If there is one spike denoting "1" in the domain of max-pooling perceived, this module works out a spike denoting "1". If there are not any spikes of "1", this module works out a spike of "0". Note, this module is optional based on SNN.

5.4 On-Chip Buffers

The on-chip buffers contain three parts (i.e., the spikes buffer, the weights buffer, and the results buffer). To hide the time of input data preparing, we implement the input data buffers (i.e., the spikes buffer and the weights buffer) in double. We can reduce the waiting time of input data preparing with the technology of Ping-Pong. Each time, we prepare the data of current simulation cycle and start up SIES. Then, we fill the data of the next simulation cycle in the copy of input buffers. Note, SIES is processing the data of the current simulation cycle at the same time. When we process the data of the next simulation cycle, we fill new data in the former input buffers. In this manner, we can significantly reduce the waiting time of input data preparing and improve both the efficiency of SNN simulating and the utilization rate of hardware resources.

5.5 Internal Controllers

The internal controllers consist of two parts (i.e., the systolic controller and the post controller). The systolic controller module controls the behaviors of the systolic_array part, and the post controller module plays the same role in the post process mechanism part. They both implement internal counters to count the clock cycles. Each counter shares the same value. In this way, we can control the simulating procedure accurately at every clock cycle.

6 Data Setup

The data setup module is designed for the convolution layer on SIES. It can change the order of spikes to fit the computation timing requirement of the systolic array. It means reordering the spikes of the input feature map (from the host processor or other accelerators) into an appropriate order so that each spike can compute with the corresponding weight correctly. Although we can do the same thing with a software program, we need a hardware data setup mechanism for the reason of efficiency. If we use the software program to reorder all the spikes, we should transform the spikes back to the host. When the software program finishes the reordering process, we need to transform the ordered spikes back to SIES. That will lead to a series of additional data moving between the host and SIES. What is more, if we use a large number of SIESs to build a big system, we need to reorder the output spikes of one SIES to fit the need of the next SIES. It

will significantly increase the processing time using the software on the host. Therefore, we need a hardware module to handle the task of spikes reordering. Concerning the hardware design, we have three choices (i.e., indirect setup, direct setup, and multi-lane setup).

6.1 Indirect Setup Strategy

The first one is called Indirect Setup strategy (shown in Fig.6). First, we store the input feature map of spikes in a RAM. Then, we use an address generator to generate the fetching addresses. The buffer fetches the spikes in sequence based on the fetching addresses. We can reorder the spikes with the help of ordered addresses sequence. It uses a few of hardware resources, but its efficiency is restricted by the reading channels of RAMs. Since most of the RAMs only have two reading channels, we will never get the full speed of spikes reordering. On the other hand, if we implement a specialized RAM with multi-reading channels, we need to implement the corresponding address generator for each of the reading channels. It leads to a significant waste of hardware resource. What is more, concerning the convolution algorithm, we need to fetch the spike with the same address repeatedly, which will aggravate the problem of reordering inefficiency.

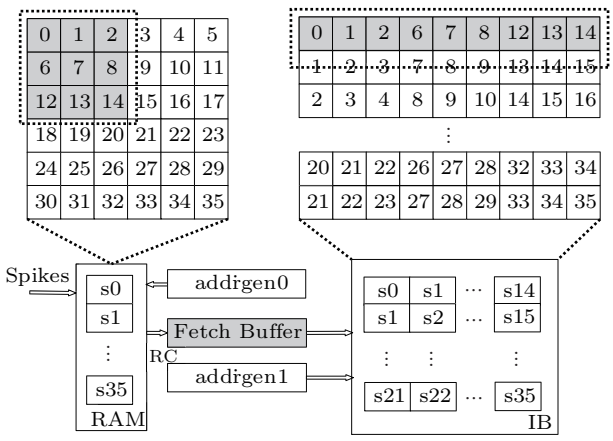

Fig. 6. Implementation details of the indirect setup strategy where the RC symbolizes the read channel.

6.2 Direct Setup Strategy

The second way to reorder the spikes is a direct setup strategy (shown in Fig.7). The mechanism receives the spikes one by one and reorders them with a series of buffers. The number of buffers equals the kernel size of the current convolution layer. Each column of the buffer banks denotes a complete convolution kernel. Every time we get a spike, we store this spike in the corresponding banks of the buffers. The banks where this spike will be stored are decided by the corresponding arranging information called pre_arg. Note, this arranging information is calculated by a software program in advance and stored in the on-chip RAM. The buffer fetches the arranging information and the spike in pair. In this way, we only need to fetch the same spike once and reorder it as soon as the buffer fetches it. The buffer stores the spike in the proper banks of different convolution kernels. With this strategy, we can improve the efficiency of spikes reordering. We will never be restricted by the reading channels of RAM.

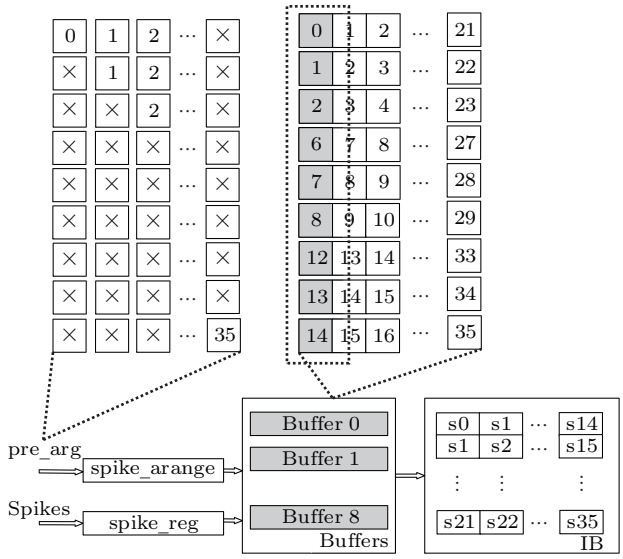

Fig.7. Implementation details of the direct setup strategy. The symbol of × in pre_arg denotes not to store the spike in the corresponding buffer bank.

6.3 Multi-Lane Setup Strategy

The above two strategies both work in a serial way. It means we need to fill in the spikes one by one. If we try to increase the speed of spikes reordering, we are limited by the pace of the spikes filled in. When we reorder a big input feature map, we should wait a long time to prepare the input spikes. Facing this, we propose the third strategy. It is a data reusing scheme called Multi-Lane Setup strategy (shown in Fig.8). Note, the data-parallel lanes [37] which are used for the zero-skipping computation, are used for data setup which reorders spikes in the right order to fulfill the computation timing of SIES. The procedure is as follows. First, we divide the input spikes into some smaller overlapping sequences. We combine all the convolution kernels in the same row of input feature map into one group. We process

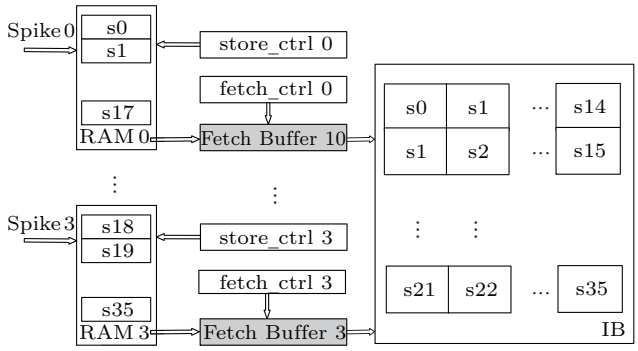

Fig.8. Implementation details of the multi-lane setup strategy.

them with the same series of hardware units (called hardware lane). We use a hardware lane to process a part of the origin spikes sequence. Since there are many overlapping pixels between different convolutional kernels, we can reuse these pixels to reduce the fetching operation of spikes. We only need to implement a series of small fetching buffers. The bank number of each buffer equals the size of the convolution kernel. Then, we fetch the spikes one by one, and all the spikes in the local RAM are fetched once. When the first convolution kernel of each hardware lane finishes buffering, we put spikes into the IB. After that, the buffer only fetches the new spikes. The buffer reuses the spikes of the former convolutional kernel to find the rest convolution kernels. In each clock cycle, the buffer fetches one spike. Fig.9(a) shows the locations of spikes in the input feature map. A series of spikes found one convolution kernel. Fig.9(b) shows the status of the fetch buffer at different clock cycles. We fill the fetching buffer with spikes and send the spikes to the IB as soon as all the spikes of the same convolutional kernel are reached. The color of grey denotes the spikes which can be reused in the fetching buffer. The buffer reuses these spikes to find the convolutional kernel with internal data moving. The array line shows the direction of internal data moving.

Since we should store all the smaller spikes sequences with on-chip RAM, this strategy will waste many resources of RAM. However, we can process all the smaller spikes sequences in parallel. That is a significant advantage in the field of reordering speed. Besides, it is easy to control the behavior of the hardware setup mechanism for the regular pattern of spikes moving in the fetch buffer. Compared with the two schemes we propose before, it has the potential for accelerating. It can significantly reduce the processing time of reordering procedure. It shows a significant advantage of efficiency among all the schemes we propose, especially for the big input feature map.

7 Working Procedure of SIES

We simulate the SNN layers in a sequence which means we simulate the layers one by one. We need to simulate the neurons in the same layer with multi simulation cycles if the neuron number of the current layer is over 4 096. We prepare the input feature map of the current simulating SNN layer and reorder the spikes ac-

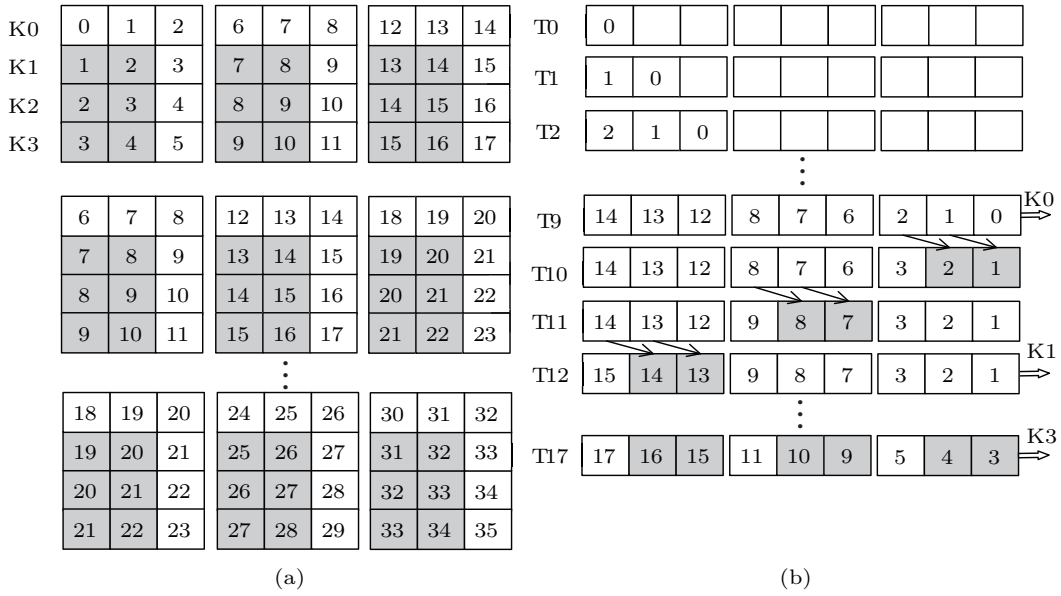

Fig.9. Principle of data reusing in multilane strategy where K symbolizes the kernel and T denotes in the cycle. (a) Spikes needed for convolution kernels. (b)Status of the fetch buffer.

cording to the convolution algorithm. Note, the original order of spikes in the input feature map is in a sequence which we call a pure feature map and the spikes do not order in a convolution way which the SNN hardware core processing needs. Thus we need a mechanism to reorder the spikes (shown in Fig.10). We typically use software to accomplish this task.

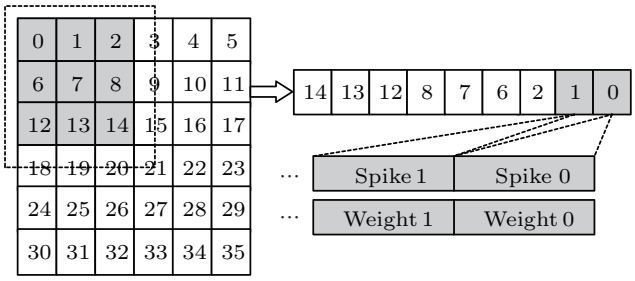

Fig.10. Procedure of data reordering. Each digital symbolizes a biological neuron which we should simulate in the layer of SNN. The size of input feature map we use here is 6×6.

The processing procedure is as follows. We store the sequences of spikes and weights into the corresponding input buffers and start up SIES (shown in Fig.11). We fill the spikes and weights into SIES one by one, and each of the spikes and weights differs one cycle. Note, we use 32 bits to denote one weight and 1 bit to denote one spike. Since the weights can be reused in multi simulation cycles while simulating the neurons in the same layer of SNN, we only need to fill the weights into the hardware accelerator once. In contrast, the spikes need to fill in every time we start a new simulation cycle. After a series of processing, SIES finishes the simulation and stores all the results (i.e., store the output feature map into the buffers).

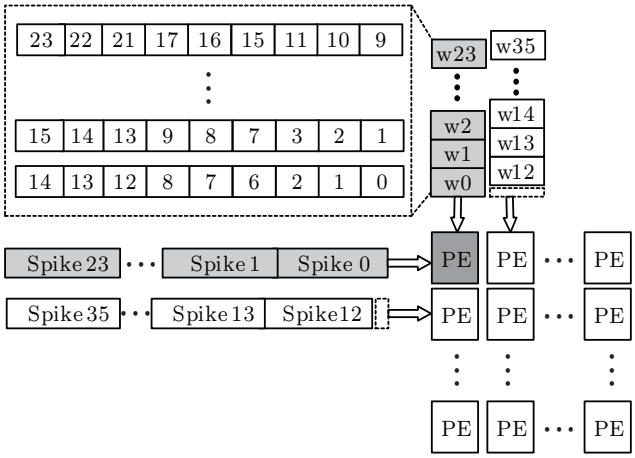

Fig.11. Results of data reordering and the timing arrangement of the systolic array computation.

8 Experiment and Analysis

8.1 Experimental Setup

The SNN we tried to speed up is transformed from four networks, i.e., three models and VGG-16. The architecture of the three models is shown in Table 1. The architecture of VGG-16 is shown in Fig.12 [32]. These models were used to verify the function of SIES. They can prove the fact that SIES can support the simulation of different max-pooling layers and convolutional layers. The image recognition datasets we used were MNIST [40], SVHN [41], and CIFAR-10 [42]. The FPGA platform we used was XCVU440, and it can easily handle the task of large-scale hardware design. We used Vivado 2018.3① for synthesizing and used it to make a standard evaluation of power, hardware resources consumption, and timing. We designed SIES with 4 096 (64×64) PEs. We also synthesized SIES with Vivado and implemented it on the FPGA board.

Table 1. Architecture of the Three Models

Layer No.	Model 1	Model 2	Model 3
1	Conv(12c5)	Conv(32c5)	Conv(32c3)
2	MP	MP	Conv(32c3)
3	Conv(64c5)	Conv(64c5)	MP(p3s2)
4	MP	MP	Conv(64c3)
5	FC(10)	Conv(32c5)	Conv(32c3)
6		MP	MP(p3s2)
7		FC(100)	FC(512)
8		FC(10)	FC(10)

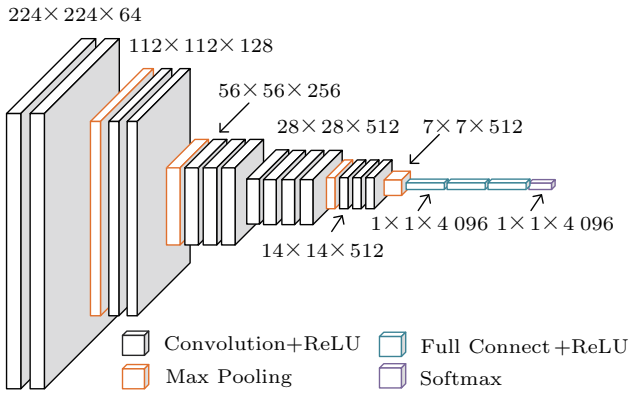

Fig.12. Architecture of VGG-16 [32].

We evaluated the RAM resources requirement of SIES with different scales PEs and show the hardware

①The Vivado Design Suite HLx editions supply design teams with the tools and methodology needed to leverage C-based design and optimized reuse, IP sub-system reuse, integration automation, and accelerated design closure.

resources consumption of the SIES with the scale of 64×64 PEs in detail. To verify the function of SIES, we chose a VGG-16 SNN to be the simulating target. We firstly implemented a software DNN with the same topology as the target SNN. Then, we transformed it into the corresponding SNN style with MATLAB. Then, we reorganized the input spikes and weights to fulfill the timing requirement of SIES and started up it to get the results. At last, we compared the speed between the software SNN running on the PC of Intel Core i7-7700 and the hardware simulation SNN running on the SIES.

For validation, we evaluated the hardware data setup module of convolution adopting three setup strategies we proposed with the software simulator and made a further comparison. We used input feature maps of different layers in VGG-16 to evaluate the RAM resource requirement of different scales of feature maps. We also used a smaller layer of 36 (6×6) for time consumption evaluation.

Though there are a few studies which use the same method as us, we compared SIES with the five famous neuromorphic schemes (i.e., TrueNorth [1], Neurogrid [2], BrainScaleS [3], SpiNNaker [4], Loihi [5]) on the field of neurons and synapses supported in the single hardware core. Concerning the fact that we implemented SIES on the platform of FPGA, we also compared the working frequency and throughput with a series of researches implemented on FPGA.

8.2 Experimental Result

8.2.1 Performance and Consumption

We compared the consumption of RAM resources with different parts of SIES (shown in Fig.13) and found that the parts which are related to the membrane increment computing and membrane potential accumulating waste most of the RAM resources. This is because we used 32 bits to symbolize one membrane potential while using 1 bit to denote one spike. We compared the RAM requirement of SIES with different scales of PEs in Table 2 and found the RAM requirement increased in an exponential form. This was caused by the array form of PEs, and every time we increased the PEs, the RAM resources increased in the same way. The hardware resources consumption is shown in Table 3. Note that since the spike was 1 or 0, the procedure of multiplication could be treated as a selection of weights. We implemented a multiplier with an adder and a selector, which means we did not need DSP.

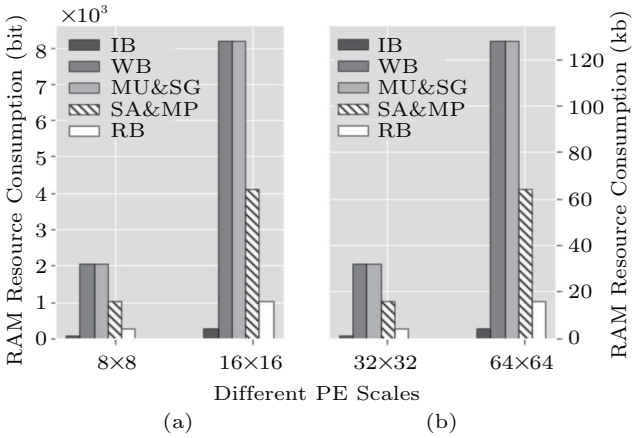

Fig.13. RAM resources requirements for different parts of the SIES with different scales of PEs.

Table 2. Total RAM Resource Requirements of SIES

Scale of PEs	RAM Size (Kb)
16 × 16	21.25
32 × 32	85.00
64 × 64	340.00
128 × 128	1 360.00
256 × 256	5 440.00

Note: The number of PEs is the result of the multiplication, for example 16 × 16 means 256 PEs.

Table 3. Consumption Ratio of Devices for SIES on FPGA XCVU440

Resources	Consumption	Utilization Rate (%)
LUT	302 797	11.95
FF	421 230	8.31
BRAM	192	7.62
I/O	646	44.37
BUFG	3	0.21

Note: The scale of PEs is 64 × 64.

We also compared the speed of data processing between the software-implemented SNN running on the host processor and the hardware-implemented SNN using SIES. The simulation target was a VGG-16 SNN. The number of the total operations of add which hardware platforms need to handle is up to 89×10^6. We employed a software written by Python to simulate all the add operations of VGG-16 SNN on the Intel® Core® i7-7700 PC and used another program to evaluate the total running time of VGG-16 SNN on SIES with the same scale of add operations. We finally found that the SIES worked better with the working frequency of 200 MHz and we could use the hardware accelerator to get a 670× speedup compared with the host processor. SIES could accomplish 4 096 × 2 operations (one multiply and one add) at each clock cycle, and got a peak performance of 1.562 5 TOPS. The error rates of

the DNNs and the corresponding SCNNs are shown in Table 4. As shown, the SCNNs we ran on SIES could get good accuracies, and the error rates of them were almost the same as the corresponding DNNs.

Table 4. Error Rates of Different Networks

Model	Dataset	DNN.ER (%)	SCNN.ER (%)
Model 1	MNIST	0.76	0.84
Model 2	SVHN	3.74	4.66
Model 3	CIFAR-10	14.07	18.20
VGG-16	CIFAR-10	8.30	8.54

Note: .ER denotes the error rate of the network. SCNN is converted from DNN. The error rates come from [38, 39].

8.2.2 *Evaluation of Different Setup Strategies*

For the comparison of different hardware data setup strategies, we evaluated the RAM requirement of reordering using different hardware data setup strategies with software which we wrote by ourselves according to the designs we proposed.

We first fixed the condition of input feature map with 224 × 224 (shown in Fig.14) and found that the mechanism employing the indirect setup strategy wasted the least RAM resources. The multi-lane setup strategy wasted less of RAM resources compared with that of the direct setup strategy.

Second, we fixed the condition of kernel size with 3×3 (shown in Fig.15) and found that the indirect setup strategy mechanism wasted the least RAM resources. The mechanism employing the multi-lane setup strategy still wasted less RAM resources compared with the direct setup strategy.

At last, we compared the cycles needed for reordering using different strategies with the same input feature map and kernel size (shown in Fig.16). We found that the mechanism using the multi-lane strategy worked the fastest. Though RAM resources wasted more compared with the indirect setup strategy, we still designed our hardware data setup mechanism using the multi-lane strategy concerning the factor of spikes preparing speed.

8.2.3 *Comparison with State of the Art*

In our scheme, the number of neurons supported corresponds to the number of PEs and every 64 hardware neurons share 64 synapses for the character of input spikes and weights sharing in the same row. With the comparison of neurons and synapses supported in single hardware core with different neuromorphic systems [1–5], we finally found that the SIES could support up to 4 000 neurons and 256 000 synapses and that was a considerable number though less than the Neurogrid scheme in Table 5. We also compared the neurons supported with [33], and the neurons supported by the SIES were twice the number of [33].

Since we implemented SIES on FPGA, We compared the working frequency with [12, 15, 34]. We found that the SIES we proposed worked better with the frequency of 200 MHz while [12] ran with the frequency of

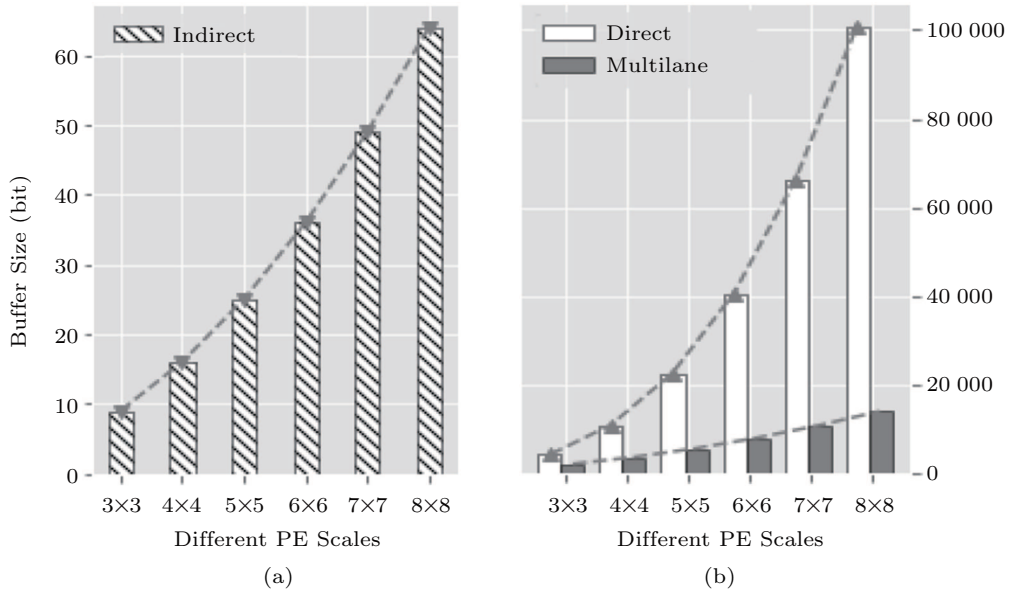

Fig.14. Trends of RAM requirements with different kernel sizes using different hardware data setup strategies under the condition of using the same input feature map. The input feature map we used is the first layer of VGG-16 which has 50 176 (224 × 224) input neurons. (a) RAM requirements trend of indirect setup strategy. (b) RAM requirements of direct and multilane setup strategies.

135 MHz, [34] ran with 150 MHz and [15] ran with 100 MHz. We also compared the throughput with [35, 36]. As we know, SIES can reach a peak performance of 1.562 5 TOPS. We finally found SIES gets a 492x improvement compared with [35], and a 128x improvement compared with [36].

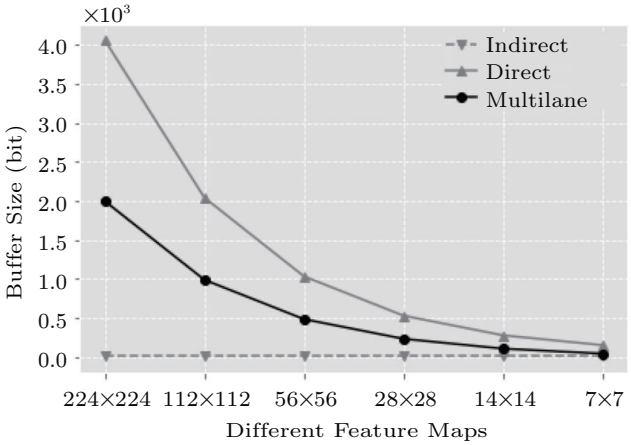

Fig.15. Trends of RAM requirements with different input feature maps using different hardware data setup strategies under the condition of using the same kernel size. The kernel size we used is the kernel size of VGG-16 which is 3×3 in general.

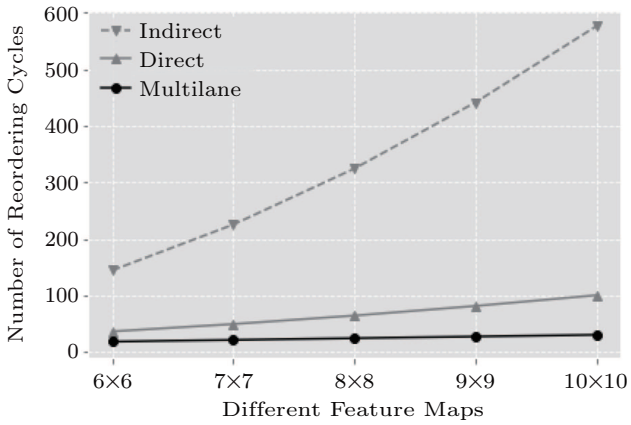

Fig.16. Cycles needed for reordering with the same input feature map using different hardware data setup strategies under the condition of using the same kernel size. For simplify, the input feature map we used here is 6×6 and the kernel size is 3×3.

Table 5. Neurons and Synapses Supported in the Single Hardware Core

Scheme	Number of Neurons ($\times 10^3$)	Number of Synapses ($\times 10^3$)
TrueNorth [1]	0.25	256
Neurogrid [2]	64.00	100 000
BrainScaleS [3]	0.50	128
SpiNNaker [4]	1.00	1 000
Loihi [5]	1.00	16
SIES	4.00	256

Note: The concept of single hardware core mentioned here is the basic processing units working independently in the system.

9 Conclusions

Inspired by the high performance of human brain, the research of SNN has become more and more popular, and the demand for speeding up the data processing with hardware accelerators is increasing. However, there are a few researches proposed in the area of SCNN accelerating. Concerning this, we proposed SIES which can speed up the inference of SCNN. To improve the on-chip data utilization of SIES, we introduced a systolic array to accelerate the computation of membrane potential increment. We implemented an optional max-pooling module which can significantly reduce the data moving between the host and the SIES. We also designed a hardware data setup mechanism so that we can reduce the waiting time of data preparing. Note, though the architecture of SIES is similar to that of the DNN accelerator, it can support the updating of membrane potential while the DNN accelerators cannot. It is designed for SNN. We finally implemented SIES on FPGA XCVU440. Experiments proved SIES can work with a frequency of 200 MHz and get a peak performance of 1.562 5 TOPS. In the future, we would like to implement SIES with ASIC and organize SIESs into a big array for the need of SNN simulation.

References

[1] Akopyan F, Sawada J, Cassidy A, Alvarez-Icaza R, Arthur J, Merolla P, Imam N, Nakamura Y, Datta P, Nam G J. TrueNorth: Design and tool flow of a 65mW 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015, 34(10): 1537-1557.

[2] Geddes J, Lloyd S, Simpson A C *et al.* NeuroGrid: Using grid technology to advance neuroscience. In *Proc. the 18th IEEE Symposium on Computer-Based Medical Systems*, June 2005, pp.570-572.

[3] Schemmel J, Grübl A, Hartmann S *et al.* Live demonstration: A scaled-down version of the BrainScaleS wafer-scale neuromorphic system. In *Proc. the 2012 IEEE International Symposium on Circuits Systems*, May 2012, p.702.

[4] Furber S B, Lester D R, Plana L A, Garside J D, Painkras E, Temple S, Brown A D. Overview of the spiNNaker system architecture. *IEEE Transactions on Computers*, 2013, 62(12): 2454-2467.

[5] Davies M, Jain S, Liao Y *et al.* Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 2018, 38(1): 82-99.

[6] Diehl P U, Neil D, Binas J, Cook M, Liu S C, Pfeiffer M. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Proc. the 2015 International Joint Conference on Neural Networks*, July 2015.

[7] Rueckauer B, Lungu I A, Hu Y, Pfeiffer M, Liu S C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 2017, 11: Article No. 682.

[8] Rueckauer B, Lungu L A, Hu Y H, Pfeiffer M. Theory and tools for the conversion of analog to spiking convolutional neural networks. arXiv: 1612.04052, 2016. https://arxiv.org/pdf/1612.04052.pdf, Nov. 2019.

[9] Du Z D, Fasthuber R, Chen T S, Ienne P, Li L, Luo T, Feng X B, Chen Y J, Temam O. ShiDianNao: Shifting vision processing closer to the sensor. In *Proc. the 42nd ACM/IEEE International Symposium on Computer Architecture*, June 2015, pp.92-104.

[10] Guan Y J, Yuan Z H, Sun G Y, Cong J. FPGA-based accelerator for long short-term memory recurrent neural networks. In *Proc. the 22nd Asia and South Pacific Design Automation Conference*, January 2017, pp.629-634.

[11] Zhou Y M, Jiang J F. An FPGA-based accelerator implementation for deep convolutional neural networks. In *Proc. the 4th International Conference on Computer Science Network Technology*, December 2016, pp.829-832.

[12] Neil D, Liu S C. Minitaur, an event-driven FPGA-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration Systems*, 2014, 22(12): 2621-2628.

[13] Wang R, Thakur C S, Cohen G, Hamilton T J, Tapson J, van Schaik A. Neuromorphic hardware architecture using the neural engineering framework for pattern recognition. *IEEE Trans. Biomed Circuits Syst.*, 2017, 11(3): 574-584.

[14] Glackin B, Mcginnity T M, Maguire L P, Wu Q X, Belatreche A. A novel approach for the implementation of large scale spiking neural networks on FPGA hardware. *In Lecture Notes in Computer Science 3512*, Cabestany J, Prieto A, Sandoral (eds.), Springer, 2005, pp.552-563.

[15] Cheung K, Schultz S R, Luk W. A large-scale spiking neural network accelerator for FPGA systems. In *Proc. the 22nd International Conference on Artificial Neural Networks*, September 2012, pp.113-130.

[16] Benton A L. Foundations of physiological psychology. *Neurology*, 1968, 18(6): 609-612.

[17] Hodgkin A L, Huxley A F, Katz B. Measurement of current-voltage relations in the membrane of the giant axon of *Loligo. J. Physiol.*, 1952, 116(4): 424-448.

[18] Izhikevich E M. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 2003, 14(6): 1569-1572.

[19] Brunel N, van Rossum M C W. Lapicque's 1907 paper: From frogs to integrate-and-fire. *Biological Cybernetics*, 2007, 97(5/6): 337-339.

[20] Liu Y H, Wang X J. Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron. *Journal of Computational Neuroscience*, 2001, 10(1): 25-45.

[21] Brette R, Gerstner W. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of Neurophysiology*, 2005, 94(5): 3637-3642.

[22] Paninski L, Pillow J W, Simoncelli E P. Maximum likelihood estimation of a stochastic integrate-and-fire neural encoding model. *Neural Computation*, 2014, 16(12): 2533-2561.

[23] Tsumoto K, Kitajima H, Yoshinaga T, Aihara K, Kawakami H. Bifurcations in Morris-Lecar neuron model. *Neurocomputing*, 2006, 69(4-6): 293-316.

[24] Linares-Barranco B, Sanchez-Sinencio E, Rodriguez-Vazquez A, Huertas J L. A CMOS implementation of the Fitzhugh-Nagumo neuron model. *IEEE Journal of Solid-State Circuits*, 1991, 26(7): 956-965.

[25] Yadav R N, Kalra P K, John J. Time series prediction with single multiplicative neuron model. *Applied Soft Computing*, 2007, 7(4): 1157-1163.

[26] Maguire L P, Mcginnity T M, Glackin B, Ghani A, Belatreche A, Harkin J. Challenges for large-scale implementations of spiking neural networks on FPGAs. *Neurocomputing*, 2007, 71(1): 13-29.

[27] Gerstner W, Kistler W. Spiking Neuron Models: Single Neurons, Populations, Plasticity (1st edition). Cambridge University Press, 2002.

[28] Gerstner W. Spiking neuron models. In *Encyclopedia of Neuroscience*, Squire L R (ed.), Academic Press, 2009, pp.277-280.

[29] Lopresti D P. P-NAC: A systolic array for comparing nucleic acid sequences. *Computer*, 1987, 20(7): 98-99.

[30] Samajdar A, Zhu Y, Whatmough P, Mattina M, Krishna T. SCALE-Sim: Systolic CNN accelerator simulator. *Distributed, Parallel, and Cluster Computing*, 2018.

[31] Jouppi N P, Young C, Patil N *et al.* In-datacenter performance analysis of a tensor processing unit. In *Proc. International Symposium on Computer Architecture*, May 2017.

[32] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In *Proc. the 3rd International Conference on Learning Representations*, May 2015, Article No. 4.

[33] Shen J C, Ma D, Gu Z H, Zhang M, Zhu X L, Xu X Q, Xu Q, Shen Y J, Pan G. Darwin: A neuromorphic hardware co-processor based on spiking neural networks. *SCIENCE CHINA Information Sciences*, 2016, 59(2): Article No. 023401.

[34] Kousanakis E, Dollas A, Sotiriades E *et al.* An architecture for the acceleration of a hybrid leaky integrate and fire SNN on the convey HC-2ex FPGA-based processor. In *Proc. the 25th IEEE International Symposium on Field-programmable Custom Computing Machines*, April 2017, pp.56-63.

[35] Fang H, Shrestha A, Ma D *et al.* Scalable NoC-based neuromorphic hardware learning and inference. arXiv:1810.09233, 2018. https://arxiv.org/pdf/1810.09233v1.pdf, Dec. 2019.

[36] Cheung K, Schultz S R, Luk W. NeuroFlow: A general purpose spiking neural network simulation platform using customizable processors. *Frontiers in Neuroscience*, 2015, 9: Article No. 516.

[37] Albericio J, Judd P, Hetherington T *et al.* Cnvlutin: Ineffectual-neuron-free deep neural network computing. *ACM SIGARCH Computer Architecture News*, 2016, 44(3): 1-13.

[38] Guo S, Wang L, Chen B, Dou Q. An overhead-free max-pooling method for SNN. *IEEE Embedded Systems Letters*. doi:10.1109/LES.2019.2919244.

[39] Sengupta A, Ye Y T, Wang R, Liu C, Roy K. Going deeper in spiking neural networks: VGG and residual architectures. arXiv:1802.02627, 2018. https://arxiv.org/pdf/1802.02627.pdf, Dec. 2019.

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

[41] Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng A Y. Reading digits in natural images with unsupervised feature learning. In *Proc. the NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, December 2011.

[42] Krizhevsky A. Learning multiple layers of features from tiny images. Technical Report, University of Toronto, 2009. http://www.cs.utoronto.ca/∼kriz/learning-features-2009-TR.pdf, Dec. 2019.

Shu-Quan Wang was a Master student in College of Computer Science and Technology, National University of Defense Technology, Changsha. He received his B.S. degree in computer science and technology in 2011. His research interests include computer architecture, asynchronous circuit, artificial intelligence and neuromorphic computation.

Lei Wang is an associate professor in the College of Computer Science and Technology, National University of Defense Technology, Changsha. She received her B.S. degree in 2000 and Ph.D. degree in 2006, both in computer science and technology, from National University of Defense Technology. Currently, her research interests include computer architecture, asynchronous circuit, artificial intelligence, and neuromorphic computation.

Yu Deng is an associate professor in the College of Computer Science and Technology, National University of Defense Technology, Changsha. He received her B.S. degree in 2000 and her Ph.D. degree in 2006, both in computer science and technology, from National University of Defense Technology, Changsha. Currently, his research interests include computer architecture, and asynchronous circuit.

Zhi-Jie Yang is a Master student in College of Computer Science and Technology, National University of Defense Technology, Changsha. He received his B.S. degree in 2014 and M.S. degree in 2016, both in computer science and technology. His research interests include computer architecture, asynchronous circuit, artificial intelligence and neuromorphic computation.

Sha-Sha Guo is a Ph.D. candidate in College of Computer Science and Technology, National University of Defense Technology, Changsha. She received her B.S. degree in computer science and technology, in 2017. Her research interests include computer architecture, neuromorphic computing and dynamic vision sensor signal processing.

Zi-Yang Kang is a Ph.D. candidate in College of Computer Science and Technology, National University of Defense Technology, Changsha. He received his B.S. degree in computer science and technology, in 2014. His research interests include computer architecture, neuromorphic computing and dynamic vision sensor signal processing.

Yu-Feng Guo is an associate professor in the College of Computer Science and Technology, National University of Defense Technology (NUDT), Changsha. He received his B.S. degree in 2000 and M.S. degree in 2002 and Ph.D. degree in 2011 from NUDT, all in computer science and technology. His research interests include computer architecture and high-performance microprocessor.

Wei-Xia Xu is a professor in the College of Computer Science and Technology, National University of Defense Technology (NUDT), Changsha. He received his B.S. degree from Nanjing University of Science and Technology in 1984 and his M.S. degree in 1993 and Ph.D. degree in 2018 from NUDT, all in computer science and technology. His research interests include computer architecture and high-performance microprocessor.