Efficient Design of Spiking Neural Network With STDP Learning Based on Fast CORDIC

Jiajun Wu, *Member, IEEE*, Yi Zhan, *Member, IEEE*, Zixuan Peng, *Member, IEEE*, Xinglong Ji, Guoyi Yu, *Member, IEEE*, Rong Zhao, *Member, IEEE*, and Chao Wang, *Senior Member, IEEE*

***Abstract*— In emerging Spiking Neural Network (SNN) based neuromorphic hardware design, energy efficiency and on-line learning are attractive advantages mainly contributed by bio-inspired local learning with nonlinear dynamics and at the cost of associated hardware complexity. This paper presents a novel SNN design employing fast COordinate Rotation DIgital Computer (CORDIC) algorithm to achieve fast spike timing–dependent plasticity (STDP) learning with high hardware efficiency. In this study, a system design and evaluation method of CORDIC-based SNN is proposed for finding optimal CORDIC type and precision, from theoretical CORDIC-level error to application-level learning performance. From the proposed design and evaluation method, a reconfigurable SNN design based on fast-convergence CORDIC is designed to achieve high classification accuracy on MNIST, fast on-line learning and good energy efficiency. By utilizing SNN's fault tolerance and time-division-multiplexing (TDM) strategy, the reconfigurable SNN design employs 8-bit fast-convergence CORDIC and TDM-based hardware accelerator for high efficiency. FPGA implementation results confirm that the proposed fast-convergence CORDIC SNN design outperforms the state-of-the-art CORDIC method by 38.5%– 45.3% in terms of learning speed and energy efficiency, with the STDP learning of 30.2 ns/SOP, energy efficiency of 176.6 pJ/SOP, processing speed of 6.1 ms/image, and on-line learning convergence of 21.4 s (time to reach the final accuracy, on average), on MNIST benchmark.**

Manuscript received December 2, 2020; revised February 2, 2021; accepted February 22, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61974053 and in part by the Fundamental Research Funds of the Central Universities under Grant 2019KFYXJJS049. This article was recommended by Associate Editor J.-O. Klein. *(Jiajun Wu and Yi Zhan contributed equally to this work.) (Corresponding author: Chao Wang.)*

Jiajun Wu, Yi Zhan, Zixuan Peng, and Guoyi Yu are with the School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan 430074, China.

Xinglong Ji was with the Engineering and Product Department, Singapore University of Science and Technology, Singapore 487372. He is now with the Center for Brain-Inspired Computing Research, Tsinghua University, Beijing 100084, China.

Rong Zhao was with the Engineering and Product Department, Singapore University of Science and Technology, Singapore 487372. She is now with the Department of Precision Instruments, Tsinghua University, Beijing 100084, China, also with the Center for Brain-Inspired Computing Research, Tsinghua University, Beijing 100084, China, and also with the Innovation Center for Future Chip, Tsinghua University, Beijing 100084, China.

Chao Wang is with the School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan 430074, China, and also with the Wuhan National Laboratory of Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: chao.wang.1978@hotmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2021.3061766.

Digital Object Identifier 10.1109/TCSI.2021.3061766

***Index Terms*— Synaptic circuit, neuron circuit, fast-convergence CORDIC, STDP, LIF neuron, on-line learning, spiking neural network, neuromorphic hardware.**

I. Introduction

SPIKING Neural Network (SNN) has been attracting significant attention in the fields of neuromorphic computing and artificial intelligence, as it promises excellent fault tolerance, high energy efficiency and fast on-line learning [1]. This is due to its unique brain-inspired properties including high parallelism, event-driven processing, spike-based information coding and local learning rules. Because SNN is more biologically plausible than conventional Artificial Neural Network (ANN) by mimicking biological neural network much more closely, SNN has been used to study and understand mammal brains, and also been applied in many Artificial Intelligent (AI) applications, such as image classification, language and speech recognition, and autonomous robot applications [2], [3]. In the past five years, researchers have proposed many neuromorphic hardware designs of SNNs with on-line learning, which can be classified into two major categories, i.e., analog or mixed analog-digital [4], [5], [6], [7], [10], and digital (including Application Specific Integrated Circuits (ASIC) and Field Programmable Gate Array (FPGA)) [8], [9], [11], [12], [15], [16].

Recently, on-line learning is preferred especially in the intelligent IoT edge devices to reduce the latency, minimize the wireless power, and protect the privacy, which is a trend for AI accelerator design research. In the research of neuromorphic hardware design, efficient design and implementation of bio-inspired neurons and synapses to enable on-line learning is very essential, because the choice of different synapse/neuron model-level circuit designs will result in a direct impact on the application performance, hardware cost and energy consumption. In most bio-inspired SNN models usually with nonlinear dynamics, including Izhikevich neuron, Integrated and Fire (IF) neurons, and Spiking-Timing Dependent Plasticity (STDP) based synapses [1]–[3], the hardware implementation involves many complex calculations, such as multiplication, exponential and differential function. Specifically, both STDP-based synapses [4], [9] and other STDP-like synapses derived from bio-plausible machine learning models [2], [3], [13], [14], involve exponential calculations to implement efficient local learning, which are very computation-intensive, hardware-expensive and power-hungry. These issues are exacerbated for massive-scale SNN with on-line STDP learning. Compared to the analog and mixed

analog-digital circuits that are more susceptible to process-voltage-temperature variation and less immune to device nonidealities and noises, digital solutions benefit more from CMOS technology scaling and are very popular to massive neuromorphic hardware implementation in both industry and academia [8], [9], [15], [16].

In the recent research of digital SNN hardware, STDP-based synapses are implemented based on Look Up Tables (LUT) [9], triangular approximation [17] and Coordinate ROtation DIgital Computer (CORDIC) [18]. Among these implementations, CORDIC has gained a lot of interests to implement neuron and synapse circuits, because CORDIC can use simple shift & add operations to approximate multiplication and exponentiation in a number of iterations, which is generally more accurate than the triangular approximation and costs fewer resources than the LUT based computation. In the latest literature [19], [20], CORDIC-based SNN designs have been proposed to implement bio-inspired neuron models and STDP-based synapses. Heidarpour *et al.* propose a CORDIC-based exponential core to implement the adaptive exponential IF neuron model [19]. Heidarpur *et al.* also use the CORDIC exponential core to compute STDP function and use a CORDIC-based square circuit to implement the Izhikevich neuron model with lower hardware resources and higher frequency rate than previous Izhikevich neuron designs [20]. These works have opened a new research direction of efficient design of SNN neuromorphic hardware based on CORDIC, for simulating and evaluating specific neural models. Error analysis has been performed to ensure the model-level accuracy of the proposed CORDIC neuron.

However, there still has some critical issues to study and solve. First, the existing research has only assessed the model-level precision and constructed a relatively simple network to demonstrate unsupervised Hebbian learning in terms of weight distribution. Actually, it is essential to evaluate the CORDIC SNN solution in a practical application, e.g., the digit classification on Modified National Institute of Standards and Technology (MNIST) data set [2]. Second, as the CORDIC STDP learning might result in longer training time due to inherent CORDIC iterations, it is necessary to evaluate the CORDIC-based learning speed in a given standard test. Third, the precision evaluation in the existing works has only analyzed the local spiking-train errors instead of network-level accuracy with respect to learning results. More importantly, the fault-tolerance of SNN has not been studied to find the optimum CORDIC-based SNN design with lowest iteration number or affordable errors of CORDIC. In summary, the efficient design of CORDIC SNN in terms of learning performance and energy consumption has not been systematically studied.

To solve the aforementioned issues, this study proposes a systematic design and evaluation method of CORDIC-based SNN, from theoretical CORDIC-level error analysis to application-level learning performance on MNIST data set, as shown in Fig. 1. From the proposed method, optimal CORDIC type from different algorithms and lowest bit-width precision (i.e., fewest iterations and computations) can be identified by trading affordable performance loss and hardware overhead for overall SNN hardware efficiency. In this paper, a reconfigurable SNN design based on selected 8-bit fast-convergence CORDIC proposed in our previous work [21] is presented. The FPGA implementation results confirm that based on the proposed efficient design and evaluation method, the fast-convergence CORDIC SNN design outperforms the conventional and state-of-the-art CORDIC methods in terms of the hardware efficiency, including of STDP-based learning performance and energy efficiency.

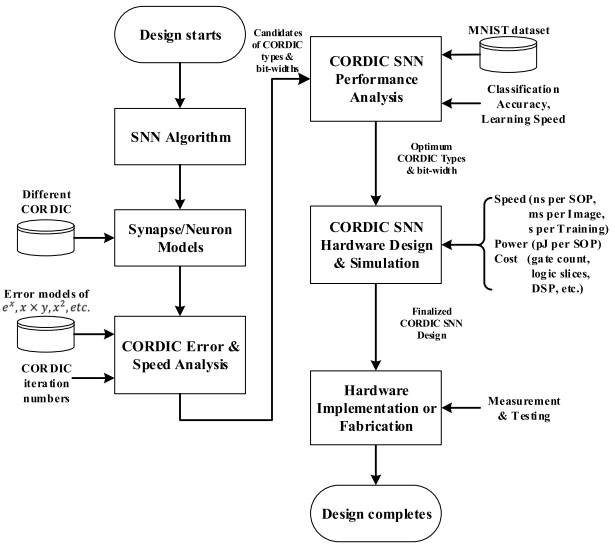

Fig. 1. Design and evaluation flow of the proposed CORDIC SNN.

The rest of this paper is organized as follows. Section II introduces STDP-based SNN, reviews different CORDIC algorithms, and proposes the SNN design and evaluation method from the theoretical CORDIC-level error analysis to network-level learning performance. Section III presents the proposed CORDIC based SNN design and describes the details of FPGA hardware implementation. Analysis and discussion of implementation results including hardware resources, learning speed and energy consumption is given in Section IV. Finally, Section V makes a conclusion.

II. CORDIC in Spiking Neural Networks

A. Network Model and Learning Rule

1) SNN Model: To perform a more practical evaluation of CORDIC-based SNN design, intensive review of SNN models has been conducted. In the literature [2], [9], [17], [22]–[28], supervised, unsupervised and hybrid learning has been researched for the training of SNN models. In this study, unsupervised STDP-variant rules are chosen because they reduce the need for labeled data, which is important for intelligent IoT edge devices. Specifically, the unsupervised SNN model and STDP-variant rules in [2] are suitable for the study as they have competitive accuracy advantage (i.e., 95% classification accuracy) for unsupervised learning setups and are still competitive with other previous supervised learning setups. Therefore, we chose the SNN model and STDP variants in [2] for the case study to demonstrate the efficiency of the CORDIC-based SNN design.

In this study, a state-of-the-art SNN model based on Leaky IF (LIF) neurons and unsupervised STDP-variant learning rule is chosen due to the re-implemented robust performance from the open source codes in [2], [34]. This SNN model has the

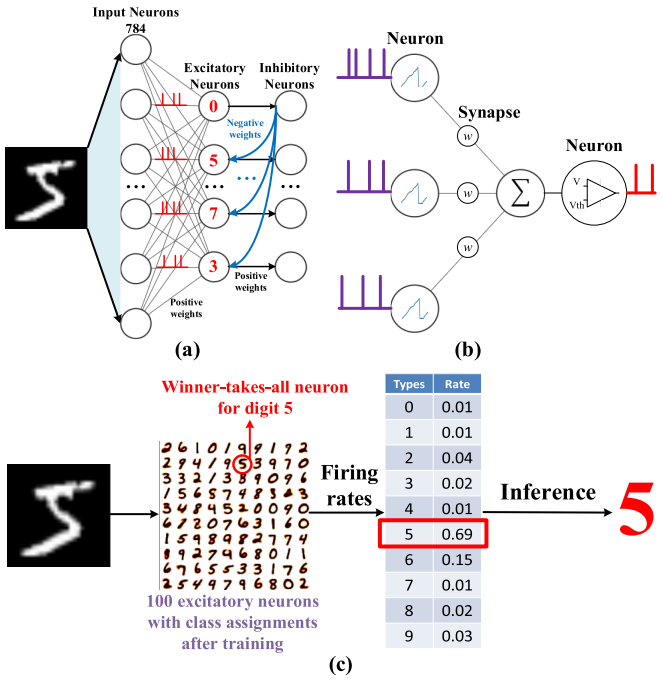

Fig. 2. (a) SNN architecture, (b) local detail of SNN architecture and (c) spiking-based working mechanism. Note: Synapses between input and excitatory neurons are excitatory synapses (black) with positive weights. Synapses from excitatory neurons to inhibitory neurons are excitatory synapses (black) with positive weights, while synapses from inhibitory neurons to excitatory neurons are inhibitory synapses (blue) with negative weights.

following features: (1) It can achieve high accuracy in MNIST benchmark with a scalable architecture (accuracy up to 95% with 6400 excitatory neurons, which is one of the highest accuracies among the unsupervised STDP SNN models in [2], [25]–[28]); (2) Inspired by biological observations, inhibitory neurons are used to generate competition among neurons and facilitate the implementation of winner-takes-all (WTA) mechanism; (3) The SNN uses bio-plausible STDP on-line learning, which is local and highly parallel, and therefore it can implement on-line learning without massive data propagation/movement across the network. The SNN model is illustrated in Fig. 2 (a) and (b).

As shown in Fig. 2 (a), the input layer contains 28×28 neurons, which is corresponding to 784 pixels from an image in the MNIST dataset. The input neurons are used for encoding a pixel to an encoding time window. There are two major encoding strategies for image encoding, i.e., spiking rate coding (e.g., the Poisson coding) [29] and spiking latency coding [30]–[33]. In this study, the Poisson coding is selected in our SNN model for a fair comparison with the existing work in [19]. The second layer is an excitatory layer, which is consisting of a variable number of excitatory neurons to perform digit classification. In the third layer, the SNN model utilizes inhibitory neurons to facilitate the excitatory layer to implement WTA method. The inhibitory layer is composed of the same number of neurons as the excitatory layer. In this SNN model, the synaptic connections between input and excitatory layers are full connections with positive weights. However, there are two types of uni-directional connections between the excitatory and inhibitory layers: (1) The forward synaptic connections from excitatory neurons to inhibitory neurons are one-to-one connections with positive weights; (2) The backward synaptic connections are from each inhibitory neuron to all excitatory neurons, excluding the one with a forward connection to the inhibitory neuron. Based on this topology, a firing excitatory neuron can inform its corresponding inhibitory neuron to generate lateral inhibition over all other excitatory neurons, as illustrated in Fig. 2 (a).

Fig. 2 (b) illustrates the LIF neuron used in the SNN model [2]. To facilitate efficient hardware implementation, we discretize the LIF neuron model by describing the membrane voltage (V) as following:

$$V[t+1] = e^{-\frac{1}{\tau}}(V[t] - E_{rest}) + E_{rest} + I$$
$$I = \sum_{i=0}^{n} w_i f_i, \quad (1)$$

where τ is time constant, E_{rest} is resting potential, w is synaptic weight, and I is the summation of weighted connections from the other n neurons. If a synaptic connection gets a spike from a pre-synaptic neuron, parameter f equals 1, otherwise it is 0. When V exceeds threshold voltage V_{th}, the neuron will fire a spike and its potential will be reset to E_{rest}. The leaky behavior of membrane potential is described by the exponential component with respect to the time constant τ.

Both excitatory and inhibitory neurons can be described by (1). The difference between the excitatory and inhibitory neurons is that the excitatory neuron receives spikes over excitatory and inhibitory synapses (with positive and negative w, respectively), from both the input and inhibitory layers, respectively, while the inhibitory neuron only receives spikes over excitatory synapse (with positive w only) from the corresponding single excitatory neuron, as shown in Fig. 2(a). In addition, the excitatory synapses are plastic, while the inhibitory synapses are non-plastic.

Fig. 2 (c) illustrates the working mechanism of SNN classification. After training, excitatory neurons get assignments to classes, based on their highest average response to a digit class over the training set. For example, in a batch of training, if a neuron is particularly sensitive to "5", which means this neuron has the highest firing rate associated with digit "5" among all digit classes, then the neuron will be assigned to predict "5". After training, every excitatory neuron has its own assignment. In the inference, the firing rates of each assignment group of excitatory neurons are summed up during the whole encoding time of an input image to be classified. According to the assigned class of a neuron group with the highest firing rates, the SNN predicts a digit class for the input image, which presents the WTA mechanism.

2) SNN Learning Rule: During the training, when spikes propagate forward in the neural network, all excitatory synapses will update their weights concurrently based on a local bio-plausible STDP rule, which can be efficiently implemented by parallel processing in hardware. From the observation of STDP learning mechanism, pre-synaptic neurons that cause the post-synaptic neuron's excitation are more likely to contribute in the future and thus the corresponding connection is strengthened. On the contrary, pre-synaptic neurons that don't or rarely cause the firing of post-synaptic neurons are less likely to contribute in the future, so the synaptic connection is weakened. The contribution can be measured by time difference between two consecutive spikes, i.e., a spike pair of the pre-synaptic and post-synaptic neurons.

This bio-phenomenon inspires a variant of STDP learning rule that is described by an exponential weight-dependent update equation in [2] as following:

$$\Delta w = \eta(x_{pre}e^{-\beta w} - x_{tar}e^{-\beta(w_{max}-w)}). \quad (2)$$

With comparable classification accuracies (up to 95%), the selected STDP rule involves exponentiation, which is less complex than the other STDP variants based on power-law functions in [2]. The updating of synaptic weight occurs at post-synaptic neuron firing. Here, w is synaptic weight, ranging from -1 to $1.\eta$ is learning rate. β (a positive number less than 1) and w_{max} is weight-dependence coefficient and maximum weight, respectively. Each synapse uses x_{pre} called pre-synaptic trace to record the recent pre-synaptic spike history. When a pre-synaptic spike arrives at the synapse, x_{pre} increases by 1. When there is no spike coming, a dynamic leakage for the x_{pre} is described by an exponential component. However, the leakage dynamics of pre-synaptic trace can be removed to simplify hardware implementation, as we found from simulation that the classification performance is not affected without the dynamic leakage, e.g. 0.7% loss for an SNN model with 100 excitatory neurons. The x_{tar} is an offset value, which stands for the target value of the pre-synaptic trace at the moment of a post-synaptic spike [2]. The bias by the second term in (2) makes Δw able to be positive or negative, which is to strengthen or weaken the corresponding synaptic connection, respectively. In this work, the x_{tar} is calculated as the average number of pre-synaptic traces in synapses that connect input layer and excitatory layer.

By collecting the pre-synaptic spiking count in a time window, x_{pre} actually depicts the STDP dynamic of the synaptic strength between two neurons. As (2) shows, when a post-synaptic neuron fires, a larger x_{pre} statistically indicates smaller difference between the spiking time of two neurons and means a stronger contribution from the pre-synaptic neuron, thus Δw increases larger to strengthen the neuron connection. In the other hand, if the pre-synaptic neuron contributes little to the post-synaptic neuron's firing and the spiking time difference is bigger, the Δw can be negative to weaken the synaptic connection with the help of target trace x_{tar} to adjust the second exponential term. The study in [2] shows that the exponential weight dependence of the learning rule actually strengthens the learning robustness and makes the convergence faster.

B. CORDIC for Exponentiation

CORDIC is an iterative algorithm to calculate many complex functions including exponential, multiplication, hyperbolic and trigonometric functions, with simple shift & add operations, which can be easily and efficiently implemented in digital ASIC and FPGA [18], [35]. In most bio-plausible SNN models including the selected SNN, the nonlinear dynamic properties of neurons and synapses are described by exponential functions, which can be implemented by CORDIC very cost-effectively. In our proposed system design method of CORDIC SNN, different CORDIC algorithms, including the conventional CORDIC [18], Heidarpur's simplified CORDIC [20], and fast-convergence CORDIC [21], are evaluated and compared in terms of the learning and classification performance, in this paper.

Algorithm 1 Conventional CORDIC for calculating exp(θ)

Input: Scaling factor s, variable to be calculated θ
CORDIC iterations n,
CORDIC angles lookup table *List* // *List*[i] = arctanh(2^{-i})
Output: Calculation results *result*
1: $x = s$; $y = 0$; $z = \theta$; // Initialization
2: **for** $i = 1 : n$ **do**
3: **if** $z > 0$ **then**
4: $x = x + y \times 2^{-i}$; $y = y + x \times 2^{-i}$; $z = z$ - *List* [i]; // In parallel
5: **else**
6: $x = x - y \times 2^{-i}$; $y = y - x \times 2^{-i}$; $z = z$ + *List* [i]; // In parallel
7: **end if**
8: **end for**
9: *result* = $x + y$; // $e^{\theta} = sinh(\theta) + cosh(\theta)$
10: Return *result*;

Fig. 3. Pseudocode for conventional CORDIC algorithm to calculate exponentiation. Note that the scaling is omitted here as the scaling factors can be merged to a single pre-calculated factor for pre/post-scaling [36].

1) Conventional CORDIC: The basic idea of CORDIC algorithm is decomposing a rotation of a vector by a target angle into a sequence of micro-rotations with predefined elementary angles. The accumulated angle from the micro-rotations approximates the target angle iteratively. The iteration equations of CORDIC-base exponentiation can be depicted as:

$$\begin{cases} x^{(i+1)} = x_s^{(i)} + d_i y_s^{(i)} 2^{-i} \\ y^{(i+1)} = y_s^{(i)} + d_i x_s^{(i)} 2^{-i} \\ z^{(i+1)} = z^{(i)} - d_i a^{(i)} \end{cases} \quad (3)$$

$$\begin{cases} x_s^{(i+1)} = S_i x^{(i+1)} \\ y_s^{(i+1)} = S_i y^{(i+1)} \end{cases} \quad (4)$$

$$e^{\theta} = cosh\theta + sinh\theta = x_s^n + y_s^n. \quad (5)$$

Here, (3) is the rotation equation, (4) is the scaling equation, and (5) is the exponentiation equation. i and n are the index of current iteration, and the number of total iterations, respectively. And $a^{(i)}$ is the angle selected from the LUT (i.e., $a^{(i)} = tanh^{-1}2^{-i}$). S_i is the scaling factor which is determined by the i. x and y are the coordinates of the vectors, while z are the residue angles, during the iterations. d_i is the direction of rotation, and $d_i = \pm 1$.

In the rotation mode, the CORDIC can be configured to calculate the hyperbolic sine and hyperbolic cosine of a target angle efficiently [36]. Equation (5) shows that the exponential result can be calculated by adding hyperbolic sine and hyperbolic cosine of the target angle θ. As shown in Fig. 3, the conventional CORDIC uses a number of iterations (denoted by n) to minimize the residue angle (z) to 0 by a sequence of micro- rotation angles from the LUT to ensure convergence to the target angle θ. In this study, the n-iteration angle approximation is depicted by n-bit CORDIC accuracy, while n-bit precision is also used for input CORDIC data with an extra sign bit. The operation of conventional CORDIC for exponentiation is also illustrated in Fig. 4 (a).

2) Fast-Convergence CORDIC: The previous studies show that the conventional CORDIC algorithm has a main drawback of slow convergence with redundant iterations that

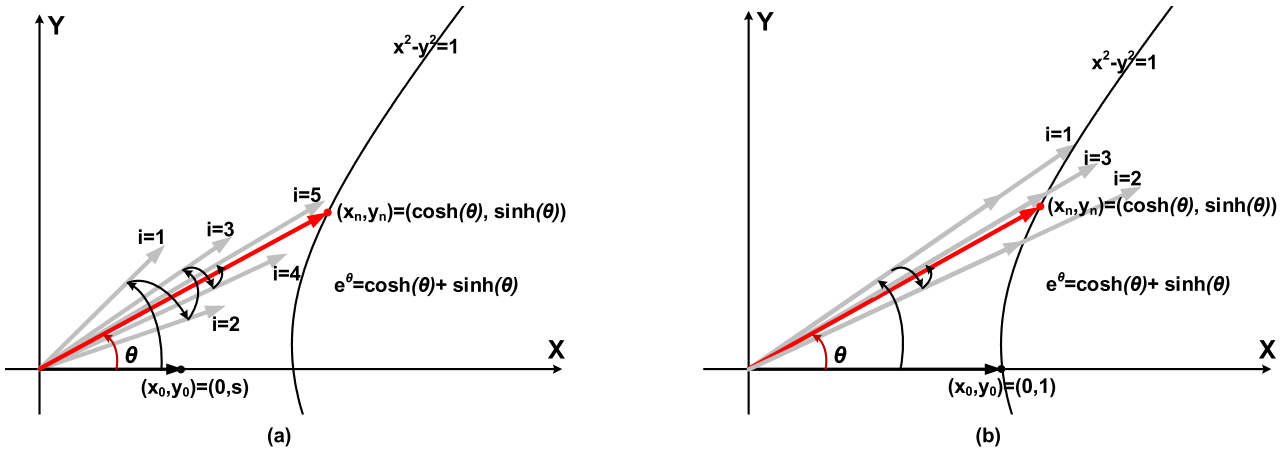

Fig. 4. Illustration of CORDIC algorithms for exponentiation: (a) conventional CORDIC with many redundant iterations and (b) fast-convergence CORDIC with least number of iterations.

TABLE I
THE AVERAGE NUMBER OF ITERATIONS FOR EXPONENTIATION BY DIFFERENT BITS FAST-CONVERGENCE CORDIC

Iteration No. of Conventional/Heidarpur's CORDIC	6	8	16	24
Average No. of Fast-convergence CORDIC	2.27	4.18	7.88	10.48
Saving of Redundant Iterations	62.2%	47.8%	50.8%	56.3%

can be up to 50% before its final convergence to a target angle [37]. In the past, fast-convergence CORDIC algorithms have been proposed for solving the issue to reduce the redundant computation and long latency. Fig. 5 shows an efficient fast-convergence CORDIC with Angle Recoding methods, making effort for converging to the final target with the least number of elementary iterations by finding the optimum micro-rotation angle at every iteration [21], [35]–[39]. The principle of fast-convergence CORDIC for exponentiation is illustrated in Fig. 4 (b) too.

The key idea of Angle-Recoding (AR) based CORDIC is finding the optimum next-to-rotate angle by selecting the closest micro-rotation angle to the target angle, and therefore achieve the least absolute rotational angle accumulation [37]. For efficient hardware implementation, there are parallel-based and pipeline-based strategies of angle selection to reduce angle selection latency. The parallel AR scheme searches all optimum micro-rotation angles at the cost of high hardware complexity [38]. In our previous work [21], the pipeline AR method selects the optimum angles during the iterations in a pipeline fashion, so that at lower hardware cost, both higher hardware utilization and better pipeline performance can be achieved for efficient hardware implementation.

Therefore, in this study, the fast-convergence CORDIC with pipeline Angle Recoding (AR) method is employed to design CORDIC SNN for achieving fast STDP learning and reducing energy consumption. For compactness, the terms "fast-convergence CORDIC" and "fast-convergence CORDIC with pipeline AR method" are interchangeably in this paper. Our study compares the fast-convergence CORDIC against the conventional CORDIC and Heidarpur's CORDIC

Algorithm 2 Fast-convergence CORDIC for calculating exp(θ)

Input: variable to be calculated θ, CORDIC iterations n,
CORDIC angles lookup table *ListAngle* // *ListAngle*[i] = arctanh(2^{-i})
CORDIC scaling factors table *ListScale* // *ListScale: scaling factors*
Output: Calculation results *result*
1: $x = 1$; $y = 0$; $z = \theta$; // Initialization
2: **for** i = 1 : n **do**
3: **if** z < 2^{-n} **then**
4: break; // Fast-iteration finishes
5: **end if**
6: *AngleIndex* = findProperAngle(z, *ListAngle*) // Find proper angle
7: **if** z > 0 **then**
8: $x = x + y \times 2^{-i}$; $y = y + x \times 2^{-i}$; $z = z$ - *ListAngle* [i]; // In parallel
9: **else**
10: $x = x - y \times 2^{-i}$; $y = y - x \times 2^{-i}$; $z = z$ + *ListAngle* [i]; // In parallel
11: **end if**
12: $x = x \times$ *ListScale[AngleIndex]*; // Scaling
13: y = y $\times$ *ListScale[AngleIndex]*; // Scaling
14: **end for**
15: *result* = $x + y$; // $e^{\theta} = \sinh(\theta) + \cosh(\theta)$
16: Return *result*;

Fig. 5. Pseudocode for fast-convergence CORDIC algorithm to calculate exponentiation. Note that scaling at each iteration can also be implemented by simple shift-and-add operations efficiently [21].

algorithm [20], in terms of CORDIC convergence speed. Table I shows the average number of iterations for exponential calculation from -1 to 1 (the input is with a step size of 0.01, i.e., 201 input samples) as required by the STDP learning rules (2). Across the different numbers of bit accuracy, the fast-convergence CORDIC for exponentiation can reduce iterations by over 50%, which can achieve significant improvement in learning speed and energy consumption.

3) Error Analysis of CORDIC Exponentiation: As described previously, the main difference between the fast-convergence CORDIC and conventional CORDIC are: a) the former significantly reduces unnecessary micro-rotations by the optimum angle selection, which results in fewer number of iterations and fewer computations; b) the latter's scaling factors can be multiplied directly on the input x/y as a pre-scaling to save computation, while the former has to perform the necessary scaling on the intermediate output

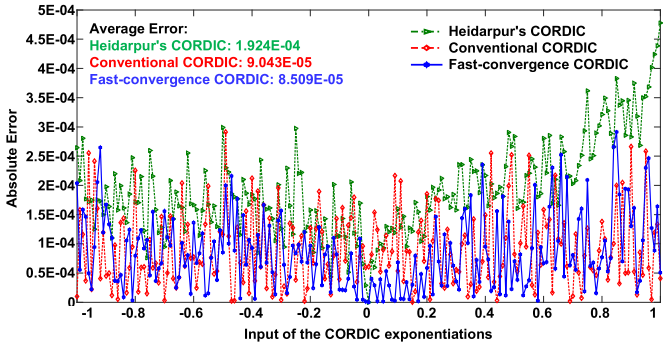

Fig. 6. Error comparison of exponentiation results based on conventional CORDIC, Heidarpur's CORDIC [20], and fast-convergence CORDIC at 16 bit-width. Note that the plot is the absolute value of actual errors against floating-point results based on exponential function in Matlab.

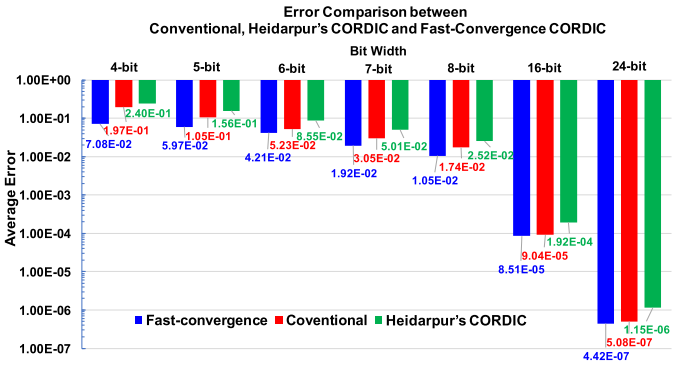

Fig. 7. Comparison of average errors in exponentiation results based on conventional CORDIC, Heidarpur's CORDIC [20], and fast-convergence CORDIC with different bit-width precisions.

depending on angle selection result at each iteration [21]. The error of different CORDIC at various bit-widths are simulated and analyzed in this sub-section for investigating the accuracy of CORDIC-based exponential operations in SNN learning and classification.

Fig. 6 presents a case study of investigating the error performance of different CORDIC that are used to implement exponentiation functions for STDP learning in the selected SNN model in [2]. Note that the Heidarpur's CORDIC in [19] has also been reproduced and simulated. In this study, the input of these CORDIC-based exponential calculations are $(-1, 1)$, as required by the STDP rule. Without loss of generality, the step size of the input is set to 0.01 and 16-bit precision is chosen as an example to evaluate the error performance. In terms of absolute values, the error distribution shows that the fast-convergence CORDIC has the smallest errors in average, while the Heidarpur's CORDIC is poorer than both the conventional and fast-convergence CORDIC. This is because the Heidarpur's CORDIC not only has redundant iterations accumulating many errors similarly to the conventional CORDIC, it also has multiplicative errors in the micro-rotation to increase error larger than other two CORDIC.

Fig. 7 presents the comparison of average errors in exponential calculations based on conventional CORDIC, Heidarpur's CORDIC, and fast-convergence CORDIC at different bit-widths. Simulation result further demonstrates the great advantage of the fast-convergence CORDIC in error. Under various bit-widths, the average error of fast-convergence CORDIC is the smallest, which is mainly contributed by the least number

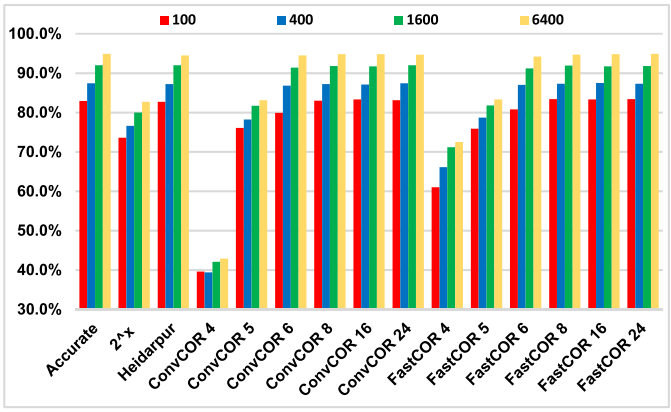

Fig. 8. Classification accuracy of SNN based on different CORDIC under various bit-width precision. The red, blue, green and yellow bars correspond to the SNN consisting of 100, 400, 1600 and 6400 excitatory neurons, respectively. "Accurate" stands for exact exponential operation in floating point. Note that 2^x and "Heidarpur" are two versions of modified CORDIC algorithm with 8-bit precision in [20]. "ConvCOR n" presents n-bit conventional CORDIC, while "FastCOR n" means n-bit fast-convergence AR-based CORDIC.

of iterations. In summary, considering the exponentiation accuracy and number of iterations required, the fast-convergence CORDIC is better than both the conventional and Heidarpur's CORDIC, which indicates that the fast-convergence CORDIC could achieve faster STDP learning in CORDIC-based SNN.

C. Performance Analysis of CORDIC SNN

As mentioned previously in Section I, although previous CORDIC-based works [19], [20] have great contribution in neural function and behavior emulation, they lack in performance evaluation in network-level application, because no clear cut-off values of error metrices can be chosen as the indicator to find an optimal CORDIC design based on the analysis of CORDIC-based synapses and neurons only [19], [20]. Therefore, comprehensive performance analysis at network level is proposed to evaluate the CORDIC in SNN models. We evaluate the CORDIC SNN in system application level by comparing the STDP learning and classification performance of SNN with respect to different CORDIC algorithms under various bit-width precisions.

Extensive simulation has been conducted to evaluate the SNN classification on MNIST data sets by substituting accurate exponential calculations in [2] with different CORDIC-based exponentiation functions. According to the error analysis results of CORDIC exponentiation in Fig. 6, the closer CORDIC input is to 1, the greater the error is caused by the fixed-point CORDIC implementation. Therefore, the weight-dependence coefficient β in (2) should be set to a small value to constrain the CORDIC input to a range close to 0, e.g., 0.25 in our simulation. In the simulation, all data sets are divided into training, validation and test set, which include 55000, 5000 and 10000 images, respectively. In the training set, 250 images are regarded as a batch. After each batch of training, all excitatory neurons are reassigned to digit types, and then 1000 random validation images are selected to test classification accuracy in training phase. After training phase completes, 10000 test images are used for testing classification accuracy based on the last-batch digit assignment.

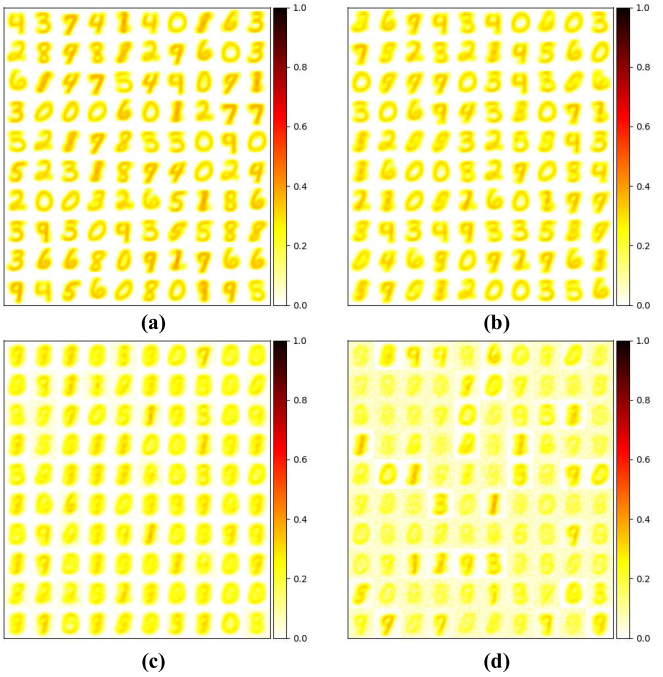

Fig. 9. Rearranged weights of excitatory synapses in an SNN with 100 excitatory neurons, based on (a) proposed 8-bit fast-convergence, (b) 8-bit conventional, (c) 8-bit simplified 2^x and (d) 4-bit conventional CORDIC, respectively. Note the 784 weights that are connected to each excitatory neuron are rearranged to a 28 × 28 pixel matrix, and all the 100 images are visualized as 10 × 10 grids in this figure, i.e. each grid stands for 784 synaptic weights which are connected to a certain excitatory neuron.

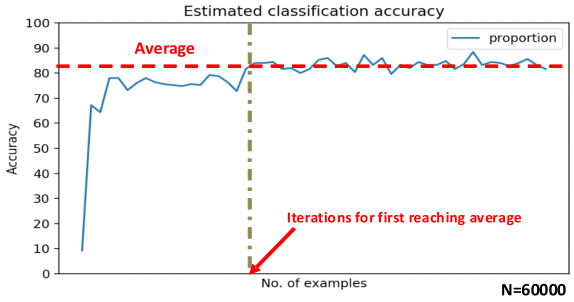

Fig. 10. Definitions of ave-iterations to evaluate convergence speed of CORDIC-based STDP. In detail, ave-iteration is equal to the number of training iterations used by the SNN model for the first time to reach average accuracy.

Fig. 8 presents the accuracies of SNN with different bit-widths of the conventional CORDIC, the fast-convergence CORDIC, and the modified CORDIC in [20]. Thanks to the excellent fault tolerance of SNN, CORDIC with a precision bit-width equaling or larger than 6 bits can be used for efficient hardware implementation as the classification accuracy is not degraded too much. Compared to other CORDIC implementations, SNN based on the fast-convergence CORDIC exhibits excellent classification accuracy, and at the same time, it can also achieve much faster STDP learning contributed by its fast-convergence advantage.

Fig. 9 shows the rearranged weights of excitatory synapses after training of 784 × 100 SNN, based on 8-bit fast-convergence CORDIC, 8-bit conventional CORDIC, 8-bit simplified 2^x CORDIC [20] and 4-bit conventional CORDIC, respectively. For each excitatory neuron, the 784 synaptic weights that are connected to the input neurons are rearranged into a 28 × 28 pixel matrix to visualize the learning performance. Well-trained excitatory synaptic weights tend to be bimodal distribution. In detail, most weights approximately tend to zero, while the rest non-zero weights are distributed as characteristics of training set. For example, in the MNIST dataset, Fig. 9 visually illustrates the training performance based on different CORDIC algorithms. It can be seen clearly that the training based on 8-bit 2^x CORDIC and 4-bit conventional CORDIC achieve worse performance (i.e., much blur images) than the 8-bit fast-convergence CORDIC and 8-bit conventional CORDIC.

TABLE II

OVERALL PERFORMANCE RESULTS OF SNN BASED ON DIFFERENT CORDIC AND BIT-WIDTH PRECISIONS (100 EXCITATORY NEURONS)

CORDIC used in STDP	Ave-iterations (CORDIC)	Accuracy (Classifi.)	Ave-iterations (Training)
No CORDIC (Accurate STDP)	-	82.9%	2750
6-bit conventional	6	79.9%	4500
8-bit conventional	8	83.0%	3500
8-bit 2^x [20]	8	73.6%	5500
8-bit Heidarpur [20]	8	82.7%	3250
6-bit fast-convergence	2.27	80.8%	4500
8-bit fast-convergence	4.18	83.4%	3500
16-bit fast-convergence	7.88	83.3%	3250
24-bit fast-convergence	10.48	83.4%	2750

Table II presents the overall performance results including the average accuracy and convergence speed of SNN based on different CORDIC. The convergence speed is defined as average iterations, which is illustrated in Fig. 10. Due to the excellent learning and classification results with much fewer CORDIC iterations, the fast-convergence CORDIC is selected for efficient SNN-based neuromorphic hardware design. It is also worthwhile to note that the SNN's fault tolerance has its limits on the errors of CORDIC-based STDP calculations. When the calculation error caused by CORDIC is too big, the overall performance degradation cannot be tolerated (e.g., the simplified 2^x CORDIC in [20]). When the calculation error is small enough (e.g., the 8-bit, 16-bit and 24-bit fast-convergence CORDIC), the classification accuracy could be negligible (i.e., less than 1% loss).

However, a little bit performance slowdown can be traded off to reduce the hardware complexity and energy consumption of CORDIC-based STDP circuits significantly. It is worthy to note that although the 8-bit fast-convergence CORDIC looks slower than the Heidarpur's CORDIC in terms of learning iterations, the actual STDP learning speed is much faster by evaluating the product of CORDIC iteration number and SNN learning iteration number. Thanks to its faster CORDIC convergence, the fast-convergence CORDIC is also more energy efficient than both the conventional and Heidarpur's CORDIC. In the next section, FPGA implementation and simulation of SNN based on different CORDIC with 6-bit and 8-bit

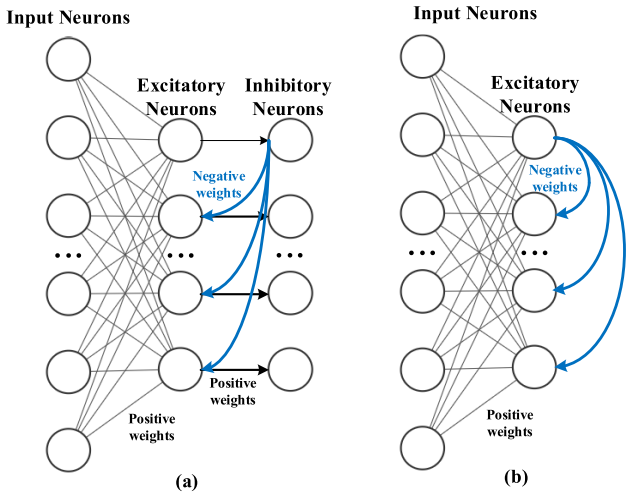

Fig. 11. Proposed SNN architecture by substituting inhibitory neurons in original model with intra-layer inhibitory synapses: (a) original model and (b) optimized model.

TABLE III

PERFORMANCE COMPARISON OF PROPOSED SNN WITH INTRA-LAYER INHIBITORY SYNAPSES AGAINST ORIGINAL SNN

CORDIC Precision	Average accuracy difference	Ave-iteration difference
6bits	0.6%	0
8bits	-0.2%	0
16bits	0.3%	0

Note that both SNN designs are based on fast-convergence CORDIC.

precisions are carried out to investigate the actual hardware efficiency of proposed fast-convergence CORDIC SNN design.

III. HARDWARE DESIGN AND IMPLEMENTATION OF PROPOSED FAST-CONVERGENCE CORDIC SNN

A. System and Architecture Design

In addition to the CORDIC SNN design and evaluation based on theoretical analysis and algorithm simulation in Section II, hardware implementation is essential to evaluate the efficiency of proposed fast-convergence CORDIC SNN design. Considering the flexibility, reconfigurability and massive parallelism of FPGA, this study selects a Xilinx's Zynq FPGA device (xc7z035fbg676-2) to implement the SNN design to perform evaluation of the hardware efficiency, including hardware utilization, learning speed and energy efficiency.

1) Design of CORDIC Synapses and Neurons: As discussed in Section II, the selected SNN model uses one inhibitory neuron following each excitatory neuron to implement lateral inhibition (Fig. 11 (a)). In this one-to-one synaptic connection, the weights from excitatory neurons to inhibitory neurons are actually fixed to high values after tuning, making inhibitory neurons to always fire immediately after the excitatory neurons firing. Thus, intra-layer inhibitory synapses among the excitatory neurons is proposed to optimize the SNN architecture to reduce hardware complexity and energy consumption by removing the inhibitory neurons, as depicted in Fig. 11 (b). Table III presents the Python simulation result to prove that the optimized SNN model only sacrifices negligible loss of accuracy in terms of ave-iteration.

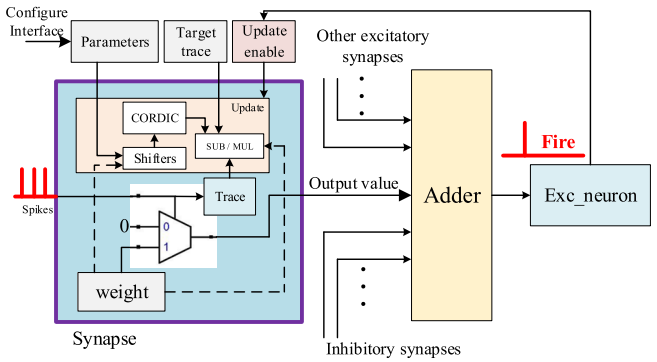

Fig. 12. Architecture of the proposed fast-convergence CORDIC synapse.

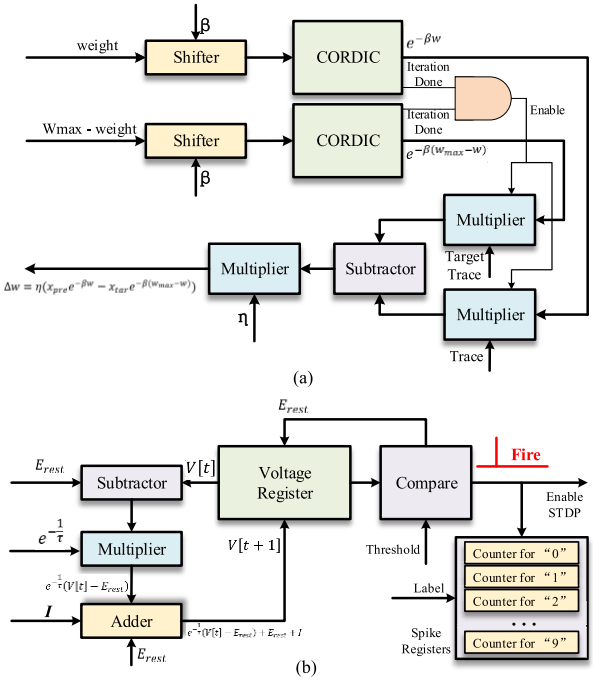

Fig. 13. Data flow of (a) fast-convergence CORDIC STDP circuit and (b) LIF neuron circuit.

Fig. 12 presents the architecture of proposed fast-convergence CORDIC based synapses in hardware design. Synapses are the core module responsible for the spike propagation and on-line STDP learning in SNN. In the synapse module, static parameters such as η and β in (2) are stored in the 8-bit parameter registers through configuration interface. The offset value x_{tar}, which is the average number of pre-synaptic traces shared in the excitatory layer, is calculated by an adder and a shifter in the trace counter block.

The synaptic weight and pre-synaptic trace are dynamically updated and stored in local 10-bit and 8-bit registers, respectively. When a spike from the pre-synaptic neuron arrives, the synapse will forward the local weight via the multiplexer to the 16-bit adder for performing weighted summation. At the same time, the pre-synaptic spike will increase the trace counter by 1, i.e., recording the number of pre-synaptic spikes. When the post-synaptic neuron fires, the excitatory neuron will enable all pre-synaptic synapses to update the local weights following the STDP rule. The on-line STDP weight update is implemented by the fast-convergence CORDIC circuit in [21].

TABLE IV

HARDWARE UTILIZATION AND MAX SPEED FOR SNN WITH DIFFERENT IMPLEMENTATIONS IN XILINX'S ZYNQ FPGA DEVICE (XC7Z030FBG484)

Implementation			Slice LUTs	Slice Registers	DSPs	BRAM	Max Speed
Heidarpur's CORDIC [20]	8 bits	Synapse	611	163	3	0	195.4 MHz
		14 × 10 SNN	87324	22596	430	0	
Conventional CORDIC	6 bits	Synapse	200	101	3	0	303.4 MHz
		14 × 10 SNN	28310	15083	430	0	
	8 bits	Synapse	244	123	3	0	301.8 MHz
		14 × 10 SNN	34912	17602	430	0	
Fast-convergence CORDIC	6 bits	Synapse	373	138	3	0	303.4 MHz
		14 × 10 SNN	51007	19502	430	0	
	8 bits	Synapse	533	178	3	0	301.8 MHz
		14 × 10 SNN	78053	23320	430	0	
LUT Based Implementation [9]	6 bits	Synapse	138	62	3	1	303.4 MHz
		14 × 10 SNN	19734	8930	430	140	
	8 bits	Synapse	180	86	3	1	301.8 MHz
		14 × 10 SNN	25740	12576	430	140	

Fig. 13 (a) illustrates the data flow of STDP-based weight calculation as depicted by (2). The multiplications with β are implemented by two shifters to save hardware resources and power consumption by optimizing β to one of 1/4, 1/8 and 1/16, which can also constrain the exponential input close to 0 for suppressing the CORDIC errors and increasing classification accuracy, as suggested in the Part B of Section II. Two 8-bit fast-convergence CORDIC exponentiation units and two 10-bit × 8-bit multipliers are used to calculate the two exponential terms of the STDP equation in parallel, before the 18-bit subtraction and final 18 × 8 multiplication with the η.

Fig. 13 (b) illustrates the data flow of LIF neuron. The decay term $e^{-\frac{1}{\tau}}$ in (1) is implemented as an 8-bit static register to the 8-bit × 16-bit multiplier for generating the final 16-bit membrane potential. For each excitatory neuron, there are 8-bit local counters for recording the firing information with respect to the corresponding class labels during training. After every spike firing in the training, the membrane potential will be reset to the resting potential, and one of the spike counters will increase by 1 according to the label of current input image. After each training batch, every excitatory neuron can update its digit class depending on the maximum firing rate recorded by the spike counters, as discussed in the Part A of Section II.

2) Reconfigurable Design of CORDIC SNN: Considering both the limited resources in hardware implementation and the excellent scalability in selected SNN model, a Time-Division Multiplexing (TDM) strategy is employed to design a small-scale SNN accelerator to implement the scalable SNN model with different numbers of excitatory neurons ranging from 100 to 6400 in [2].

Fig. 14 depicts the overall system architecture of the proposed reconfigurable CORDIC SNN design in the Zynq FPGA device. Initial synaptic weights and input spike sequences are generated by the Processing System (PS) based on a Cortex-A9 processor in the Zynq and then stored in a DDR3 memory. Controlled by the PS, initial weights and spike sequences are transmitted from the DDR3 to the block memory (BRAM) in the Programmable Logic (PL) through a Direct Memory Access (DMA) on the FPGA. In the PL, an initialization module is designed to configure parameters and allocate two BRAMs for buffering the input sequences and synaptic weights, i.e., BRAM-I and BRAM-S, respectively. Note

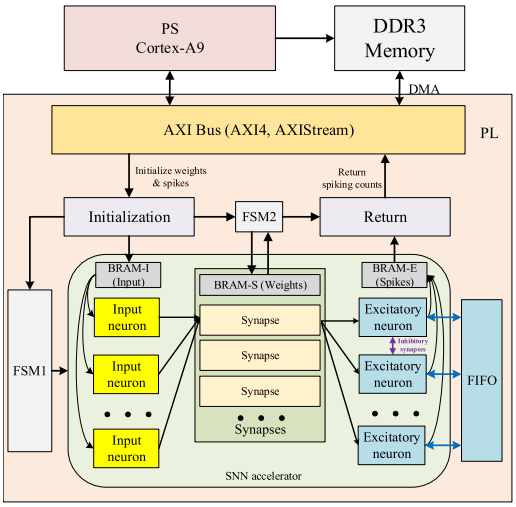

Fig. 14. System architecture of proposed SNN accelerator based on Time-Division Multiplexing to compute scalable SNN model in [2].

that because all the weights are buffered inside the FPGA during training, DDR accesses are only needed to retrieve the input spikes data after the initialization. After that, two Finite State Machines (FSMs) start to facilitate the proposed SNN accelerator to implement the TDM strategy. In the SNN accelerator, each excitatory neuron is allocated with a FIFO to buffer intermediate membrane potentials during one time slot of multiple TDM slots, while a BRAM (BRAM-E) is used to store final membrane potential and spiking information of every neuron. Moreover, a return module is designed for feeding the spiking information to the PS for image classification.

Considering the limited PL resources in the selected Zynq device, a 14×10 network is proposed to implement the TDM-based hardware accelerator in this study. In general, when calculating a $M \times N$ SNN model with a certain of spiking time units (i.e., $T_{encoding}$, the encoding time window size), the spiking data of M neurons are divided into $\lceil \frac{M}{14} \rceil$ TDM slots (i.e., $\lceil \frac{784}{14} \rceil = 56$, for MNIST) to be sequentially fed into the 14×10 network, which is controlled by the FSM1. By the TDM manner, the calculation of $M \times 10$ neurons can be sequentially performed. The final membrane potentials of

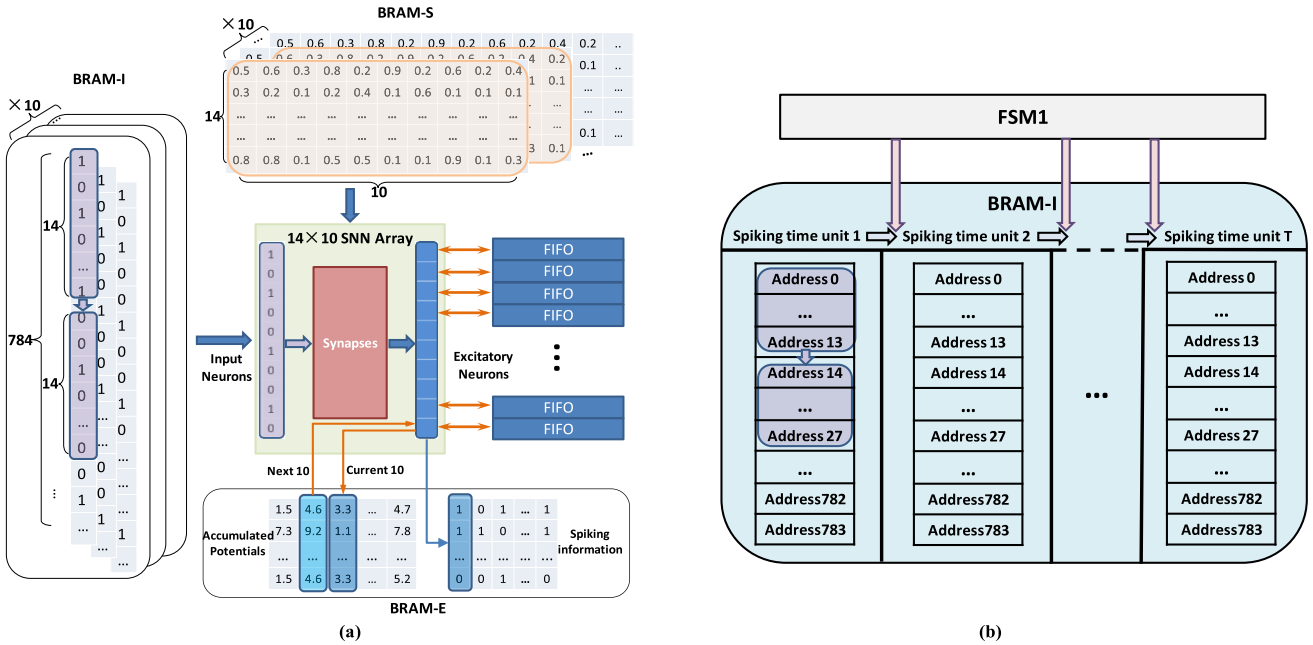

Fig. 15. Data flow of proposed TDM-based SNN accelerator for reconfigurable SNN design: (a) Datapath of the 14×10 SNN acceleration array to calculate 784×100 SNN model in one spiking time unit and (b) input data access from BRAM in different spiking time units. The scheduling schemes of BRAM-S and BRAM-E are repeated in every spiking time unit. BRAM-S stores all the synaptic weights which are updated after the calculations of excitatory neurons. BRAM-E is divided into two parts, storing membrane potentials and spiking information (i.e., whether this neuron spikes or not in current spiking time unit) of 100 excitatory neurons, respectively. As for BRAM-I, the time window of input neurons in (a) corresponds to the address window in (b), which contain spiking data of input neurons. Controlled by FSM1, the addresses of next spiking time unit are generated after the calculations of current unit are finished.

TABLE V

RUNNING SPEED OF DIFFERENT IMPLEMENTATIONS BASED 784×100 SNN MODEL WITH ENCODING TIME WINDOW SIZE 250

	Time per SOP	Time per image	Time for convergence of training
CPU (Core i7), Pytorch Library	N/A	340 ms	935 s
8-bit Heidarpur's CORDIC [20]	56.3 ns	10.7 ms	34.8 s
8-bit conventional CORDIC	36.4 ns	6.9 ms	24.1 s
8-bit fast-convergence CORDIC	30.2 ns	6.1 ms	21.4 s
8- bit LUT based implementation	26.6 ns	5.6 ms	19.6 s
6-bit conventional CORDIC	29.7 ns	6.0 ms	27.0 s
6-bit fast-convergence CORDIC	26.5 ns	5.4 ms	24.3 s
6- bit LUT based implementation	25.8 ns	5.2 ms	23.5 s

Note that the clock is set to the maximum frequency for a fair comparison.

the 10 excitatory neurons is stored into the BRAM-E, and the update of the corresponding synaptic weights is controlled by the FSM2. After this process is repeated by $\lceil \frac{N}{10} \rceil$ times, the calculation of $M \times N$ SNN model is completed for a spiking time unit.

Fig. 15 illustrates the proposed TDM-based SNN accelerator for reconfigurable SNN design, by an example of how to calculate a 784×100 SNN model in $T_{encoding}$ spiking time units. Fig. 15 (a) presents the specific data flow of one spiking time unit. The 784 input neurons are allocated into the 14×10 synapse array in 56 TDM slots. For each slot, the input spikes are fed to the 14 input neurons, while the corresponding synaptic weights are properly fetched from the BRAM-S to the synapses. Intermediate membrane potentials are calculated and then buffered into the FIFOs for the next-slot accumulation. After 56 slots, the final potentials are transferred to the BRAM- E, and then the STDP-based weight update is carried out depending on the firing information of the 10 excitatory neurons. Meanwhile, the potentials of the next 10 excitatory neurons are fetched from BRAM-E. The above process repeats 10 times for total 100 excitatory neurons, until the calculations of this spiking time unit are finished. Then, the sliding window will slide to the next spiking time unit to read the spikes of out of the BRAM-I. The above operation is repeated for all spiking time units of T controlled by FSM1, as shown in Fig. 15(b).

B. FPGA Implementation and Test

The proposed reconfigurable CORDIC SNN design with the 14×10 hardware accelerator is synthesized and implemented into the selected Zynq device. After the implementation, a test code in C runs in the PS to implement the control tasks including the data transfer of the initial weights and input spiking sequences, the spiking information calculated from the SNN accelerator. Simulation has also been performed to compare the 784×100 network in Python program and the hardware network in FPGA implementation to verify the function of the proposed reconfigurable CORDIC SNN design.

IV. IMPLEMENTATION RESULTS AND DISCUSSION

A. Hardware Utilization and Timing

Table IV presents the results of FPGA's utilization and timing for comparing the fast-convergence AR-based CORDIC against the conventional CORDIC, Heidarpur's CORDIC in [20] and LUT based non-CORDIC implementation method in [9]. Although the LUT-based exponentiation takes less slice LUTs and slice registers, it consumes extra considerable numbers of BRAMs compared with the other CORDIC implementation methods. The fast-convergence CORDIC SNN

TABLE VI
ON-FPGA POWER ANALYSIS OF ENERGY PER SOP WITH DIFFERENT IMPLEMENTATIONS BASED SNN DESIGNS

	Dynamic energy (pJ per SOP)	Static energy (pJ per SOP)	Total energy (pJ per SOP)	Average iterations of CORDIC	Average SOP duration (ns)
8-bit Heidarpur's CORDIC [20]	308.4	22.6	331.0	8	110.0
8-bit conventional CORDIC	302.5	20.3	322.8	8	110.0
8-bit fast-convergence CORDIC	162.7	13.9	176.6	4.18	71.8
8- bit LUT based implementation	525.1	18.5	543.6	N/A	50.0
6-bit conventional CORDIC	207.3	11.2	218.5	6	90.0
6-bit fast-convergence CORDIC	125.6	8.5	134.1	2.77	57.7
6- bit LUT based implementation	397.4	17.9	415.3	N/A	50.0

Note that the clock is set to 100 MHz for a fair comparison.

consumes more utilization than conventional and Heidarpur's CORDIC, mainly because the fast-convergence CORDIC requires extra LUT to store scaling factors and logic slices to perform shift & add based scaling [21], [35]–[39], as discussed in Section II. However, the fast-convergence CORDIC can achieve faster STDP learning and better energy-efficiency over the conventional and Heidarpur's CORDIC. Note that the maximum frequency of Heidarpur's CORDIC based SNN is lower than the others. This is because Heidarpur's CORDIC requires more shift & add operations to multiply the pre-calculated values in the iterations [19], which causes a much longer carry chain than the other CORDIC designs.

B. Running Speed and Power

To evaluate the learning speed performance of the proposed CORDIC SNN design, time consuming per Synaptic Operation (SOP), total time for processing an image, and total time for training convergence are defined as:

$$T_{perSOP} = (N_{iteration} + N_{other} + N_{extra}) \times T_{clock}, \tag{6}$$

$$T_{perimage} = \frac{N_{input} \times N_{exc}}{14 \times 10} \times T_{encoding} \times \left(T_{perSOP} + T_{tr}\right), \tag{7}$$

$$T_{convergence} = T_{perimage} \times ave - iterations. \tag{8}$$

Equation (6) describes the time per SOP by the proposed CORDIC STDP circuit in Fig. 13 (a), where $N_{iteration}$, N_{other} andN_{extra} stand for the average iteration cycles of CORDIC unit, the total cycles of other non-CORDIC units and the extra overhead cycles caused by the pipeline's architecture, respectively.T_{clock} is the synthesized clock period. Equation (7) describes the total time for processing an MNIST image by the proposed reconfigurable SNN design in Fig. 14, where N_{input} and N_{exc} are the number of input neurons and excitatory neurons, respectively. $T_{encoding}$ is the size of encoding time window, while T_{tr} is the consumed time of intermediate value's transfer between FIFO and neurons, i.e., approximately 4 clock cycles. Equation (8) describes the approximated time for training convergence by the proposed SNN design, by ignoring the data transfer time between the PS and PL. Here, the *ave-iterations* is the average iteration number of learning based on CORDIC presented in Table II.

Table V presents the running speed results of the proposed reconfigurable SNN design based on different STDP circuit implementations to compute the 784 × 100 SNN model. The learning performance of proposed fast-convergence CORDIC SNN is much better than the other methods. Compared to the Heidarpur's CORDIC based SNN, the fast-convergence CORDIC SNN achieves significantly faster STDP learning by 46.4%, 43.0%, and 38.5%, in terms of time per SOP, time per image, and time for learning convergence, respectively. This is because the fast-convergence CORDIC SNN takes much fewer CORDIC iterations for exponentiation in STDP learning rule.

TABLE VII
ON-FPGA POWER ANALYSIS OF ENERGY CONSUMPTION PER TRAINING WITH DIFFERENT IMPLEMENTATIONS BASED SNN DESIGNS

	Dynamic energy (mJ per image)	Static energy (mJ per image)	Total energy (mJ per image)
8-bit Heidarpur's CORDIC [20]	7.461	2.589	10.050
8-bit conventional CORDIC	7.345	2.443	9.788
8-bit fast-convergence CORDIC	4.605	2.041	6.646
8- bit LUT based implementation	11.708	2.183	13.891
6-bit conventional CORDIC	5.479	1.988	7.467
6-bit fast-convergence CORDIC	3.878	1.924	5.802
6- bit LUT based implementation	9.205	2.179	11.384

Note that the clock is set to 100 MHz for a fair comparison.

Table VI presents the power simulation results on FPGA evaluation kit. Energy per SOP is used for the energy efficiency evaluation, which is based on synaptic module level simulation. The results in this table do not include power of DDR3 accesses, as the proposed SNN doesn't need to buffer the weights in the DDR3 during the learning, and the power for retrieving input spikes data from DDR3 has been neglected. The energy efficiency of proposed CORDIC-based SNN is evaluated by energy consumption per SOP during the STDP on-line learning. Note that dynamic and static energy consumption is used in this FPGA-based comparison. As the fast-convergence CORDIC takes much fewer iterations and computation to finish STDP calculation, the proposed fast-convergence CORDIC SNN design achieves dynamic energy saving significantly. In terms of static energy consumption, although the fast-convergence CORDIC requires more logic, the static energy consumption per SOP based on fast-convergence CORDIC is still the lowest due to the less

iterations and therefore shorter computational time (resulting in lower leakage energy consumption). Therefore, the total energy consumption based on 8-bit fast-convergence CORDIC achieves 40.7% and 45.3% energy saving than the other two CORDIC designs, respectively. Although the LUT-based non-CORDIC implementation achieves faster training speed, the power consumption of LUT-based implementation is much more than the CORDIC-based methods due to the power consumption contributed by the large number of BRAMs. Table VII presents the energy consumption per training (i.e., one image) of different CORDIC and LUT based implementations, which exhibits that the fast-convergence CORDIC based SNN design can also reduce training energy significantly.

In summary, the above analysis of hardware implementation results confirms that the proposed fast-convergence CORDIC SNN design is more hardware-efficient than both the conventional and the-state-of-the-art CORDIC designs in terms of the STDP learning performance and energy efficiency.

V. Conclusion

This paper presents a novel CORDIC based Spiking Neural Network (SNN) design with on-line STDP learning and high hardware efficiency. A system design and evaluation method of CORDIC SNN is proposed to evaluate the hardware efficiency of SNN based on different CORDIC algorithm types and bit-width precisions. The comprehensive evaluation proves that selected 8-bit fast-convergence CORDIC with pipeline AR method outperforms both the conventional and the-state-of-the-art CORDIC designs, in terms of the theoretical CORDIC-level error analysis, the application-level learning and classification performance on MNIST benchmark. Based on the theoretical and systematic analysis, a reconfigurable SNN design employing the 8-bit fast-convergence CORDIC and utilizing time division multiplexing strategy is implemented in a Xilinx Zynq FPGA device. The implementation results confirm that the proposed fast-convergence CORDIC SNN design has better hardware efficiency than the conventional and state-of-the-art CORDIC methods, in terms of the STDP learning speed and energy efficiency, the image processing speed and learning convergence speed. The presented research work on CORDIC SNN design also extends a solid step toward systematic design and evaluation of SNN with bio-plausible on-line learning to achieve different hardware design metrics by exploring various CORDIC algorithms [35] in digital platforms including FPGAs and ASICs, which can be studied in the future.

References

[1] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *Int. J. Neural Syst.*, vol. 19, no. 4, pp. 295–308, Aug. 2009.

[2] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers Comput. Neurosci.*, vol. 9, p. 99, Aug. 2015.

[3] A. Sboev, D. Vlasov, R. Rybka, and A. Serenko, "Solving a classification task by spiking neurons with STDP and temporal coding," *Procedia Comput. Sci.*, vol. 123, pp. 494–500, Jan. 2018.

[4] M. Chu *et al.*, "Neuromorphic hardware system for visual pattern recognition with memristor array and CMOS neuron," *IEEE Trans. Ind. Electron.*, vol. 62, no. 4, pp. 2410–2419, Apr. 2015.

[5] C. Mayr *et al.*, "A biological-realtime neuromorphic system in 28 nm CMOS using low-leakage switched capacitor circuits," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 1, pp. 243–254, Feb. 2016.

[6] B. V. Benjamin *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[7] R. M. Wang, T. J. Hamilton, J. C. Tapson, and A. van Schaik, "A mixed-signal implementation of a polychronous spiking neural network with delay adaptation," *Frontiers Neurosci.*, vol. 8, p. 51, Mar. 2014.

[8] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.

[9] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, Apr. 2019.

[10] N. Qiao *et al.*, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Frontiers Neurosci.*, vol. 9, p. 141, Apr. 2015.

[11] D. Neil and S.-C. Liu, "Minitaur, an event-driven FPGA-based spiking network accelerator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 12, pp. 2621–2628, Dec. 2014.

[12] S. Gomar and A. Ahmadi, "Digital multiplierless implementation of biological adaptive-exponential neuron model," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 4, pp. 1206–1219, Apr. 2014.

[13] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," *Frontiers Neurosci.*, vol. 7, p. 272, Jan. 2014.

[14] Z. Liu, T. Q. S. Quek, and S. Lin, "Variational probability flow for biologically plausible training of deep neural networks," in *Proc. AAAI*, 2018, pp. 1–8. [Online]. Available: http://arxiv.org/abs/1711.07732

[15] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[16] J. Pei *et al.*, "Towards artificial general intelligence with hybrid Tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, Aug. 2019.

[17] A. S. Cassidy, J. Georgiou, and A. G. Andreou, "Design of silicon brains in the nano-CMOS era: Spiking neurons, learning synapses and neural architecture optimization," *Neural Netw.*, vol. 45, pp. 4–26, Sep. 2013.

[18] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. 8, no. 3, pp. 330–340, Sep. 1959.

[19] M. Heidarpour, A. Ahmadi, and R. Rashidzadeh, "A CORDIC based digital hardware for adaptive exponential integrate and fire neuron," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 11, pp. 1986–1996, Nov. 2016.

[20] M. Heidarpur, A. Ahmadi, M. Ahmadi, and M. R. Azghadi, "CORDIC-SNN: On-FPGA STDP learning with Izhikevich neurons," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 7, pp. 2651–2661, Jul. 2019.

[21] C. Wang, J. Luo, and J. Zhou, "A 1-V to 0.29-V sub-100-pJ/operation ultra-low power fast-convergence CORDIC processor in 0.18-μm CMOS," *Microelectron. J.*, vol. 76, pp. 52–62, Jun. 2018.

[22] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers Neurosci.*, vol. 10, p. 508, Nov. 2016, doi: 10.3389/fnins.2016.00508.

[23] A. Tavanaei, Z. Kirby, and A. S. Maida, "Training spiking ConvNets by STDP and gradient descent," in *Proc. IJCNN*, Jul. 2018, pp. 1–8.

[24] J. M. Brader, W. Senn, and S. Fusi, "Learning real-world stimuli in a neural network with spike-driven synaptic dynamics," *Neural Comput.*, vol. 19, no. 11, pp. 2881–2912, Nov. 2007.

[25] C. Lee, P. Panda, G. Srinivasan, and K. Roy, "Training deep spiking convolutional neural networks with STDP-based unsupervised pre-training followed by supervised fine-tuning," *Frontiers Neurosci.*, vol. 12, pp. 435–447, Aug. 2018.

[26] A. Cassidy, A. G. Andreou, and J. Georgiou, "A combinational digital logic approach to STDP," in *Proc. IEEE ISCAS*, May 2011, pp. 673–676.

[27] A. Sboev, D. Vlasov, R. Rybka, and A. Serenko, "Solving a classification task by spiking neurons with STDP and temporal coding," in *Proc. 8th Annu. Int. Conf. Biol. Inspired Cogn.*, 2017, pp. 1–7.

[28] Q. Wang, Y. Li, B. Shao, S. Dey, and P. Li, "Energy efficient parallel neuromorphic architectures with approximate arithmetic on FPGA," *Neurocomputing*, vol. 221, pp. 146–158, Jan. 2017.

[29] B. Nessler, M. Pfeiffer, L. Buesing, and W. Maass, "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity," *PLoS Comput. Biol.*, vol. 9, no. 4, Apr. 2013, Art. no. e1003037.

[30] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," *Neural Netw.*, vol. 99, pp. 56–67, Mar. 2018.

[31] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, and T. Masquelier, "Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks," *Pattern Recognit.*, vol. 94, pp. 87–95, Oct. 2019.
[32] P. Falez, P. Tirilly, I. M. Bilasco, P. Devienne, and P. Boulet, "Multi-layered spiking neural network with target timestamp threshold adaptation and STDP," in *Proc. IJCNN*, Jul. 2019, pp. 1–8.
[33] R. Vaila, J. Chiasson, and V. Saxena, "Feature extraction using spiking convolutional neural networks," in *Proc. ICONS*, Jul. 2019, pp. 1–8.
[34] H. Hazan *et al.*, "BindsNET: A machine learning-oriented spiking neural networks library in Python," *Frontiers Neuroinform.*, vol. 12, p. 89, Dec. 2018.
[35] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 9, pp. 1893–1907, Sep. 2009.
[36] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. London, U.K.: Oxford Univ. Press, 2000.
[37] Y. H. Hu and S. Naganathan, "An angle recoding method for CORDIC algorithm implementation," *IEEE Trans. Comput.*, vol. 42, no. 1, pp. 99–102, Jan. 1993.
[38] T. K. Rodrigues and E. E. Swartzlander, "Adaptive CORDIC: Using parallel angle recoding to accelerate rotations," *IEEE Trans. Comput.*, vol. 59, no. 4, pp. 522–531, Apr. 2010.
[39] P. K. Meher and S. Y. Park, "CORDIC designs for fixed angle of rotation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 2, pp. 217–228, Feb. 2013.

Jiajun Wu (Member, IEEE) was born in Fujian, China, in 1999. He is currently pursuing the bachelor's degree with the School of Optical and Electronic Information, Huazhong University of Science and Technology (HUST), Wuhan, China.

His research interests include energy-efficient VLSI architecture design and ultra-low-voltage circuit design for machine learning, neuromorphic computing, and robot applications.

Yi Zhan (Member, IEEE) was born in Hubei, China, in 1998. He received the B.Eng. degree in electronics engineering from Northwestern Polytechnical University, Xi'an, China, in 2019. He is currently pursuing the master's degree with the School of Optical and Electronic Information, Huazhong University of Science and Technology (HUST), Wuhan, China.

His research interests include energy-efficient digital signal processor design and ultra-low-voltage circuit design for biomedical/healthcare, wireless sensor, neuromorphic computing, and robot applications.

Zixuan Peng (Member, IEEE) was born in Wuhan, China, in 1997. He is currently pursuing the bachelor's degree with the School of Optical and Electronic Information, Huazhong University of Science and Technology (HUST), Wuhan, China.

His research interests include energy-efficient VLSI architecture design and ultra-low-voltage circuit design for biomedical/healthcare, neuromorphic computing, and robot applications.

Xinglong Ji received the Ph.D. degree from the Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, in 2016. He was with the Department of Engineering Product Development, Singapore University of Technology and Design. He is currently with the Center for Brain-Inspired Computing Research, Tsinghua University. His current research interests include non-volatile memory technologies and emerging nanoelectronics for neuromorphic applications.

Guoyi Yu (Member, IEEE) received the B.Eng., M.Eng., and Ph.D. degrees in electronics engineering from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2000, 2003, and 2006, respectively.

Since 2007, he had been with the Department of Electronic Science and Technology, HUST, as a Lecturer. He is currently an Associate Professor with the School of Optical and Electronic Information, HUST. His research interests include analog-mixed signal circuit design, sensor interface circuit design, and heterogeneous 3D-IC design for biomedical/healthcare, wireless sensor, wearable and robot applications. He is also the Treasurer of the IEEE CASS-EDS-SSCS Wuhan Joint Chapter. He is also an active Reviewer of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS, IEEE ACCESS journal, and *Microelectronics Journal* (Elsevier).

Rong Zhao (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the National University of Singapore, Singapore, in 1999.

She is currently with the Department of Precision Instruments, the Center for Brain-Inspired Computing Research, and the Innovation Center for Future Chip, Tsinghua University. Prior to joining Tsinghua University, she has been an Associate Professor in engineering product development with the Singapore University of Technology and Design (SUTD). She had been worked as a Senior Scientist, a Principle Investigator, and the Assistant Division Manager at the Data Storage Institute, Agency for Science, Technology and Research (A∗STAR). She is the author or the coauthor of over 100 publications in international journals and international conference proceedings. Her main research interests include non-volatile memories (phase-change memory and resistive memory) and reconfigurable devices covering from material synthesis and device design/fabrication to chip design/prototyping. More recently, she has broadened her activities, entering the field of artificial cognitive memory and energy harvesters. She was the Co-Chair of the Technical Committee of the IEEE Non-Volatile Memory Technology Symposium from 2012 to 2014 and a Lead Organizer of the Material Research Society in 2011 and 2012 spring meetings of the Phase Change Symposium.

Chao Wang (Senior Member, IEEE) received the B.Eng. degree in electronics engineering from the Huazhong University of Science and Technology (HUST), China, in 2000, and the Ph.D. degree in electronics engineering from Nanyang Technological University (NTU), Singapore, in 2008.

From 2005 to 2007, he was with the Center for Signal Processing, NTU, as a Research Engineer. He worked as an IC Design Engineer with STMicroelectronics, Asia-Pacific Design Centre, Singapore, from 2008 to 2010. He participated in the development of STM Bayer/YUV CMOS image sensing and processing ICs/SoCs. From 2010 to 2017, he was a Research Scientist and the Project Leader with the Institute of Microelectronics (IME), Agency for Science, Technology and Research (A∗STAR), Singapore, where he led projects on wearable medical devices for cardio-engineering applications, and MEMS Accelerometer ASICs for medical and navigation applications. He is currently a Professor with HUST and the Wuhan National Laboratory of Optoelectronics (WNLO), Wuhan, China. His major research interests include energy-efficient digital signal processor design, ultra-low-voltage circuit design, MEMS sensor ASIC design, and heterogeneous 3D-IC design, especially for biomedical/healthcare, wireless sensor, IoT, neuromorphic computing, and robot applications.

Dr. Wang has served as a TPC member for a number of international devices, circuits, and systems conferences. He is also the Chair of the IEEE CASS-EDS-SSCS Wuhan Joint Chapter. He used to serve as a Committee Member, the Vice Chair, and the Chair of the IEEE SSCS Singapore Chapter. He served as a Guest Editor for *Sensors* journal, Hindawi in 2016, and IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS in 2019. He is also an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, *IEEE Circuits and Systems Magazine*, IEEE ACCESS journal, and *Microelectronics Journal* (Elsevier).