Neurocomputing 221 (2017) 146–158

Contents lists available at ScienceDirect

Neurocomputing

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

Energy efficient parallel neuromorphic architectures with approximate arithmetic on FPGA

Qian Wang[a,*], Youjie Li[a], Botang Shao[b], Siddhartha Dey[a], Peng Li[a]

[a] Department of Electrical and Computer Engineering, Texas A & M University, College Station, TX 77843 USA
[b] Freescale Semiconductor, Inc., Austin, TX 78735, USA

ARTICLE INFO

Communicated by Duan Shukai

Keywords:
Spiking neural network
Neuromorphic system
Approximate computing
Parallel architecture

ABSTRACT

In this paper, we present the parallel neuromorphic processor architectures for spiking neural networks on FPGA. The proposed architectures address several critical issues pertaining to efficient parallelization of the update of membrane potentials, on-chip storage of synaptic weights and integration of approximate arithmetic units. The trade-offs between throughput, hardware cost and power overheads for different configurations are thoroughly investigated. Notably, for the application of handwritten digit recognition, a promising training speedup of 13.5x and a recognition speedup of 25.8x are achieved by a parallel implementation whose degree of parallelism is 32. In spite of the 120 MHz operating frequency, the 32-way parallel hardware design demonstrates a 59.4x training speedup over the single-thread software program running on a 2.2 GHz general purpose CPU. Equally importantly, by leveraging the built-in resilience of the neuromorphic architecture we demonstrate the energy benefit resulted from the use of approximate arithmetic computation. Up to 20% improvement in energy consumption is achieved by integrating approximate multipliers into the system while maintaining almost the same level of recognition rate achieved using standard multipliers. To the best of our knowledge, it is the first time that the approximate computing and parallel processing are applied to FPGA based spiking neural networks. The influence of the parallel processing on the benefits of approximate computing is also discussed in detail.

1. Introduction

The human brain controls all our body movements, cognitive activities, emotions and other complex tasks. When it comes to complex tasks such as face recognition and language learning, a human brain can solve such problems with ease demonstrating much improved energy and space efficiency and show even better performance than supercomputers [1]. Brain-inspired computing has attracted much research interest, not only because of its application as a practical tool in areas such as pattern recognition, but also as a means of developing an understanding of mammalian brains and ultimately increasing our understanding of intelligence and consciousness.

Although most real world applications such as the processing of sensory inputs and pattern recognition can be realized by software models on Von Neumann machines, software simulation of complex biologically plausible models is intrinsically slow and may require tremendous energy consumption and space resources to solve these real-world problems. Brain-inspired neuromorphic hardware systems provide an appealing architectural solution to the above problems. They show good energy efficiency, potentially improved scalability and great suitability for pattern recognition problems. In addition, since one important property of neural networks is their parallel distributed nature, it is highly desirable to develop efficient parallel neuromorphic architectures for significantly acceleration. Meanwhile, the inherent error resilience and fault tolerance offered by brain-inspired architectures provide promising opportunities for leveraging approximate computing for additional energy and silicon area benefits. [2–10].

Traditionally, analog circuits are used to implement silicon neurons [11,12]. However, they are difficult to reconfigure and intrinsically sensitive to process, voltage and temperature (PVT) variations [13,14]. For example, a same analog circuit design may probably demonstrate require different performance under different environments. In addition, large-scale integration of spiking neurons is hindered by the use of area consuming capacitors to keep synaptic weights [15]. The impact of PVT variations on the performance of digital neuromorphic designs is thoroughly investigated by [16], which provides guidance on the design of robust digital spiking neural circuits.

FPGAs offer great flexibility and reconfigurability for fast prototyp-

* Corresponding author.
E-mail addresses: qwangku@tamu.edu (Q. Wang), lyj2013apply@tamu.edu (Y. Li), jackieshao2011@gmail.com (B. Shao), sidhart.de@email.tamu.edu (S. Dey), pli@tamu.edu (P. Li).

http://dx.doi.org/10.1016/j.neucom.2016.09.071
Received 15 January 2016; Received in revised form 11 June 2016; Accepted 2 September 2016
Available online 02 October 2016

ing and hardware acceleration of software algorithms. To facilitate the application of SNNs in embedded systems and develop processing acceleration for large data sets, there have been several attempts to implement software algorithms in FPGA. Numerous examples of SNNs have been implemented using FPGAs in the past [17–20], which show promising speedups compared with software programs running on general purpose CPUs. However, none of these works investigate the potential benefits of approximate computing and parallel processing in terms of energy consumption and efficiency of resource utilization. The main goal of this paper is to develop parallel digital neuromorphic architectures on FPGA and investigate the potential application of approximate arithmetic units to reduce hardware cost and power consumption.

The proposed architectures are demonstrated under the context of an FPGA based spiking neuromorphic learning system, which fully explores the parallelism in key processing steps. We also integrate a recent approximate Booth multiplier design [21] to replace the relatively bulky full precision multipliers, which contribute significantly to the area and energy estate of the overall system. Importantly, through the use of a real-life pattern recognition application, we show how such arithmetic units can be employed without incurring any significant loss of recognition performance for the end application. In return, the use of approximate computing offers noticeable energy and area benefits. Such reduction in energy dissipation and/or area overhead provides room for further throughput improvement via increased parallelism.

Realizing FPGA-based spiking neural networks (SNNs) entails addressing a number of critical issues pertaining to memory organization, parallel processing, hardware reuse for different operating modes and tradeoffs between throughput, area, and power overheads. The proposed neuromorphic system makes use of a large number of available block-RAMs for storing synaptic weights. To support parallel processing, multiple block-RAMs are instantiated in the system which allows multiple synaptic weights to be accessed simultaneously. We systematically demonstrate the tradeoffs between processing speed, power or energy and area overheads as a result of employment of varying levels of parallelisms and/or approximate multiplications.

The proposed neuromorphic processor is implemented on a Xilinx Virtex-6 FPGA. The handwritten digits from the MNIST dataset [23] are used to test the recognition performance of the system. The architectures with standard multipliers achieve a recognition rate of 89.1%, and those utilizing the approximate multipliers maintain an excellent recognition rate of 87.7%. The proposed spiking neural network involves 1591 neurons and 638,208 synapses, which shows comparable performance to a recent software reference [34] although our network has a smaller size. Energy consumption of the architecture without parallel processing is reduced by 20% when the approximate multipliers are used. A promising 13.5x training speedup and a 25.8x recognition speedup are achieved by the parallel architecture whose degree of parallelism is 32.

Algorithm 1. Pseudocode of the learning algorithm based on STDP.

```
Given an input training pattern (i.e. a digit image)
for t=1 to Iteration_num
/* Update membrane potential of each neuron */
  for i=1 to N
    $V_{mem}(i) = V_{mem}(i) + K_{SYN} \sum_{j=1}^{N} W(j, i)\cdot S(j)$
              $+K_{EXT}\cdot E(i) - V_{LEAK}$
  end for
/* Check the firing activity of each neuron */
  for i=1 to N
    if ($V_{mem}(i) \geq V_{Threshold}$)
        $S(i) = 1, \quad T_{fire}(i) = t, \quad V_{mem}(i) = V_{rest}$
    else
        $S(i) = 0$
    end if
  end for
/* Update synaptic weights with STDP rule */
  for i=$L_{first}$ to $L_{last}$
    if ($S(i)$== 1)
      for j=$M_{first}$ to $M_{last}$
        $\Delta T(j) = t - T_{fire}(j)$
        $A_+(j, i) = A_+(j, i)\cdot e^{(\frac{\Delta T(j)}{\tau 1})} + offset_1$
        $A_-(j, i) = A_-(j, i)\cdot e^{(\frac{\Delta T(j)}{\tau 2})} + offset_2$
        $\Delta W(j, i) = A_+(j, i) + A_-(j, i) + offset_3$
        $W(j, i) = W(j, i) + \Delta W(j, i)$
      end for
    end if
  end for
end for (sufficient iteration cycles)
return $W$
```

2. Spiking neural networks

The proposed spiking neural network is designed for character recognition where each training pattern (i.e. a letter or a digit) enters the input layer as encoded external input spikes. Fig. 1(a) shows one example of such spiking neural networks. This example involves 212 neurons. The neurons labeled from 1 to 196 are the excitatory neurons in the input layer, and those labeled from 197 to 205 are the excitatory neurons in the output layer. The neurons with labels from 206 to 212 are the inhibitory neurons, which introduce strong negative feedback to the excitatory neuron population, so as to realize the winner-take-all (WTA) mechanism in this network. As shown in Fig. 1(a), the synaptic weights involving inhibitory neurons are all fixed, while the feed-forward synapses connecting the input layer to the output layer are plastic. These plastic synapses are able to change their strength based on an adopted biologically-inspired STDP (Spike-timing-dependent plasticity) learning rule [24]. During the learning process, each synapse adjusts its synaptic weight based on the relative timing of spikes of the corresponding presynaptic and postsynpatic neurons. Fig. 1(b) uses a conceptual 212×212 crossbar array to illustrate the corresponding interconnections of this example. Each column in Fig. 1(b) represents the connections between a particular neuron and all its pre-synapses. For example, the neuron labeled 197 is connected to all the excitatory neurons in the input layer. Since the synaptic weights involving the inhibitory neurons are constants and there are no interconnections between neurons within the same layer, the only synapses to be updated are the feed-forward synapses which correspond to the shaded region in Fig. 1(b).

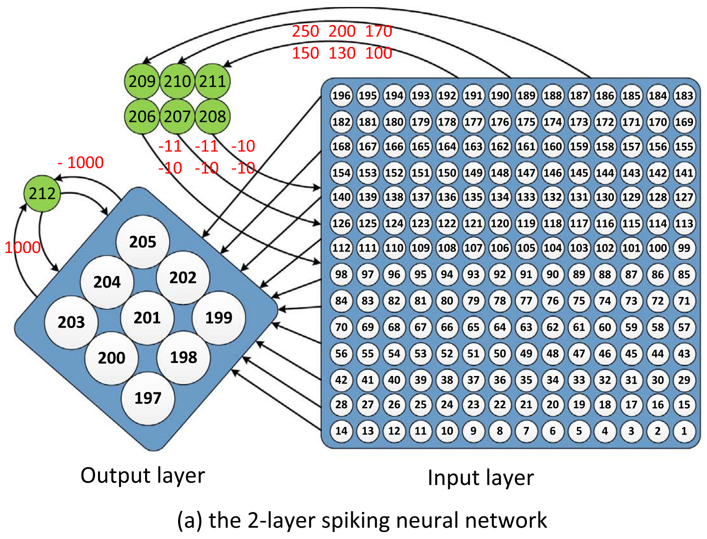
The neurons in this work are based on the widely adopted Leaky Integrate & Fire (LIF) neuron model [25]. The dynamics of the LIF neuron is described by

$$\tau_m \frac{dv(t)}{dt} = -v(t) + R\cdot I(t) \tag{1}$$

where $v(t)$ represents the membrane potential at time t, τ_m is the membrane time constant and R is the membrane resistance. This equation describes a simple resistor-capacitor (RC) circuit where the leakage term is due to the resistor and the integration of $I(t)$ is due to the capacitor that is parallel to the resistor. For the digital hardware implementation, the dynamics of the LIF neurons is simplified and digitized as the follows

$$v(t) = v(t-1) + \sum I_{pre} - v_{leak} \tag{2}$$

where $\sum I_{pre}$ represents the input stimulation from all its pre-synaptic



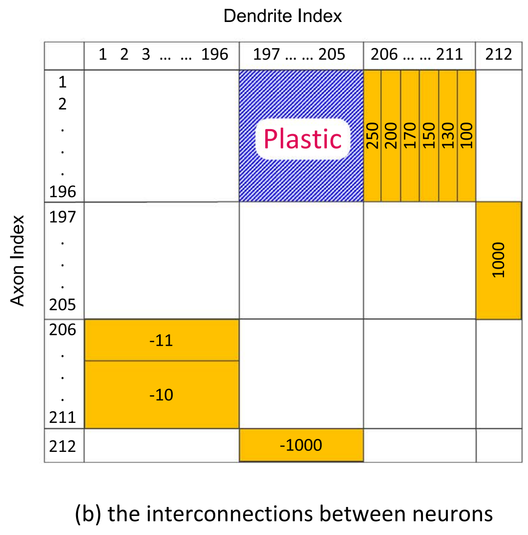


Fig. 1. (a) An exemplary spiking neural network for character recognition. There are 212 neurons in this example, and neurons labeled from 206 to 212 are the inhibitory neurons. (b) conceptual crossbar diagram of the synaptic connections. The synapses involving inhibitory neurons are constant while the feed-forward synapses are plastic.

neurons, and v_{leak} is a constant leakage value. Algorithm 1 shows the pseudo code of the SNN learning algorithm based on the STDP learning rule. V_{mem} is the membrane potential. W is the synaptic weight. E is the external input spike to each neuron, and S indicates if a neuron fires or not. N is the total number of neurons. L_{fisrt} and L_{last} are the indices of the first excitatory neuron and the last excitatory neuron in the output layer, respectively. M_{fisrt} and M_{last} are the indices of the first excitatory neuron and the last excitatory neuron in the input layer, respectively. For any general neural network topologies with N neurons, the time complexity of the computation in one training iteration is O(N^2). However, for the network topology illustrated in Fig. 1, the time complexity will be O(nm), where n and m represent the number of excitatory neurons for the input and output layers, respectively, since most synapses in such networks are the feed forward synapses between the two layers.

For each training pattern entering the input layer, the STDP learning process is performed for a certain number of iterations, which is the outer most loop of the pseudo code. In one run of this procedure, the membrane potential V_{mem} of each ith neuron is updated considering the spikes from its firing presynaptic neurons (i. e. by incrementing the membrane potential bya scaled version of W(j, i)×S(j), where $W(j, i)$ represents the weight of the synapse from the jth neuron to the ith neuron and S(j) is the flag indicating if a neuron fires or not.) and the external input spike (denoted by $E(i)$). There also exists a constant leakage for V_{mem}, which is denoted by V_{LEAK}. It should be noted that the amplitude of the external input spike K_{EXT} is a random number, which emulates the random injected currents to a biological neuron. Once the membrane potential of each neuron has been updated, it is compared with a threshold value $V_{threshold}$ to check the firing activity. If V_{mem} is higher than the threshold, the corresponding neuron fires and its firing flag S is set. Meanwhile, the corresponding firing time T_{fire} is stamped with the current biological time t (iteration index), and V_{mem} is reset to the resting potential V_{rest}. On the other hand, the firing flag S is reset if V_{mem} is below the threshold. During the above operation, the update of V_{mem}s can be parallelized in hardware implementations.

After that, the change of each synaptic weight is calculated following the STDP learning rule. When a particular neuron fires during the current iteration, all its pre-synaptic neurons are accessed to receive their most recent firing times. Then the change of a particular synaptic weight is calculated from the relative firing time difference between the post-synaptic and pre-synaptic firing events. According to the STDP learning rule, a smaller ΔT tends to result in larger change of the synaptic weight, accordingly, the parameters τ_1 and τ_2 are chosen to certain negative values. The synaptic parameters A_+ and A_- determine the maximum amounts of synaptic modification. After the synaptic weights are updated, a new iteration will start.

3. Serial baseline neuromorphic processor architecture

In this section, we describe in detail the proposed neuromorphic processor architecture for the 2-layer spiking neural network. A number of critical issues such as memory organization, efficient parallel processing and the application of approximate arithmetic units in system, are addressed.

The overall experimental platform of our neuromorphic system is shown in Fig. 2. The Matlab program on the PC converts the training patterns to spike sequences and sends them to a Xilinx ML605 evaluation board through a UART (universal asynchronous receiver/transmitter) cable. Once the training is finished, the results (i.e. output spikes and synaptic weights) are sent back to the PC through the same cable. The proposed FPGA-based neuromorphic processor is composed of three major components: Neuron Unit, LIF (Leaky-Integrate-and-Fire) Arithmetic Unit, and STDP Unit. The synaptic weights of the plastic synapses (i.e. the synapses from the input layer to the output layer) are stored in the block RAMs (BRAMs) on the FPGA chip. The access to these BRAMs is realized by a synapse Read/Write interface. The overall operations of the system are managed by a system controller through a clocking based synchronous control and the system operates in a synchronous manner as shown in Fig. 3. Each step corresponds to a biological time unit and consumes many hardware clock cycles. The control flow of the processor involves three processing stages, namely, the spike I/O stage, the neuron operation stage (NOS) and the learning operation stage (LOS). During the spike I/O stage, the processor communicates with the PC through a spike I/O buffer and the UART interface. During the NOS, the membrane potential of each neuron is calculated and the corresponding firing activity and firing time are recorded. Then the processing moves onto the LOS, and the synaptic weights are updated based on the STDP learning rule. The three stages are executed in a pipelined manner. The spike I/O and the LOS can work simultaneously because there is no data and control hazards between them.

Fig. 4 demonstrates the baseline architecture of the proposed neuromorphic processor. The synaptic parameters such as W, $A+$ and $A-$ are stored in the block RAMs. The synaptic weights are read out from the BRAM sequentially and the membrane potentials which

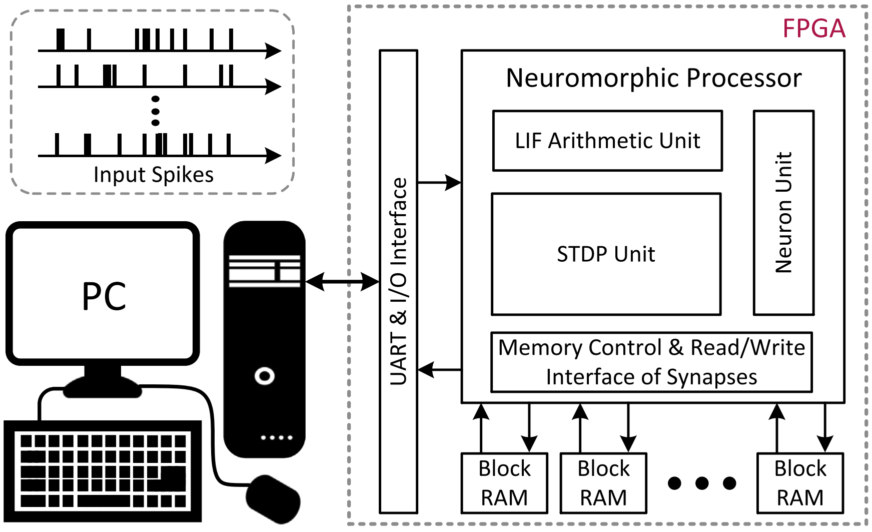


Fig. 2. Top-level schematic of the proposed neuromorphic processor running on Xilinx ML605 evaluation board, with the synaptic weights stored in block RAMs. The communication between PC and FPGA is realized by a UART cable.

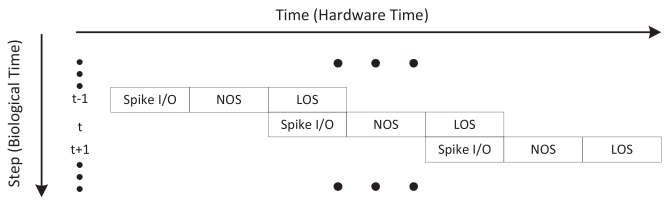
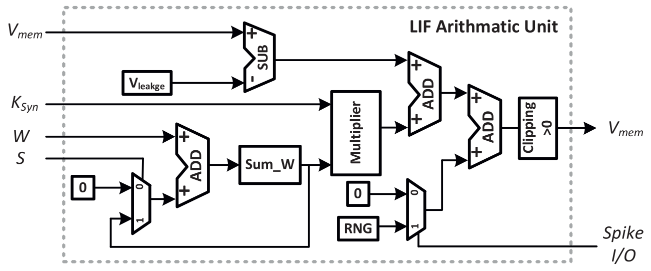


Fig. 3. Flow diagram of the digital neuromorphic processor. NOS represents the neuron operation stage and LOS represents the learning operation stage. The LOS is necessary for training, but not required for recognition.



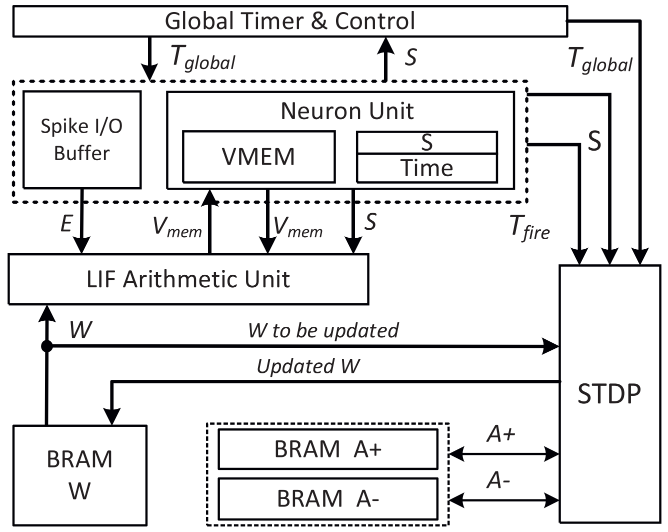


Fig. 4. The block diagram of the serial baseline architecture without parallel computing. The synaptic weights W's are stored in a single-port block RAM, and the synaptic parameters $A+$ and $A-$ are stored in another two block RAMs.

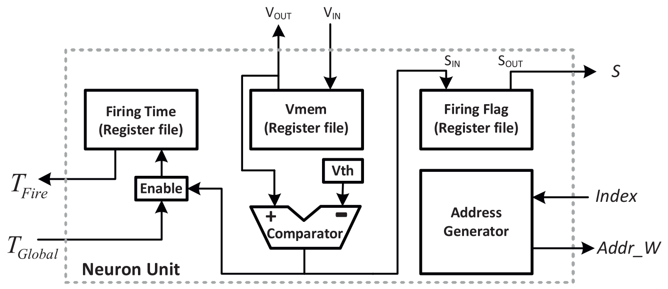


Fig. 5. The proposed LIF Arithmetic Unit (LAU) and Neuron Unit (NU). LAU is used to update the membrane potentials of all the neurons. NU is used to store the membrane potential, firing time and firing activity flag of each neuron. The synaptic weights are stored in the BRAM.

Table 1
Comparison of different multipliers in terms of hardware cost and delay. The precision of each multiplier is 16×16. In the Virtex-6 FPGA chip, each CLB consists of 2 slices and each slice contains 4 LUTs and 8FFs.

	Approximate	Standard	Xilinx IP	Built-in
Slice LUT	273	376	378	97
DSP48E1	0	0	0	1
Delay (ns)	7.537	7.258	8.538	8.314

are recorded by the Neuron Unit are updated one after another. Fig. 5 shows the design details of the Neuron Unit (NU) and the LIF Arithmetic Unit (LAU). The NU involves three important register files which store the membrane potentials (V_{mem}'s), the firing times (T_{fire}'s) and the firing activity flags (S's) of all the neurons. During the NOS, the LAU first reads out the V_{mem}'s and the S's from the NU, the synaptic weights from the BRAM and the external input spikes from the spike I/O buffer, and then writes the updated V_{mem} back to the NU. The calculation of $\sum W(j,\ i)\cdot S(j)$ as in the membrane potential update section of Table 1 takes many clock cycles to complete. And the time consumed by the NOS usually dominates the entire processing runtime. Once all the membrane potentials inside the NU are updated for the current iteration, the NU compares all the membrane potentials with $V_{threshold}$ to detect firing neurons and update the firing activity flags and firing times. T_{Global} is a signal representing the current biological time, which is generated by a global timer inside the top-level control logic. The amplitude of the external input spike is determined by a random number generator (RNG) based on Linear Feedback Shift

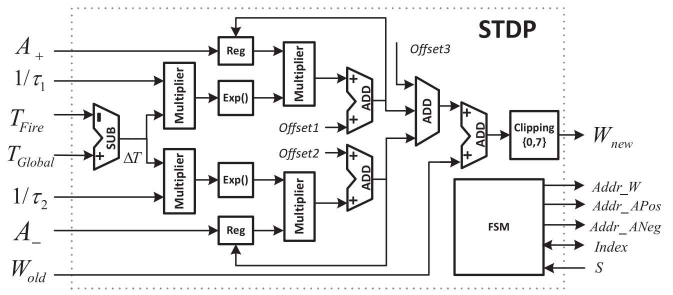


Fig. 6. The proposed STDP unit which is used to update the synaptic weights. T_{fire} and S are obtained from the neuron unit. W, A_+ and A_- are from the BRAMs.

Registers (LFSRs).

Fig. 6 shows the design details of the proposed STDP unit, which is used to update the synaptic weights based on the difference of firing times between the pre-synaptic neuron and the post-synaptic neuron. Assuming there are N_{output} neurons in the output layer and N_{input} neurons in the input layer, the total number of the plastic synapses to be updated is $N_{output} \times N_{input}$. Each plastic synapse is associated with two parameters A_+ and A_- which may depend on the current status of the synapse. And the change of the synaptic weight W is calculated with these two parameters during the LOS. The exponential function used to update $A+$ and $A-$ is realized by a pre-computed lookup table.

The proposed neuromorphic processor has two operating modes, namely, the training mode and the recognition mode. The recognition mode is much simpler than the training mode because the synaptic weights need not to be updated during recognition. The NU, the LAU and the BRAM for the synaptic weights are reused in the recognition mode, which leads to noticeable reduction of area overhead since no additional functional block is added.

4. Parallel architectures

4.1. Motivation for parallel architectures

As illustrated in Fig. 5, the calculation of $\sum W(j, i) \cdot S(j)$ (see the membrane update section of Table 1)is realized by an accumulator of W's controlled by S's. Therefore, the proposed architecture in Fig. 4, which is referred to as the serial baseline, takes N_{pre} cycles to update one V_{mem} if this particular neuron has N_{pre} pre-synapses. The V_{mem} of each neuron is calculated one after another, during which a LAU takes many clock cycles to accumulate the pre-synaptic weights. During the LOS, we only scan the excitatory neurons in the output layer and the update of the plastic pre-synapses is not performed unless the excitatory neuron in the output layer fires. If it does not fire, the update of pre-synaptic weights will be skipped. This approach enjoys a low hardware cost at the expense of processing speed due to lack of parallelism.

Storage of synaptic weights is an important issue for both the serial baseline architecture and several parallel architectures that will be discussed later. Block RAMs are based upon the embedded memory blocks of the FPGA chips. We take advantage of the fact that it is normally more efficient to implement memories on FPGAs using the embedded block RAMs, each of which can be quite large while supporting high-speed operations. Take an SNN with 800 output neurons and 784 input neurons as an example, there are 627,200 (800×784) variable weights associated with the plastic synapses but only 10 different constant numbers are used as the weights of the inhibitory synapses. Therefore, since the weights of all the inhibitory synapses are fixed and have limited number of values, these constant weights are integrated into the arithmetic logic circuits. So only the feed-forward synapses require a large storage and we choose to store them in the block RAMs.

To see why parallel architectures for the targeted spiking neural

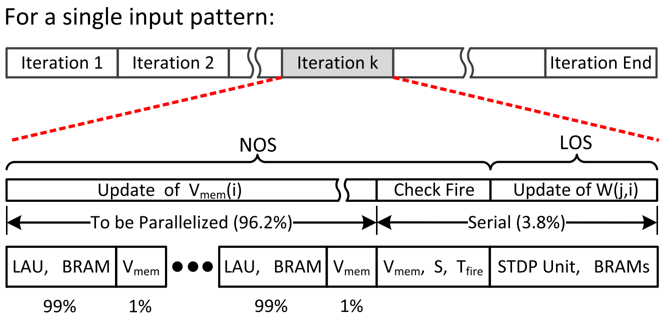


Fig. 7. The detailed timing diagram of the baseline design. A large number of biological time steps need to be processed for a single input pattern (i.e. a handwriting digit image) in the training phase. Each iteration is divided into NOS and LOS. The updating of membrane potentials during NOS is parallelized, which consumes 96.2% of the total runtime.

networks are desirable, we analyze temporal processing steps involved in the baseline serial architecture. Fig. 7 uses a detailed timing diagram to demonstrate the operations of the serial baseline architecture without parallel processing. The timing diagram is based on the functional simulation with Verilog HDL, which shall correlate well with the actual design running on the FPGA. The time utilization of each functional block in one biological time step is also illustrated in this figure. Thousands of biological time steps are required for the training of one input pattern, which corresponds to the outermost loop of the pseudo code in Table 1. As mentioned earlier, processing of each biological time step is divided into two stages, namely, the NOS and the LOS. According to Fig. 7, the LAU and the block RAM which stores the synaptic weights have the highest utilization, while other blocks consume a much less portion of the overall processing time. Because the update of synaptic weights is only performed for the firing post-synaptic neurons, the workload for LOS can be very small considering the low firing rate of the output neurons. Obviously, the runtime of the NOS is much longer than the LOS, and the runtime of updating the membrane potentials dominates the NOS runtime, so this work only focuses on the parallel readout of synaptic weights during the NOS. The updating of the synaptic weights during the LOS is still a sequential process.

4.2. Proposed parallel architectures and memory organization

We propose several parallel architectures. In Fig. 8, $W(j, i)$ represents the weight of the synapse from the jth neuron to the ith neuron. Assume that there are N input layer neurons and M output layer neurons. The neurons in the input layer are labeled from 1 to N, and the neurons in the output layer are labeled from $N+1$ to $N+M$. Fig. 8(a) shows a parallel architecture which supports K-way parallel processing during the NOS where K membrane potentials are updated simultaneously. The processing may take a large number of clock cycles to complete since for each update many pre-synaptic weights need to be read out in sequence. In this scheme, the same LAU design of Fig. 5 is used.

In addition, we explore another parallel scheme as shown in Fig. 8(b) which allows the calculation of $\sum W_{ji} \cdot Sj$ for one neuron to be completed in one clock cycle. And N block RAMs need to be instantiated to support the parallel synapse readout. According to Fig. 8(b), to update the V_{mem} of one particular neuron, the weights associated with all its pre-synapses are read out simultaneously and sent to a modified LAU.

In Table 1, the index of the current post-synapse neuron is denoted by i and the index of its pre-synapses is denoted by j. Therefore, to be convenient, the parallel scheme in Fig. 8(a) is referred to as the "Loop-I Parallelism (LIP)", and the scheme in Fig. 8(b) is referred to as the "Loop-J Parallelism (LJP)".

The architecture based on the LIP is illustrated by Fig. 9. The

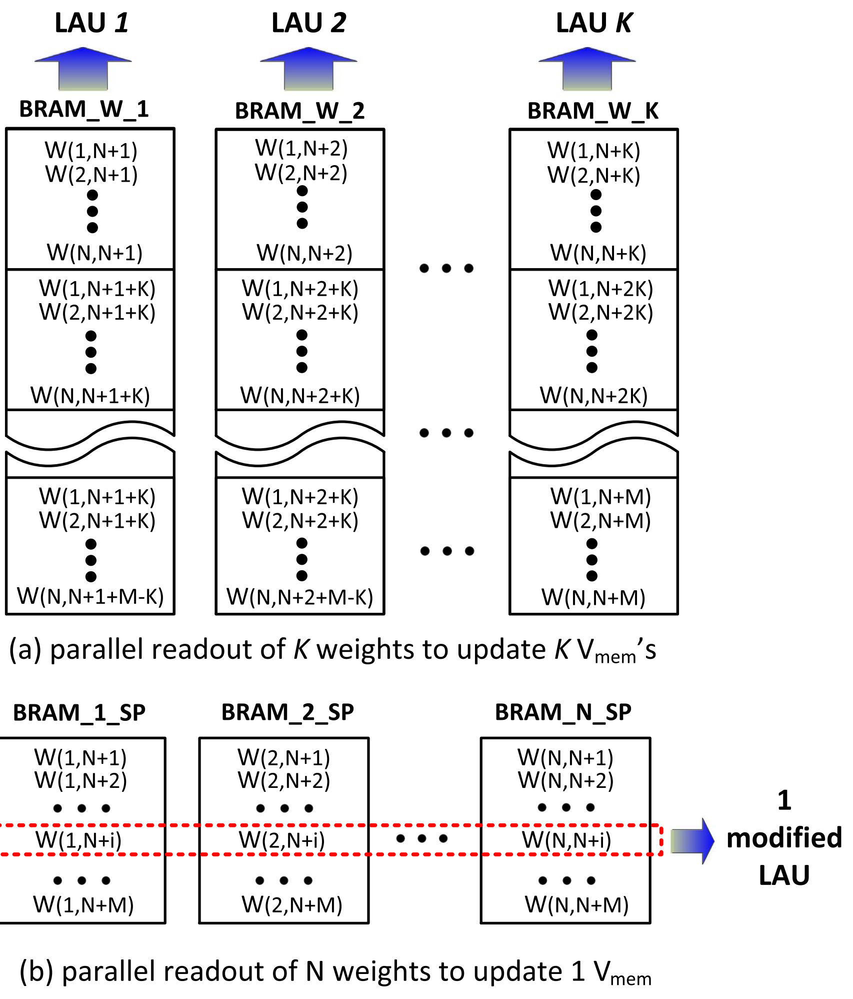

LAU 1
LAU 2
LAU K
BRAM_W_1
BRAM_W_2
BRAM_W_K
W(1,N+1) W(2,N+1) W(N,N+1)
W(1,N+2) W(2,N+2) W(N,N+2)
W(1,N+K) W(2,N+K) W(N,N+K)
W(1,N+1+K) W(2,N+1+K) W(N,N+1+K)
W(1,N+2+K) W(2,N+2+K) W(N,N+2+K)
W(1,N+2K) W(2,N+2K) W(N,N+2K)
W(1,N+1+K) W(2,N+1+K) W(N,N+1+M-K)
W(1,N+2+K) W(2,N+2+K) W(N,N+2+M-K)
W(1,N+M) W(2,N+M) W(N,N+M)
(a) parallel readout of K weights to update K Vmem's
BRAM_1_SP
BRAM_2_SP
BRAM_N_SP
W(1,N+1) W(1,N+2) W(1,N+i) W(1,N+M)
W(2,N+1) W(2,N+2) W(2,N+i) W(2,N+M)
W(N,N+1) W(N,N+2) W(N,N+i) W(N,N+M)
1 modified LAU
(b) parallel readout of N weights to update 1 Vmem

Fig. 8. Parallel processing schemes for *N* excitatory neurons in the input layer and *M* excitatory neurons in the output layer: (a) simultaneous updates of *K* membrane potentials with synaptic weights stored in *K* parallel block RAMs. (b) update of one V_{mem} by reading out the synaptic weights stored in *N* parallel single-port block RAMs during one clock cycle.

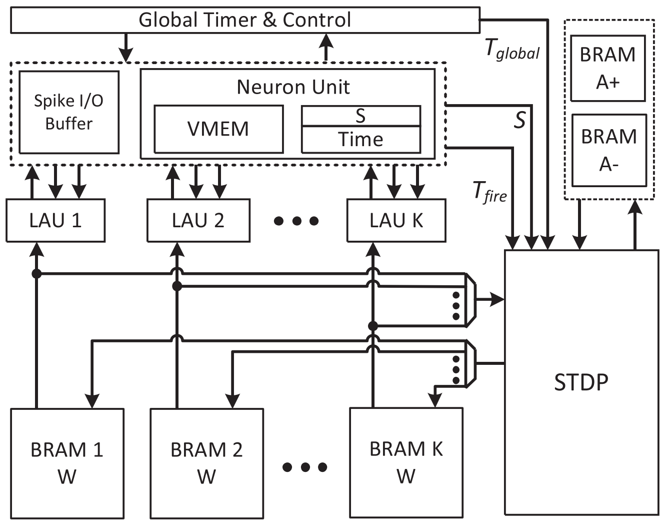


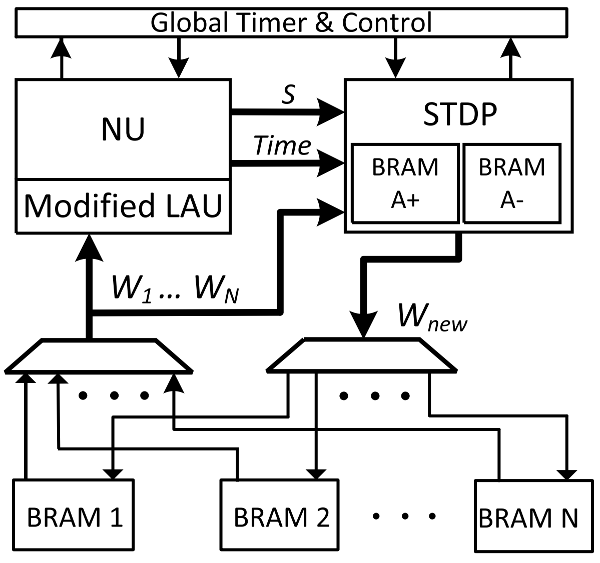
Fig. 9. The proposed parallel neuromorphic processor which develops *K*-way parallel processing based on LIP. *K* block RAMs are used to store synaptic weights, and *K* LAUs work in parallel to update *K* membrane potentials at the same time.

weights of the synapses from the input layer to the output layer are stored in *K* block RAMs. The weights associated with each output layer neuron are all in the same block RAM. Ideally, if the workload of the NOS is well balanced, each LAU performs the V_{mem} update of M/K excitatory neurons in the output layer and *K* LAUs work in parallel. The V_{mem} update of other neurons is parallelized in the same way. Although the total capacity of the register files (V_{mem}, S and T_{fire}) inside the neuron unit (NU) remains the same, multiple data ports are created for the NU to enable the *K* LAUs to access the data inside NU in parallel.

Fig. 10 shows the parallel architecture based on LJP, which corresponds to Fig 8(b). *N* synaptic weights are read out simultaneously and sent to a modified LAU. The modified LAU is illustrated by Fig. 10(b), which involves an N-input adder tree. Unlike the original LAU in Fig. 5, this modified LAU updates each V_{mem} in only one clock cycle.

If the size of the SNN is not very large, the architecture in Fig. 10 will significantly accelerate the NOS. However, this parallel scheme suffers from bad scalability. To support the calculation of $\sum W_{ji}\cdot Sj$ in one single clock cycle, a large *N*-input adder is required. Such adders can introduce high hardware cost and propagation delay, which limits the clock rate. Even though the critical path delay may be reduced by pipelining, both the power and the slice utilization would increase rapidly with the size of the network in this architecture. Therefore, LIP is more competitive than LJP for large networks.

Meanwhile, the efficiency of the proposed parallel schemes also depends on the number of output neurons and the number of input neurons. Specifically, the size of the adder in a LJP design is based on the number of pre-synaptic neurons. For an SNN with *N* excitatory neurons in the input layer and *M* excitatory neurons in the output layer, each excitatory neuron in the input layer has 1 external input and also a small number of inhibitory pre-synaptic neurons. However, each



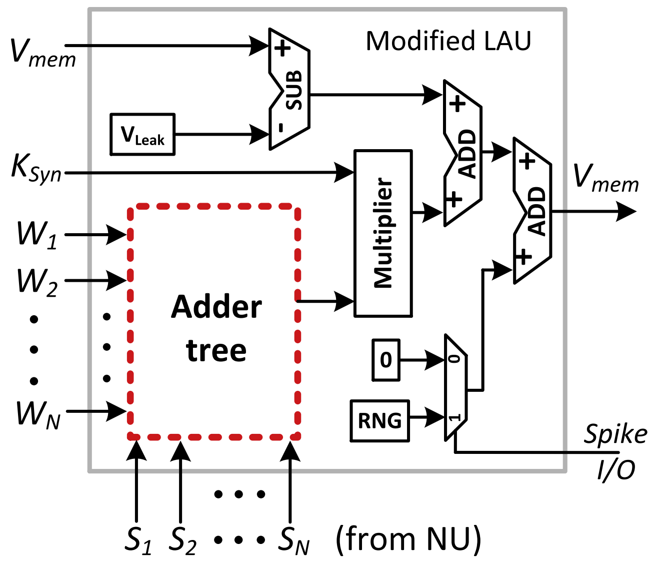


Fig. 10. The proposed parallel neuromorphic processor based on LJP, which uses N block RAMs to store synaptic weights and one modified LAU to update one V_{mem} in one clock cycle. (a) the parallel architecture based on LJP; (b) the modified LAU with a large adder tree.

excitatory neuron in the output layer has totally M excitatory pre-synaptic neurons and 1 inhibitory pre-synaptic neuron. Therefore, when M is much larger than N, but not too large, the LJP can be more efficient than LIP.

5. Approximate multiplier with error compensation

Booth multipliers are ideal for high speed applications and the Radix-4 Modified Booth multipliers are most widely applied [26,27]. The main blocks of a Radix-4 Modified Booth multiplier are shown in Fig. 11. Two main advantages of the Booth multipliers are the speed and the suitability for the multiplications of two's complements without any extra conversions. The encoding block applies the Radix-4 Booth Algorithm to encode the multiplier B, allowing the selection block to generate only half number of partial products needed for array multipliers with each product being one of the following: 0, A, 2A, −A, −2A (A is the multiplicand). Then, the compressors in the compression block compress the number of partial products to two [28,29]. Finally, a 2n-bit adder is used to generate the final product. Multipliers take up about 20% of the slices used in the serial baseline architecture and also consume a large portion of power consumption.

Approximate multiplication circuits consume fewer hardware resources and less power and are usually faster than exact multipliers.

Nevertheless, use of approximate multiplications would incur unavoidable errors in these low-level arithmetic operations. However, a more meaningful question to ask is that to what extent such errors may impact the targeted machine learning applications. We argue that due to the inherent resilience of our focused neuromorphic computing architecture, when well controlled, these arithmetic errors may only lead to negligible performance degradation of the neural network.

While several approximate multipliers have been proposed in the literature, there has not been any attempt to evaluate the application of approximate multipliers under the context of spiking neural networks. To explore the potential energy and area savings offered by approximate multiplication in our neuromorphic architectures, we adapt the

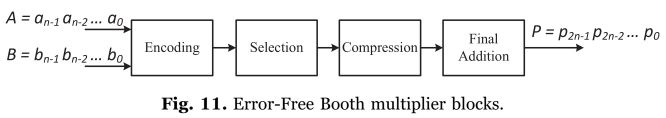


Fig. 11. Error-Free Booth multiplier blocks.

ideas presented in [21,22] to develop approximate multipliers for the FPGA platform. Fig. 12 shows the full 8-partial product array for a standard 16×16 Booth multiplier where each dot row (PP_0 to PP_7) is a partial product. The fixed-point operands in the multiplication are 2's complements with 8 integer bits and 8 fraction bits. The 16 dots (bits) in each PP_i are denoted by $pp_{i,15}pp_{i,14}\cdots pp_{i,0}$ from left to right. c_i is the correction constant required to generate the negative partial product, and s_i is the sign of the ith partial product. In addition, an extra 1 is added to the 15th column to exactly round the carry-out to the 16th column, and finally output the exact 16-bit integer part of the final product. The partial products are then added to obtain the final result. In contrast, the adopted approximate Booth multiplier only generates the signals of the partial products from the 15th column to the 31st column, which corresponds to the low-precision computing unit (LPCU) in Fig. 13. An error compensation which is for the signals associated with column 0 to column 14 is finally combined with the product obtained by the LPCU. To generate the appropriate error compensation, all the input patterns are classified into a number (in this case three) of groups, each of which is associated with a predetermined optimal error compensation [21]. Instead of providing a fixed error compensation for all input patterns, the key idea in achieving good accuracy is to have a fine tuned error compensation for each subset (group) of input patterns such that a given error metric (e.g. average or mean squared error) is minimized. As shown in Fig. 13, in the case of 16×16 approximate multipliers we use for our spiking neural networks, the optimal error compensation is 2, 1 and 0, respectively for three different groups of input patterns. The inputs are classified by a compact signature generator block (Fig. 13) that is based on Boolean operations of certain existing internal Booth encoding signals so as to minimize the area and energy overhead of the approximate multiplier design.

Table 1 compares the approximate multiplier with the standard booth multiplier and the multipliers provided by Xilinx. Because we need to convert the operands and product of the multiplier into our desired format, additional hardware overhead will be added to the Xilinx multipliers. The first Xilinx multiplier is based on LogiCORE Multiplier v11.2, a soft IP core provided by Xilinx [31]. This multiplier core can be configured to realize an operand precision ranging from 2 to 64 bits and is implemented with generic FPGA resources like CLBs (Configurable Logic Blocks). The other Xilinx multiplier is based on the

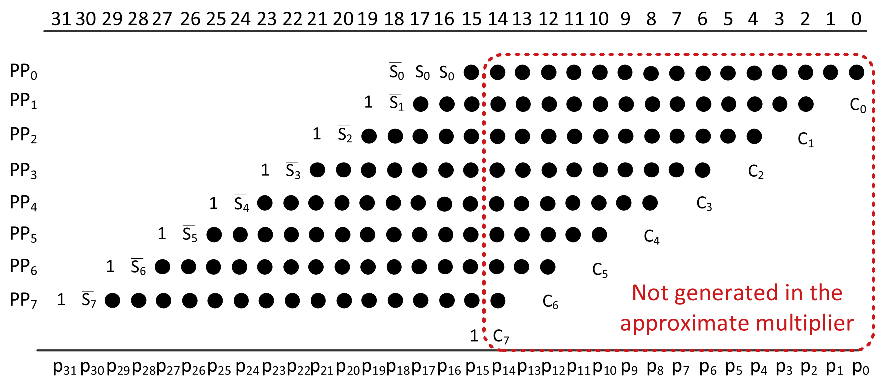


Fig. 12. Partial product diagram for fixed-width 16×16 Booth multipliers. Although all the partial products of the standard multiplication are shown in this diagram, only the signals on the left hand side are implemented. The signals on the right hand side are not generated.

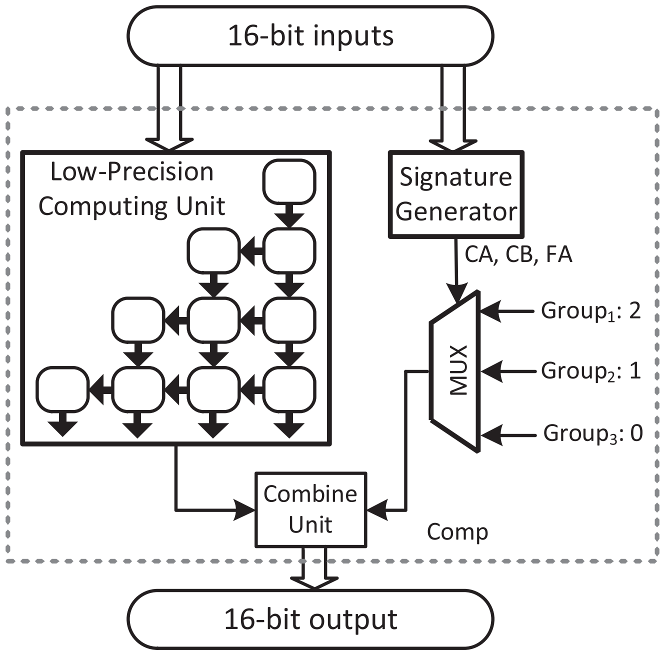


Fig. 13. The architecture of the 16×16 approximate multiplier with optimal error compensation for each input group.

dedicated and full-custom DSP slice DSP48E1 [32]. DSP48E1 involves a 25×18 hardwired ASIC multiplier, which is highly optimized by Xilinx. One DSP48E1 slice is flexible enough to realize any arithmetic precision below 25×18. Cascading multiple DSP48E1 slices allows for multiplications with higher precisions. According to Table 1, the booth multipliers used in this work enjoy smaller delays than the Xilinx multipliers. This is in part due to the fact that the desired operand format has been integrated into the proposed multipliers such that no additional logic is needed for format conversion. Although DSP48E1 itself is a highly optimized built-in resource provided by Xilinx, connecting the DSP48E1 slices with CLBs may incur a very large routing delay. In contrast, LUT-based multipliers may enjoy much reduced routing delays within a local area of reconfigurable FPGA resources. In this sense, the LUT-based booth multipliers are more suitable for this work.

To minimize the extra delay, the signature generator and multiplexer for generating an error compensation are designed to run in parallel with the selection block and compression block, which compose the LPCU (as shown in Fig. 13). Finally, a 16-bit Carry-Propagation Adder (CPA) [26] is designed as the Combine Unit (CU) to combine the error compensation with the result outputted by the LPCU, generating the final product. To further reduce delay of the multiplier, a 16-bit Carry-Lookahead Adder (CLA) [26] might be designed as the CU, which, however, may consume more power and hence present a different tradeoff between delay and power.

6. Experimental results

6.1. Design platform

The proposed neuromorphic processors are designed in Verilog HDL and synthesized using a Xilinx Synthesis tools. A Xilinx ML605 Evaluation Board [30], making use of an FPGA Virtex 6 core, has been employed in order to develop and test our designs. We follow the typical FPGA design flow to perform functional simulation, logic synthesis, placement & routing, and generate the configuration bit-stream. According to the timing analysis conducted as part of the synthesis flow, the proposed neuromorphic processors are able to run at 133.288 MHz. We employ an MMCM (Mixed Mode Clock Manager) block to generate the actual clock rate which is 120 MHz. The proposed designs are synthesized in a hierarchical/bottom-up manner, to allow straightforward reuse of baseline building blocks such as the Neuron Unit, LIF Arithmetic Unit and STDP Unit among targeted architectural variants. In order for the proposed architectures to communicate with the Matlab program running on PC, we also implement an UART(Universal Asynchronous Receiver/Transmitter) to support the serial communication. The power consumption of each architecture is obtained by using XPower Analyzer, which offers detailed power analysis of the designs on Xilinx FPGA [33]. The same with our hardware architectures, our C++ software SNN simulator is also developed according to the algorithm described in Section 2.

6.2. Performances for handwritten digit recognition

To demonstrate the performance of different neuromorphic processor architectures, we use the proposed architectures to solve a handwritten digit recognition problem with images from MNIST, a popular public domain dataset of handwritten digits with the 28×28 resolution [23]. MNIST involves 60,000 images for training and 10,000 images for recognition. Each 28×28 image is converted into a pattern with 28×28 pixels, which are used to generate the external input spikes to the input layer of the spiking neural network. In order to obtain an acceptable performance for this particular test bench, we instantiate a spiking neural network with 784 excitatory neurons in the input layer and 800 excitatory neurons in the output layer, as illustrated by Fig. 14(a). There are also 6 inhibitory neurons in the input layer and 1 inhibitory neuron in the output layer. This setup provides a good experimental study for examining the performance of the proposed architectures especially when approximate arithmetic computing is incorporated.

Fig. 14 (b) demonstrates how each 28×28 image is converted to 784 parallel spike trains which are the inputs to the input layer of the neural

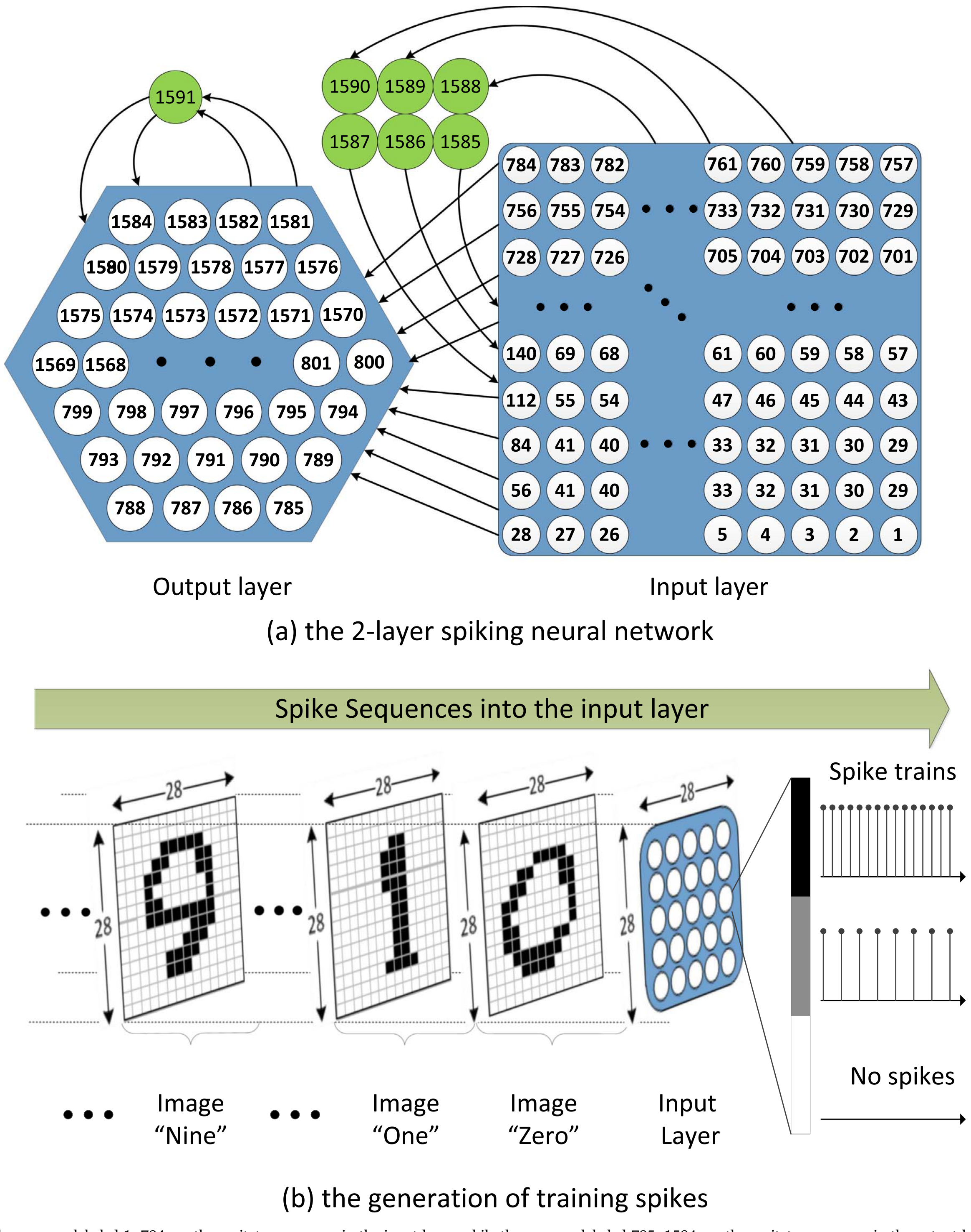

Output layer
Input layer
(a) the 2-layer spiking neural network
Spike Sequences into the input layer
Spike trains
No spikes
Image "Nine"
Image "One"
Image "Zero"
Input Layer
28
(b) the generation of training spikes

Fig. 14. (a) The neurons labeled 1–784 are the excitatory neurons in the input layer, while the neurons labeled 785–1584 are the excitatory neurons in the output layer. The other neurons are the inhibitory neurons. (b) Each pixel of the input image is sent to the corresponding input layer neuron in the form of a spike train.

network. The occupation rate of each spike train depends on the grey level of the corresponding pixel. As illustrated in Fig. 14, a "black" pixel is converted to a spike train with the highest occupation rate, while there is not any spike generated for the"white" pixel. These spike trains are considered as the external input spikes of the neuromorphic processor.

The receptive fields of the output layer neurons after the training are illustrated in Fig. 15. As can be seen, the receptive fields are well shaped by the training. The proposed neuromorphic processors with the standard Booth multipliers achieve a 89.1% recognition rate over the complete MNIST dataset. If the standard booth multipliers in the proposed architectures are replaced by the approximate multipliers, the recognition rate over the same data set becomes 87.7%, demonstrating that the use of approximate multiplications has no significant impact on recognition performance for this application.

Table 2 compares our work with a recent publication [34], which proposes software-based spiking neural networks with unsupervised learning, in terms of the classification accuracy for the MNIST dataset. In [34], a neural network with 400 excitatory neurons in the output layer, 1584 neurons and 473,600 synapses in total achieves a recogni-

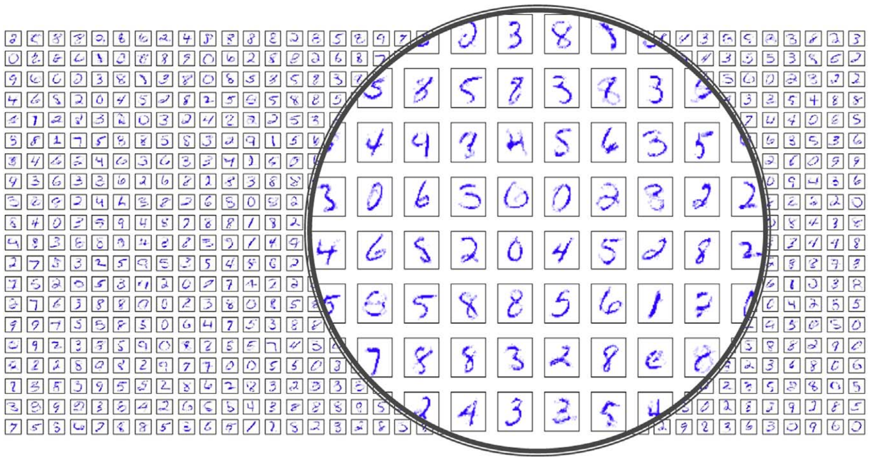


Fig. 15. The 800 receptive fields obtained after the training over 60,000 MNIST images of handwritten digits.

Table 2
Classification accuracy of various spiking neural networks on MNIST test set. The software (sw) simulation in this work is based on floating point arithmetic, while the hardware implementations with standard booth multipliers (std hw) and approximate multipliers (apx hw) are based on fixed point arithmetic.

		Size of output layer	# of neurons (synapses)	Accuracy
Ref. [22] Peter et al., 2015		800	2384 (1,267,200)	88.6%
This work	sw	800		90.0%
	std hw		1591(638,208)	89.1%
	apx hw			87.7%

tion accuracy of 87.0%. However, when 1600 excitatory neurons are used in the output layer, 3984 neurons and 3814,400 synapses are used for the entire network, an accuracy of 91.9% is obtained from their network. The recognition accuracy vs. network size plot provided in [34] shows that an accuracy level of 88.6% can be achieved by a network with 800 excitatory neurons in the output layer, 2384 neurons and 1,267,200 synapses in total. As shown in Table 2, compared with this reference network simulated using floating points in software, our proposed work achieves highly competitive recognition performances with a much smaller overall network complexity, i.e. 1591 neurons and 638,208 synapses. This is the case even for our hardware-based implementations operating on the fixed-point arithmetic.

6.3. *Tradeoffs between power, energy and hardware overheads of the parallel architectures*

In Section 4, we have presented two parallel architectures. Among these, the architecture in Fig. 8(b) involves LJP-based parallelization. To provide a good recognition performance for the digit recognition task, the number of input layer neurons has to be large enough to provide a sufficient resolution for input patterns. This gives rise to a large network size. As discussed in Section 4, this disqualifies LJP-based architectures as a competitive solution. Therefore, we only experimentally focus on the LIP architecture.

Table 3 lists the power consumption and slice utilization of each building blocks in the baseline serial neuromorphic processor design. The membrane potentials of all neurons are updated one after another. The summation of the synaptic weights for each neuron is realized by a single accumulator, which is consistent with Fig. 5. The slice utilization of each building block is obtained after place and route. The powers of the building blocks are obtained from XPower Analyzer (XPA) and Xilinx Power Estimator (XPE), two commercial tools for power analysis. Although the clock frequency is 120 MHz for the neuro-

Table 3
Power and resource utilization of the building block components of the baseline architecture, which accumulates the pre-synaptic weights serially for each output layer neuron. All the neurons are processed one by one in a serial manner.

	Slice LUTs	Slice FFs	Power (mW)
(a) The baseline design with standard booth multipliers			
LIF Arithmetic	626	96	2.98
Neuron Unit	69,265	50,688	1.07
STDP Unit	2352	199	10.44
Glue Logic	68	16	0.55
Block RAM	2508,800 bits for W		0.39
Block RAM	5,017,600 bits for $A+$		0.48
Block RAM	5,017,600 bits for $A-$		0.48
(b) The baseline design with approximate multipliers			
LIF Arithmetic	516	83	2.21
Neuron Unit	69,265	50,688	1.07
STDP Unit	1817	134	7.71
Glue Logic	68	16	0.55
Block RAM	2,508,800 bits for W		0.39
Block RAM	5,017,600 bits for $A+$		0.48
Block RAM	5,017,600 bits for $A-$		0.48

morphic processor, the actual switching frequency of each building block can be much lower than the main system clock. Therefore, the Neuron Unit has a low power consumption although the number of used flip-flops is large. The design information of the baseline design without and with the approximate multipliers are shown in Table 3(a) and (b), respectively. When comparing these two variants of the baseline serial design, we can easily see adoption of the approximate multipliers helps reduce both power consumption and slice utilization.

Table 4 reports the power consumption and slice utilization of each building block in a parallel design based on the LIP architecture illustrated in Fig. 9. This design has 2 single-port BRAMs to store the weights of the plastic synapses, and 2 LAUs are instantiated to support the 2-way parallel processing. The original neuron unit is modified to support two parallel accesses to V_{mem} and S. Table 5 reports the power consumption and slice utilization of the building blocks of the LIP architecture in Fig. 9 with a higher degree of parallelism (K=32). This architecture involves 32 parallel LAUs and 32 block RAMs to store the weights of the plastic synapses.

Table 6 compares five LIP designs with different degrees of parallelism and different multipliers in terms of the runtime, energy consumption and hardware resource cost. The runtime is the processing time of one image during training. These designs follow the LIP architecture of Fig. 8(a). The energy consumption of each design is calculated from the actual runtime, the power consumption and utilization of all its building blocks. For the serial baseline architecture,

Table 4
Power and resource utilization of the building block components of the 2-way LIP parallel architecture, in which synaptic weights of each neuron are accumulated in serial while two neurons are processed simultaneously.

	Slice LUTs	Slice FFs	Power (mW)
(a) The 2-way parallel LIP architecture with standard booth multipliers			
LIF Units (2)	1252	192	5.92
Neuron Units	71,107	51,858	1.78
STDP Unit	2352	199	10.44
Glue Logic	96	33	1.01
Block RAM (2)	2,508,800 bits for W		0.79
Block RAM	5,017,600 bits for $A+$		0.48
Block RAM	5,017,600 bits for $A-$		0.48
(b) The 2-way parallel LIP architecture with approximate multipliers			
LIF Units (2)	1032	166	4.42
Neuron Unit	71,107	51,858	1.78
STDP Unit	1817	134	7.71
Glue Logic	96	33	1.01
Block RAM (2)	2,508,800 bits for W		0.79
Block RAM	5,017,600 bits for $A+$		0.48
Block RAM	5,017,600 bits for $A-$		0.48

Table 5
Power and resource utilization of the building block components of the 32-way LIP parallel architecture, in which 32 neurons are processed simultaneously.

	Slice LUTs	Slice FFs	Power (mW)
(a) The 32-way parallel LIP architecture with standard booth multipliers			
LIF Units (32)	20,032	3072	94.72
Neuron Units	74,674	55,388	14.33
STDP Unit	2352	199	10.44
Glue Logic	229	167	13.82
Block RAM (32)	2,508,800 bits for W		12.86
Block RAM	5,017,600 bits for $A+$		0.48
Block RAM	5,017,600 bits for $A-$		0.48
(b) The 32-way parallel LIP architecture with approximate multipliers			
LIF Units (32)	16,512	2656	70.72
Neuron Unit	74,674	55,388	14.33
STDP Unit	1817	134	7.71
Glue Logic	229	167	13.82
Block RAM (32)	2,508,800 bits for W		12.86
Block RAM	5,017,600 bits for $A+$		0.48
Block RAM	5,017,600 bits for $A-$		0.48

we use the detailed timing diagram shown in Fig. 7 to determine the execution times for various building blocks for the purpose of energy estimation. Similar functional analysis is performed for the architectures with K=2–32. Obviously, the parallel design with K=32 achieves a speedup of 13.5×over the baseline design with K=1.

Fig. 16 compares these designs with respect to runtime and energy consumption. As the degree of parallelism increases, the runtime gets shorter and shorter but the energy consumption goes up. This is because additional resource and power overheads are introduced to support parallel processing. The energy improvement introduced by the approximate multipliers can reach up to 20.1% for the serial design. This improvement gets somewhat smaller when the degree of parallelism is increased. It is due to the fact that the runtime of the parallelized part takes up a smaller portion of the total runtime. In this case, while a larger number of approximate multipliers are used to increase the number of LAUs so as to increase parallelism, the relative benefit in energy saving is getting smaller. Therefore, to further parallelize the serial part of the design (i.e. STDP Unit) can be a solution, which prevents the energy improvement due to approximate computing from decreasing. However, we expect a higher energy consumption from such a design, and the corresponding improvement of runtime is not obvious because the LOS consumes much shorter runtime than the NOS.

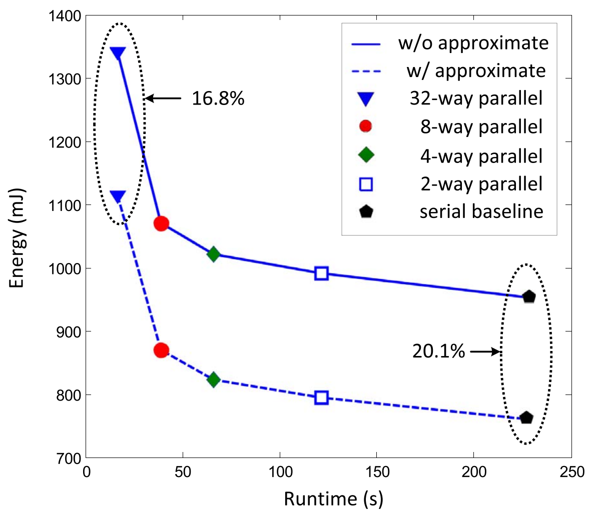


Fig. 16. Comparison of different designs in terms of runtime and energy consumption. The solid curve represents the designs using standard booth multipliers. The dashed curve represents the designs using the approximate multipliers.

The C++ program which corresponds to the serial baseline hardware design is evaluated on the AMD Opteron 6174 processor, which is a general purpose CPU clocked at 2.2 GHz. This single-thread program takes 989.7 s to process an image during training, which is 4.4×longer than the runtime of the proposed serial hardware design with K=1. Our 32-way parallel hardware design is 59.4×faster than the single-thread C++ program for processing one image.

The LJP-based design which realizes the parallel processing schemes in Fig. 8(b) is also evaluated for the same benchmark. In this case, the large adder tree in the LAU involves 1568 inputs, namely, 784 parallel synaptic weights and 784 parallel spike events. This introduces noticeable propagation delay, so the maximum operating frequency becomes only 57 MHz. For such an LJP design, the training runtime for processing one image is 18.8 s, which is slightly longer than the 32-way LIP design. Meanwhile, in spite of the lower clock frequency, the corresponding energy consumption is 1358.5 mJ, which is higher than that of the 32-way LIP design.

When the neuromorphic processors are in the recognition mode,

Table 6
The comparison of the serial design and 5 LIP designs in terms of the runtime, energy and hardware cost. All the designs here are based on the parallel LIP architecture in Fig. 8(a). K is the degree of parallelism. *Std* means the design is based on the standard booth multipliers, and *Apx* means the design is based on the approximate multipliers.

	K=1		K=2		K=4		K=8		K=32	
	Std	Apx	Std	Apx	Std	Apx	Std	Apx	Std	Apx
Runtime (s)	226.95		121.86		66.53		38.8		16.8	
Energy (mJ)	953.56	761.82	990.60	794.69	1021.56	824.38	1071.06	869.8	1339.83	1115.02
Slice LUTs	72,311	71,666	74,717	73,962	77,043	76,068	79,956	78,541	97,287	93,232
Slice FFs	50,999	50,921	52,282	52,191	53,800	53,683	55,348	55,179	58,826	58,345
Block RAMs	3		4		6		10		34	

Table 7
The comparison of 5 LIP designs in terms of the runtime and energy in the recognition mode. *K* is the degree of parallelism. *Std* means the design is based on the standard booth multipliers, and *Apx* means the design is based on the approximate multipliers.

	K=1		K=2		K=4		K=8		K=32	
	Std	Apx	Std	Apx	Std	Apx	Std	Apx	Std	Apx
Runtime (s)	218.33		112.95		57.60		29.10		8.40	
Energy (mJ)	847.36	674.66	872.90	706.12	898.34	725.58	923.16	768.52	1127.25	925.65

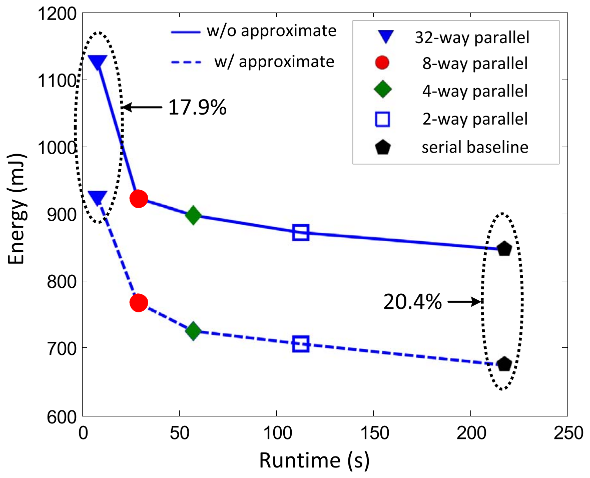


Fig. 17. Comparison of different designs in terms of runtime and energy consumption during recognition. The solid curve represents the designs using standard booth multipliers. The dashed curve represents the designs using the approximate multipliers.

the STDP unit and the BRAMs for A+ and A− remain inactive. Therefore, the runtime for processing one image can be shorter than that of the training mode, because there is no LOS in the recognition phase. Table 7 compares the five proposed LIP designs in terms of runtime and energy consumption in the recognition mode.

Obviously, since LOS which is not parallelized does not exist in the recognition mode, the speedup increases almost linearly with the degree of parallelism in NOS. For example, when K=32, the speedup over the baseline design (K=1) for the recognition mode is 25.8x. In addition, as the degree of parallelism goes up, the recognition mode shows a more linear runtime reduction than the training mode. Therefore, its energy dissipation increases slower than that of the training mode, that latter of which is shown in Table 6. The comparison of parallel designs in the recognition mode is illustrated in Fig. 17.

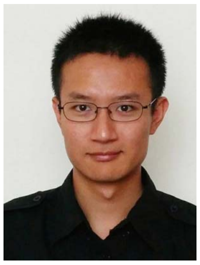
7. Conclusions

In this paper, we present an FPGA-based digital neuromorphic processor and several parallel architectures. The proposed architectures successfully address several critical issues pertaining to efficient parallelization in membrane potential computation, on-chip storage of synaptic weights, and integration of approximate arithmetic units. The trade-offs between throughput, hardware cost and power overheads for different configurations have been thoroughly investigated. A promising training speedup of 13.5x and a recognition speedup of 25.8x are achieved by a parallel design whose degree of parallelism is 32. The total training speedup provided by the 32-way parallel design running at 120 MHz over the serial software simulation running on a 2.2 GHz CPU is 59.4x. Up to 20% reduction in energy consumption is achieved when using the approximate multipliers in our baseline processor design, while maintaining pretty much the same level of recognition performance for handwritten digit recognition.

References

[1] J.V. Arthur, P.A. Merolla, et al., Building Block of a Programmable Neuromorphic Substrate: A Digital Neurosynaptic Core, IJCNN, pp. 1–8.
[2] Zhang Yong, et al., Linking brain behavior to underlying cellular mechanisms via large-scale brain modeling and simulation, Neurocomputing 97 (2012) 317–331.
[3] Q. Wang, Y. Li, P. Li, Liquid state machine based pattern recognition on FPGA with firing-activity dependent power gating and approximate computing, in: Proceedings of IEEE International Symposium of Circuits and Systems, May 2016.
[4] Q. Wang, Y. Jin, P. Li, General-purpose LSM learning processor architecture and theoretically guided design space exploration, in: Proceedings of IEEE Biomedical Circuits and Systems, Oct. 2015, pp. 1–4.
[5] J. Jin, P. Li, AP-STDP: a novel self-organizing mechanism for efficient reservoir computing, The International Joint Conference on Neural Networks, July 2016.
[6] Q. Wang, Y. Kim, P. Li, Neuromorphic processors with memristive synapses: synaptic interface and architectural exploration, in: ACM Journal on Emerging Technologies in Computing Systems, 2016.
[7] Koickal Thomas Jacob, Luiz C. Gouveia, Alister Hamilton, A programmable spike-timing based circuit block for reconfigurable neuromorphic computing, Neurocomputing 72 (16) (2009) 3609–3616.
[8] Diaz Carlos, et al., An efficient hardware implementation of a novel unary Spiking Neural Network multiplier with variable dendritic delays, Neurocomputing (2016).
[9] O. Abu Arqub, M. AL-Smadi, S. Momani, T. Hayat, Numerical solutions of fuzzy differential equations using reproducing kernel Hilbert space method, Soft Comput. (2015). http://dx.doi.org/10.1007/s00500-015-1707-4.
[10] O.Abu Arqub, Adaptation of reproducing kernel algorithm for solving fuzzy Fredholm-Volterra integrodifferential equations, Neural Comput. Appl. (2015). http://dx.doi.org/10.1007/00521-015-2110-x.
[11] S. Mitra, et al., Real-time classification of complex patterns using spike-based learning in neuromorphic VLSI, IEEE Trans. Biomed. Circuits Syst. 3 (1) (2009) 32–42.
[12] A. van Schaik, et al., Building blocks for electronic spiking neural networks, Neural Netw. 14 (2001) 617–628.
[13] Lanny L. Lewyn, et al., Analog circuit design in nanoscale CMOS technologies, Proc. IEEE 97 (10) (2009) 1687–1714.
[14] Harpe Pieter, Andrea Baschirotto, Kofi AA Makinwa, (eds.), High-performance AD and DA converters, IC Design in Scaled Technologies, and Time-domain Signal Processing: Advances in Analog Circuit Design 2014, Springer, 2014.
[15] G. Indiveri, et al., A VLSI array of low-power spiking neurons and bistable synapses with spiking-timing dependent plasticity, IEEE Trans. Neural Netw. 17 (1) (2006) 211–221.
[16] Bashaireh Ahmad, Peng Li, Design robustness analysis of digital spiking neural circuits, circuits and systems (MWSCAS), in: Proceedings of the IEEE 57th International Midwest Symposium, 2014.
[17] Andres Upegui, Andrés Peña-Reyes Carlos, Eduardo Sanchez, et al., An FPGA platform for on-line topology exploration of spiking neural networks, Microprocess. Microsyst. 29 (5) (2005) 211–223.
[18] Martin J. Pearson, et al., Implementing spiking neural networks for real-time signal-processing and control applications: a model-validated FPGA approach, IEEE Trans. Neural Netw. 18 (5) (2007) 1472–1487.
[19] Kenneth L. Rice, et al., FPGA implementation of Izhikevich spiking neural networks for character recognition, in: Proceedings of the IEEE International Conference on Reconfigurable Computing and FPGAs, 2009, ReConFig'09, 2009.
[20] David B. Thomas, Wayne Luk, FPGA accelerated simulation of biologically plausible spiking neural networks, in: Proceedings of the 17th IEEE Symposium on Field Programmable Custom Computing Machines, 2009, FCCM'09, 2009.
[21] Botang Shao, Peng Li, A model for array-based approximate arithmetic computing with application to multiplier and squarer design, in: Proceedings of IEEE/ACM International Symposium on Low Power Electronics and Design, 2014.
[22] Botang Shao, Peng Li, Array-based approximate arithmetic computing: a general model and applications to multiplier and squarer design, IEEE Trans. Circuits Syst. I (2014).
[23] The MNIST database of handwritten digits by Yann Lecun, Corinna Cortes.
[24] Song Sen, Kenneth D. Miller, Larry F. Abbott, Competitive Hebbian learning through spike-timing-dependent synaptic plasticity, Nat. Neurosci. 3 (9) (2000) 919–926.
[25] Anthony N. Burkitt, A review of the integrate-and-fire neuron model: i. Homogeneous synaptic input, Biol. Cybern. 95 (1) (2006) 1–19.
[26] Neil HE Weste, David Money Harris, CMOS VLSI Design: A Circuits and Systems Perspective, Addison-Wesley Publishing Company, 2010.
[27] E. de Angel, E.E. Swartzlander, Jr., Low Power Parallel Multipliers, Workshop VLSI Signal Process., IX, 1996.

[28] Christopher S. Wallace, A suggestion for a fast multiplier, IEEE Trans. On Electronic Computers, vol. EC-13, 1964.
[29] Vojin G. Oklobdzija, David Villeger, Simon S. Liu, A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach, IEEE Trans. Computers, 1996.
[30] XILINX, Virtex-6 FPGA ML605 Evaluation Kit. ⟨http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm⟩.
[31] XILINX, Xilinx DS255 Multiplier v11.2, Data Sheet.⟨http://www.xilinx.com/support/documentation/ip-documentation/mult-gen-ds255.pdf⟩
[32] XILINX, Virtex-6 FPGA DSP48E1 Slice User Guide. ⟨http://www.xilinx.com/support/documentation/user-guides/ug369.pdf⟩.
[33] Xilinx ISE Xpower Analyser⟨http://www.xilinx.com/products/logic-design/verification/xpoweran.htm⟩.
[34] P.U. Diehl, M. Cook, Unsupervised learning of digit recognition using spike-timing-dependent plasticity, Front. Comput. Neurosci. 9 (2015).



Qian Wang received the B.Eng. degree in electronics information science and technology from the Harbin Institute of Technology, Harbin, China, in 2009, and the M.S. degree in electrical engineering from the University of Kansas, Lawrence, KS, USA, in 2012. He received the Ph.D. degree in computer engineering with Texas A & M University, College Station, TX, USA, in 2016. He is involved in research on VLSI implementation of machine learning algorithms, such as support vector machine and spiking neural networks.

Youjie Li received the M.S. degree in Department of Electrical and Computer Engineering, Texas A & M University, College Station, TX, USA, in 2016. His research interests are in the areas of neuromorphic computing and approximate computing. He received an ISCAS best paper award in 2016.

Botang Shao received the M.S. degree in Department of Electrical and Computer Engineering, Texas A & M University, College Station, TX, in 2014. He is currently a design engineer at NXP Semiconductors, Austin, TX. His research interests are in the areas of low-power arithmetic circuits and VLSI design.

Siddhartha Dey received the B.Tech. degree in Electronics and Communication Engineering from Heritage Institute of Technology, Kolkata, India, in 2013. He is currently working towards the M.Eng. degree in Computer Engineering at Texas A & M University, College Station, TX, USA. He joined Synopsys, Inc., Austin, TX, USA in May 2016 where he works as a Verification Engineer specializing in emulation platform. His research interests are in the areas of neuromorphic systems, and VLSI design and verification.

Peng Li received the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University. He is a professor with the Department of Electrical and Computer Engineering, Texas A & M University, and a faculty member of Texas A & M Institute for Neuroscience and Texas A & M Health Science Center. His research interests are in integrated circuits and systems, electronic design automation, and computational neuroscience. He has authored and co-authored over 180 publications and edited two books.

His work has been recognized by various distinctions including an IEEE/ACM William J. McCalla ICCAD Best Paper Award, three IEEE/ACM Design Automation Conference Best Paper Awards, two SRC Inventor Recognition Awards, two MARCO Inventor Recognition Awards, the National Science Foundation CAREER Award. At Texas A & M University, he received the ECE Outstanding Professor Award, and was named a TEES Fellow and a William O. and Montine P. Head Faculty Fellow. He currently serves the IEEE Council of Electronic Design Automation as its Vice President for Technical Activities. He is a Fellow of the IEEE.