Towards the Optimal Design and FPGA Implementation of Spiking Neural Networks

Wenzhe Guo, *Graduate Student Member, IEEE*, Hasan Erdem Yantir, *Member, IEEE*,
Mohammed E. Fouda, *Member, IEEE*, Ahmed M. Eltawil, *Senior Member, IEEE*,
and Khaled Nabil Salama, *Senior Member, IEEE*

Abstract—The performance of a biologically plausible spiking neural network (SNN) largely depends on the model parameters and neural dynamics. This paper proposes a parameter optimization scheme for improving the performance of a biologically plausible SNN and a parallel on-FPGA online learning neuromorphic platform for the digital implementation based on two numerical methods, namely Euler and the 3rd-order Runge-Kutta (RK3) methods. The optimization scheme explores the impact of biological time constants on information transmission in the SNN and improves the convergence rate of the SNN on digit recognition with a suitable choice of the time constants. The parallel digital implementation leads to a significant speedup over software simulation on a general-purpose CPU. The parallel implementation with Euler method enables around $180\times$ ($20\times$) training (inference) speedup over a Pytorch-based SNN simulation on CPU. Moreover, compared with previous work, our parallel implementation shows more than $300\times$ ($240\times$) improvement on speed and $180\times$ ($250\times$) reduction on energy consumption for training (inference). In addition, due to the high-order accuracy, RK3 method is demonstrated to gain $2\times$ training speedup over Euler method, which makes it suitable for online training in real-time applications.

Index Terms—Neuromorphic computing, spiking neural network, On chip learning, unsupervised STDP learning, Euler method, Runge-Kutta method, parameter optimization, FPGA platform, parallel architecture.

I. INTRODUCTION

IN recent years, non-Von Neumann computing architectures have become popular to satisfy demands for high throughput computations [1]. Originally proposed in 1990, a neuromorphic system defines an entire system built by means of the organizing principles existing in nervous systems [2]. Neuromorphic systems feature parallel operations and collocated memory and processor.

While artificial neural networks (ANNs) have proved to be extraordinarily successful in various applications [3]–[5], their architecture inherently requires a massive amount of computing power due to intensive matrix multiplications and memory access. In contrast, spiking neural networks (SNNs) compute like a human brain by emulating and modeling the structure of nervous systems, which transmit information in the form of action potential (or spikes) [6]. In the network, the information transmission is driven by different events that are

generally sparse in time. Thus, with the event-driven property, SNNs perform energy-efficient computation and have shown high potential in real-time applications [7]–[10]. Due to its functional similarity to biological nervous system, SNNs have been explored in a wide range of applications, such as image classification [11], object detection [12], navigation [13] and motor control [14]. In this paper, we construct SNNs based on the leaky integrate-and-fire (LIF) model which is the most widely used neuron model in neuromorphic systems. The model is composed of one simple first-order linear differential equation and a threshold condition [15]. The synapse is modelled by ionic channels with time-varying conductance [16].

SNNs can be trained with different supervised and unsupervised learning algorithms. Recently, to enhance the learning capability of SNNs, different variants of supervised backpropagation (BP) algorithm has been introduced for SNNs tackling the issue of non-differentiable neural models [17]–[19]. These methods perform gradient descent technique on the modified continuous membrane potential function. By incorporating BP algorithms, SNNs start to perform closely to ANNs in various recognition tasks. However, this comes at the cost of long training time and high computational power, making BP algorithms not suitable for developing neuromorphic systems, which motivates local learning techniques such as DECOLLE [20]. On the other hand, SNNs can be directly trained with biologically plausible unsupervised learning rules (e.g., spike-timing-dependent plasticity (STDP)) which update synaptic weights using local spiking information between a presynaptic neuron and postsynaptic neuron. Although STDP learning rule still lags behind supervised BP algorithms, recent efforts on applying STDP learning rule in deep networks have shown potential in closing the gap [16], [21], [22]. Besides, STDP learning rule is able to analyze spatio-temporal information and demonstrated to be very useful in solving difficult computational problems [23]–[25]. Because of its biological plausibility, STDP is also of great interest to neuroscientists to investigate the learning mechanism happening in human brains. Moreover, with local event-driven updating features, STDP learning rule can be efficiently implemented on hardware and enables scalable online learning in various neuromorphic systems [26]–[28]. In this work, a triplet-based spike-timing-dependent plasticity (STDP) model is considered because of its biological plausibility and easy implementation [29].

To perform time-domain computation, SNNs need to be implemented as a highly-parallel hardware system similar to the structure of a human brain. Towards that end, hard-

W. Guo, H. E. Yantir, A. E. Eltawil, and K. N. Salama are with the Division of Computer, Electrical and Mathematical Sciences and Engineering, King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia. (e-mail: {wenzhe.guo, hasan.yantir, ahmed.eltawil, khaled.salama}@kaust.edu.sa)

M. E. Fouda is with the Department of EECS, University of California–Irvine, Irvine, CA 92612 USA (e-mail: foudam@uci.edu)

We acknowledge the funding support from King Abdullah University of Science and Technology (KAUST) AI Initiative, Saudi Arabia.

ware systems using both analog or digital circuits have been proposed [26], [27], [30]–[34]. Despite the advantages of small area and low power consumption, analog systems suffer from poor flexibility, noise, device variabilities, and lack of area-efficient storage elements [35], [36]. In contrast, digital systems provide great reconfigurability, stability, and efficient embedded memory. Particularly, field programmable gate array (FPGA) provides a low-cost reconfigurable platform that supports the design of parallel processing architectures, which makes it very suitable for the functional and topological exploration of large-scale spiking neural networks.

Numerous examples of the implementations of SNNs on FPGA have been reported for different applications [37]–[41]. Yang et al. reported a real-time digital neuromorphic system that was used to simulate the cortico-basal ganglia-thalamocortical network with 1 million neurons [38]. Neil et al. developed an event-driven deep spiking network accelerator which was demonstrated to operate with very low power consumption [39]. Wang et al. implemented a spiking neural network on FPGA for the application of digit recognition [41]. With the consideration of parallel processing and approximate computing in the architecture design, they managed to achieve around 60x speedup over the software program on general-purpose CPU and reduce the energy consumption by 20%.

The previous works have focused on the architectural design of SNNs on FPGA towards low-cost and low-power development. In these works, the neurodynamics models in SNNs are implemented using Euler method which requires a small number of hardware resources and low power. However, this numerical method produces a first-order accuracy, which potentially may degrade network performance. Recent efforts considered more accurate numerical approaches such as high-order Runge-Kutta (RK) methods [42], [43]. While these works provide useful insights into the implementation of neuron models with different numerical methods, they fail to present a comprehensive study of the end-to-end performance of SNN utilizing different numerical methods. Therefore, in this paper, we investigate the performance of two different numerical methods, namely Euler method and 3rd-order Runge-Kutta (RK3) method in the spiking neural network implemented on FPGA for pattern classification, and provide a comprehensive comparison in terms of classification accuracy, resource utilization, energy consumption and FPGA runtime. The main contributions of this paper are listed as follows.

- 1) We introduce an optimization scheme for the biological parameters in the SNN and investigate the impact of the time constants on the information transmission in the network. The optimization scheme improves the classification accuracy and convergence rate.
- 2) A digital implementation of SNNs with parallel architecture capable of online learning is proposed in this paper. The parallel implementation significantly accelerates image processing in the network over software simulation. An efficient method for memory storage of synaptic weights is presented.
- 3) The implementations of SNNs based on two numerical methods are presented. A detailed comparison between the two methods reveals that RK3 method is promising for online training in real-time application and Euler method is suitable for low-cost implementation and low-

energy inference applications.

The rest of this paper is organized as follows. Section II introduces the network architecture, the numerical methods and optimization scheme. Section III provides the details of digital implementation and the schemes to parallelize the network architecture. Section IV summarizes the simulation results for the parameter optimization and implementation results for both numerical methods. Section V compares our study with previous work and discusses the potential benefits of our design. Section VI concludes the paper.

II. METHODS

A. Neural Models

1) *LIF neuron model*: LIF model is composed of one first-order linear differential equation and a threshold condition. The dynamics of membrane potential v is described as

$$\begin{aligned} C_m \frac{dv}{dt} &= g_l(v_r - v) - I_s \\ v &= v_r, \text{ if } v > v_{th} \end{aligned} \quad (1)$$

where C_m is the membrane capacitance, v_r is the resting membrane potential, v_{th} is the threshold potential, g_l is the leaky conductance, and I_s is the input synaptic current.

2) *Synaptic dynamics*: Synapse is modelled by different ionic channels with time-varying conductance. The total postsynaptic current I_s can thus be modelled as

$$I_s = g_e(v - E_{exc}) + g_i(v - E_{inh}) \quad (3)$$

where g_e is the conductance associated with the excitatory channel, E_{exc} is the reverse potential of the channel, g_i is the conductance associated with the inhibitory channel, E_{inh} is the reverse potential of the channel. The dynamics of the conductance is governed by

$$\tau_g \frac{dg}{dt} = -g + \sum_j w_{ij} \delta(t - t_j^f) \quad (4)$$

where g is the conductance, τ_g is the time constant, w_{ij} is the synaptic weight from the presynaptic neuron j to the postsynaptic neuron i , and t_j^f is the firing time of the presynaptic neuron j .

3) *Triplet STDP*: A triplet-based STDP model considers sets of three spikes (one presynaptic and two postsynaptic spikes) and each spike leaves a time-varying trace whose dynamics is described as below [29],

$$\frac{dx_j}{dt} = -x_j / \tau_x \quad (5)$$

$$\frac{dy_i}{dt} = -y_i / \tau_y \quad (6)$$

where x_j is the trace variable associated with the firing event of the presynaptic neuron j , y_i is the trace variable associated with the firing event of the postsynaptic neuron i , τ_x and τ_y are the corresponding time constants. y represents a fast and slow trace variables denoted as y_i^1 and y_i^2 respectively. When the presynaptic neuron (postsynaptic) fires, the relative trace x_j (y_i) is reset to 1. The weight updates are carried out on the occurrence of presynaptic or postsynaptic spikes.

$$\Delta w_{ij} = \begin{cases} -\mu_{pre} y_i^1, & \text{if neuron } j \text{ fires,} \\ +\mu_{post} x_j y_i^2, & \text{if neuron } i \text{ fires.} \end{cases} \quad (7)$$

Fig. 1: (a) Illustration of the triplet-based STDP model. (b) The architecture of a spiking neural network. An input layer consists of 784 neurons, receiving the pixel values from one input image and converting them into Poisson spike trains in time. A processing layer is built in a WTA network that includes 100 excitatory (Exc) and 100 inhibitory (Inh) neurons.

where μ_{pre} and μ_{post} are the corresponding weight updating rates. The weight updating process is illustrated in Fig. 1a.

B. Network Architecture

The architecture of the spiking neural network is constructed according to the reference [16]. The schematic of the network is shown in Fig. 1b. Hand-written digit images from the MNIST dataset are used as inputs that are converted into Poisson spike trains [44]. A winner-take-all (WTA) network and threshold adaption scheme are applied to induce the competition for activation among neurons so that each neuron gets to learn different input pattern. We chose this network architecture for two main reasons; 1) Reasonable performance: despite the simplicity of the network which consists of two layers, it can achieve 95% classification accuracy with an unsupervised STDP learning rule only [16], and 2) Biological plausibility: the network structure uses a combination of different biological plausible models, including LIF neuron model, conductance-based synaptic model, STDP model, WTA network model, and intrinsic plasticity model. It closely mimics biological processes and provides a good platform to study

how a biological neural network performs computations unsupervised. It also enables us to study the impact of temporal information transmission on network performance, as all the neural models largely depend on spike timing.

C. Numerical methods

In this paper, we adopt two numerical methods with different complexity and accuracy to implement the SNN, namely the Euler method [45] and the third order Runge-Kutta method [46].

Consider the differential equation, $y' = f(t, y)$, where $f(\cdot)$ is some arbitrary function of time t and y .

1) *Euler's method*: The iterative numerical formula at the time point n is defined as,

$$y[n+1] = y[n] + hf(t[n], y[n]) \quad (8)$$

where h is the step size. This method produces first-order accuracy, where the global error at a given time point is proportional to the step size.

2) *3rd-order Runge-Kutta (RK3) method*: RK3 method produces third-order accuracy at the cost of higher computational complexity as shown below [46],

$$y[n+1] = y[n] + \frac{1}{6}h(k_1 + 4k_2 + k_3) \quad (9)$$

where k_1 , k_2 and k_3 are defined as:

$$k_1 = f(t[n], y[n]) \quad (10)$$

$$k_2 = f\left(t[n] + \frac{1}{2}h, y[n] + \frac{1}{2}k_1h\right) \quad (11)$$

$$k_3 = f(t[n] + h, y[n] - k_1h + 2k_2h) \quad (12)$$

D. Parameter optimization

Neural model biological parameters play a crucial role in the capability of the SNN. To simulate the SNN, we have to consider all key parameters involved in different neural models [47]. Namely, we aim to optimize the time constants of the conductance model and STDP model, the learning rates of the STDP model, and the threshold adaption constant, as these parameters are critical to the learning capability of the network. Table I lists the corresponding model parameters.

In order to find the best training parameters that achieve the best performance metrics, a search optimization algorithm is needed. In this work, we chose a genetic algorithm (GA) since it avoids being trapped in local optimal points compared to gradient-based methods where a GA searches the entire

TABLE I: Model parameters to be optimized.

Parameters	Description
τ_x	the time constant for the presynaptic trace x in the STDP model
τ_1, τ_2	the time constants for the postsynaptic traces y_1 (fast) and y_2 (slow) in the STDP model, respectively. $\tau_2 = 2\tau_1$
$\tau_{ge_exc}, \tau_{gi_exc}$	the time constants for the excitatory and inhibitory conductance channels of an excitatory neuron, respectively
τ_{ge_inh}	the time constant for the excitatory conductance channel of an inhibitory neuron
μ_{pre}, μ_{post}	the learning rates of weight updates in the STDP model
θ^+	threshold adaption constant

TABLE II: GA Parameters.

Parameters	Value
The number of generations N_{gen}	20
The number of individuals in a population N_i	20
The number of parents N_p	7
The number of offspring N_{off}	18

solution space and reaches the optimal point. In addition, a GA is inherently parallel and can be easily distributed [48]. The GA consists of five phases, including parameter initialization, fitness calculation, parent selection, crossover, and mutation process, as described below. The algorithm is described in Algorithm 1, and the corresponding parameters are listed in Table II.

- **Parameter initialization:** randomly initialize the selected parameters to be optimized for each individual in a population.
- **Fitness calculation:** calculate the fitness score for each individual in a population.
- **Parent selection:** select N_p individuals with the best fitness scores as parents.
- **Crossover:** Randomly select a crossover point in the parameter set. Produce an offspring by combining the parameters selected from the first parent before the point and the parameters from the second parent after the point.
- **Mutation:** Randomly select a parameter from the produced offspring's parameter set and mutate its value.

The fitness function is defined by considering both the classification accuracy and total time constant. The time constants in the differential equations determine how fast the variables change with time and hence have a direct impact on the efficiency and effectiveness of information transmission in the network. In other words, we need to do fast training and inference while achieving the highest possible accuracy. Thus, the optimization problem is defined as

$$\begin{aligned} \max_{\mathbf{x}} \quad & w\text{Accuracy} + (1 - w)\tau_n \\ \text{s.t.} \quad & \mathbf{x} \geq 0 \end{aligned} \quad (13)$$

where $\mathbf{x} = \{\tau_x, \tau_1, \tau_{ge_exc}, \tau_{gi_exc}, \tau_{ge_inh}, \mu_{pre}, \mu_{post}, \theta^+\}$. These parameters are as defined in Table I and τ_n is the normalized total time constant, defined as

$$\tau_n = (\tau_{max} - \tau_{tot}) / (\tau_{max} - \tau_{min}) \quad (14)$$

$$\tau_{tot} = \tau_x + \tau_1 + \tau_{ge_exc} + \tau_{gi_exc} + \tau_{ge_inh} \quad (15)$$

τ_{max} and τ_{min} are the maximum value and the minimum value of the total time constant respectively, and w is a weighting factor. The impact of the time constants on the optimization can be adjusted with the weight factor w . Since τ_n is inversely proportional to the total time constant, the optimization problem defined in (13) aims to search for a set of parameters that ensure high classification accuracy and small time constants at the same time. By solving this optimization problem, one can compare the impact of different parameter sets with regard to the number of time steps required per input image to reach the maximum classification accuracy.

III. HARDWARE IMPLEMENTATION

In this section, the digital design and considerations of SNNs are discussed in detail. Fig. 2 shows the overall system

Algorithm 1: Genetic Algorithm

```

Specify input parameters:  $N_{gen}, N_i, N_p, N_{off}$ .
for  $k = 1$  to  $N_{gen}$  do
  if the first generation then
    | Parameter initialization.
  end
  else
    | Create a new population.
    | Parent selection.
    | Pass 10% of individuals of the current
      | population with the best fitness values to the
      | new population.
    for  $i = 1$  to  $N_{off}$  do
      | Select a pair of parents if the pair has not
        | been selected before.
      | Apply crossover to produce an offspring.
      | Mutate the offspring's parameter.
      | Add the mutated offspring into the new
        | population.
    end
  end
  | Fitness calculation.
end

```

design of the spiking neural network. Initially, all the input images are transferred from PC to the external memory (i.e., DDR SDRAM) on FPGA through a microprocessor. The microprocessor manages the communication between direct memory access (DMA) and the external memory. So image pixels are transferred from the external memory to the network through DMA and converted into Poisson spike trains. These spike trains control the read operations of synaptic weights from BRAMs so that certain weights are sent to the neuron processing cores. Then the neuron processing cores update their membrane potentials and generate spikes if the membrane potentials surpass the thresholds. The generated spikes from the SNN trigger the STDP learning unit before being sent out to the DDR memory and then extracted to PC. The SNN is mainly composed of Poisson spikes generator, neuron units, STDP learning unit, and memory blocks.

Fig. 2: The overview of the proposed digital online learning system. The processing unit manages the communication between PC and the SNN unit. The SNN unit processes the input images and sends out spikes. In the WTA network, w_{exc} and w_{inh} are the constant synaptic weights between the excitatory layer and inhibitory layer.

Fig. 3: The digital implementation of LIF neuron model with different numerical methods. (a) details the design of LIF neuron model based on Euler method. The current module produces the input synaptic current. The spike detection module checks the occurrence of an output spike. (b) shows the design based on RK3 method. The current module is repeatedly used through the multiplexer (MUX) until the three intermediate variables are updated.

A. Poisson spike generator

The Poisson spike generator is used to convert and encode the image pixels to stochastic spike trains with Poisson process. The generated spike trains are fed to spiking neurons in the next layer. This encoding scheme is commonly referred to as rate coding in SNNs [49]. A random number generator is implemented by using a 16-bit linear-feedback shift register (LFSR). Within a particular time window, at each time step, one random number is generated and then compared with the input pixel value. If the random number is smaller, it outputs a logic-1. Otherwise, it outputs a logic-0. In this way, a time series of 0 or 1 forms the Poisson spike train.

B. Neuron processing core

The neuron processing core consists of two parts, an arithmetic unit (AU) and a storage unit (SU). The arithmetic unit updates the state variables of a neuron, like synaptic conductance and membrane potential, by solving the related differential equations. The storage unit provides the memory space for storing the state variables. For the implementation of an AU, we apply two different numerical methods to solve the differential equations. The design of AUs is discussed below for each case.

1) *Euler-method based AU design*: The Euler method provides a straightforward way to update the variable. With the Euler method, the membrane potential in the LIF model can be updated according to the following formula.

$$I(v[n]) = g_l(v_r - v[n]) - I_s(v[n]) \quad (16)$$

$$v[n+1] = v[n] + h/C_m * I(v[n]) \quad (17)$$

where $I_s(v)$ is the synaptic current defined as a function of the membrane potential v as in (3), $v[n]$ and $v[n+1]$ are the membrane potential at time n and $n+1$ respectively, and h is the time step size. The leaky conductance, g_l , is set as 1. Fig. 3a illustrates the design diagram of solving the LIF neuron equation. First, the current module generates the synaptic current and leakage current by multiplying the conductances with the voltage drop. Then the membrane potential is updated and compared with the threshold voltage. An output spike will be detected once the membrane potential surpasses the threshold. In the design, the multiplications between the conductance and voltage drops are implemented by multipliers. The multiplication by constants is realized by shift operation by properly choosing the constants as powers of two.

2) *RK3-method based AU design*: RK3 method requires more iterative steps and complicates the design. The formulas used to update the membrane potential are expressed as below.

$$k_1 = I(v[n])/C_m \quad (18)$$

$$k_2 = I(v[n] + \frac{1}{2}k_1h)/C_m \quad (19)$$

$$k_3 = I(v[n] - k_1h + 2k_2h)/C_m \quad (20)$$

$$v[n+1] = v[n] + \frac{1}{6}h(k_1 + 4k_2 + k_3) \quad (21)$$

The detailed design is shown in Fig. 3b. We use one multiplexer (MUX) and one demultiplexer (DEMUX) to realize the time-division multiplexing function so that the current module can be used repeatedly at each iteration. The MUX and DEMUX gates are controlled by three selection signals (S_1 , S_2 , and S_3), and each signal selects a different input to the MUX gate and a different output for the DEMUX gate. The three selection signals are activated sequentially in non-overlap time windows, and activation starts after a signal for membrane potential update arrives. The intermediate variables k_1 , k_2 , and k_3 are iteratively computed by multiplexing the relative inputs. After obtaining the values of all intermediate variables, the membrane potential is updated. In the final update of the membrane potential, the multiplication by a model coefficient is reduced to shift-and-add operations by approximating the coefficient to the sum of powers of two [46]. Because of the lengthy iterative steps, it takes more resources and clock cycles to finish one time-step update in the RK3-method based AU design.

C. STDP unit

The function of the STDP learning unit is to update the synaptic weights on the occurrence of an output spike. We implemented the triplet-based STDP model [29], and the formulas used to compute weight updates based on Euler method are shown below.

$$x_j[n+1] = \begin{cases} 1, & \text{if a presynaptic neuron fires,} \\ x_j[n] - x_j[n]h/\tau_x, & \text{otherwise.} \end{cases} \quad (22)$$

$$y_i[n+1] = \begin{cases} 1, & \text{if a postsynaptic neuron fires,} \\ y_i[n] - y_i[n]h/\tau_y, & \text{otherwise.} \end{cases} \quad (23)$$

The design is shown in Fig. 4 where the positive weight update is computed. The negative update follows the similar design.

Fig. 4: The digital implementation of the STDP learning unit for positive weight update. s_{pre} and s_{post} represent the presynaptic and postsynaptic spike trains respectively. μ is the learning rate. Δw_{ij} is the weight change.

This unit is constructed of two parts, an arithmetic unit and a storage unit. Both traces are stored in the storage unit. The arithmetic unit updates the values of both traces by solving the differential equations with different numerical methods. Fig. 4 only shows the design with Euler solver. The design of RK3 solver follows the same flow as in the Fig. 3b. Subsequently, the change of the synaptic weight is computed by multiplying both traces with the learning rate μ when a postsynaptic spike comes in. All the synaptic weights associated with the firing postsynaptic neuron are read out from memory and updated by adding the change. Afterward, the updated weights are written back into the memory.

D. Parallel architecture

The synaptic connection between two layers guarantees the transmission of information. So, for the hardware implementation, reading and writing the synaptic weights are crucial operations in the whole process, which depend on the memory allocation of the synaptic weights on FPGA. In our implementation, the synaptic weights between the input layer and the second layer are stored in the Block RAMs (BRAMs) inside the FPGA, as BRAMs provide low-power, high-speed operations and ample storage space. With different schemes of storing the weights, there are two degrees of parallelism that we can consider in the design. Accordingly, two parallel architectures are depicted in the Fig. 5.

1) *Parallelizing synaptic weight accumulation*: Input spike trains act as the selectors that activate the read operation of the synaptic weights from the BRAMs only if there is an input spike. After that, the selected synaptic weights are summed and sent into the conductance channel of each excitatory neuron in the second layer. This process is equivalent to calculating the accumulation, $\sum_j w_{ij} \delta(t - t_j^f)$, where w_{ij} is the synaptic weight between the presynaptic neuron j and the postsynaptic neuron i , t_j^f is the firing time of the

(a) Parallel accumulation of synaptic weights for one neuron core.

(b) Parallel neuron processing cores.

Fig. 5: The memory allocation of synaptic weights in BRAMs and parallel structures. N_{pre} is the number of the presynaptic neurons in the input layer. N_{post} is the number of the postsynaptic neurons in the second layer. (a) Synaptic weights associated with one neuron are stored in m BRAMs. So m weights can be read at the same time and sent to the adder tree for accumulation. (b) n neuron cores can be implemented to update the membrane potentials of n neurons in the second layer in parallel. N_i denotes the storage of the synaptic weights associated with the neuron i .

presynaptic neuron j . Because of the fully-connected structure, the accumulation over all the presynaptic neurons j needs to be computed for each postsynaptic neuron i in the second layer. The straightforward implementation is to store all the weights in one single BRAM and read one weight at one time. At each clock cycle, we accumulate one weight sequentially. In this way, it takes around N_{pre} cycles to finish the accumulation, where N_{pre} is the total number of presynaptic neurons in the input layer. However, with a large number of neurons, the accumulation process could take many cycles to complete and hence, slow down the training process. Thus, a parallel design is favored to accelerate the computation. We can divide the synaptic weights associated with one postsynaptic neuron into m BRAMs as shown in the Fig. 5a, through which m weights are simultaneously read out and passed to the m -input adder tree. By pipelining the read and accumulation process, the processing time is taken up by the time for reading the weights and the time for computing the m -input addition. Since the depth of each BRAM becomes N_{pre}/m , the time for reading the weights is reduced to N_{pre}/m cycles. The m -input addition is computed in a pipelined adder tree to avoid large

Fig. 6: The timing analysis for processing one input image.

propagation delay and ensure high throughput. In general, it takes $\lceil \log_2(m) \rceil + 1$ cycles to complete the computation. Since the number of inputs is generally very large, the total time for computing weight accumulation is dominated by the weight access time. As a result, the weights accumulation process can be accelerated significantly.

2) *Parallel neuron processing cores*: The neuron processing cores at the second layer are used for updating membrane potential and detecting output spikes. To update the membrane potential of one neuron, we have to go through the weights accumulation process and solve the neural equations. In the case of one neuron core, all the operations have to be repeated for each neuron sequentially, which will consume a significant amount of time. Alternatively, we can implement n neuron processing cores and compute the membrane potentials of the n neurons in parallel. By using the time-multiplexing design, we can update the membrane potentials of all the neurons in the second layer in N_{post}/n stages, where N_{post} is the total number of neurons in the second layer. The parallel processing requires that n processes of the weights accumulation need to be carried out simultaneously. Thus, as depicted in the figure, we can allocate all the synaptic weights into n different blocks of BRAMs. Each block consists of m BRAMs and is partitioned into N_{post}/n parts, and each part contains all the synaptic weights associated with the neuron i , N_i .

E. WTA network

The WTA network is implemented with the help of inhibitory neurons. Whenever the first excitatory neuron fires, the inhibitory neuron that it is connected to produces inhibitory signals that stop the other neurons from firing for the rest of the training window. The digital implementation is straightforward as shown in Fig. 2. The network implements an inhibitory neuron layer that applies the same design of the proposed neural processing cores. As shown in Fig. 1b, the excitatory neuron layer is connected to the inhibitory neuron layer in a one-to-one fashion with a large positive constant synaptic weight, $w_{exc} = 10$. The inhibitory neurons accumulate this constant weight to their excitatory conductance channels so that the produced synaptic current is positive enough to generate a spike. The inhibitory feedback signal is sent to all the other excitatory neurons except for the firing excitatory neuron. A large negative constant weight, $w_{inh} = -17$, is used

(a) Euler method

(b) RK3 method

Fig. 7: The classification accuracy vs training time per image for different time constants. The inset figures show the required training time per image to reach the maximum classification accuracy for each total time constant.

to ensure instant inhibition. The excitatory neurons receive the negative weight in their inhibitory conductance channels so that the produced synaptic current is negative enough to prevent the firing.

F. Timing analysis

The timeline of processing one image is shown in Fig. 6. With the presentation of each input pattern, the network requires t time steps to learn or classify the features. To complete the computation in one time step, the membrane potentials of all the neurons at the second layer need to be updated and STDP units will be activated to update the synaptic weights if there are any output spikes. One time step takes N_{post}/n iterations to complete, which depends on the number of the parallel neuron cores as discussed above. Within each iteration window, the runtime mainly consists of two parts: the time for reading and accumulating synaptic weights (T_W), and the time for updating the membrane potential (T_{mem}). The time for running STDP unit can be excluded from the runtime analysis. The reason lies in the fact that the STDP unit stays inactive most of the time due to very sparse output spikes. As a result, the total runtime for learning or classifying one pattern can be approximately calculated as

$$T_{tot} \approx (T_W + T_{mem}) \cdot \frac{N_{post}}{n} \cdot t$$

where t is the number of time steps required to reach the maximum accuracy.

IV. RESULTS

A. Parameter optimization

As previously discussed in Section II-D, several parameters are optimized, including the time constants of conductance channels, the time constants of STDP traces, the learning rates of STDP weight update, and the threshold adaption constant. The weight coefficient of the total time constant in the objective function is set as 6 different values: 0, 0.1, 0.3, 0.5, 0.7, 0.9 respectively. The genetic algorithm runs for 20 generations to ensure that the best fitness score is achieved. The simulation program was written in Python language and run on the Intel Xeon Gold 6130 CPU. And the simulation time step size is taken as 0.5 ms. During the simulation, six sets of the optimized parameters and corresponding classification accuracy of MNIST dataset are obtained from the genetic algorithm. After the optimization, the maximum classification achieved is 86.54% which is much higher than 82.9% reported in [16].

The convergence of the network for different total time constants is compared with regard to training time and inference time per image, as shown in the Fig. 7. From both plots, it

can be observed that a large time constant tends to decrease the convergence rate but with higher classification accuracy (up to a point, after which accuracy reduces), while small time constants deteriorate the performance of the network. Clearly, large time constants slow down the variable update and hence the learning process, while in the case of small time constants, the variables change so fast that the input information could not be transmitted and processed effectively. For example, in the triplet STDP model (7), the weight update depends on the values of presynaptic and postsynaptic trace variables. If the trace variables change slowly over time, different relative timing between presynaptic and postsynaptic spikes results in similar weight update. The time correlation is not effectively reflected in the STDP model, and the network can not efficiently use the input information to learn the synaptic weights. In the case of small time constants, the variables change very fast so that some input information could be lost before the neuron responds by firing. The input information can not be transmitted and processed efficiently in a short time window. This observation is further clarified by the insets in Fig 7, which show the required training time per image to reach the maximum classification accuracy for each total time constant. As a result, one can identify the optimum total time constants for the fastest convergence in both cases, which are 16 ms and 19 ms for the SNNs with Euler method and RK3 method respectively. In addition, the inference convergence for different total time constants is also compared in the Fig. 8. The difference caused by different time constants becomes less prominent, as the impact of the learning time constants is excluded.

This behavior can also be explained by the raster plots of firing patterns shown in the Fig. 9. The firing patterns are obtained by inputting 1000 images with the training time of 70 ms in the case of RK3 method. When the total time constant is 19 ms, multiple neurons are activated to learn the input patterns and most of the spikes are fired by these neurons. However, in other cases, only few neurons tend to learn the input patterns and the spikes are distributed more evenly across all neurons.

It is noteworthy to show that with the optimum total time constants, the convergence rate in the case of RK3 method is faster than that in the case of Euler method. For Euler method, the convergence times are 150 ms for the training and 100 ms for the inference. While for RK3 method, the convergence time is 70 ms for both training and inference. The difference in the convergence rate is caused by the different numerical accuracy of the solutions to the network dynamics. The dynamics of updating the synaptic conductance and membrane potential determine how the input information is conveyed and processed. The learning capability of the network is also largely dependent on the dynamics of updating the synaptic traces in the STDP model. RK3 method computes the network dynamics more accurately and hence, provides more precise transmission and processing of the input information. In contrast, Euler method suffers from more information loss during the transmission and processing. Therefore, the SNN with RK3 method is able to learn the input patterns more efficiently within a short time window.

Biological parameters are essential in defining neural dynamics. The GA used in this work has provided a fast and

Fig. 8: The classification accuracy vs inference time per image for different time constants. The inset figures show the required inference time per image to reach the maximum classification accuracy for each total time constant.

Fig. 9: The firing patterns for three different time constants obtained from RK3 method.

(a) Resource utilization and power consumption.

(b) Throughput and energy consumption.

Fig. 10: The implementation results of the SNN with Euler method for different number of neuron cores n . n varies from 4, 10, 25, 50, to 100. Accordingly, (a) resource utilization and power consumption are plotted. (b) Throughput and energy consumption per image are calculated for both training phase and inference phase.

efficient optimization process for configuring these parameters. The algorithm runs for 20 generations with a population of 20 individuals to reach the optimal solution. With the defined objective function, the genetic algorithm produces a set of optimized parameters which improves the training and inference speed. It helps accelerate the training and inference

process with no loss of accuracy. As shown in Fig. 7 and Fig. 8, the improvement of training time and inference time can be up to 5 times.

Additionally, to verify the generality, the SNNs with the same sets of parameters are also tested on Fashion-MNIST dataset. The simulation results suggest the same observations of the convergence change with the time constants for both methods.

B. FPGA implementation of parallel neuron cores

The spiking neural network was programmed using Verilog hardware description language (HDL). The Verilog code was synthesized and implemented in Xilinx Vivado Design Suite. The generated bitstream was downloaded to the Xilinx Virtex-7 VC709 evaluation board for verification. Power consumption estimates were obtained after routing by using the power analysis tool in Vivado Design Suite that provides detailed analysis and accurate estimation [50]. We would like to highlight that the power estimation only reports dynamic power which includes the power for logic operations, BRAM memory access, I/O operation, signal transmission, and clock switching. The static power and DDR memory power are not included as they remain the same for all the designs. The implemented SNNs run at 100 MHz clock frequency. In addition, for the SNNs to work properly in fixed-point arithmetic, 16 integer bits and 9 fraction bits are chosen to avoid overflow and precision loss respectively.

We implemented the SNN based on Euler method. Firstly, the number of parallel synaptic weight readouts m is set as 4 for fast processing. Then, we considered the parallelism in the implementation of neuron cores. Implementation results are shown in Fig. 10 with $n = 4, 10, 25, 50, 100$, including slice LUT, slice FF and BRAM utilization, power consumption, throughput, and energy consumption per image. Fig. 10a shows that both slice look-up table (LUT) and flip flop (FF) utilizations are increased with the number of neuron cores. On the FPGA, BRAMs are implemented by combining the embedded primitives that include 18Kb RAMs and 36 Kb RAMs. It can be seen that the number of BRAM primitives is not changed much with the number of neuron cores. Especially, when the number of neuron cores is between 10 and 50, the usage of BRAM primitives is very efficient.

Moreover, the power consumption grows with the number of neuron cores, as more neuron cores are operating at the

same time. We measured the throughput of the network which is defined as the number of images processed in one second. Fig. 10b demonstrates that the throughput is improved linearly by increasing the number of neuron cores. It can be explained by the timing analysis in the Fig. 6. The number of iterations spent to complete the updates in one time step is reduced proportionally when more neuron cores are used. Therefore, more images can be processed in one second. Besides, during the inference phase, fewer time steps are required to achieve the maximum accuracy. So the throughput during the inference phase is higher. Additionally, energy consumption for processing one image is calculated in terms of the runtime and power consumption. From the Fig. 10b, we can see that the energy consumption is reduced when the number of neuron cores goes up. It is because the reduction of the runtime is directly proportional to the increase of n , but the power consumption scales up at a slower rate. In the network, the power consumption also results from many other modules except the neuron cores, like control units, Poisson spike generator, and memory blocks. So when the number of neuron cores is small, the energy consumed by these modules is comparable with that by the neuron cores. The plot shows that 25 is the turning point from where the energy consumption starts to change very slowly, as the neuron cores become dominant in the network. Therefore, we choose the number of neuron cores as 25 by considering the trade-off between resource and energy consumption.

To study the effect of the number of parallel weight readouts m , we present the implementation results on resource utilization, throughput, and energy consumption for different m in Fig. 11. The number of neuron cores n is fixed at 25. As presented in Fig. 5a, multiple BRAMs are used to store the weights associated with one postsynaptic neuron in order to access multiple weights simultaneously. The usage of a BRAM becomes inefficient because some parts of the BRAM could be left empty. So the utilization of BRAMs increases with m . To sum up all the weights, a large m -input adder tree needs to be implemented with a lot of logic resources. So the LUT utilization and power increase rapidly with m . With the pipelining design, a large adder tree consists of multiple stages, and the weight accumulation process is finished in many clock cycles. Thus, the runtime reduction becomes less significant when m gets large. The increase in throughput also slows down. Moreover, the results show the lowest energy point at $m = 16$. However, a large-scale neuromorphic system generally requires many on-chip memories inside (or near) neural cores to store synaptic weights for fast memory access [26], [32], [33]. The size of on-chip memories is limited, so the usage of memories has to be very efficient, which makes a large m not practical in the design of a large-scale neuromorphic system. Therefore, we choose m as 4 under the consideration of better BRAM usage and fast processing.

TABLE III: The implementation results for LIF neuron equation solved by Euler method and RK3 method.

Numerical method	FF	LUT	Delay (ns)	Power (mW)
Euler	85	95	80	0.25
RK3	297	196	300	0.48

(a) Resource utilization and power consumption.

(b) Throughput and energy consumption.

Fig. 11: The implementation results of the SNN with Euler method for different number of parallel weight readouts m . m varies from 4, 8, 16, 32, to 64. Accordingly, (a) resource utilization and power consumption are plotted. (b) Throughput and energy consumption per image are calculated for both training phase and inference phase.

C. Comparison between Euler method and RK3 method

Table III listed the implementation results of LIF neuron model based on Euler method and RK3 method. Clearly, Euler method takes up fewer resources and consumes less power. Because of one-iteration update, the delay for the Euler method is also much smaller.

Furthermore, the SNN based on RK3 method is implemented on FPGA with 25 neuron cores. In Table IV, resource utilization, power, runtime and energy consumption are compared between the implementations with Euler method and RK3 method. Because of the complexity, RK3 method uses more FFs to store the intermediate variables and more LUTs to implement the logic. Accordingly, more power is dissipated to run the network.

In spite of the disadvantages of more resource utilization and power consumption, RK3 method requires less training time steps (140 time steps) for processing each image to achieve the maximum accuracy, as opposed to the time steps required by Euler method (300). The actual training runtimes per image on FPGA for both methods are 1.36 ms and 2.59 ms respectively. While RK3 method consumes more power, the network can process images much faster. According to the results of energy consumption per image, RK3 method con-

sumes less energy since it requires less number of time steps per image. Therefore, the speedup improvement compensates the extra power consumption due to higher resource utilization.

In our online learning system, the STDP unit is essential for the unsupervised online learning capability. When the network runs in the inference phase, the STDP unit is deactivated by a control signal to ensure that all the registers stay idle and no activity is allowed. Thus, power consumption is reduced for both methods. In addition, RK3 method requires less runtime to complete the inference. However, there is less difference between the runtimes for the two methods. That is because in the inference phase, Euler method needs fewer time steps to classify one image than in the training phase. As a result, the energy consumed in the network with RK3 method becomes more than that in the network with Euler method.

V. COMPARISON AND DISCUSSION

A. Comparison with previous works

The comparison among different implementation methods of the same SNN structure is presented in Table V. In our work, the software simulation follows a straightforward implementation scheme in Python and runs in a single process on an Intel Xeon Gold 6130 CPU at 2.1 GHz. The long simulation time is because solving the differential equations associated with neural models requires sequential operations, and hence the membrane potential of each neuron is updated sequentially. Besides, because of the temporal behavior of the SNNs, a long time loop is required for training and inference per image. For all the reference works, Euler method was used. In [16], a Brian simulator was used to run the SNN, and the simulation time for both training and inference is longer than that in our work with Euler method. In [51], a Pytorch-based implementation was proposed, which allows the simulation to run in a batch mode. The batch size is chosen as 32 to ensure satisfactory accuracy. The Pytorch-based simulation is able to process multiple input images in parallel by using multiple CPU cores and more memory. The simulation results show that the inference phase is much faster but the training phase is slightly improved. The slight improvement on the training time is because the communications between multiple processes across CPU cores for updating the synaptic weights slow down the parallel processing speed [51]. The Brian-based simulation and our works are both run on a single core and consume less memory. Compared with our software simulation with Euler method, the FPGA-based implementation with Euler method can achieve around $250\times$ speedup during the training phase and around $180\times$ speedup during the inference phase. For RK3 method, the speedups are more significant, around $750\times$ for the training and $330\times$ for the inference. Compared with the Pytorch-based simulation, our FPGA-based implementation with Euler method can achieve around $180\times$ speedup during the training phase and around $20\times$ speedup during the inference phase.

The performance comparison for FPGA implementation between our work and previous work is summarized in Table VI. In [41], Euler method was used to implement the SNN. To make a fair comparison, we implemented our SNNs on the same FPGA with the same clock frequency. The number of BRAM primitives for the reference work is estimated according to the reported memory size in the paper. To compare the area occupation of different designs, we estimate the equivalent NAND gate counts for both slices and BRAMs following the estimation method presented in [52]. As shown in the table, our SNNs produce comparable classification accuracy and the SNN with Euler method requires less hardware resource. The comparison also shows more than $300\times$ ($240\times$) improvement in the runtime and $180\times$ ($250\times$) improvement in energy consumption during training (inference) phase with the same number of output neurons. This significant difference is a result of different architecture design and input spikes transmission. In the reference design, the input spikes are sent from the PC to FPGA board through universal asynchronous receiver/transmitter (UART). Due to the slow transmission rate, it takes long time to send all the input spikes from one image to FPGA board. As a consequence, the runtime and energy consumption becomes huge. In contrast, in our design, the input spikes are generated on FPGA and immediately sent to the next layer. And the generation of input spikes is pipelined with the weight accumulation process. Little time is actually spent on the generation and transmission of input spikes in our SNNs. Except for the large communication overhead, the design in [41] is not fully parallelized. The runtime is not proportionally improved as the number of neuron processing units increases. In contrast, our design proposes a fully parallelized architecture, which reduces the runtime proportionally with the increasing number of neuron processing units. Therefore, the fully parallel design also contributes to the speedup over the previous work.

Furthermore, with 400 output neurons, our SNNs can manage to produce around 1% less classification accuracy and largely reduce the usage of hardware resource. Specifically, for Euler method, there are around 65% reduction on the number of BRAMs and more than 40% reduction on the equivalent gate counts. Moreover, the runtime and energy consumption are also reduced by around 35% (20%) and 70% (61%) respectively during training (inference) as the network size becomes much smaller. Therefore, with the proposed optimization scheme and parallel architecture design, we can design low-cost, high-speed, and energy-efficient SNNs with high classification accuracy.

B. Impact of RK3 method on Scalability

We have demonstrated that SNNs with RK3 method require less training time and achieve higher classification accuracy. The improvement is attributed to the fact that neural models with RK3 method can more accurately process temporal

TABLE IV: Implementation results for the SNN with Euler method and RK3 method. The number of neuron cores is set as 25 for both methods. The runtime is the time used for processing one image.

Phase	Numerical method	Slice FF	Slice LUT	Power (mW)	Runtime (ms)	Energy (mJ)
Training	Euler	17,407	32,548	52.09	2.59	0.13
	RK3	59,070	51,305	88.19	1.36	0.12
Inference	Euler	17,407	32,548	44.09	1.69	0.075
	RK3	59,070	51,305	76.34	1.30	0.099

TABLE V: Comparison among different implementation methods of the same SNN structure. CPU simulations were run on a multi-core Intel Xeon Gold 6130 CPU. The runtime is measured for processing one image. Simulations in all the references used Euler method for solving differential equations. All the results are obtained in the SNN with 100 output neurons.

Ref	Programming language	Processing device	Runtime (ms) Training/inference	CPU usage Training/inference	Memory (MB) Training/inference	Execution mode
[16]	Python with Brian library	CPU	2850.32/347.44	3.12%/3.12%	163.63/153.58	Serial
[51]	Python with Pytorch library	CPU	473.78/34.48	96.92%/75.80%	286.03/270.16	Parallel
Our work (Euler)	Python	CPU	643.23/312.43	3.12%/3.12%	116.66/116.23	Serial
Our work (RK3)	Python	CPU	1023.57/427.21	3.12%/3.12%	117.36/117.22	Serial
Our work (Euler)	Verilog	FPGA	2.59/1.69	—	—	Parallel
Our work (RK3)	Verilog	FPGA	1.36/1.30	—	—	Parallel

TABLE VI: Comparison for FPGA implementation between our methods and previous published work during training and inference phase. Euler method was used in [41]. The results of both our work and the previous work are taken from the design with $n = 32$. Gate counts are estimated by considering the equivalent number of NAND gates for both slices and BRAMs [52]. For runtime and energy, at each entry, the first number is from training phase and the second is from inference phase.

Ref	# output neurons	Accuracy	Slice FF	Slice LUT	BRAM	Gate counts (MGates)	Runtime (s) Training/inference	Energy (mJ) Training/inference
[41] (standard multiplier)	800	89.1%	58,826	97,287	340	15.48	16.8/8.4	1339.83/1127.25
[41] (approximate multiplier)	800	87.7%	58,345	93,232	340	15.24	16.8/8.4	1115.02/925.65
Our work (Euler method)	800	89.5%	50,614	45,531	289	10.97	0.053/0.035	7.37/4.45
Our work (Euler method)	400	88.3%	31,099	32,983	101	5.66	0.034/0.028	2.43/1.73
Our work (RK3 method)	800	89.7%	91,281	88,053	289	16.12	0.029/0.024	6.78/5.19
Our work (RK3 method)	400	88.6%	67,984	54,313	101	9.16	0.019/0.018	1.74/1.39

information. For example, the STDP model usually performs weight updates according to the precise timing correlations between a presynaptic spike and a postsynaptic spike. Different timing correlation can result in totally different weight updates. Low-precision numerical solutions can cause large uncertainty in the spike timing from a neuron because of large numerical error, especially in a biological neural network involving different neural dynamics. So a positive update could be mistaken as a negative update if a presynaptic spike appears too late or a postsynaptic spike gets delayed due to numerical error. Compared with the Euler method, RK3 method can lower the possibility of the occurrence of such behavior by providing a more stable solution. Therefore, RK3 method contributes to the improvement of network performance by enabling more accurate computation of neural dynamics. The RK method has been widely used with STDP model to solve neural dynamics because of its high precision in many applications, such as pattern recognition and motion detection [53], [54]. Although RK3 method requires more resources than Euler method, it processes the data much faster during training. Thus, when the network scales, RK3 method would still outperform Euler method for the same number of neurons in terms of training speed, as RK3 method requires fewer time steps due to precise information transmission and processing. Since the RK3-based design takes up more resources, the RK3-based design is less scalable than Euler-based design under the same constrain of hardware resources. RK3 method is promising for developing fast online learning systems, while Euler method is suitable for building low-cost systems.

For an unsupervised deep spiking CNN trained with STDP as presented in [22], the network is trained based on the temporal information of spikes. Due to sparse spikes in the network, information is mainly encoded in the precise timing of spikes. Thus, the learning capability relies on the temporal dynamics of neural models. RK3 method could offer accuracy and speed advantages for such networks. Furthermore, RK3

method can also be useful in different networks. In complex biological SNNs that aim to emulate the nervous system, information processing largely depends on spike timings, and the precision of neural dynamics becomes very important to network performance. Because of complex connections in the network, a tiny error in computing the spike timings could lead to totally different responses. For example, to understand and learn spatio-temporal brain data, the network needs to be biologically plausible and provide a similar information processing flow to what happens in the brain [25], [55]. A high-precision numerical solution is important to ensure that the network dynamics can function correctly and the network can capture precise temporal information to learn complex spatiotemporal sequences or features. Moreover, a liquid state machine (LSM) network is a type of biological SNN with a high computational capability to solve different spatio-temporal tasks [56]. It also uses precise spike timing to process information. Therefore, RK3 solution could also be useful to improve the performance of the biological SNN by providing a more accurate solution to neural dynamics.

C. Impact of RK3 method on supervised SNNs

However, in the SNNs trained with supervised BP algorithms, neural models are generally simplified and modified to enable the implementation of BP algorithms [17], [19]. The impact of numerical solutions on the network is reduced. Moreover, although the Euler method offers low precision to neural dynamics, the supervised BP algorithms can reduce the impact of this low precision by incorporating the numerical error into the total network output error and adjust the synaptic weights accordingly. In other words, the numerical error from the neural dynamics becomes a part of the network output error and could be effectively minimized by BP algorithms. On the other hand, unsupervised learning with STDP highly relies on the spike timing and precision, which require accurate calculation to achieve the best performance. Thus, RK3 method would not play a significant role in improving the

network performance. [19] studied the impact of the precision of neuronal dynamics on the performance of SNNs trained with a supervised BP algorithm by changing the time step size. They used two different step sizes, $t = 0.1ms$ and $1ms$. Their results showed that within the same time window, when $t = 1ms$, the best accuracy is 97.80% compared with 98.17% achieved when $t = 0.1ms$. For a 10x large difference of precision, the supervised network can reduce the resulting accuracy difference to less than 0.4%. While this study is not on the impact of different numerical solutions on the network performance, it provides an insight into how the precision of neuronal dynamics affects the network performance.

D. Impact of hyper-parameter optimization on SNNs

SNNs are sensitive to the hyper-parameters in neural models. For example, the time constants control the change of neural variables with time and hence have a direct impact on the efficiency and effectiveness of information transmission in the network. Synaptic conductance determines how much information is integrated from the previous layer. These biological hyper-parameters have profound impacts on learning efficiency in the network. In other words, they can affect the network accuracy and learning or inference convergence rate. When input information is inaccurately captured and transmitted through neural units, even with a supervised BP algorithm, the network can not learn the input features efficiently. In supervised SNNs, in order to correctly implement the BP algorithm, more hyper-parameters are introduced. Thus, it becomes more difficult to tune the hyper-parameters. Some works have reported using extensive grid search or manual search for hyper-parameter tuning [57], [58]. As for deep SNNs trained with an unsupervised STDP, hyper-parameters also significantly impact network performance. These parameters affect not only the neural dynamics but also the STDP learning efficiency. [59] studied the effect of different hyper-parameters on classification accuracy and network sparsity. The results revealed that network performance has a large dependence on these parameters. In [22], it requires to tune its hyper-parameters carefully, especially the neuron threshold, to reach high accuracy.

The novelty of our proposed optimization scheme lies in defining the optimization problem with a genetic algorithm which enables us to search for a set of optimal parameters and investigate the impact of the time constants. Due to the large dimension of the parameter space, a grid search or manual tuning is extremely time-consuming and not practical. The first value the genetic algorithm adds to our works is to provide a fast and efficient optimization process with which we can easily investigate the impact of the time constants. 20 generations are enough to find a set of optimized parameters with sufficient performance. Secondly, the genetic algorithm allows us to define an optimization problem to search for a set of optimal parameters that ensure high classification accuracy and small time constants. As a result, a set of optimized time constants with other model parameters can accelerate both training and inference phase by up to 5 times with no loss of accuracy. The proposed optimization method has shown good potential to improve the performance of different networks. However, due to the slow simulation time on CPU, the runtime for a single SNN simulation with Euler

TABLE VII: Scaling of the proposed digital system with the number of neuron cores for both numerical methods. For LUT and FF, the linear functions of n were derived by data fitting.

Euler-based design						
Neuron cores	100	200	400	600	800	n
LUT	85961	179196	352026	527140	702254	$877n+828$
FF	66163	131467	257033	385142	517080	$642n+1731$
DSP	400	800	1600	2400	3200	$4n$
36Kb BRAM	50	100	200	300	400	$0.5n$
RK3-based design						
Neuron cores	100	200	400	600	800	n
LUT	139562	284008	559669	828853	1118037	$1389n+2491$
FF	175604	349172	693605	1050884	1386958	$1735n+2237$
DSP	400	800	1600	2400	3200	$4n$
36Kb BRAM	50	100	200	300	400	$0.5n$

method for both training and inference is around 11h. With a many-core computing workstation, it takes around 220h to finish the 20-generation optimization process and find a set of optimized parameters. The long duration is because of the long simulation time of the SNN itself. Thus, to reduce the optimization runtime cost, we can incorporate our hardware accelerator in the optimization loop since the SNN simulation can be significantly accelerated. As a result, the optimization time would be reduced to around 19h.

E. Impact of the proposed digital system

We have proposed a parallel SNN digital system with online STDP training. The parallel design can significantly accelerate synaptic operations and neuron membrane updates. The proposed digital design can be scaled up proportionally with the number of neuron cores. VII shows how the system based on both numerical methods is scaled up with the number of neuron cores in terms of different components of the system, namely, LUT, FF, DSP, and BRAM. Clearly, for all the components, the number grows proportionally as the number of neuron cores increases. This is due to our parallel design, where neuron cores can be simply added to the system without modifying the system architecture when the size increases. We have derived LUT and FF usage as linear functions of the number of neurons n by data fitting. When the number of neuron cores is large, the neuron cores become the dominant modules in the system and consume most of the resources. So, the scaling behavior is linear. However, some modules like control units in the system are not scaled linearly with the neuron cores, resulting in a bias term in the functions. Moreover, each neuron core requires one 18Kb BRAM to store the associated weights and 3 DSPs to compute the synaptic current for excitatory and inhibitory neurons. Each STDP unit requires one DSP to compute weight updates. The RK3-based design takes up more LUTs and FFs due to the complexity of RK3 method. The proposed digital system is also scalable for more complex network architectures by cascading additional neural layers with the same neural cores design or using the time-multiplexing method to share the same neural cores with the addition of extra state memory and synaptic weight memory. The parallel design for synaptic memory allocation and neuron processing cores is not network-specific and can be applied in multiple layers. The STDP digital module can

be either shared by multiple layers or easily implemented for each layer since it is based on local information. Supervised BP algorithms can also be integrated with our digital system. For example, BP algorithms such as direct feedback alignment (DFA) [60] or deep continuous local learning (DECOLLE) [20] can directly replace the STDP unit with an addition of an output error generator in the system since the DFA/DECOLLE unit updates the local synaptic weights by only receiving the error information from the net/local output. Furthermore, other neuromorphic platforms, such as Loihi [26] and NSAT [61], are designed to accommodate different kinds of network structures for general purposes. These neuromorphic systems use Euler method for solving neural dynamics with low precision. According to our results, the RK3 method makes the network capable of capturing more precise temporal information, which leads to less training and inference time and higher accuracy. The proposed FPGA architecture together with the RK3 method can potentially provide more speedup benefits and higher accuracy.

To sum up, we summarize two broader takeaways for neuromorphic research community as follows.

- 1) Hyper-parameter optimization is essential for configuring SNNs because these parameters have a profound impact on neural dynamics. Considering large parameter space, our proposed optimization method with a genetic algorithm offers a fast and efficient method for optimizing hyper-parameters and improving network performance.
- 2) Runge-Kutta method is very useful in biological SNNs. It enables the network to capture temporal information more accurately compared with Euler method and allows for more accurate information transmission and processing in the network. It can improve network performance and accelerate the training process, which makes it promising for online training in real-time application.

VI. CONCLUSION

In this paper, an optimization scheme for biological parameters was proposed to improve the performance of the spiking neural network and two numerical methods were applied to implement the spiking neural network on FPGA. We investigated the impact of biological time constants on the information transmission in the network. It was concluded that proper choice of the time constants improves the convergence rate of the network significantly for digit recognition. In particular, RK3 method enables much faster learning and classification thanks to the high-order accuracy it produces. In the digital implementation, the design with a parallel degree of 25 leads to very low energy consumption and efficient usage of resources.

Compared with a Pytorch-based simulation on CPU, the FPGA-based implementation with Euler method achieves around $180\times$ training speedup and $20\times$ inference speedup. Furthermore, the detailed comparison of implementations between Euler method and RK3 method has revealed that RK3 method takes up more resources but provides around $2\times$ training speedup. During the inference phase, Euler method leads to similar runtime and lower energy consumption. In summary, Euler method is suitable for low-cost implementation and provides an energy-efficient solution for inference.

RK3 method could be a promising solution for online training as it allows for accurate, fast, and low-energy information processing in the real-time application.

REFERENCES

- [1] D. Monroe, "Neuromorphic computing gets ready for the (really) big time," *Commun. ACM*, vol. 57, no. 6, p. 13–15, 2014.
- [2] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, Oct 1990.
- [3] M. Mahmud, M. S. Kaiser, A. Hussain, and S. Vassanelli, "Applications of deep learning and reinforcement learning to biological data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2063–2079, 2018.
- [4] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53 040–53 065, 2019.
- [5] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, Nov 2019.
- [6] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. USA: Cambridge University Press, 2014.
- [7] J. M. Cruz-Albrecht, M. W. Yung, and N. Srinivasa, "Energy-efficient neuron, synapse and stdp integrated circuits," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, no. 3, pp. 246–256, 2012.
- [8] A. P. Johnson, J. Liu, A. G. Millard, S. Karim, A. M. Tyrrell, J. Harkin, J. Timmis, L. J. McDaid, and D. M. Halliday, "Homeostatic fault tolerance in spiking neural networks: A dynamic hardware perspective," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 2, pp. 687–699, Feb 2018.
- [9] E. Nichols, L. J. McDaid, and N. Siddique, "Biologically inspired snn for robot control," *IEEE Transactions on Cybernetics*, vol. 43, no. 1, pp. 115–128, Feb 2013.
- [10] M. Osswald, S.-H. Ieng, R. Benosman, and G. Indiveri, "A spiking neural network model of 3d perception for event-based neuromorphic stereo vision systems," *Scientific Reports*, vol. 7, no. 1, p. 40703, 2017.
- [11] D. Liu and S. Yue, "Event-driven continuous stdp learning with deep structure for visual pattern recognition," *IEEE Transactions on Cybernetics*, vol. 49, no. 4, pp. 1377–1390, April 2019.
- [12] S. J. Kim, S. Park, B. Na, and S. Yoon, "Spiking-yolo: Spiking neural network for real-time object detection," *CoRR*, vol. abs/1903.06530, 2019.
- [13] S. Koul and T. K. Horiuchi, "Waypoint path planning with synaptic-dependent spike latency," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 4, pp. 1544–1557, April 2019.
- [14] F. Naveros, N. R. Luque, E. Ros, and A. Arleo, "Vor adaptation on a humanoid icub robot using a spiking cerebellar model," *IEEE Transactions on Cybernetics*, pp. 1–14, 2019.
- [15] N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biol. Cybern.*, vol. 95, no. 1, p. 1–19, 2006.
- [16] P. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, p. 99, 2015.
- [17] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, vol. 10, p. 508, 2016.
- [18] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3227–3235, 2018.
- [19] S. R. Kulkarni and B. Rajendran, "Spiking neural networks for handwritten digit recognition—supervised learning and network optimization," *Neural Networks*, vol. 103, pp. 118–127, 2018.
- [20] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020.
- [21] A. Tavanaei and A. S. Maida, "Bio-inspired spiking convolutional neural network using layer-wise sparse coding and stdp learning," *ArXiv*, vol. abs/1611.03000, 2016.
- [22] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "Stdp-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018.
- [23] T. Masquelier, R. Guyonneau, and S. J. Thorpe, "Competitive stdp-based spike pattern learning," *Neural Computation*, vol. 21, no. 5, pp. 1259–1276, 2009.
- [24] N. Kasabov, K. Dhoble, N. Nuntalid, and G. Indiveri, "Dynamic evolving spiking neural networks for on-line spatio- and spectro-temporal pattern recognition," *Neural Networks*, vol. 41, pp. 188 – 201, 2013, special Issue on Autonomous Learning.

- [25] N. K. Kasabov, "Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data," *Neural Networks*, vol. 52, pp. 62–76, 2014.
- [26] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathakutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
- [27] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 1947–1950.
- [28] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 106–122, 2018.
- [29] J.-P. Pfister and W. Gerstner, "Triplets of spikes in a model of spike timing-dependent plasticity," *Journal of Neuroscience*, vol. 26, no. 38, pp. 9673–9682, 2006.
- [30] S. A. Aamir, Y. Stradmann, P. Müller, C. Pehle, A. Hartel, A. Grübl, J. Schemmel, and K. Meier, "An accelerated lif neuronal network array for a large-scale mixed-signal neuromorphic architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4299–4312, Dec 2018.
- [31] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. Dell'Anna, G. Indiveri, J. L. Krichmar, N. D. Dutt, S. Schaafsma, and F. Catthoor, "Mapping spiking neural networks to neuromorphic hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 76–86, 2020.
- [32] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.
- [33] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.
- [34] S. Yang, B. Deng, J. Wang, H. Li, M. Lu, Y. Che, X. Wei, and K. A. Loparo, "Scalable digital neuromorphic architecture for large-scale biophysically meaningful neural network with multi-compartment neurons," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 1, pp. 148–162, 2020.
- [35] L. L. Lewyn, T. Ytterdal, C. Wulff, and K. Martin, "Analog circuit design in nanoscale cmos technologies," *Proceedings of the IEEE*, vol. 97, no. 10, pp. 1687–1714, Oct 2009.
- [36] M. Fouda, F. Kurdahi, A. Eltawil, and E. Neftci, "Spiking neural networks for inference and learning: A memristor-based design perspective," *arXiv preprint arXiv:1909.01771*, 2019.
- [37] J. Park and S. Jung, "Presynaptic spike-driven spike timing-dependent plasticity with address event representation for large-scale neuromorphic systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–12, 2020.
- [38] S. Yang, J. Wang, B. Deng, C. Liu, H. Li, C. Fietkiewicz, and K. A. Loparo, "Real-time neuromorphic system for large-scale conductance-based spiking neural networks," *IEEE Transactions on Cybernetics*, vol. 49, no. 7, pp. 2490–2503, July 2019.
- [39] D. Neil and S. Liu, "Minitaur, an event-driven fpga-based spiking network accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2621–2628, Dec 2014.
- [40] M. Heidarpur, A. Ahmadi, M. Ahmadi, and M. Rahimi Azghadi, "Cordic-snn: On-fpga stdp learning with izhikevich neurons," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 7, pp. 2651–2661, July 2019.
- [41] 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, 2017.
- [42] M. J. Skocik and L. N. Long, "On the capabilities and computational costs of neuron models," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 8, pp. 1474–1483, Aug 2014.
- [43] R. R. Osorio, "Pipelined fpga implementation of numerical integration of the hodgkin-huxley model," in *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2016, pp. 202–206.
- [44] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.
- [45] *Euler's method*. John Wiley & Sons, Ltd, 2011, ch. 2, pp. 15–36.
- [46] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations*, 2nd ed., ser. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2008.
- [47] F. Jug, "On competition and learning in cortical structures," Ph.D. dissertation, ETH Zurich, Zurich, 2012.
- [48] P. Chu and J. Beasley, "A genetic algorithm for the generalised assignment problem," *Computers and Operations Research*, vol. 24, no. 1, pp. 17–23, 1997.
- [49] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers in Neuroscience*, vol. 7, p. 178, 2013.
- [50] F. B. Muslim, L. Ma, M. Roozmeh, and L. Lavagno, "Efficient fpga implementation of opencl high-performance computing applications via high-level synthesis," *IEEE Access*, vol. 5, pp. 2747–2762, 2017.
- [51] D. J. Saunders, C. Sigrist, K. Chaney, R. Kozma, and H. T. Siegelmann, "Minibatch processing in spiking neural networks," *ArXiv*, vol. abs/1909.02549, 2019.
- [52] V. Berg, J. Doré, and S. Mayrargue, "Silicon area of fbmc receivers for cmos 65nm and comparison to ofdm receivers," in *2017 European Conference on Networks and Communications (EuCNC)*, 2017, pp. 1–5.
- [53] S. Knieling, K. S. Sridharan, P. Belardinelli, G. Naros, D. Weiss, F. Mormann, and A. Gharabaghi, "An unsupervised online spike-sorting framework," *International Journal of Neural Systems*, vol. 26, no. 05, p. 1550042, 2016, PMID: 26711713.
- [54] A. P. Shon, R. P. N. Rao, and T. J. Sejnowski, "Motion detection and prediction through spike-timing dependent plasticity," *Network: Computation in Neural Systems*, vol. 15, no. 3, pp. 179–198, 2004, PMID: 15468734.
- [55] A. Maes, M. Barahona, and C. Clopath, "Learning spatiotemporal signals using a recurrent spiking network that discretizes time," *PLOS Computational Biology*, vol. 16, no. 1, pp. 1–26, 01 2020.
- [56] D. Florescu and D. Coca, "Learning with precise spike times: A new decoding algorithm for liquid state machines," *Neural Computation*, vol. 31, no. 9, pp. 1825–1852, 2019.
- [57] A. Shrestha, H. Fang, Q. Wu, and Q. Qiu, "Approximating back-propagation for a biologically plausible local learning rule in spiking neural networks," in *Proceedings of the International Conference on Neuromorphic Systems*, ser. ICONS '19. New York, NY, USA: Association for Computing Machinery, 2019.
- [58] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers in Neuroscience*, vol. 11, p. 324, 2017.
- [59] P. Falez, P. Tirilly, I. M. Bilasco, P. Devienne, and P. Boulet, "Multi-layered spiking neural network with target timestamp threshold adaptation and stdp," *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.
- [60] J. Lee, R. Zhang, W. Zhang, Y. Liu, and P. Li, "Spike-train level direct feedback alignment: Sidestepping backpropagation for on-chip training of spiking neural nets," *Frontiers in Neuroscience*, vol. 14, p. 143, 2020.
- [61] G. Detorakis, S. Sheik, C. Augustine, S. Paul, B. U. Pedroni, N. Dutt, J. Krichmar, G. Cauwenberghs, and E. Neftci, "Neural and synaptic array transceiver: A brain-inspired computing framework for embedded learning," *Frontiers in Neuroscience*, vol. 12, p. 583, 2018.

Wenzhe Guo received his B.S. majored in integrated circuits design from University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2017. He obtained M. Sc. degree in Electrical Engineering department in King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia, in 2018. Currently, he is a Ph.D. student in KAUST, working on the design of neuromorphic hardware systems for different applications. His research interests include digital IC design, neuromorphic computing, event-based computation, and computer vision.

Hasan Erdem Yantir (S'13) received the B.Sc. degree from the computer engineering and electrical and electronics engineering, Yeditepe University, Istanbul, Turkey, in 2011, the M.Sc. degree in computer engineering from Boğaziçi University, Istanbul, and the Ph.D. degree from the University of California, Irvine, CA, USA, in 2018. He is currently a postdoctoral fellow at King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia. He was an Engineer with Ericsson, Istanbul, from 2011 to 2012, and with

Ford Motor Company, Istanbul, from 2012 to 2013. His current research interests include AI accelerator architectures, neuromorphic computing, digital IC design, in-memory computing, computer architectures, and reconfigurable computing.

Khaled Nabil Salama (SM'10) received the B.S. degree from the Department Electronics and Communications, Cairo University, Cairo, Egypt, in 1997, and the M.S. and Ph.D. degrees from the Department of Electrical Engineering, Stanford University, Stanford, CA, USA, in 2000 and 2005, respectively. He was an Assistant Professor at Rensselaer Polytechnic Institute, Troy, NY, USA, from 2005 to 2009. He joined the King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia, in 2009, where he is currently a Professor,

and was the Founding Program Chair until August 2011. He is the Director of the sensors initiative, a consortium of nine universities (KAUST, MIT Cambridge, UCLA, GATECH, Brown University, Georgia Tech, TU Delft, Swansea University, the University of Regensburg, and the Australian Institute of Marine Science (AIMS)). His work on CMOS sensors for molecular detection has been funded by the National Institutes of Health (NIH) and the defense advanced research projects agency (DARPA), awarded the Stanford–Berkeley Innovators Challenge Award in biological sciences and was acquired by Illumina Inc. He is the author of 250 articles and 14 U.S. patents on low-power mixed-signal circuits for intelligent fully integrated sensors and neuromorphic circuits using memristor devices.

Mohammed E. Fouda received the B.Sc. degree (Hons.) in Electronics and Communications Engineering and the M.Sc. degree in engineering mathematics from the Faculty of Engineering, Cairo University, Cairo, Egypt, in 2011 and 2014, respectively. Fouda received his Ph.D. degree from the University of California-Irvine, USA in 2020. Currently, He works as an assistant researcher at University of California, Irvine. Fouda published more than 100 peer-reviewed Journal and conference papers, 1 Springer book and 3 book chapters. His H-index

is 20 with more than 1300 citations. His research interests include analog AI hardware, neuromorphic circuits and systems, brain-inspired computing, memristive circuit theory, fractional circuits and systems and analog circuits. He serves as a peer-reviewer for many prestigious journals and conferences. He also serves as a review editor in Frontier of Electronics and International Journal of Circuit theory and applications in addition to a technical program committee member in many conferences. He was the recipient of the best paper award in ICM years 2013 and 2020 and the Broadcom foundation fellowship for 2016-2017

Ahmed E. Eltawil (S'97–M'03–SM'14) received the Doctorate degree from the University of California, Los Angeles, in 2003 and the M.Sc. and B.Sc. degrees (with honors) from Cairo University, Giza, Egypt, in 1999 and 1997, respectively. Since 2019 he is a Professor at the Computer, Electrical and Mathematical Science and Engineering Division (CEMSE) at the King Abdullah University of Science and Technology (KAUST), Thuwal, KSA. Since 2005 he was with the Department of Electrical Engineering and Computer Science, at the University

of California, Irvine, where he founded the Wireless Systems and Circuits Laboratory. His research interests are in the general area of low power digital circuit and signal processing architectures with an emphasis on mobile systems. Dr. Eltawil has been on the technical program committees and steering committees for numerous workshops, symposia, and conferences in the areas of low power computing and wireless communication system design. He received several awards, as well as distinguished grants, including the NSF CAREER grant supporting his research in low power systems.