Hardware Spiking Neural Networks with Pair-Based STDP Using Stochastic Computing

Junxiu Liu[1] · Yanhu Wang[1] · Yuling Luo[1] · Shunsheng Zhang[1] · Dong Jiang[1] Yifan Hua[1] · Sheng Qin[1] · Su Yang[2]

Abstract

Spiking Neural Networks (SNNs) can closely mimic the biological neural network systems. Recently, the SNNs have been developed in hardware circuits to emulate the time encoding and information-processing aspects of the human brain in real-time. However, the hardware SNN systems are suffering from large hardware resource consumption due to the high complexity of computational units. In this paper, a novel hardware SNN system based on stochastic computing is proposed to address this problem. Pair-based spiking-timing-dependent plasticity, coupled with integrate-and-fire neurons are employed to design the SNN. Stochastic computing can simplify the computational components of multipliers, adders, and subtractors in conventional hardware SNNs, hence reduce the hardware resource cost. Experimental results show that compared with the state-of-the-art approaches the proposed SNN system reduces the resource consumption by 58.0% (especially registers by ≥65.6%). In the meantime, the maximum normalized root mean square error between the proposed hardware and others is only 0.0097, which can maintain the behaviours of SNN. This work provides a beneficial alternative to the large-scale hardware SNN implementations.

Keywords Spiking neural networks · Pair-based spiking-timing-dependent plasticity · Integrate-and-fire neurons · Stochastic computing

1 Introduction

Spiking Neural Networks (SNNs) are known as the third generation of Artificial Neural Networks (ANNs), and they can more closely emulate information processing of the mammalian brain than traditional ANNs [1–4]. Spike Timing Dependent Plasticity (STDP) is a widely used learning method for SNNs, which is thought to play an important role in learning and

Yuling Luo
yuling0616@gxnu.edu.cn

1 Guangxi Key Lab of Brain-inspired Computing and Intelligent Chips, School of Electronic Engineering, Guangxi Normal University, Guilin 541004, China

2 Department of Computer Science, Faculty of Science and Engineering, Swansea University, Swansea, UK

brain computation. It is considered most suitable for unsupervised training of feedforward SNNs and is similar to the biological behaviour [5–7]. Recently, the STDP is emulated by the software and hardware platform. For the software platform, the computer systems of conventional processors are applied to emulate the STDP [8–10]. The advantage of these software systems is that the emulated neurons are more flexible and the neuron models are more complex, and the disadvantage is that software platform cannot satisfy the requirement of parallel updating of state of neurons in the human brain. Thus the analog circuits and memristors are used to implement the STDP [11–14]. However, they are not configurable, and modifying a small part of the circuit requires a long development cycle. Field Programmable Gate Array (FPGA) devices with massive reconfigurable logical, computing, and memory resources have been used to implement the STDP. For example, a digital implementation of the STDP learning rule using a stochastic approach is developed in [15], which realizes the computationally expensive exponential delay and is capable of producing the same results to a more complex STDP model. Look-Up-Tables (LUTs), piecewise linear approximation, and Base-2 approximation are used in the approach of [16] to design a new STDP model with low computational complexity. The COordinate Rotation DIgital Computer (CORDIC) algorithm is used to implement the exponential function required for STDP in [17]. These methods reduce the consumption of hardware resources in hardware implementation of exponential functions, thus reducing the hardware cost. However, these approaches still have relatively low accuracy, large hardware resources and power consumption, which is not conducive for the scalability of large-scale SNNs. To deal with this problem, the phenomenological model of STDP is developed in the approach of [5]. Each spike train is converted to a continuous local variable through a low-pass filter, and only the current pre- and post-synaptic spikes are considered for the update of synaptic weight.The multiplications of phenomenological STDP can be implemented by using the powers of 2, i.e., 2^N in [18]. However, the approximation method still has relatively low accuracy in [18]. Therefore, building a low-cost hardware SNN with STDP is still a big challenge.

To deal with this problem, another alternative to optimize the hardware cost and power consumption is proposed in [19], which is namely Stochastic Computing (SC). In the SC, the conventional arithmetic operations are replaced by the operations between stochastic bitstreams to decrease computational resources and complexity [20]. The SC has been widely used to address computing problems at a low cost. Several Finite State Machine (FSM)-based SC elements are used to implement basic digital image processing algorithms [21]. In addition, low-density parity-check codes based on the SC is proposed in [22], which has a high speed and reduces the decoding complexity. The SC is also used to implement ANN. In the research of [23], a novel extended stochastic logic is proposed, which is used for efficient ANN implementation. An efficient way of hardware ANN is proposed by combining conventional binary radix computation and the SC technique [24].

To the authors' best knowledge, the SC has not been applied to develop the STDP in the SNNs. Inspired by these works, in this paper, an efficient hardware structure of SNN is designed, especially for the optimal design of the learning algorithm in synapses. The SC technique is used to achieve this design, and a novel hardware SNN with Pair-based STDP (PSTDP) based on the SC is proposed in this paper. The SC technique is employed to simplify the computational components of multipliers, adders, and subtractors in conventional hardware SNNs. The proposed system is synthesized and implemented on an FPGA device and experimental results show that only 323 LUTs and 491 registers are required by using the proposed system, and a low hardware resource consumption performance is obtained. The area and power consumption are evaluated by the SAED 90 nm CMOS technology, and

the hardware area is 27,565 $\bar{}$m^2, the power consumption is 2.32 mW. Moreover, the maximum Normalized Root Mean Square Error (NRMSE) between the proposed and the original models is 0.0097, i.e., the model behaviour is maintained. Contributions of this paper can be summarized as follows: (a) The SC technique is employed for IF neurons, synapses and PSTDP learning rule in the SNN hardware implementation where arithmetic operations are replaced by stochastic bitstreams with minimized hardware resource consumption, leading to a hardware friendly SNN. (b) A hardware SC-based PSTDP learning rule (SC-PSTDP) is proposed to modulate the synaptic weight changing process and maintain the system scalability. (c) To validate the efficiency of the proposed SC-SNN, it is synthesized and implemented on FPGA device, and comparison results with previous works show that the proposed work can achieve a trade-off between the performance and hardware cost. The proposed SC-PSTDP is critical for the SNN hardware as it determines the learning process and system scalability. The synaptic weight regulation happens at each synapse, thus a good scalability can be maintained if the learning control component has a low hardware cost. The proposed SC-PSTDP reduces the hardware consumption by using the SC technique, especially the DSP blocks which are expensive for FPGA devices. Thus, it can aid implementing the large-scale SNNs at a relatively low cost for the hardware system. The proposed SC-PSTDP in this paper is not limited to specific tasks. Besides, while maintaining the behavior of traditional STDP, the consumption of hardware resources is reduced and the scalability is improved by using the SC technique. Thus, the proposed SC-PSTDP provides an alternative for the SNN hardware implements.

The rest of this paper is organized as follows. Section 2 describes the related works of the STDP in the SNNs. Section 3 introduces the theoretical background of the neuron, synapse, PSTDP learning rule and the SC. Section 4 describes the hardware architecture of SNN by using the proposed method and Sect. 5 provides performance analysis and comparisons with other works. Finally, the conclusion is presented in Sect. 6.

2 Related Works

As an unsupervised learning rule in the SNNs, the STDP has been widely explored recently. There are two main platforms for implementing STDP, one is the software architecture and the other is the hardware architecture. Specifically, the hardware implementation includes analog and digital circuits.

2.1 Software-Based Architectures

Recently, the computer systems employing conventional processors are applied to implement the STDP in the SNNs [8, 9, 25]. For example, a novel supervised learning approach based on an event-based STDP is proposed in [9], which is evaluated on the XOR problem. A pre-training scheme using STDP is proposed in [25], and it is used to train the deep SNNs. The experiment shows that the proposed method improves robustness and reduces the training time. However, they are all based on the von Neumann structure which is characterized by serial execution. It cannot satisfy the requirement of parallel updating of state of neurons in the human brain, thus software implementation cannot guarantee the behaviour of a large neural network in real-time.

2.2 Hardware-Based Architectures

The analog and digital circuits are applied to implement the STDP of SNNs [15–17, 26–28]. The Leaky Integrate-and-Fire (LIF) and STDP models for SNNS are implemented by the analog circuit in the approach of [26]. The analog programming is used to generate the STDP with arbitrary behaviour [27]. Although the analog circuits require very little silicon area, they have relatively low noise resistance and reliability. Besides, they are not configurable and flexible [15]. For example, even a small modification to the designed neuron model and STDP requires a long development cycle. To deal with this problem, the FPGA device is used to implement the neurons and the STDP in the SNNs [15, 16, 28]. The Izhikevich neuron model and STDP learning rule based on CORDIC method are implemented on FPGA devices in [17]. The combined circuit is designed to realize the complex computation of kernel function, and arithmetic shift is used to replace the multiplication operation in [29]. In [30], a neural computing hardware unit and a neuromorphic system architecture based on LIF neuron model are proposed. In [31], a digital hardware architecture for spiking force is designed. The Euler method and the RK3 method are applied to implement the SNNs on FPGA [32]. In [33], an SNN based on CORDIC with on-line STDP learning is presented. These methods aid to reduce the hardware cost, especially, the usage of multipliers. The SC is also used to implement the neuron and synapse models in the SNNs [15, 23, 28]. The Izhikevich neuron model is implemented by using the SC technique in [28], which can reduce area and power consumptions. This research shows that the SC technique can reduce the hardware cost of neuron and synapse models. Thus, the Integrate-and-Fire (IF) neuron and synapse models are implemented by the SC technique in this paper. Furthermore, the SC-PSTDP learning rule is proposed to modulate the synaptic weight changing process to enhance the scalability of the large-scale SNNs. The proposed method in this paper requires less hardware resources and has a high accuracy compared with other approaches.

3 Preliminaries

The SNN is composed of neurons and synapses. The neurons receive and respond to input spikes from other neurons, and synapses transmit information via spikes between neurons. In this section, spiking neurons, synapse models, the PSTDP learning rule, and the SC are introduced.

3.1 Neuron Model

Neurons are the basic components of the biological neural network systems. Massive neurons are interconnected in a network, which gives the nervous system powerful capacities for information processing [34, 35]. Researchers have proposed various mathematical models of neurons to closely mimic the behaviour and function of biological neurons. For example, Hodgkin–Huxley model is able to express many biological behaviours of neurons [36]. However, its circuit implementation is quite complex and consumes lots of resources, which constraints large-scale network simulation in real-time. The Izhikevich model aims to closely mimic the biological neurons [37], and the IF model has higher computational efficiency and few parameters of neurons, and it exhibits essential biological features [38]. Therefore, the

IF model is used as the neuron model in this paper, which is given by

$$\tau_m \frac{dV}{dt} = R_m \, I(t), \tag{1}$$

where τ_m is the membrane potential time constant, V represents the membrane potential of the neuron, R_m denotes the membrane resistance, and $I(t)$ is the current injected into the membrane from the synapse. When the membrane potential V exceeds the threshold V_{th}, it will reset to V_{rest}, and the neuron will fire a spike.

3.2 Synapse Model

The synapse is responsible for transmitting spikes from presynaptic to postsynaptic neurons and plays a significant role in memory and learning in the SNNs. The strength or performance of a synapse can be modulated by presynaptic and postsynaptic activities. Some mathematical models of synapses are developed to closely emulate the behaviour and function of synapses [39–41]. Considering the low-cost implementation of synapse model in hardware circuits, the α dynamic synapse model in [39] is employed in this work, and it is described by

$$\tau \frac{dI_{syn}}{dt} = -I_{syn} + C \sum_{k=1}^{n} w_{ij} \delta\left(t - t_j^f\right), \tag{2}$$

where I_{syn} denotes the injected current, τ is the time constant for the decay of I_{syn}, C is a constant, n represents the number of spikes fired from presynaptic neuron j, w_{ij} denotes the synaptic weight between postsynaptic neuron i and presynaptic neuron j, t_j^f is the arrival time of a presynaptic spike, and $\delta(t)$ denotes the Dirac delta function. Particularly, if a presynaptic spike arrives, the synaptic current I_{syn} is increased, otherwise, it is decreased.

3.3 PSTDP Learning Rule

The STDP is one of plasticity rules and it is used to learn and memorize in the SNNs [42]. Specifically, synaptic weight is changed according to the time difference between the pre- and post-synaptic spikes. If the presynaptic spike arrives before the postsynaptic spike, the synaptic weight will be potentiated. Otherwise, it will be depressed [43]. Particularly, the STDP learning rule has several variants [44, 45]. Among them, PSTDP has less computational complexity, and it is used in this paper. The PSTDP can be easily represented with two variables of x_j and y_i. The x_j and y_i are updated by

$$\frac{dx_j}{dt} = -\frac{x_j}{\tau_+} + \sum_f \delta\left(t - t_j^f\right), \tag{3}$$

$$\frac{dy_i}{dt} = -\frac{y_i}{\tau_-} + \sum_f \delta\left(t - t_i^f\right), \tag{4}$$

where the x_j and y_i are the low-pass filtered version of the presynaptic spike train and postsynaptic spike train and they are only affected by the presynaptic spike and postsynaptic spike. When the presynaptic neuron fires a spike, the variable x_j will increase; otherwise, it will decrease exponentially with time constant τ_+. Similarly, the postsynaptic spike leaves a trace y_i. When the postsynaptic neuron fires a spike, the y_i will increase; otherwise, it will decrease exponentially with the time constant τ_-. The τ_+ and τ_- are the time constants

of traces of presynaptic and postsynaptic spikes, and they affect the trajectory attenuation. Generally their values are based on the findings of biological experiments, but they may vary according to different approaches. In this paper, referring the approach of [5], τ_+ and τ_- are set as 10 ms. t_j^f and t_i^f represent the firing times of the presynaptic neuron and postsynaptic neuron, respectively. $\delta(t)$ is the Dirac delta function. When the pre- or post-synaptic spike arrives, the change of synaptic weight is calculated by

$$\frac{dw_{ij}}{dt} = -B_i y_i(t) \sum_f \delta\left(t - t_j^f\right) + B_j x_j(t) \sum_f \delta\left(t - t_i^f\right), \tag{5}$$

where w_{ij} is the synaptic weight between postsynaptic neuron i and presynaptic neuron j, B_i and B_j denote the maximum weight change amplitudes. The change of the weight is proportional to the postsynaptic spike trace y_i or the presynaptic spike trace x_j.

3.4 Stochastic Computing

The SC technique requires a longer computation time than the traditional arithmetic operation [46], as the computing data of SC is represented by a stochastic sequence. The stochastic sequence is a pseudo-random bitstream generated by Stochastic Number Generator (SNG) [46, 47]. However, complex arithmetic operators are replaced by the simple logic gates in the SC, thus the consumption of hardware resources is reduced. In the SC, there are two coding formats: unipolar and bipolar. A pseudo-random bitstream S with a length of L-bit is used to represent a random number X, and has $L1$-bit for logic one and $(L-L1)$-bit for logic zero. For unipolar representation, the probability of random number X is $p_x \in [0, 1]$, which is calculated by $p_x = L1/L$. For bipolar representation, the probability of random number X is p_y [$\in$ 1, 1], which is calculated by a mapping function: p_y 2 p_x 1 (2$L1$ L)/L. Some basic computation elements, such as an SNG, a Stochastic Computing Element (SCE), and a de-randomizer are required in the SC. The circuit structure of SC addition is described in Fig. 1 a, which contains SNG1, SNG2, a multiplexer, and a counter. Among them, the SNG1 and 2 convert the traditional binary numbers of the inputs into stochastic sequences. In addition, the SNG is composed of a Linear Feedback Shift Register (LFSR) and a comparator, where the LFSR can generate pseudo-random numbers, then it is compared to the input binary number. The comparator outputs one if the pseudo-random number is less than the input binary number, otherwise, zero is the output [47]. In Fig. 1a, the X 1 represents the binary number, and the Y1 represents the pseudo-random number. X1 and Y1 is the input of comparator, and SN1 is the output of comparator. The counter is responsible for converting the output stochastic sequence of the SCE into a binary number. The range of a random number is [0, 1] in unipolar, so the addition result of two random numbers ranges in [0, 2]. To satisfy the range of SC in unipolar, a scale factor (0.5) is needed in multiplexer. Particularly, two separate SNG blocks are used to make the generated stochastic bitstreams uncorrelated. Different seeds are selected in the SNG to improve the accuracy of SC. Moreover, the SCE is composed of basic arithmetic operations, mainly including multiplication, addition, and subtraction, which are introduced as follows.

(a) *Multiplication* In the unipolar format, a multiplication operation of two stochastic bit-streams is implemented by a simple two-input AND-gate. In the bipolar format, it is implemented by an XNOR-gate. These operations are shown in Fig. 1b, where A, B, and C denote the binary number, (e.g., $A = 0.100_2$ is represented by $p(A) = 4/8$), i.e., the

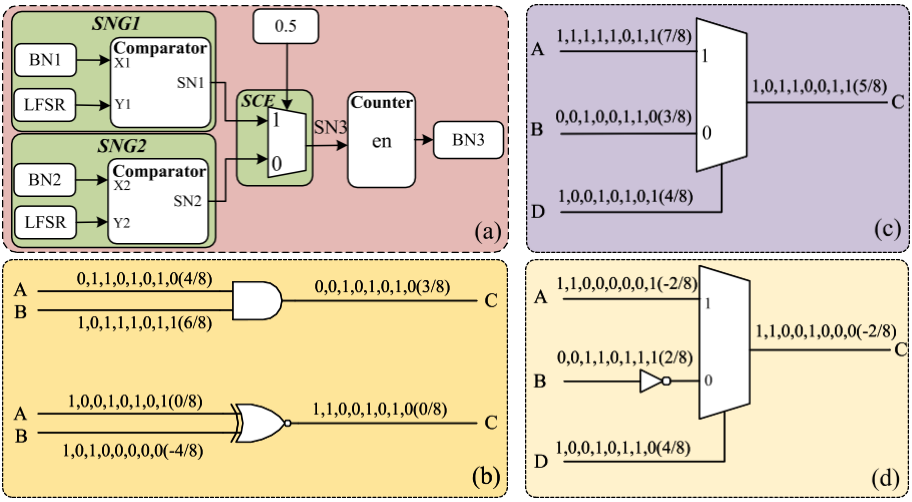

Fig. 1 The structure of SC arithmetic operations. **a** Hardware structure of SC addition; **b** SC multiplication; **c** SC addition; **d** SC subtraction

number of logic ones is 4 in stochastic bitstreams and the length of stochastic bitstreams is 8.

(b) *Addition* As can be seen from the Fig. 1b, the range of a random number is [0, 1] in unipolar format, so the addition result of two random numbers ranges in [0, 2]. A multiplexer is used to scale the addition operation to [0, 1] to satisfy the range of SC in unipolar. Therefore, the addition formula is $p(C) = p(A) + p(B)/2$, and its implementation is shown in Fig. 1c.

(c) *Subtraction* The only difference between subtraction and addition is that an NOT-gate is added in subtraction, which is described as Fig. 1d, and this processing structure is only suitable for bipolar format.

4 Methodology

In this section, the SNN hardware system based on the SC technique, called SC-SNN, is presented. Specifically, the proposed system is composed of SC-IF, SC-Synapse, and SC-PSTDP. The overall architecture of the system is shown in Fig. 2 a. Their implementation circuits are described in detail in this section. In addition, considering the multiple uses of multiplication, addition, and subtraction operations in hardware implementation, the multiplexing method is also used in this work to reduce hardware resource cost. Particularly, it can be seen from the circuit structure that there are no complex and luxurious multipliers which are widely used in conventional designs.

4.1 SC-IF NeuronModel

The function of neurons is to receive, process, and fire spikes in the SNNs. The IF neu- ron implementation based on the SC technique, namely SC-IF, is designed. Specifically,

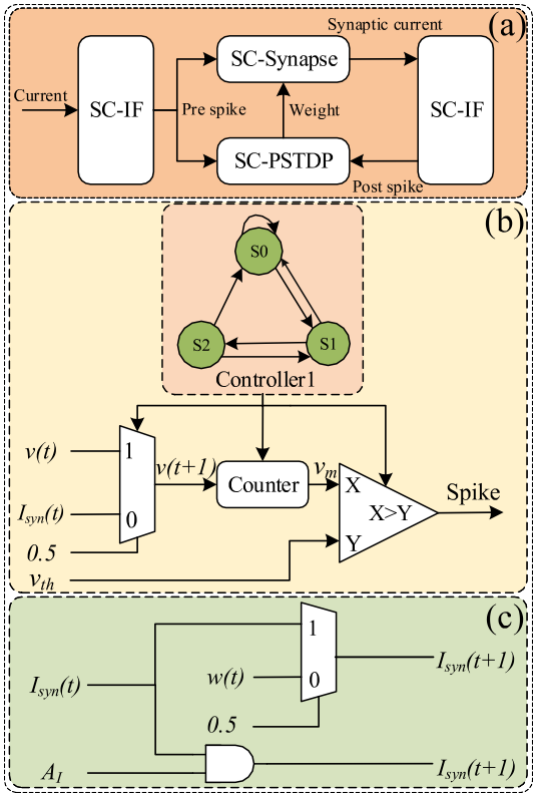

Fig. 2 a The overall architecture of system; **b** SC-IF; **c** SC-Synapse

according to the Euler method [48], differential Eq. 1 of neuron model can be simplified as

$$v(t+1) = v(t) + \frac{h}{\tau_m} R_m I(t), \tag{6}$$

where τ_m is the membrane potential time constant, v represents the membrane potential of the neuron, R_m denotes the membrane resistance, $I(t)$ is the current injected into the membrane from the synapse, h is time step, and an adder and a multiplier are needed for hardware design. Specifically, the adder is implemented by the multiplexer and the multiplier is implemented by the AND-gate in the SC. The architecture of SC-IF hardware is described in Fig. 2b, which is composed of a controller, multiplexer, counter, and comparator. The multiplexer is responsible for the accumulation of membrane potential. Because the output of multiplexer is a stochastic sequence, it is converted into a binary number of membrane potential by a counter. Then the membrane potential v_m is compared to the threshold v_{th}. If the v_m is greater than the threshold, a spike is fired by SC-IF. Moreover, the controller 1 is an FSM, which is used to control neurons to fire spikes. The function of each state is presented as follows. S0 is the initial state of the neuron. The state will go to S1 after a time step. In state S1, the neuron utilizes the circuit of adder to update membrane potential v_m. The neuron will transfer from S1 to S2 and fire a spike in state S2 until membrane potential is greater than the threshold v_{th}. Note that input and output data of the multiplexer in Fig. 2 are stochastic bitstreams.

4.2 SC-Synapse Model

The α dynamic synapse model is used in the SNN. Its low-cost hardware implementation is a great challenge due to its complex mathematical operations, hence the α synapse model based on the SC technique is designed in this work, which is defined as the SC-Synapse model. Specifically, differential Eq. 2 of synaptic model can be simplified as

$$I_{syn}(t+1) = I_{syn}(t) - \frac{hI_{syn}(t)}{\tau} + \frac{hC}{\tau}\sum_{k=1}^{n} w_{ij}\,\delta\left(t - t_j^f\right), \tag{7}$$

where I_{syn} denotes the injected current, τ is the time constant for the decay of I_{syn}, C is a constant, h is the time step, n represents the number of spikes fired from presynaptic neuron j, w_{ij} denotes the synaptic weight between postsynaptic neuron i and presynaptic neuron j. $\delta(t)$ is the Dirac delta function, and t_j^f represents the firing times of the presynaptic neuron. To simplify the arithmetic operations, Eq. 7 can be further simplified to

$$I_{syn}(t+1) = I_{syn}(t)\,A_I + B\sum_{k=1}^{n} w_{ij}\,\delta\left(t - t_j^f\right), \tag{8}$$

where $A_I = 1 - h/\tau$, $B = hC/\tau$. The architecture of the SC-Synapse is shown in Fig. 2c. If a presynaptic spike arrives, the synaptic current I_{syn} is increased by synaptic weight w_{ij}, i.e., the multiplexer is needed to update the synaptic current; otherwise I_{syn} has an exponential decay where the AND-gate is used. As can be seen that only an AND-gate and a multiplexer are contained in the architecture of the SC-Synapse, so the aim of reducing hardware resource cost is achieved.

4.3 SC-PSTDP Learning Rule

The trace equations for implementing the PSTDP learning rule are simplified by the Euler method. Then the SC method is used to implement the PSTDP learning rule, namely SC-PSTDP. The differential Eqs. 3 and 4 can be simplified as

$$x_j(t+1) = x_j(t) - \frac{hx_j(t)}{\tau_+} + h\sum_{f}\delta\left(t - t_j^f\right), \tag{9}$$

$$y_i(t+1) = y_i(t) - \frac{hy_i(t)}{\tau_-} + h\sum_{f}\delta\left(t - t_i^f\right), \tag{10}$$

where the x_j and y_i are the low-pass filtered version of the presynaptic spike train and postsynaptic spike train, respectively, τ_+ and τ_- are the time constants for the decay of x_j and y_i, h denotes time step. To reduce the hardware resource cost by using the SC technique, Eqs. 9 and 10 can also be simplified as

$$x_j(t+1) = x_j(t)\,A_j + h\sum_{f}\delta\left(t - t_j^f\right), \tag{11}$$

$$y_i(t+1) = y_i(t)\,A_i + h\sum_{f}\delta\left(t - t_i^f\right), \tag{12}$$

where $A_j = 1 - h/\tau_+$, $A_i = 1 - h/\tau_-$. When a presynaptic spike arrives, the trace x_j is increased. Equation 5 can be simplified as $\frac{dw_{ij}}{dt} = -B_i\,y_i(t)$. Then according to the Euler

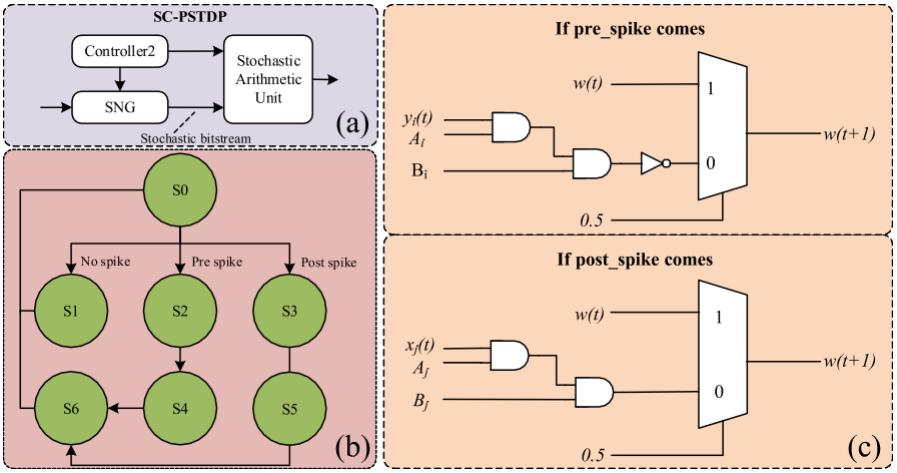

Fig. 3 a The overall architecture of SC-PSTDP; **b** Controller2; **c** Stochastic arithmetic unit (All input and output data are stochastic bitstreams)

method, it can be simplified as $w_{ij}(t+1) - w_{ij}(t) = -B_i y_i(t)$. Similarly, when a postsynaptic spike arrives, the trace y_i is increased. Equation 5 can be simplified as $\frac{dw_{ij}}{dt} B_j x_j(t)$, then $w_{ij}(t+1) - w_{ij}(t) = B_j x_j(t)$ can be obtained. Thus, the weight change Δw can be described as

$$\Delta w1(t+1) = y_i(t) B_i, \tag{13}$$

$$\Delta w2(t+1) = x_j(t) B_j, \tag{14}$$

where $\Delta w1$ and $\Delta w2$ denote the synaptic weight changes, and they only depend on the time difference of pre- and post-synaptic spikes. B_i and B_j represent amplitudes. The synaptic weight is obtained by

$$w(t+1) = w(t) - \Delta w1(t), \tag{15}$$

$$w(t+1) = w(t) + \Delta w2(t). \tag{16}$$

If a presynaptic spike arrives, the synaptic weight is depressed by Eq. 15. Similarly, it is potentiated by Eq. 16. In this work, the SC technique is used to calculate the Eqs. 11 to 16. The overall architecture of SC-PSTDP is shown in Fig. 3a, which contains three components: a controller 2, an SNG, and a stochastic arithmetic unit.

Specifically, the stochastic arithmetic unit is responsible for logic operations, which is shown in Fig. 3c. x_j and y_i denote the trace of pre- and post-synaptic spikes, A_j and A_i represent the attenuation coefficients of pre- and post-synaptic spikes trace, respectively, B_i and B_j are amplitudes. $w(t)$ describes the synaptic weight at time t. 0.5 represents a stochastic bitstream that the number of 1's is 50% in the bitstream (e.g. 01011010...). All the input and output data in Fig. 3c are stochastic bitstreams. As can be seen that only four AND-gates, a NOT-gate and two multiplexers are required in the SC-PSTDP learning rule. Figure 3b describes the controller 2 in detail. According to Eqs. 11 to 16, seven states are required to control the implementation of the SC-PSTDP. The function of each state is presented as follows: The system goes to state S0 where the initial values of SC-PSTDP are set. If a presynaptic spike arrives (i.e., presynaptic spike signal is high), the system is changed to S2 where the value of the presynaptic spike trace is increased. If a postsynaptic spike is detected,

the system is changed to S3 where the value of the postsynaptic spike trace is increased. If there is no spike coming, the next state is S1 where the traces of pre- and post-synaptic spikes and synaptic current decay exponentially. If the trace value of pre- or post-synaptic spike is increased, the system is switched from S2 to S4 or S3 to S5. In state S4 or S5, weight change is calculated. Then when the weight change is calculated completely, the system is turned to S6 where the weight is updated. After all the computations are completed, the system returns to the S0 state. In this way, the synaptic weight has been updated once.

5 Results

In this section, the proposed SC-SNN hardware system with LFSR of three different precisions is implemented using Verilog Hardware Description Language and verified on the FPGA device. Several performance metrics (Root Mean Square Error (RMSE), NRMSE, correlation, and hardware resources) are used to measure the SC-SNN. Experimental results of the proposed system are given and compared with others.

5.1 Performance Evaluation

A. Performance of SC-IF The performance metrics of SC-IF mainly include the RMSE, NRMSE, and correlation, which represent the similarity between the proposed SC-IF hardware and conventional IF neuron. Specifically, the RMSE and NRMSE are calculated by

$$RMSE(V_{err}) = \left(\frac{1}{M}\sum_{i=1}^{M} V_{err}^2\right)^{1/2}, \tag{17}$$

$$NRMSE(V_{max}, V_{min}) \frac{RMSE}{V_{max} - V_{min}}, \tag{18}$$

where V_{err} (V_{sc} V_{ori}), V_{sc} is the membrane potential of the SC-IF neuron, V_{ori} denotes the membrane potential of the original IF neuron model, M denotes the number of available data points in the selected region, V_{max} , and V_{min} are the maximum and minimum membrane potential values of the original model. Moreover, correlation shows the strength and direction of the linear relationship between two random variables in probability theory and statistics. The correlation between SC-IF neuron and the original IF neuron model can be described as

$$corr(V_{sc}, V_{ori}) = \frac{cov(V_{sc}, V_{ori})}{\sigma_{sc}\sigma_{ori}} = \frac{\sum_{i=1}^{M} (V_{scerr})(V_{orierr})}{\left(\sum_{i=1}^{M} (V_{scerr})^2\right)^{1/2} \left(\sum_{i=1}^{M} (V_{orierr})^2\right)^{1/2}}, \tag{19}$$

$= \quad - \overline{\quad} \qquad = \quad - \overline{\quad} \quad \overline{\quad} \qquad \overline{\quad}$

where V_{scerr} V_{sc} V_{sc}, V_{orierr} V_{ori} V_{ori} , V_{sc} and V_{ori} are the average values

of membrane potential of SC-IF and IF neuron, respectively. In addition, the time step, threshold, resting potential and other parameters used in the SC-IF are shown in Table 1 . Figure 4 shows the membrane potential waveforms of SC-IF and IF neurons. v_{th} represents the threshold of the IF neuron. The black line represents the experimental results of the original IF neuron, while the red line shows the experimental results of SC-IF. The experiment shows that the $RMSE\ (V_{err})$ and $NRMSE\ (V_{max}, V_{min})$ between

Table 1 The parameters of the proposed SNN structure

Parameters	Description	Value
C	Constant	100
B_i	Amplitude	0.3994
B_j	Amplitude	0.3994
h	Time step	0.1ms
V_{th}	Threshold	0.9mV
V_{rest}	Resting potential	0mV
R_m	Membrane resistance	10M¨
τ	Synapse time constant	10ms
τ_+	Trace x_j time constant	10ms
τ_-	Trace y_i time constant	10ms
τ_m	Membrane potential time constant	10ms

Fig. 4 The membrane potential of SC-IF and IF neurons

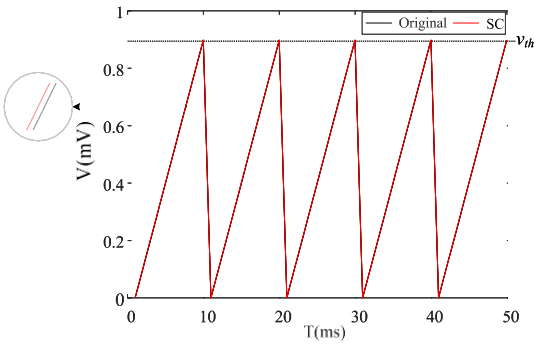

Table 2 Performance comparisons of SC-PSTDP with different bit lengths

Model	SC-PSTDP8	SC-PSTDP10	SC-PSTDP12
Slice LUTs	171	202	253
Slice registers	259	310	371
RMSE	0.2364	0.0165	0.0046
NRMSE	0.1110	0.0352	0.0097
CORR(%)	96.73	99.86	99.98

the proposed SC-IF with 12-bit length and original neurons are only 0.13% and 0.17%, respectively. Besides, its correlation is 99.99%. Overall, these evaluation metrics confirm that the SC-IF neuron of the SC-SNN has similar behaviour to the original IF neuron.

B. Performance of SC-PSTDP The accuracy of the SC is affected by the length of LFSR [47]. It is particularly important to achieve a trade-off between the hardware resource consumption and accuracy. Thus, the SC-PSTDP with the LFSR of different bit lengths (e.g., 8-bit, 10-bit and 12-bit) is implemented. Particularly, similar to SC-IF, the RMSE, NRMSE, correlation and hardware resource consumption are used to evaluate the performance of SC-PSTDP. Table 2 shows performance comparisons of SC-PSTDP with different bit lengths. Among them, the SC-PSTDP8 denotes the proposed SC-PSTDP

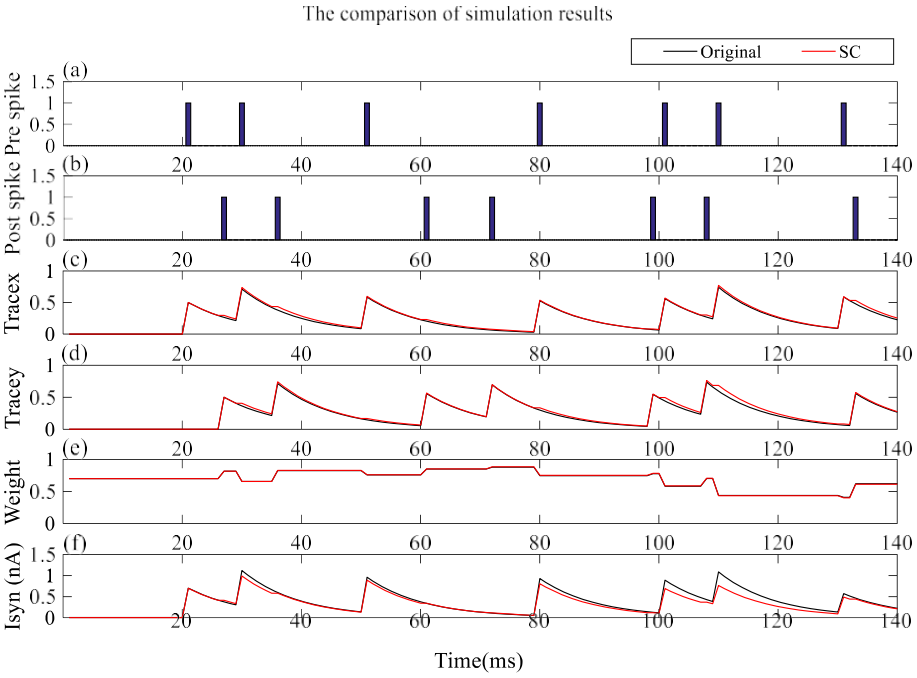

Fig. 5 The comparisons of results between the original PSTDP and the SC-PSTDP, the closer the better. **a** The presynaptic spike; **b** The postsynaptic spike; **c** The trace of the presynaptic spike; **d** The trace of the postsynaptic spike; **e** The synaptic weight; **f** The synaptic current

with the LFSR of 8-bit length. 171 LUTs and 259 registers are consumed, and its correlation is 96.73%. For the SC-PSTDP12, 253 LUTs and 371 registers are needed, and its RMSE is 0.46%, NRMSE is 0.97%, correlation is 99.98%, which are greater than the approach of [49, 50]. It can be seen that the improvement of precision leads to increasing amount of hardware resources. This is because the increase in the sequence length of LFSR leads to more hardware resources. To achieve a trade-off between the accuracy and hardware resource consumption, this experiment adopts the LFSR of 12-bit length to generate a random sequence for the SC-PSTDP and the SC-IF, and its sequence length is 2^{12} ~~1~~ 4095, i.e., a stochastic number is generated every 4095 clock cycles. The frequency of proposed hardware system is 100 MHz, it takes 40 us to generate a random number, i.e., the generation rate of random number is 25 KHz. The pre- and post-synaptic spikes, the traces of pre- and post-synaptic spikes, weight, and synaptic current of the proposed SC-PSTDP with 12-bit length are simulated on the FPGA device. Figure 5 shows the experimental results. The black line represents the original PSTDP, and the red line represents the SC-PSTDP. As can be seen that the results of two simulations are very similar, which shows that the proposed SC-PSTDP can maintain the ability of the PSTDP to adjust the synaptic weight. In addition, the synaptic weight change pro- duced by the time difference between pre- and post-synaptic spikes is shown in Fig. 6 a. $Ot\ t_{\overline{post}}\ t_{pre}$ denotes the time difference of pre- and post-synaptic spikes. The synaptic weight is potentiated if $Ot > 0$, otherwise it is depressed. Figure 6b further shows the error of weight more clearly. It shows weight error reaches a maximum value

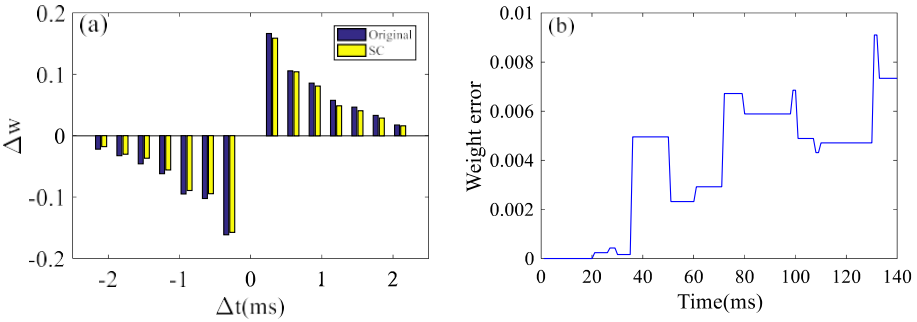

Fig. 6 The weight change of PSTDP. **a** The comparison of weight change of original PSTDP and SC-PSTDP; **b** The absolute value of the difference between original PSTDP and SC-PSTDP

Table 3 Utilized resources to implement SC-IF neuron and original IF neuron

Resource	SC-IF (This work)	Original
Slice LUTs	83	56
Slice registers	134	51
DSPs	0	2

Table 4 Utilized resources to implement SC-PSTDP and original PSTDP model

Resource	SC-PSTDP (This work)	Original
Slice LUTs	253	135
Slice registers	371	140
DSPs	0	10

of 0.0091 at 130 ms. Overall, the SC-PSTDP is very close to the original method of regulating the synaptic weight. These further show the learning rule of SNNs is maintained in the proposed SC-SNN. Furthermore, the area and power consumption of the SC-SNN are evaluated, which follows the standard ASIC cell design flow based on a SAED 90 nm CMOS technology. The experiment shows that the hardware area of SC-SNN is 27,565 μm^2 and the power consumption is 2.32 mW. Overall, the SC-PSTDP is very close to the original method of regulating the synaptic weight. These further show the learning rule of SNNs is maintained in the proposed SC-SNN.

5.2 Comparison of Resource Consumption

The hardware cost and power consumption are two important metrics in the hardware SNN [15–17]. The proposed SC-SNN is implemented on the FPGA device. The utilized resources for hardware implementation of the SC-IF and original IF are presented in Table 3 . 83 LUTs and 134 registers are consumed by the proposed SC-IF. In addition, the hardware resource consumption of the SC-PSTDP and original PSTDP implementations is presented in Table 4 . As can be seen, the proposed SC-PSTDP needs 253 LUTs and 371 registers. Though the proposed SC-IF and SC-PSTDP consume more LUTs and registers than the original implementation, they do not require the limited and expensive DSPs resources on the FPGA devices. This is because the arithmetic operations in the original implementation

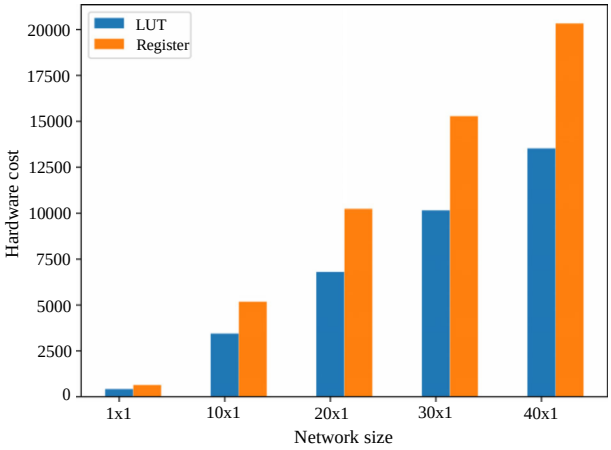

Fig. 7 Comparisons of hardware resources under different network sizes of SC-SNN

are replaced by the logical gates in the SC and the logic gates are implemented by the resources of LUTs and registers on the FPGA device.

To further verify the scalability of SC-SNN, the proposed SC-IF and SC-PSTDP are used to build the SC-SNN of different network sizes, and they are implemented on FPGA device. The hardware resources consumption is shown in Fig. 7. It can be seen the number of LUTs and registers increase as the network size increases. The SC-SNN with the input layer containing 40 SC-IF neurons and the output layer containing one SC-IF neuron requires 13,523 LUT and 20,334 registers. However, they still do not consume the DSP blocks which are very expensive for the FPGA devices. Thus, the proposed SC-SNN can be applied in large-scale SNNs.

Moreover, in Table 5, the LUTs, registers, DSP blocks, max speed, NRMSE and device descriptions of SC-SNN are provided and compared with other works. The resource consumption of the same network size is calculated and compared. The approach of [41] contains a neuron and a synapse. Compared to it, the LUTs consumption of the proposed work is reduced by 58%, and registers are reduced by 65.6%. In [17], the two-layer network is designed with the input layer of 20 neurons and the output layer of a single neuron. If using the proposed method to build the same network in [17], 6,803 LUTs and 10,234 registers are consumed, i.e., the LUTs are reduced by 4%, and registers are reduced by 1.3%. In [29], the SNN consists of 48 input neurons, three output neurons, and 144 synapses. In [30], the input layer of SNN consists of 25 dummy neurons, the hidden layer consists of five LIF neurons, and the output layer contains one LIF neuron. The number of synapses is 130 as the full connection is adopted in the SNN network. In [31], the 510 neurons are included in the SNN with binary FORCE learning. In [32], 100 neuron cores are included based on the Euler method and RK3 method. If the proposed method is used to build the same networks in [29–32], 40,665 LUTs and 60,258 registers, 35,463 LUTs and 52,384 registers, 219,430 LUTs and 328,040 registers, 67,200 LUTs and 101,000 registers are consumed, respectively. This work consumes more LUTs and registers compared with [29–32], however it does not consume the DSP blocks which are expensive for the FPGA devices. In [33], the synapse model is designed by the CORDIC method, and it consumes 373 LUTs, 138 registers and three DSP blocks. The SC-Synapse model designed in this paper only needs 253 LUTs and 371 registers, and it does not require DSP blocks. Besides, the max speed of the SC-SNN is 191.1 MHz, which is

Table 5 Resource consumption comparisons between SC-SNN and other works

Approach	# of neurons/synapses	LUTs	Registers	DSPs	Max speed (MHz)	NRMSE (%)	Device
Liu [41]	1/1	800	1468	0	100	NA	Zynq-7000
SC-SNN	1/1	336	505	0	191.1	0.97	Zynq-7000
Heidarpur [17]	21/20	7088	10376	0	84.1	0.003	Spartan-6
SC-SNN	21/20	6803	10234	0	191.1	0.97	Zynq-7000
Zhang [29]	51/144	13862	5487	144	178	NA	Virtex-7
SC-SNN	51/144	40665	60258	0	191.1	0.97	Zynq-7000
Farsa [30]	31/130	11339	1023	124	189.1	3.53	Virtex-6
SC-SNN	31/130	35463	52384	0	191.1	0.97	Zynq-7000
Akbarzadeh [31]	510/700	8077	2274	42	170.7	NA	Artix-7
SC-SNN	510/700	219430	328040	0	191.1	0.97	Zynq-7000
Guo [32]	200/200	85961	66163	400	100	NA	Virtex-7
SC-SNN	200/200	67200	101000	0	191.1	0.97	Zynq-7000
Wu [33]	0/1	373	138	3	303.4	NA	Zynq-7000
SC-SNN	0/1	253	371	0	191.1	0.97	Zynq-7000
Çağdaş [51]	1/0	267	38	11	190	NA	ZCU104
SC-SNN	1/0	83	134	0	191.1	0.97	Zynq-7000
Guo [52]	100/0	10163	3159	21	100	NA	Virtex-7
SC-SNN	100/0	8300	13400	0	191.1	0.97	Zynq-7000

higher than other works except the approach of [33]. Though the NRMSE in [30] is smaller than this paper, it consumes more LUTs and registers. Besides, the max speed in [30] is 84.1 MHz, which is smaller than this work. In the approach of [51], the Izh neuron module is implemented, where 38 registers, 267 LUTs and 11 DSPs are consumed for the unfolded architecture. If the proposed method is used to implement the neuron module, 83 LUTs and 134 registers are needed, i.e., the LUTs are reduced by 68.9%. Though the consumed registers are more than the work of [51], the proposed method does not consume DSP blocks. In the approach of [52], a neuromorphic hardware containing 100 neurons is proposed, which uses 10163 LUTs, 3159 registers and 21 DSPs. If the proposed method is used to build the same network in [52], 8300 LUTs, 13400 registers are consumed, i.e., the LUTs are reduced by 18.3% and no DSP blocks are required.

6 Conclusion

In this paper, a novel SNN hardware architecture by using the SC technique is proposed. The conventional arithmetic operations (e.g. multipliers, adders, and subtractors) are replaced by the stochastic bitstreams to reduce the hardware cost and power consumption. The proposed SC-SNN is implemented on the FPGA device, and the RMSE, NRMSE and curve correlation are used for performance evaluation, where the NRMSE between the proposed hardware SC-SNN and others is only 0.0097. In addition, the hardware resource consumption is compared with the previous approaches. Results show that only 323 LUTs and 491 registers are consumed by the proposed SC-SNN containing two neurons and one synapse, and most importantly the proposed work does not require the DSP blocks (which are very expensive resources for FPGA devices). The proposed work achieves a trade-off between the performance and the hardware resources. Therefore, the scalability of the hardware SNN is improved by using the SC technique. Future work will further optimize the network hardware architecture.

Acknowledgements This research is supported by the National Natural Science Foundation of China under Grant 61976063, the Guangxi Natural Science Foundation under Grant 2022GXNSFFA035028, research fund of Guangxi Normal University under Grant 2021JC006, the AI+Education research project of Guangxi Humanities Society Science Development Research Center under Grant ZXZJ202205.

References

1. Liu J, Huang Y, Luo Y, Harkin J, McDaid L (2019) Bio-inspired fault detection circuits based on synapse and spiking neuron models. Neurocomputing 331(1):473–482
2. Auge D, Hille J, Mueller E, Knoll A (2021) A survey of encoding techniques for signal processing in spiking neural networks. Neural Process Lett 5(5):1–18
3. Luo Y, Wan L, Liu J, Harkin J, Cao Y (2018) An efficient, low-cost routing architecture for spiking neural network hardware implementations. Neural Process Lett 48(3):1777–1788
4. Singh AK, Saraswat V, Baghini MS, Ganguly U (2022) Quantum tunneling based ultra-compact and energy efficient spiking neuron enables hardware snn. IEEE Trans Circuits Syst I Regul Pap 13(6):1–13
5. Morrison A, Diesmann M, Gerstner W (2008) Phenomenological models of synaptic plasticity based on spike timing. Biol. Cybern. 98(6):459–478
6. Quintana FM, Perez-Pena F, Galindo PL (2022) Bio-plausible digital implementation of a reward modulated stdp synapse. Neural Comput Appl 1(1):1–12
7. Daddinounou S, Vatajelu EI (2022) Synaptic control for hardware implementation of spike timing dependent plasticity. In: International symposium on design and diagnostics of electronic circuits and systems (DDECS), pp 106–111

8. Liu J, Lu H, Luo Y, Yang S (2021) Spiking neural network-based multi-task autonomous learning for mobile robots. Eng Appl Artif Intell 104(104):362
9. Tavanaei A, Maida A (2019) Bp-stdp: approximating backpropagation using spike timing dependent plasticity. Neurocomputing 330:39–47
10. Peterson DG, Nawarathne T, Leung H (2022) Modulating stdp with back-propagated error signals to train snns for audio classification. IEEE Trans Emerg Topics Comput Intell 5(1):1–12
11. Pani D, Meloni P, Tuveri G, Palumbo F, Massobrio P, Raffo L (2017) An FPGA platform for real-time simulation of spiking neuronal networks. Front. Neurosci. 11(2):90–103
12. Neil D, Liu SC (2014) Minitaur, an event-driven FPGA-based spiking network accelerator. IEEE Trans Very Large Scale Integr Syst 22(12):2621–2628
13. Wijesinghe P, Ankit A, Sengupta A, Roy K (2018) An all-memristor deep spiking neural computing system: a step toward realizing the low-power stochastic brain. IEEE Trans Emerg Top Comput Intell 2(5):345–358
14. Babacan Y, Yesil A, Tozlu OF, Kacar F (2022) Investigation of stdp mechanisms for memristor circuits. AEU Int J Electron Commun 151(1):154–230
15. Wang R, Thakur CS, Hamilton TJ, Tapson J, van Schaik A (2016) A stochastic approach to STDP. In: International Symposium on Circuits and Systems, pp 2082–2085
16. Gomar S, Ahmadi M (2018) Digital realization of PSTDP and TSTDP learning. In: International Joint Conference Neural Networks, pp 1–5
17. Heidarpur M, Ahmadi A, Ahmadi M, Rahimi Azghadi M (2019) CORDIC-SNN: on-FPGA STDP learning with Izhikevich neurons. IEEE Trans Circuits Syst I Regul Pap 66(7):2651–2661
18. Lammie C, Hamilton TJ, van Schaik A, Rahimi Azghadi M (2019) Efficient FPGA implementations of pair and triplet-based STDP for neuromorphic architectures. IEEE Trans Circuits Syst I Regul Pap 66(4):1558–1570
19. Sartori J, Kumar R (2011) Stochastic computing. Found Trends Electron Des Autom 5(3):153–210
20. Brown BD, Card HC (2001) Stochastic neural computation I: computational elements. IEEE Trans Comput 50(9):891–905
21. Li P, Lilja DJ (2011) Using stochastic computing to implement digital image processing algorithms. In: International on conference computer design, pp 154–161
22. Sarkis G, Hemati S, Mannor S, Gross WJ (2013) Stochastic decoding of LDPC codes over GF(q). IEEE Trans Commun 61(3):939–950
23. Canals V, Morro A, Oliver A, Alomar ML (2016) A new stochastic computing methodology for efficient neural network implementation. IEEE Trans Neural Networks Learn Syst 27(3):551–564
24. Nguyen DA, Ho HH, Bui DH, Tran XT (2018) An efficient hardware implementation of artificial neural network based on stochastic computing. In: Conf. Information and Computer Science pp 237–242
25. Lee C, Panda P, Srinivasan G, Roy K (2018) Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. Front Neurosci 12:435
26. Yang Z, Huang Y, Zhu J, Ye TT (2020) Analog circuit implementation of lif and stdp models for spiking neural networks. In: Proceedings of the 2020 on great lakes symposium on VLSI, pp 469–474
27. Panwar N, Rajendran B, Ganguly U (2017) Arbitrary spike time dependent plasticity (stdp) in memristor by analog waveform engineering. IEEE Electron Device Lett 38(6):740–743
28. Ismail AA, Shaheen ZA, Rashad O, Salama KN, Mostafa H (2018) A low power hardware implementation of izhikevich neuron using stochastic computing. In: 2018 30th international conference on microelectronics (ICM), pp 315–318. IEEE
29. Zhang G, Li B, Wu J, Wang R, Lan Y, Sun L, Lei S, Li H, Chen Y (2020) A low-cost and high-speed hardware implementation of spiking neural network. Neurocomputing 382(1):106–115
30. Farsa EZ, Ahmadi A, Maleki MA, Gholami M, Rad HN (2019) A low-cost high-speed neuromorphic hardware based on spiking neural network. IEEE Trans Circuits Syst II Express Briefs 66(9):1582–1586
31. Akbarzadeh-Sherbaf K, Safari S, Vahabie AH (2020) A digital hardware implementation of spiking neural networks with binary FORCE training. Neurocomputing 412(1):129–142
32. Guo W, Yantir HE, Fouda ME, Eltawil AM, Salama KN (2021) Toward the optimal design and FPGA implementation of spiking neural networks. IEEE Trans Neural Netw Learn Syst 24(6):1–15
33. Wu J, Zhan Y, Peng Z, Ji X, Yu G, Zhao R, Wang C (2021) Efficient design of spiking neural network with stdp learning based on fast cordic. IEEE Trans Circuits Syst I Regular Pap 68(6):2522–2534
34. L Wan, Y Luo, S Song, J Harkin, J Liu (2016) Efficient neuron architecture for FPGA-based spiking neural networks. In: Signals and Systems Conference, pp 1–6
35. Liu J, Harkin J, Maguire LP, McDaid LJ, Wade JJ (2018) SPANNER: a self-repairing spiking neural network hardware architecture. IEEE Trans Neural Netw Learn Syst 29(4):1287–1300
36. Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117(4):500–544

37. Izhikevich EM (2004) Which model to use for cortical spiking neurons? IEEE Trans Neural Netw 15(5):1063–1070
38. Brunel N, Hakim V (1999) Fast global oscillations in networks of integrate-and-fire neurons with low firing rates. Neural Comput 11(7):1621–1671
39. Destexhe A, Mainen ZF, Sejnowski TJ (1994) Synthesis of models for excitable membranes, synaptic transmission and neuromodulation using a common kinetic formalism. J Comput Neurosci 1(3):195–230
40. Liu J, Harkin J, Maguire L, McDaid L, Wade J, McElholm M (2016) Self-repairing hardware with astrocyte-neuron networks. In: International symposium on circuits and systems, pp 1350–1353
41. Liu J, Liang Z, Luo Y, Huang J, Yang S (2019) Hardware tripartite synapse architecture based on stochastic computing. In: International symposium on theoretical aspects of software engineering, pp 81–85
42. Pfister JP (2006) Triplets of spikes in a model of spike timing-dependent plasticity. J Neurosci 26(38):9673–9682
43. Froemke RC, Dan Y (2002) Spike-timing-dependent synaptic modification induced by natural spike trains. Nature 416(6879):433–438
44. Gjorgjieva J, Clopath C, Audet J, Pfister JP (2011) A triplet spike-timing-dependent plasticity model generalizes the bienenstock-cooper-munro rule to higher-order spatiotemporal correlations. Proc Natl Acad Sci USA 108(48):19383–19388
45. Wang HX, Gerkin RC, Nauen DW, Bi GQ (2005) Coactivation and timing-dependent integration of synaptic potentiation and depression. Nat Neurosci 8(2):187–193
46. Alaghi A, Qian W, Hayes JP (2018) The promise and challenge of stochastic computing. IEEE Trans Comput Des Integr Circuits Syst 37(8):1515–1531
47. Alaghi A, Hayes JP (2013) Survey of stochastic computing. ACM Trans Embed Comput Syst 12(2):1–19
48. Hahn GD (1991) A modified Euler method for dynamic analyses. Int J Numer Methods Eng 32(5):943–955
49. Nouri M, Jalilian M, Hayati M, Abbott D (2017) A digital neuromorphic realization of pair-based and triplet-based spike-timing-dependent synaptic plasticity. IEEE Trans Circuits Syst II Express Briefs 65(6):804–808
50. Azghadi MR, Al-Sarawi S, Iannella, N, Abbott D (2012) Efficient design of triplet based spike-timing dependent plasticity. In: The 2012 International joint conference on neural networks (IJCNN), pp 1–7
51. Çağdaş S, Şengör NS (2022) A folded architecture for hardware implementation of a neural structure using izhikevich model. In: Pimenidis E, Angelov P, Jayne C, Papaleonidas A, Aydin M (eds.) Artificial neural networks and machine learning – ICANN 2022, pp 508–518. Springer Nature Switzerland
52. Guo W, Fouda ME, Eltawil AM, Salama KN (2022) Efficient hardware implementation for online local learning in spiking neural networks. In: 2022 IEEE 4th international conference on artificial intelligence circuits and systems (AICAS), pp 387–390