TYPE Original Research
PUBLISHED 28 September 2022
DOI 10.3389/fnins.2022.929644

Check for updates

OPEN ACCESS

EDITED BY
Lining Zhang,
Peking University, China

REVIEWED BY
Hongwei Mo,
Harbin Engineering University, China
Mutsumi Kimura,
Ryukoku University, Japan

*CORRESPONDENCE
Guosheng Yi
guoshengyi@tju.edu.cn

SPECIALTY SECTION
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

RECEIVED 27 April 2022
ACCEPTED 07 September 2022
PUBLISHED 28 September 2022

CITATION
Gao T, Deng B, Wang J and Yi G (2022) Highly efficient neuromorphic learning system of spiking neural network with multi-compartment leaky integrate-and-fire neurons. *Front. Neurosci.* 16:929644. doi: 10.3389/fnins.2022.929644

Highly efficient neuromorphic learning system of spiking neural network with multi-compartment leaky integrate-and-fire neurons

Tian Gao, Bin Deng, Jiang Wang and Guosheng Yi*

School of Electrical and Information Engineering, Tianjin University, Tianjin, China

A spiking neural network (SNN) is considered a high-performance learning system that matches the digital circuits and presents higher efficiency due to the architecture and computation of spiking neurons. While implementing a SNN on a field-programmable gate array (FPGA), the gradient back-propagation through layers consumes a surprising number of resources. In this paper, we aim to realize an efficient architecture of SNN on the FPGA to reduce resource and power consumption. The multi-compartment leaky integrate-and-fire (MLIF) model is used to convert spike trains to the plateau potential in dendrites. We accumulate the potential in the apical dendrite during the training period. The average of this accumulative result is the dendritic plateau potential and is used to guide the updates of synaptic weights. Based on this architecture, the SNN is implemented on FPGA efficiently. In the implementation of a neuromorphic learning system, the shift multiplier (shift MUL) module and piecewise linear (PWL) algorithm are used to replace multipliers and complex nonlinear functions to match the digital circuits. The neuromorphic learning system is constructed with resources on FPGA without dataflow between on-chip and off-chip memories. Our neuromorphic learning system performs with higher resource utilization and power efficiency than previous on-chip learning systems.

KEYWORDS

spiking neural network, multi-compartment LIF, neuromorphic system, dendrite-guided synaptic plasticity, on-chip learning system

Introduction

A spiking neural network (SNN) is considered the brain-inspired mechanism in computing. It has shown excellent efficiency in many fields. Compared with the huge acquisition of computation resources and memory bandwidth in artificial neural networks (ANNs), SNN provides a convenient way of integrating storage and computing, which contributes to high efficiency (Mead, 1990; Schuman et al., 2017; Davies et al., 2018; Bohnstingl et al., 2019). ANNs focus on continuous functions and gradient descent learning rules. SNNs learn through synaptic plasticity based on the sparse spikes of neurons. The synaptic weights in an SNN are updated based on local errors instead of global gradients backpropagated through layers, which is considered the key to neuromorphic hardware (Boybat et al., 2018; Stewart et al., 2020). Neuromorphic hardware systems have become the core of hardware acceleration and embedded systems (Vivancos et al., 2021).

However, two factors limit efficiency in computation and power. One is the high-throughout data transmission between off-chip and on-chip memory. Although high-performance memory systems have been proposed to optimize data transmission, the power consumption is irreducible (Lian et al., 2019; Vivancos et al., 2021). Dynamic random-access memory (DRAM) is usually used as the off-chip memory. Compared with a static random-access memory (SRAM), the power consumption of DRAM is significantly high. It costs 640pj to access 32 bits of data from DRAM but costs only 5pj to access 8 Kb of data from SRAM (Horowitz, 2014). This study uses a learning system to store weights in synapses between neurons. The integration of storage and computation has become the key to solving speed and energy consumption problems. This learning system is implemented on a field-programmable gate array (FPGA), which has rich memory and computation resources (Wang et al., 2016; Chang and Culurciello, 2017; Yang et al., 2018, 2020).

While implementing an on-chip learning system on FPGA, the computation of global gradients requires a large number of state variables stored in registers, which is very luxurious for the FPGA. A series of algorithms based on surrogate-gradient are proposed to replace global gradients backpropagated layer by layer from outputs (Zenke and Ganguli, 2017; Sacramento et al., 2018; Neftci et al., 2019; Nøkland and Eidnes, 2019; Kaiser et al., 2020; Debat et al., 2021; Singanamalla and Lin, 2021). Although local gradients based on loss functions for each layer reduce the memory requirement, the memory complexity is still O(NT), where N is the number of nodes and T is the number of time steps in the time window (Kaiser et al., 2020). This study applies a multi-compartment leaky integrate-and-fire (MLIF) model to a SNN, which contains the basal dendrite, the apical dendrite, and the soma. The plateau potential in the apical dendrite guides the updates of synaptic weights. State variables are no longer required to be stored all the time to calculate gradients. The memory complexity is reduced to O(N). In order to match the digital circuits, shift multipliers (shift MUL) and piecewise linear (PWL) are applied to replace multiplication and complex nonlinear operations.

Further, we perform the system to classify spike patterns and reproduce their frequency distribution. With noise applied to the outputs of each soma, we also analyze the robustness and effectiveness of the system. The architecture of the learning system realized in this study consumes fewer resources and achieves high evaluation accuracy.

Materials and methods

The architecture of spiking neural network and multi-compartment leaky integrate-and-fire model

The SNN used in this study is divided into three layers, including the input layer, hidden layer, and output layer, as shown in **Figure 1**. Spike pattern inputs are encoded into spike trains by Poisson filters. The MLIF model in the hidden layer includes three compartments representing the basal dendrite, apical dendrite, and soma. The basal dendrite receives spike trains from the input layer, and the apical dendrite receives spike trains backpropagated from the output layer. Compared with the single-compartment LIF model, the MLIF model used in this study provides independent channels for information transmission. Dendritic current is transmitted to the soma and converted to the new spike trains. The somatic voltage of the MLIF model is described as follows:

$$\begin{aligned} V^0(t+1) = \; & V^0(t) + (g_l(V_{res} - V^0(t)) + g_b(V^{0b}(t) - V^0(t)) \\ & + g_a(V^{0a}(t) - V^0(t)))/C_m, \qquad (1) \end{aligned}$$

where V^0 is the somatic voltage of the MLIF model in the hidden layer, and g_l is the leak conductance. C_m is the membrane capacitance, g_a and g_b are the conductance from apical dendrite

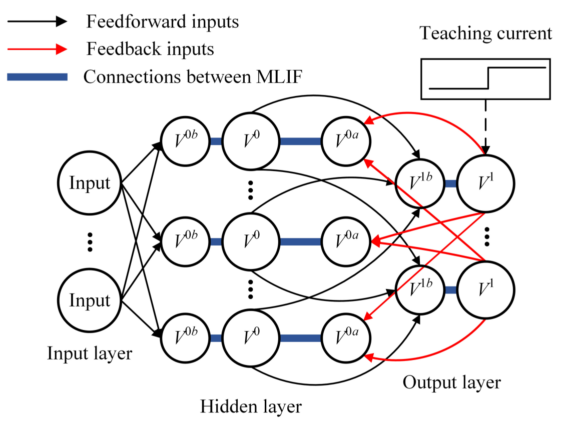

FIGURE 1
The architecture of spiking neural network (SNN) is depicted. There are three layers in the feedforward SNN. Input signals are encoded by the input layer and transmitted from the bottom to the top of SNN. An error is generated by the output layer with teaching current and backpropagated to the hidden layer with a dedicated channel different from the feedforward channel.

to soma and from the basal dendrite to soma, V_{res} is the resting potential, V^{0b} is the basal dendritic potential, and V^{0a} is the apical dendritic potential. The basal and apical dendritic potentials are given by weighted sums of the filtered spike trains:

$$V_i^{0b}(t) = \sum_j W_{ij}^0 x_j(t) + b_i^0, \tag{2}$$

$$V_i^{0a}(t) = \sum_j Y_{ij} s_j^1(t), \tag{3}$$

$$s(t) = \sum_k K(t - t_k), \tag{4}$$

$$K(t) = (e^{-t/\tau_l} - e^{-t/\tau_s})\Theta(t)/(\tau_l - \tau_s), \tag{5}$$

where x is the filtered spike trains from the input layer, s^1 is the filtered spike trains from the output layer, $K(t)$ is the kernel function, W^0 is the feedforward synaptic weights from the input layer to the hidden layer, Y is the synaptic feedback weights from the output layer to the hidden layer, b^0 is the bias of the MLIF model in the hidden layer, t_k is the kth spike time of the spike trains, τ_l and τ_s are long and short time constants, and $\Theta(t)$ is the Heaviside step function.

The firing rate of the MLIF model is described as follows:

$$\phi^0(t) = \phi_{\max}\sigma(V^0(t)) = \phi_{\max}\frac{1}{1 + e^{-V^0(t)}}, \tag{6}$$

where $\phi^0(t)$ is the firing rate of the MLIF model at time t, ϕ_{max} is the maximum firing rate, and $\sigma[V^0(t)]$ is the sigmoid function as a nonlinear activation function. The Poisson filter samples spike trains based on the firing rate of the MLIF model.

The MLIF model in the output layer includes two compartments representing the basal dendrite and soma. A teaching current is added to the soma of the MLIF model in the output layer, which contains an excitatory current and an inhibitory current. The somatic voltage of the MLIF model in the output layer is described as follows:

$$\begin{aligned} V^1(t+1) = \; & V^1(t) + (g_l(V_{res} - V^1(t)) + g_d(V^{1b} - V^1(t)) \\ & + I(t))/C_m, \end{aligned} \tag{7}$$

$$I(t) = g_E(E_E - V^1(t)) + g_I(E_I - V^1(t)), \tag{8}$$

$$V_i^{1b}(t) = \sum_j W_{ij}^1 s_j^0(t) + b_i^1, \tag{9}$$

where V^1 is the somatic voltage of the MLIF model in the output layer, g_d is the conductance from basal dendrite to soma, V^{1b} is the basal dendritic potential, $I(t)$ is the teaching current added to the MLIF model, g_E and g_I are the excitatory and inhibitory nudging conductance, s^0 is the filtered spike trains from the hidden layer, W^1 is the feedforward synaptic weights from the hidden layer to the output layer, and b^1 is the bias of the output layer.

Learning rules based on plateau potentials

Neuron models used in this study generate spikes based on the Poisson process. The training period consists of two phases: the forward phase and the target phase. Each phase continues 100 ms, and all synaptic weights and biases are updated at the end of the target phase. Notably, the synaptic feedback weights from the output layer to the hidden layer are fixed. In the forward phase, the teaching current I is kept at 0. In the target phase, $g_E = 1$ and $g_I = 0$ are applied to the output neuron according to the default label. The other neurons in the output layer are inhibited with $g_E = 0$ and $g_I = 1$. Synaptic weights and biases in the hidden layer are updated based on the plateau potentials of apical dendrites as follows:

$$W^0 = W^0 - \eta^0 P^0 \frac{g_b}{g_l + g_b + g_a}(\alpha^t - \alpha^f)\phi_{\max}\sigma'(\bar{V}^{0f}) \bullet \bar{x}^f, \tag{10}$$

$$b^0 = b^0 - \eta^0 P^0 \frac{g_b}{g_l + g_b + g_a}(\alpha^t - \alpha^f)\phi_{\max}\sigma'(\bar{V}^{0f}), \tag{11}$$

$$\sigma'(\bar{V}^{0f}) = \sigma(\bar{V}^{0f})(1 - \sigma(\bar{V}^{0f})), \tag{12}$$

$$\alpha^f = \sigma\left(\frac{1}{\Delta t}\int_{t_1 - \Delta t}^{t_1} V^{0a}(t)dt\right), \tag{13}$$

$$\alpha^t = \sigma\left(\frac{1}{\Delta t}\int_{t_2 - \Delta t}^{t_2} V^{0a}(t)dt\right), \tag{14}$$

where η^0 is the learning rate, P^0 is the scaling factor of the hidden layer, α^f and α^t are plateau potentials in the forward phase and the target phase, and $\bar{V}^{0f}$ is the average somatic voltage of the MLIF model in the hidden layer during a forward phase, $\bar{x}^f$ is the average input from the input layer during the forward phase, σ' is the derivative of the sigmoid function, "·" is the matrix multiplication, t_1 and t_2 are the duration of the forward phase and the target phase, and Δt is the unstable time. Synaptic weights and biases in the output layer are updated based on the error between predictions of SNN and labels as follows:

$$W^1 = W^1 - \eta^1 P^1 \frac{g_d}{g_l + g_d}(\bar{\phi}^{1t} - \bar{\phi}^{1f})\phi_{\max}\sigma'(\bar{V}^{1f}) \bullet \bar{s}^{0f}, \tag{15}$$

$$b^1 = b^1 - \eta^1 P^1 \frac{g_d}{g_l + g_d}(\bar{\phi}^{1t} - \bar{\phi}^{1f})\phi_{\max}\sigma'(\bar{V}^{1f}), \tag{16}$$

$$\bar{\phi}^{1f} = \phi_{\max}\sigma(\bar{V}^{1f}), \tag{17}$$

$$\bar{\phi}^t = \phi_{\max}\sigma(\bar{V}^t), \tag{18}$$

where η^1 is the learning rate, P^1 is the scaling factor of the output layer, $\bar{\phi}^{1f}$ and $\bar{\phi}^{1t}$ are the average firing rates of output neurons in the forward phase and target phase, and $\bar{s}^{0f}$ is the average filtered input from the hidden layer. We simulate a 784-500-10 SNN on a computer with MATLAB before implementing the learning system on the FPGA. The SNN is trained to classify the handwriting digits of the MNIST dataset and reaches an accuracy of 96.13% after 60 epochs. It suggests that the SNN learns satisfactorily based on the learning rule. The parameters used in this simulation are shown in **Table 1** (Guerguiev et al., 2017).

Linear approximated function

The sigmoid function is used to convert the somatic potential to a spiking frequency. Due to the exponential and reciprocal operations in the sigmoid function, it is not friendly for implementation on hardware (Lian et al., 2019; Heidarpur et al., 2020). These operations are usually realized by a large number of adders and multipliers on the FPGA. Shift and addition are the easiest operations realized on the FPGA. However, multiplication usually consumes decades more resources than addition. In this study, piecewise linearization (PWL) is used to replace nonlinear functions with several linear functions. In **Figure 2A**, the sigmoid function is divided into five PWL segments with two additional limit rules (Soleimani et al., 2012; Hayati et al., 2015). Accordingly, the PWL5 model of the sigmoid function is computed by the following equation:

$$\sigma_l(V) = \begin{cases} m_0 V + k_0, & V \le N_0 \\ m_1 V + k_1, & N_0 < V \le N_1 \\ m_2 V + k_2, & N_1 < V \le N_2, \\ m_3 V + k_3, & N_2 < V \le N_3 \\ m_4 V + k_4, & V > N_3 \end{cases} \tag{19}$$

where σ_l is the sigmoid function with PWL5, m_i and k_i are the slope and intercept of lines in PWL5, and N_i is the scope of the border. We exhaust N_i, m_i, and k_i to minimize the root-mean-square error (RMSE), which is computed by the following equation:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_{sof}(i) - X_{har}(i))^2}, \tag{20}$$

where N is the number of values in exhaustion, the steps of N_i, m_i, and k_i are 0.1, 0.01, and 0.01 in exhaustion. The sigmoid function with PWL5 is clipped to 0–1. In **Figure 2B**, the derivative of the sigmoid function is divided into six PWL segments, and it with PWL6 is clipped to 0–∞. The evaluation results of Sigmoid and their derivative functions are shown in **Table 2**. The evaluation criteria include RMSE, mean absolute error (MAE), and R-square (R^2). MAE and R^2 are computed as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|X_{sof}(i) - X_{har}(i)|, \tag{21}$$

$$R^2 = 1 - \frac{\sum_{I=1}^{N}(X_{sof}(i) - X_{har}(i))^2}{\sum_{I=1}^{N}(\bar{X}_{sof} - X_{sof}(i))^2}, \tag{22}$$

where $X_{sof}(i)$ and $X_{har}(i)$ are the results of simulation on software and implementation on FPGA at the ith iteration, N is the total number of iterations, and $\bar{X}_{sof}$ is the mean of $X_{sof}(i)$. The results of error evaluation with three indicators are shown in **Table 3**. With PWL models, only shift and addition operations are needed while calculating nonlinear functions.

TABLE 1 Parameter values of spiking neural network (SNN) (Guerguiev et al., 2017).

Parameters	Values	Parameters	Values
dt	1	g_a	0
φ_{max}	0.2	g_b	0.6
τ_l	10	g_l	0.1
τ_s	3	V_{res}	0
E_E	12	C_m	1
E_I	−12	Δt	30
g_d	0.6	η^0, η^1	0.01
P^0	$20/\varphi_{max}$	P^1	$20/\varphi_{max}{}^2$

Neuromorphic system architecture overview

The SNN used in this study comprises four parts: inputs, MLIF models, outputs, and synaptic plasticity modules. The learning system is implemented on the Altera Stratix V Advanced Systems Development Kit with Altera Stratix V 5SGXEA7N2F45C2N FPGA, which is available at https://github.com/TianGaoTJU/Learning-System-of-Spiking-Neural-Network-with-Multi-compartment-LIF-Neurons. This learning system on the FPGA is used to classify spike patterns and reproduce frequency distribution. To keep a balance between the accuracy and resource consumption of calculation, we used 24-bit fixed-point data for calculation. The value of data is computed as follows:

$$(-1)^{sign} \times (integer + fraction/2^{16}), \tag{23}$$

The 0–15th bits are the fraction part, the 16–22nd bits are integral, and the 23rd is the significant part. If the sign part is 0, the data is positive. If the sign part is one, it is negative.

Figure 3 shows the architecture of the learning system implemented on the FPGA. The controller contains a counter used as a system clock. The counter increases at the posedge of the system clock from 0 to 200. When the counter reaches 100, the ping-pong switch is sloped to the target side. When

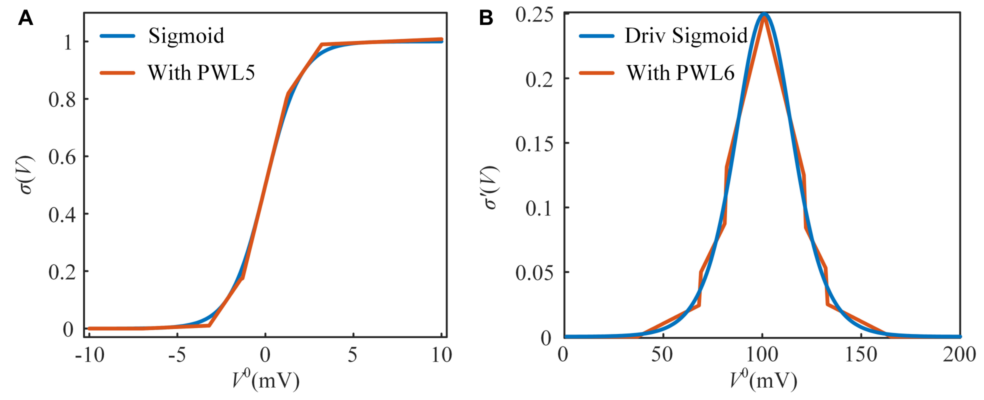

FIGURE 2
The approximated sigmoid functions and their derivative functions. **(A)** The sigmoid function and its approximated function with piecewise linear 5 (PWL5). **(B)** The derivative of the sigmoid function and its approximated function with piecewise linear 6 (PWL6).

the counter reaches 200, the ping-pong switch is sloped to the forward side. At the beginning of the forward phase, a clear signal is transmitted to all modules to reset the SNN to its initial state. There is a clear port, a forward phase enable port, and a target phase enable port in the synaptic plasticity module. In the forward phase, the forward phase enables signal = 1, and the target phase enables signal = 0. In the target phase, the settings are the opposite. More details of control signals are shown in Figure 6. The outputs of SNN are spike trains. Each output neuron has a dedicated counter that counts the soma spikes. The location of the output neuron with the most spikes is considered the prediction of SNN.

Implementation of the multi-compartment leaky integrate-and-fire model

Figure 4 shows the architecture of the MLIF model in the hidden layer. Figure 4A shows that the kernel function is calculated in a 10 ms time window. K (t) results in ten timesteps stored in shift registers shown as yellow blocks. Spike trains from the input layer are delayed from 1 to 10 ms and multiplied with ten K (t) values stored in registers. The z^{-1} block in gray is a register with a single bit and stores the spike at the latest timestep. K (t) values are multiplied by synaptic weights and accumulated in the Multiply Accumulate (MAC) module. The dendritic and somatic potential differences are multiplied by the conductance. The new somatic potential is calculated based on the sum of dendritic currents and leakage currents. The MLIF model in the output layer differs from that in the hidden layer, which only consists of the basal and somatic compartments.

In Figure 4B, we show the Poisson filter used to sample spikes from the firing rate of the MLIF model. The firing rate of each model is clipped to 0–200 Hz and distributed over ten frequency segments. Each frequency segment covers a range of 20 Hz. We use the random function in MATLAB to generate these ten Poisson spike trains on a computer. The parameter λ of the function is set as 0.02, 0.04, ..., 0.2, where λ is the average incidence of random events happening per millisecond. The spike trains are clipped to 0–1. We used the chi-square test to evaluate these Poisson spike trains. With a significance level of 0.05, these spike trains conform to the Poisson distribution. Then these spike trains are stored in shift registers as the yellow blocks in Figure 4B. The shift register outputs a bit of data at the constant address according to ϕ(t), and then the constant address is shifted right a bit. Figure 5 shows the somatic potential of the MLIF model implemented on FPGA and simulated in MATLAB on the computer with the same

TABLE 2 Coefficients of approximated functions.

Coefficients	Approximated function	
	Sigmoid	Derivative of sigmoid
(m_0, k_0, N_0)	(0.0078125, 0.05,3.4)	(0.0078125, 0.05,3.2)
(m_1, k_1, N_1)	(0.0625, 0.24,1.3)	(0.03125, 0.15,2)
(m_2, k_2, N_2)	(0.25, 0.5, 1.3)	(0.0625, 0.25, 0)
(m_3, k_3, N_3)	(0.0625, 0.76, 3.4)	(0.0625, 0.26, 2)
(m_4, k_4, N_4)	(0.0078125, 0.95, −)	(0.03125, 0.15, 3.2)
(m_5, k_5, N_5)	(−, −, −)	(0.0078125, 0.05,3.2)

TABLE 3 Error evaluation results.

	RMSE	MAE	R^2
Sigmoid	0.0101	0.0049	0.9995
Derivative of Sigmoid	0.0015	2.1643×10^{-4}	0.9948

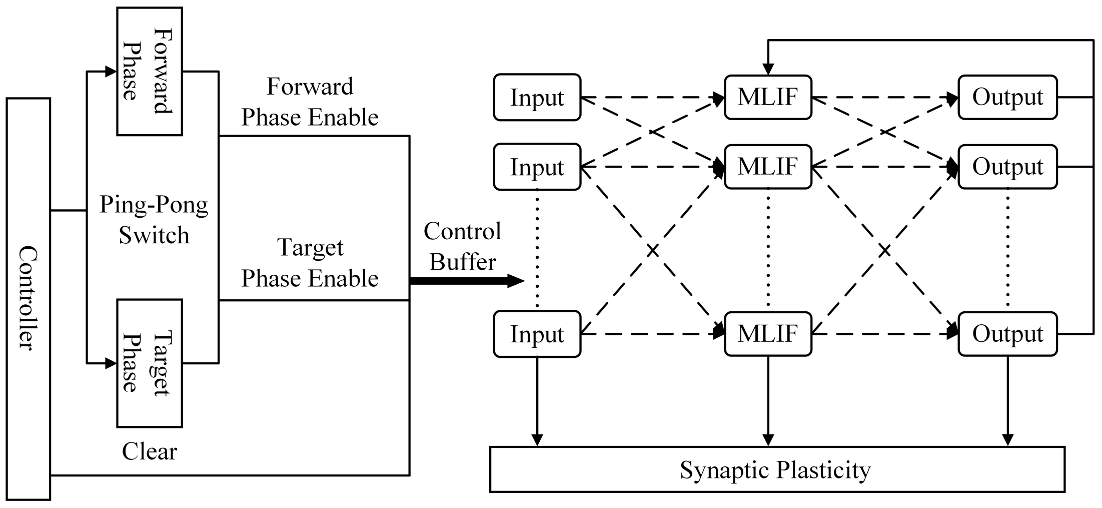

FIGURE 3
Overview of the spiking neural network (SNN) architecture implemented on the field-programmable gate array (FPGA). The controller is used as a system clock counter and enables signals. The ping-pong switch controls enable signals for the forward phase and target phase.

input. The results of the error evaluation of the MLIF model are as follows, i.e., RSME = 0.0057, MAE = 0.0042, and R^2 = 0.9944.

Implementation of synaptic plasticity

There are three input ports in the synaptic plasticity module implemented on FPGA: the forward phase enable, the target phase enable, and clear. In the controller, the forward phase enable and the target phase enable are accessed with a ping-pong switch. When the forward phase enable is set to 1, the target phase enable is set to 0. **Figure 6** shows the architecture of the synaptic plasticity module for b^0. The MUX block is a selector with three input ports and an output port. When the Sel. port is 0, the MUX block outputs the data on the first input port. When the Sel. port is 1, the MUX block outputs the data on the second input port. In the forward phase (without the first 30 ms), V^{0a} is transmitted into the multiplier and decreased to 1/70 of its original value. The sigmoid function with PWL5 converts the average potential of the apical dendrite to the firing rate in the forward phase. Due to the target phase enable being set to 0, the inputs of α^t are clamped at 0. In the target phase (without the first 30 ms), the target phase enable is set to 1, and the forward phase enable is set to 0. The input to α^f is clamped at 0, keeping the value. V^{0f} is calculated in the same way.

Two kinds of multipliers are implemented in the synaptic plasticity module. One is used for the multiplication of a constant and a variable. This multiplier is composed of shifters and adders. Another is the Shift MUL module, which multiplies two variables (Yang et al., 2021). The "Input a" and "Input b" are two variable inputs of the Shift MUL module. Note that the value of "Input a" is expected to be between 0 and 1 for the exact calculation, and thus the 16–23rd bits of "input a" are dropped, and the 0–15th bits are split into 16 single bits in a bus splitter. The outputs of the bus splitter are numbered from 16 to 1 according to the importance of each output port, from the least significant to the most significant bit. The "Input b" is shifted right in the shifter block according to the input number from 16 to 1. The MUX block is used to choose the input flow. The number 0 is added to the first data line of MUX. And the variable "Input b" is added to the second data line after being shifted right by the barrel shifter block. Outputs of the MUX blocks are added in a parallel adder. The sum of MUX blocks is considered the multiplication result. The same indicators are used to evaluate the errors of Shift MUL. RSME is 1.0703×10^{-4}, MAE is 9.8480×10^{-5}, and R^2 is 1.0000.

We also compare the compilation results of the shift MUL module and the multiplier summarized in **Table 4**. The multiplier module is an IP block provided by Quartus and DSP Builder software, which consumes 436 LUTs. A shift MUL module consumes 185 LUTs and 185 registers. One LUT and one register are integrated into a logic element (LE) of an FPGA. This result indicates that the Shift MUL module only requires 185/436 LEs of the multiplier module, suggesting high resource utilization efficiency.

Results

An $8 \times 10 \times 4$ SNN (with 120 synapses) is implemented on the FPGA to evaluate the learning system. Four spike patterns with different frequency distributions are input to the SNN. In these spike patterns, the maximum spike frequency is 200 Hz, and the minimum spike frequency is 20 Hz. **Figure 7** shows

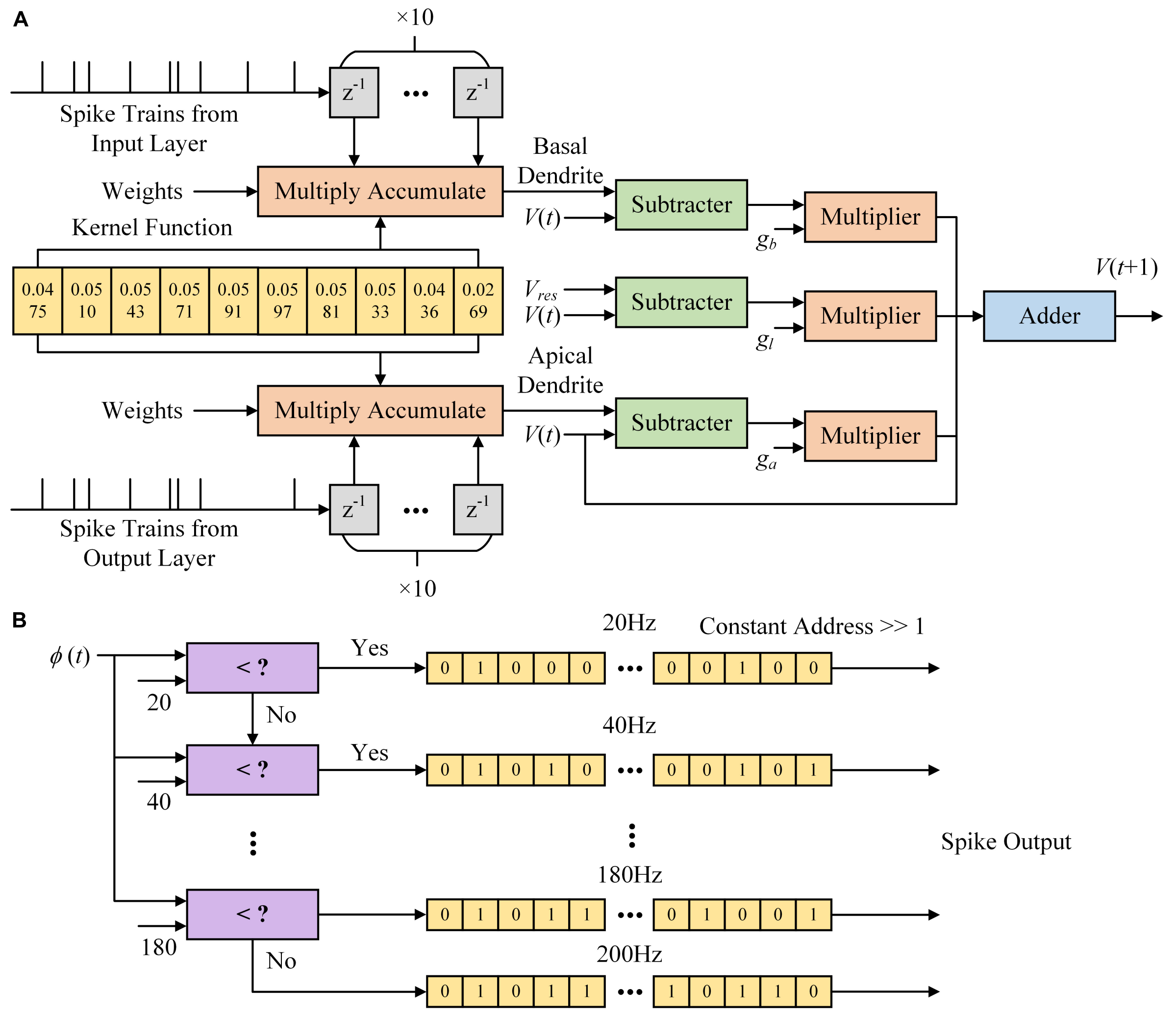

FIGURE 4
Multi-compartment leaky integrate-and-fire (MLIF) model implemented on the field-programmable gate array (FPGA). **(A)** The architecture of somatic voltage used in the MLIF model. **(B)** The architecture of the Poisson process is used to generate spike trains based on the firing rate of the MLIF model. Spike trains with different firing rates are stored in memory. When a spike output is read out, the constant address is shifted right a bit.

the inputs and outputs of the SNN on the FPGA with the SignalTap II Logic Analyzer. Spike trains in 200 Hz are input to "Data2in," "Data4in," "Data6in," "Data8in," and spike trains in 20 Hz are input to "Data1in," "Data3in," "Data5in," and "Data7in." Output ports "Data1out"–"Data4out" are spike trains output by MLIF models in the output layer. It can be found that the first neuron outputs the most spikes, so the prediction label of the SNN is 1. In this experiment, SNN realizes a high accuracy of 100.0% after 500 epochs. The resource utilization and power consumption are shown in Table 5. Compared with the other two previous works, our SNN performs higher Fmax, fewer look-up tables (LUTs), and less power consumption. The total power dissipation is 3.644 watts, as shown in Table 4. The core dynamic power dissipation is 2.2215 watts, and the static power dissipation is 989.08 mW. The I/O power dissipation is 433.44 mW. A 2 × 2 × 1 SNN with six synapses is implemented on FPGA based on the on-chip back-propagation learning algorithm designed by Vo (2017). Further, Mazouz and Bridges (2021) implement an 8 × 8 CNN based on a back-propagation learning algorithm. Table 5 shows the compilation results of neural networks implemented on FPGA, which are extended to 120 synaptic connections. The SNN implemented by Vo (2017) requires 5.6 times the logic elements of our implementation. The utilization of the CNN is 1.6 times that of the SNN in this study if it is extended to 120 neurons. These results suggest that our SNN performs more efficiently in resource utilization.

Another experiment is tested on the 8 × 10 × 8 SNN (with 160 synapses) on the FPGA. In this experiment, SNN is

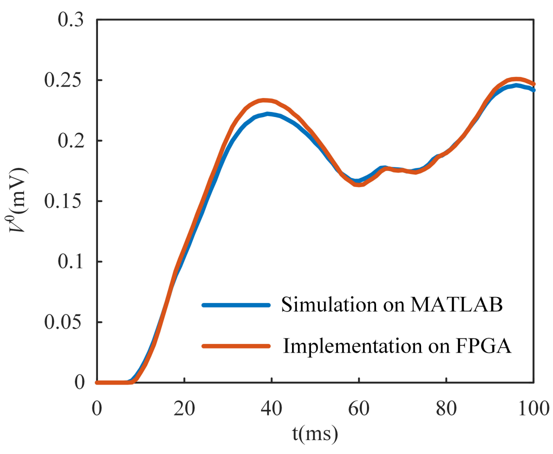

FIGURE 5
Multi-compartment leaky integrate-and-fire (MLIF) model implemented on field-programmable gate array (FPGA) is compared with it simulated in MATLAB with the same random input. There is only a small difference between the MLIF model in simulation and its implementation on the FPGA.

expected to reproduce the patterns with the specified frequency distribution (Mohemmed and Schliebs, 2012). In **Figure 8**, errors between random pattern input and frequency distribution output by SNN are evaluated in MAE and R2. **Figure 8A** shows the results of inputs without any noise. At the end of each training epoch, the SNN is tested and outputs spikes. Trained after 300 epochs, MAE and R^2 are 6.8904 and 0.9910. When compared with the range of 200 Hz, the SNN reproduces the input spike pattern accurately.

Further, we add a drop noise to spike trains and show the results trained after 300 epochs in **Figure 8B**, where the abscissa is the probability of dropping. When 50% of input spikes are decreased by noise, MAE increases to 32.46, and R^2 reduces to 69.55%. These results suggest that the learning system performs well under robustness to noise.

TABLE 4 Resource utilization of multipliers.

	LUT	Register/FF
Shift MUL	185	185
Multiplier	436	0

Discussion

This study applies dendrite-guided synaptic plasticity to the neuromorphic learning system. In contrast to the gradients backpropagated from top-down in ANNs, two dendrites in the MLIF model receive spike trains as inputs. Dendrites convert the input frequency into the potential. After a short oscillating period, the potential in dendrites reaches a plateau. If potentials in dendrites are at a high level, the presynaptic neuron fires at a high frequency. While training the SNN, the teaching current is added to neurons in the output layer. It could be transmitted to the apical dendrite, which guides the behavior of neurons in a lower layer. The gradients are not backpropagated from the output layer but calculated by the presynaptic activities and potentials in the dendrites. Benefiting from the reduction in memory complexity and synaptic plasticity, data used for learning could be stored on FPGA, not off-chip memory. It further contributes to the decrease in energy consumption in dataflow between on-chip and off-chip memories, which is considered a challenge in energy and speed. In order to match the learning system to the digital circuits, PWL and Shift MUL are applied to optimize nonlinear functions and multiplication between two fixed-point data.

As a computing unit independent of the soma, the dendrite enriches the possibility of a learning mechanism in the system. It has been proven that the prediction error between dendrites and soma can be used for synaptic plasticity. Urbanczik and Senn (2014) minimize the discrepancies between somatic firings and local dendritic potential to train their SNN. The error is generated continuously and drives synapses to be

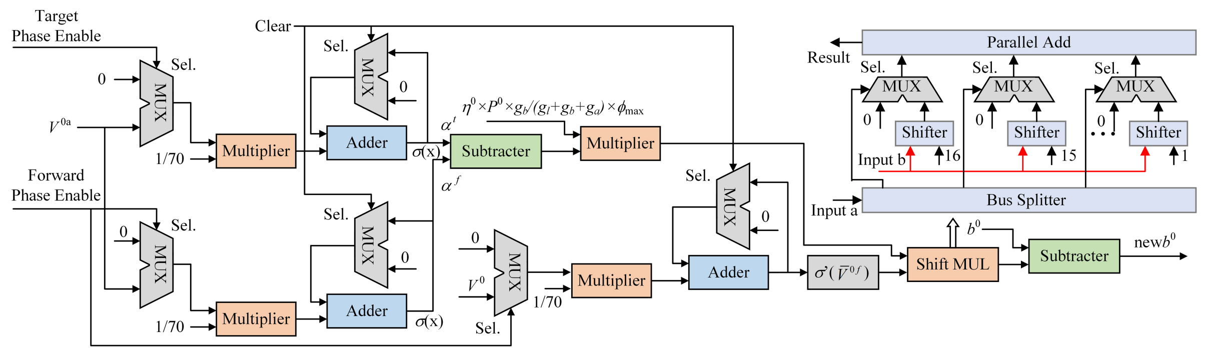

FIGURE 6
The architecture of synaptic plasticity. The synaptic plasticity module is implemented on the field-programmable gate array (FPGA) as an on-chip learning system. A shift multiplier (MUL) module is proposed to replace DSP blocks on the FPGA to reduce logical resource consumption.

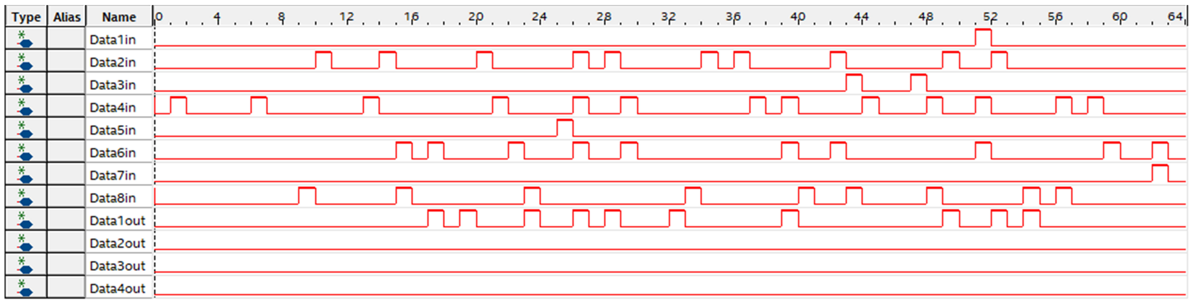

FIGURE 7
An example of spike pattern inputs and outputs of spiking neural network (SNN) on the field-programmable gate array (FPGA). Spike trains with a certain frequency distribution are input to SNN for classification. The output port "Data1out" outputs the most spikes, reflecting the prediction label according to this spike pattern.

TABLE 5 Resource utilization and comparison with previous works.

	LUT	Register/FF	Power	Fmax	Device
This study	196469	197154	3.644 W	216.54 MHz	Stratix V
Vo (2017)	1103160	584640	–	–	Spartan 3E
Mazouz and Bridges (2021)	274125	–	4.982 W	200 MHz	Zynq-7100

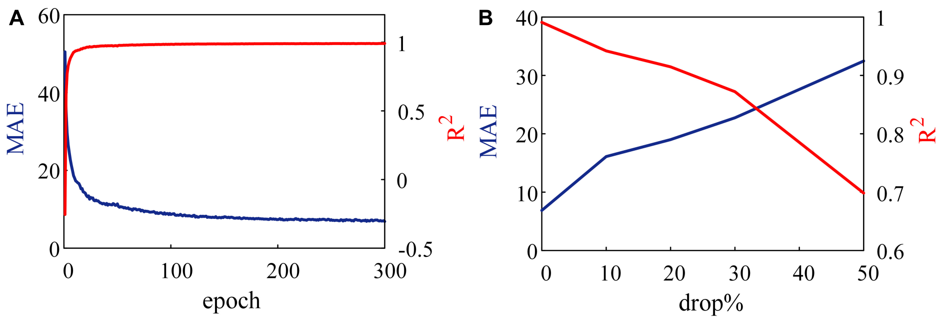

FIGURE 8
Results of reproducing spike patterns. **(A)** Without noise, spiking neural network (SNN) reproduces input patterns accurately. **(B)** Noise in the form of dropping with a constant probability is added to an SNN.

constantly updated. Sacramento et al. (2018) use the error between predictive activities from lateral interneurons and feedback activities to train the SNN. Their model does not need separate phases, and synaptic plasticity is driven by the local dendritic prediction errors continuously in time. These algorithms proved excellent and provided ways to solve credit assignment problems. However, continuous learning during training drives the synaptic plasticity module on the FPGA to work all the time at high speed. The lateral synaptic connections also increase the number of synaptic plasticity modules on FPGA. Although these two learning rules achieve extremely high accuracy in testing, we need a hardware-friendly learning rule to implement the SNN on FPGA. We divide the training period into two phases. Synaptic weights are updated at the end of the target phase. Only the plateau potentials in the apical dendrites are required to guide the updates of synaptic weights. The plateau potential is obtained by accumulating the potential in the apical dendrite, which consumes a fixed-point adder and two 24-bit registers. The power consumption of the learning system on FPGA is mainly generated by the transmission of spikes between neurons and the update of synaptic plasticity modules. We use the discrepancies between the plateau potentials of the apical dendrites in two phases to train the SNN, and weights are updated only once at the end of the target phase. The FPGA works at an extremely high speed compared to the CPU, but the memory and logical resources of the FPGA are scarce. Thus, high-efficient utilization of resources on the FPGA is the most important. Besides, while realizing the same learning system, the power consumption of FPGA is usually a few tenths of that of a GPU. Therefore, power consumption is usually considered the second most important standard for FPGA. We aim to realize a learning system with low power consumption and high resource utilization on FPGA. The SNN and architecture of implementation in this study are best suitable for this goal.

One of the main energy consumptions of the learning system occurs in data transmission. Spiking neurons generate spike trains, which only include 0 or 1 signal. Meanwhile, SNN transmits signals at 0 or 1 instead of continuous membrane potential. These 1-bit signals greatly reduce the requirements for bandwidth and energy during data transmission. After the dendrite of the MLIF model receives spike trains, the postsynaptic potential is triggered by a flip-flop (FF) without MAC operation. It has been shown that SNN works on FF and simple operations, not LUTs (Pei et al., 2019). In particular, the MLIF model used in this study generates spikes with a certain probability based on its membrane potential. As a result, the membrane potential is stored with less precision, and the neuromorphic system presents better robustness, as shown in **Figure 8**. Although neuromorphic learning systems are still being studies and have many limitations, embedded devices will present advantages in a broad category of applications with such neuromorphic learning systems.

Conclusion

In this study, a high-efficient neuromorphic learning system based on MLIF models is realized on a FPGA. We use the discrepancies between the plateau potentials in two phases to train the SNN. The synaptic weights are updated only once at the end of the target phase, which facilitates FPGA implementation with fewer memory resources and energy. The shift MUL module and PWL mechanism are applied instead of multiplication and complex nonlinear operations. The neuromorphic learning system is implemented on the Stratix V FPGA. For important units and modules in this learning system, error evaluation is applied based on computer simulations and hardware experiments. Results of resource utilization and performance in two tasks support the notion that the neuromorphic learning system works more efficiently.

Data availability statement

The original contributions presented in this study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

All authors listed have made a substantial, direct, and intellectual contribution to the work, and approved it for publication.

Funding

This work was supported by the National Natural Science Foundation of China (Grant nos. 62071324 and 62006170).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Bohnstingl, T., Scherr, F., Pehle, C., Meier, K., and Maass, W. (2019). Neuromorphic hardware learns to learn. *Front. Neurosci.* 13:483. doi: 10.3389/fnins.2019.00483

Boybat, I., Gallo, M. L., Nandakumar, S. R., Moraitis, T., Parnell, T., Tuma, T., et al. (2018). Neuromorphic computing with multi-memristive synapses. *Nat. Commun.* 9:2514. doi: 10.1038/s41467-018-04933-y

Chang, A., and Culurciello, E. (2017). "Hardware accelerators for recurrent neural networks on FPGA," in *Proceedings of the 2017 IEEE international symposium on circuits and systems (ISCAS)*, Baltimore, MD. doi: 10.1109/ISCAS.2017.8050816

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Joshi, P., Lines, A., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Debat, G., Chauhan, T., Cottereau, B. R., Masquelier, T., Paindavoine, M., and Baures, R. (2021). Event-based trajectory prediction using spiking neural networks. *Front. Comput. Neurosci.* 15:658764. doi: 10.3389/fncom.2021.658764

Guerguiev, J., Lillicrap, T. P., and Richards, B. A. (2017). Towards deep learning with segregated dendrites. *Elife Sci.* 6:22901. doi: 10.7554/eLife.22901

Hayati, M., Nouri, M., Haghiri, S., and Abbott, D. (2015). Digital multiplierless realization of two coupled biological Morris-Lecar neuron model. *IEEE Trans. Circuits Syst. Regul. Papers* 62, 1805–1814. doi: 10.1109/TCSI.2015.2423794

Heidarpur, M., Ahmadi, A., Ahmadi, M., and Azghadi, M. R. (2020). "CORDIC-SNN: On-FPGA STDP learning with Izhikevich neurons," in *Proceedings of the IEEE international symposium on circuits and systems (ISCAS)*, Seville. doi: 10.1109/TCSI.2019.2899356

Horowitz, M. (2014). "1.1 Computing's energy problem (and what we can do about it)," in *Proceedings of the 2014 IEEE international solid- state circuits conference (ISSCC)*, San Francisco, CA. doi: 10.1109/ISSCC.2014.6757323

Kaiser, J., Mostafa, H., and Neftci, E. (2020). Synaptic plasticity dynamics for deep continuous local learning (decolle). *Front. Neurosci.* 14:424. doi: 10.3389/fnins.2020.00424

Lian, X., Liu, Z., Song, Z., Dai, J., and Ji, X. (2019). High-performance fpga-based cnn accelerator with block-floating-point arithmetic. *IEEE Trans. Very Large Scale Integr. Syst.* 27, 1874–1885. doi: 10.1109/TVLSI.2019.2913958

Mazouz, A., and Bridges, C. P. (2021). Automated CNN back-propagation pipeline generation for FPGA online training. *J. Real Time Image Process.* 18, 2583–2599. doi: 10.1007/s11554-021-01147-2

Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636. doi: 10.1109/5.58356

Mohemmed, A., and Schliebs, S. (2012). SPAN: Spike pattern association neuron for learning spatio-temporal spike patterns. *Int. J. Neural Syst.* 22:1250012. doi: 10.1142/S0129065712500128

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

Nøkland, A., and Eidnes, L. H. (2019). Training neural networks with local error signals. *arXiv* [Preprint]. doi: 10.48550/arXiv.1901.06656

Pei, J., Deng, L., Song, S., Zhao, M., Zhao, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid Tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Sacramento, J., Costa, R. P., Bengio, Y., and Senn, W. (2018). "Dendritic cortical microcircuits approximate the back-propagation algorithm," in *Proceedings of the conference on neural information processing systems*, Montréal, QC. doi: 10.5555/3327546.3327550

Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., et al. (2017). A survey of neuromorphic computing and neural networks in hardware. *arXiv* [Preprint]. doi: 10.48550/arXiv.1705.06963

Singanamalla, S., and Lin, C. T. (2021). Spiking neural network for augmenting electroencephalographic data for brain computer interfaces. *Front. Neurosci.* 15:651762. doi: 10.3389/fnins.2021.651762

Soleimani, H., Ahmadi, A., and Bavandpour, M. (2012). Biologically inspired spiking neurons: Piecewise linear models and digital implementation. *IEEE Trans. Circuits Syst. Regul. Papers* 59, 2991–3004. doi: 10.1109/TCSI.2012.2206463

Stewart, K., Orchard, G., Shrestha, S. B., and Neftci, E. (2020). Online few-shot gesture learning on a neuromorphic processor. *IEEE J. Emerg. Sel. Top. Circuits Syst.* 10, 512–521. doi: 10.1109/JETCAS.2020.3032058

Urbanczik, R., and Senn, W. (2014). Learning by the dendritic prediction of somatic spiking. *Neuron* 81, 521–528. doi: 10.1016/j.neuron.2013.11.030

Vivancos, I. E., Sharify, S., Ly-Ma, D., Ameer, A., Mahmoud, M., Ciaran, B., et al. (2021). "Boveda: Building an on-chip deep learning memory hierarchy brick by brick," in *Proceedings of machine learning and systems 3 pre-proceedings*, San Jose, CA.

Vo, H. M. (2017). "Implementing the on-chip back-propagation learning algorithm on FPGA architecture," in *Proceedings of the international conference on system science & engineering*, Ho Chi Minh City. doi: 10.1007/s11265-005-4961-3

Wang, Y., Xu, J., Han, Y., and Li, H. (2016). "DeepBurning: Automatic generation of FPGA -based learning accelerators for the neural network family," in *Proceedings of the 53rd annual design, automation conference*, New York, NY. doi: 10.1145/2897937.2898003

Yang, S., Wang, J., Hao, X., Li, H., and Loparo, K. A. (2020). Bicoss Toward large-scale cognition brain with multigranular neuromorphic architecture. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 2801–2815. doi: 10.1109/TNNLS.2020.3045492

Yang, S., Wang, J., Lin, Q., Deng, B., Wei, X., Liu, C., et al. (2018). Cost-efficient fpga implementation of a biologically plausible dopamine neural network and its application. *Neurocomputing* 314, 394–408. doi: 10.1016/j.neucom.2018.07.006

Yang, S., Wang, J., Zhang, N., Deng, B., Pang, Y., and Azghadi, M. R. (2021). CerebelluMorphic: Large-scale neuromorphic model and architecture for supervised motor learning. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 4398–4412. doi: 10.1109/TNNLS.2021.3057070

Zenke, F., and Ganguli, S. (2017). Superspike: Supervised learning in multi-layer spiking neural networks. *Neural Comput.* 30, 1514–1541. doi: 10.1162/neco_a_01086