Neural Computing and Applications
https://doi.org/10.1007/s00521-023-08650-6

ORIGINAL ARTICLE

A generalized hardware architecture for real-time spiking neural networks

Daniel Valencia[1,2] · Amir Alimohammad[1]

Received: 1 February 2023 / Accepted: 2 May 2023

Abstract
This article presents an area- and power-efficient hardware architecture for the brain-implantable spiking neural networks (SNNs). The proposed generalized hardware architecture is parameterizable and reconfigurable such that the maximum supported number of neurons, the interconnection structure among neurons, and the resolution of the time step can be readily adjusted for realizing various SNN topologies. The designed SNN hardware architecture is capable of emulating moderately-sized SNNs with tens of thousands of neurons in real-time with varying degrees of parallelism, while reducing the resource utilization by 34% for similarly sized SNNs implemented on a single field-programmable gate array (FPGA). We evaluate the model using the MNIST digit recognition benchmark and show that the network can accurately classify handwritten digits with 89.8% accuracy. Compared to the other recently implemented SNN emulators based on FPGAs, the designed and implemented single-FPGA system is able to emulate moderately-sized SNNs instead of using a cluster of FPGAs or CPUs. The application-specific integrated circuit (ASIC) implementation of a moderately-sized SNN is estimated to occupy 3.6 mm^2 of silicon area. Post-layout synthesis and simulation results show that the ASIC will dissipate 3.6 mW of power from a 1.16 V supply while operating at 34.7 MHz in a standard 32-nm CMOS process.

Keywords Spiking-neural networks · Field-programmable gate arrays · Application-specific integrated circuits · Machine learning

1 Introduction

In the United States alone, over 17000 cases of spinal cord injury (SCI) or other neuro-degenerative diseases (NDDs) are reported per year [1]. SCI and NDDs can affect the ability of biological brains to communicate and/or interface with other parts of the body. Brain-machine interfaces (BMIs) have become a viable solution for creating alternative communication pathways between the human brain and external devices, such as a prosthetic arm or a computer.

Over the past decade, artificial neural networks (ANNs) have been employed in a wide range of applications. Feed-forward neural networks (FNNs) and multi-layer perceptrons (MLPs) process data in a single direction from input to output, while recurrent neural networks (RNNs) utilize feedback connections among artificial neurons (ANs) to learn about the temporal information within the input data. Convolutional neural networks (CNNs) are primarily used in image and video processing. They can decompose images with a variety of filter kernels and can be combined with FNNs to perform accurate image classification, segmentation, and recognition tasks, such as detecting and segmenting medical images to assist with disease diagnosis [2]. Temporal convolutional networks (TCNs) employ causal convolutions and strided dilations to adapt to sequential data with a receptive field larger than conventional RNNs [3]. Transformer neural networks [4] employ a self-attention mechanism to weigh the relevance of particular points in sequential data. The mechanism of self-attention has proven invaluable for natural language processing tasks, such as machine translation [5].

The early study of neural networks led to the development of more biologically-plausible neural networks.

✉ Daniel Valencia
dlvalencia@sdsu.edu

[1] Department of Electrical and Computer Engineering, San Diego State University, San Diego, USA

[2] Department of Electrical and Computer Engineering, University of California, La Jolla, USA

Published online: 24 May 2023

Spiking neural networks (SNNs) [6], often referred to as the third generation of neural networks, employ more accurate mathematical models for describing the behavior of brain neurons. The neurons in SNNs are connected to one another using synapses, and the weights associated with the synapses model the impact of one neuron's spike on its post-synaptic neighboring neurons. The values of the synaptic weights are obtained through the training process, in which weights are fine tuned so that the employed SNN can mimic desired spiking behaviors, such as responding to input stimuli at specific times. While neurons in an ANN communicate by sending discrete values to one another, spiking neurons communicate with one another via exhibiting spiking behavior, similar to those of a biological brain, through the emulation of the membrane potentials of the neuron.

Due to their biological plausibility, which is not inherent in the relatively simple mathematical models of neurons in the conventional ANNs, SNNs have become the prime candidate for interfacing with living biological brain neurons [7, 8]. Various experiments have shown that interfacing a SNN for stimulating the main respiratory muscles, such as the diaphragm, has helped patients regain ventilatory control [9]. Recently, it has been shown that information transfer from SNNs to biological neural networks is possible [10], and hence SNNs can potentially be realized as replacement neural micro-circuits. For example, in [11], one hundred synthetic spiking neurons' parameters are tuned to match the behavior of biological neurons for restoring communication between two neuronal populations separated by a focal lesion. Another effort has shown that SNNs can potentially be used as replacements for damaged biological networks in the hippocampus of behaving rats to restore and improve memory functionality [12, 13].

The real-time operation of the SNNs, however, imposes strict limitations on the feasible computational complexity of the mathematical models for SNNs and hence, on the silicon area, power consumption, and latency of the realized hardware circuits. Various neuron models have been proposed, ranging from the simple integrate and fire (I &F) model [14] to the relatively complex Hodgkin and Huxley (HH) model [15]. The Izhikevich neuron model [16], however, can reproduce the spiking dynamics of cortical neurons reliably while also being computationally more efficient than the other state-of-the-art SNN models [17].

This article focuses on the area- and power-efficient design and implementation of a generalized hardware architecture for moderately-sized SNNs, in the range of tens of thousands of neurons, based on the Izhikevich neuron model. While most of the previously-published work have elaborated on fixed dedicated hardware architectures, supporting a predefined number of neurons and synaptic interconnect topologies [18–21], the proposed architecture aims to exploit the reconfigurable nature of the field-programmable gate arrays (FPGAs) to support an arbitrary number of spiking neurons and diverse network interconnect structures, while offering an adjustable time resolution. In addition to FPGA [18–23] and programmable processor [24] realizations of SNNs, application-specific integrated circuit (ASIC) implementations of SNNs have also been reported [25–30]. While FPGA and processor realizations offer greater flexibility in realizing diverse SNNs, ASIC implementations can consume significantly lower power and can be implemented in a smaller form factor and hence, are practical for the in-vivo brain implantations. In this work, we propose to implement a tightly integrated SNN core with embedded memory and computational elements. In contrast to the state-of-the-art ASIC realizations, our design's power consumption and real-time processing are not affected by the spiking rate of the emulated SNN. Our design supports various spiking rates based on the number of neurons and allows an adjustable degree of parallelism and time step resolution. State-of-the-art ASIC implementations, such as Loihi [30], SpiNNaker [24], and Tianjic [31], aim for high-performance computing by exploiting massively-parallel architectures or distributing computations over a large amount of processing cores. While our approach is that of a generalized real-time SNN, our goal is to integrate all of the required elements of the SNN core, such as processing elements and memory, into a single unified chip. The relatively small silicon area and low power consumption of the implemented generalized SNN hardware architecture make it suitable for single-chip realizations, while emulating SNNs with various sizes and interconnect topologies in real-time. Our proposed design is scalable and additional cores can be readily utilized at the expense of increased silicon area.

The rest of this article is organized as follows. Section 2 briefly reviews the main computational requirements of the Izhikevich-based SNN model. Section 3 provides a detailed explanation of the designed generalized hardware architecture for SNNs. A memory partitioning scheme for parallel computation of the membrane potentials is presented. Section 4 discusses the limitations of the real-time SNN realizations, quantifies the implementation characteristics of the proposed design, and provides comparisons between the FPGA and ASIC implementation results of our generalized SNN architecture and those of the state-of-the-art work. Finally, Sect. 5 makes some concluding remarks.

2 A computationally-efficient SNN model

Izhikevich's neuron model iteratively reproduces the spiking dynamics of biological neurons using the following mathematical expressions [16]:

$$v' = (t \cdot (0.04v^2 + 5v + 140 - u + I)) + v, \quad (1)$$

$$u' = a(bv' - u), \quad (2)$$

$$\text{if} \quad v' \geq \quad 35 \quad \text{mV}, \begin{cases} \text{v} \leftarrow \text{c} \\ \text{u} \leftarrow \text{u}' + \text{d} \end{cases}, \quad (3)$$

where v denotes the membrane potential in terms of mV, u denotes the recovery variable, I denotes the accumulated input activity to the neuron, and t denotes the resolution of the time step. The spiking dynamics of the neurons can be controlled by the parameters a–d, where a denotes the decay rate of the recovery variable u, b denotes the coupling between v and u, which can result in sub-spiking oscillations, c denotes the value of the membrane potential after a spike, and d denotes the reset of the recovery variable u after a spike. Given a unit-step input current, Fig. 1 shows the membrane potential voltages for four different types of neurons. The regular spiking (RS) neurons exhibit the spiking behavior shown in Fig. 1a with parameters $a = 0.02$, $b = 0.2$, $c = -65$, and $d = 8$, and are the most common type of neurons in the mammalian cortex [32]. The parameters $a = 0.02$, $b = 0.2$, $c = -50$, and $d = 2$ define the chattering (CH) neurons, which fire bursts of spikes following an excitation, as shown in Fig. 1**b**. Figure 1c shows a low-threshold spiking (LTS) neuron, which can fire high-frequency spikes with adaptation (i.e., slowing down of firing rate) using $a = 0.02$, $b = 0.25$, $c = -65$, and $d = 2$. Figure 1d shows a fast-spiking (FS) neuron, which can fire rapid spikes without adaptation, with $a = 0.1$, $b = 0.2$, $c = -65$, and $d = 2$. Both RS and CH neurons are examples of excitatory cells, while the LTS and FS neurons are inhibitory cells.

An iteration of the Izhikevich-based SNNs using the neuron state Eqs. (1)–(3) can be executed in three steps: (i) receiving network inputs; (ii) performing synaptic weight accumulation; and (iii) computing the membrane potentials. In the first step, the network inputs are buffered. The second step poses the main computational bottleneck for realizing the SNN model. In traditional ANNs, the neuron's communicate by passing discrete values to one another; however, SNNs operate differently. The output of a spiking neuron is the excitation or lack thereof and hence, can be represented as a binary value, denoting whether the neuron emitted (fired) a spike. A neuron fires a spike if the membrane potential crosses a predefined threshold of 35 mV, which results in resetting the membrane potential v to the predefined neuron's potential value c. After a neuron fires a spike, the neuron cannot fire another spike for a period of time, known as the refractory period. When a neuron fires, the synaptic weights of all pre-synaptic firing neurons are accumulated and added to the current input stimuli for every connected post-synaptic neuron. Note that since Eq. (1) is on the millisecond scale, more precise spike timings can be modeled using finer temporal resolutions. This requires more iterative computations of Eq. (1) to complete one millisecond of emulation. For example, if the time resolution $t = 0.5$ ms, the membrane potential v' has to be computed twice (with $v = v'$ for the second iteration). The third and final step is to update the membrane potentials using the neuron state Eqs. (1)–(3).

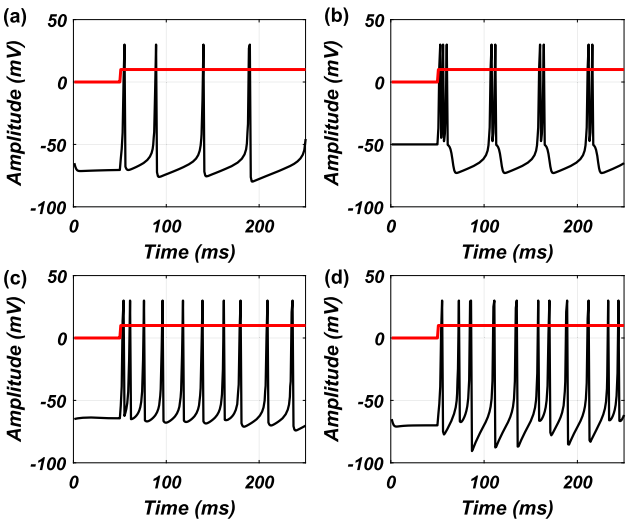

Fig. 1 Membrane potential waveforms for **a** regular spiking, **b** chattering, **c** low-threshold spiking, and **d** fast-spiking neurons

The biologically-inspired models attempt to replicate the behavior of biological neural systems, but not necessarily in a biologically-plausible manner. For example, the I &F model employs a relatively simple biologically-inspired spiking neuron model. Even though this model is less biologically realistic, it produces enough behavioral resemblance to the biological neurons to be useful in spiking neural systems. The leaky I &F model adds a leak term to the simple I &F model, which causes the potential on a neuron to decay over time. The leaky I &F model updates the membrane potential v as:

$$v' = I + a - bv,$$
$$\text{if} \quad v' \geq v_{thresh} : v \leftarrow c,$$

where I denotes the input current to the neuron and a, b, c, and v_{thresh} are the neuron parameters. Because the I &F model has only a single variable, it cannot exhibit all 20 neuro-computational features and hence, is not considered a biologically-plausible neuron model. An alteration to the I &F model decreases the frequency of spiking and is referred to as spike-frequency adaption. An extension to this model was the Adaptive Exponential (AdExp) model which changes the input current injection to conductance-based injection. These extensions represent a better reflection of the changes seen by cortical neurons. The biologically-plausible models, however, explicitly model

the types of behavior that are seen in biological neural systems. The most popular biologically-plausible neuron model is the HH model, which is relatively computationally-intensive and takes 1200 floating-point operations to evaluate one millisecond of time, which can be limiting for real-time SNN emulation. The Izhikevich spiking neuron model (IZH) was developed to produce similar bursting and spiking behaviors as can be elicited from the HH model, but do so with a much simpler computation.

3 A generalized reconfigurable hardware architecture for SNNs

For network topologies which follow the classical, layer-based structure, network inputs are only applied to the input layer. However, for realizing a generalized hardware architecture, the inputs are given to every neuron in the network. The designed scalable architecture utilizes the degree of parallelism parameter *PDeg*, which controls the number of input accumulations to be computed in parallel. Utilizing folding transformations [33], computations can be reduced to *PDeg* SNs at a time, resulting in a hybrid sequential-parallel hardware architecture. By effectively folding the computations down to *PDeg* neurons at a time, the overall resource utilization can be significantly reduced. This in effect allows more complex systems to be integrated onto a single chip or logic resources can be allocated for additional neural signal processing or computational functions.

For example, the brain-computer interface in [34] employs spike sorting in conjunction with emulated spiking neural networks to manipulate a prosthetic device. While their system utilizes multiple CPU cores for real-time SNN emulation, our proposed hardware architecture can accommodate several neural signal processing modules onto a single chip for emulation of the real-time moderately-sized SNNs. By updating *PDeg* SNs at a time, the hardware architecture can be readily scaled such that a millisecond of SNN emulation is completed within a predefined number of clock cycles. Compared to the software realizations, the total number of neurons N, and the degree of parallelism *PDeg*, supported by our hardware architecture are limited by the number of reconfigurable resources available on the target FPGA device or the silicon area and power consumption of the ASIC implementation.

3.1 SNN core memory

The top-level block diagram of the designed SNN core is shown in Fig. 2. The SNN architecture consists of several memory units for storage of: (i) the network's inputs *Input Mem.*; (ii) the neurons' membrane potential voltage values *Voltage Mem.*; (iii) the neurons' recovery variables

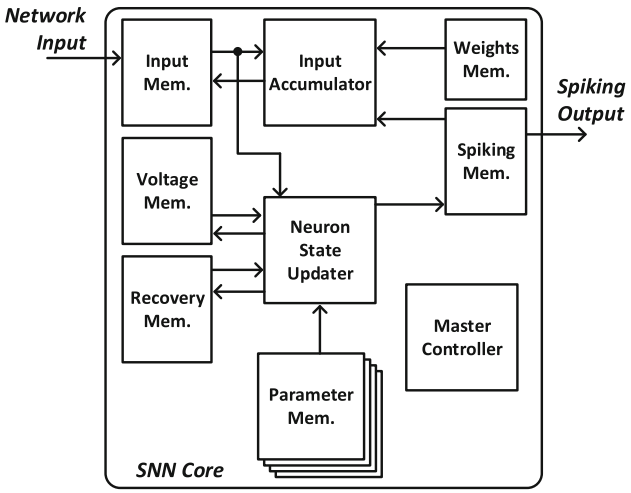

Fig. 2 The top-level block diagram of the designed and implemented SNN core

Recovery. Mem., (iv) the SN's outputs *Spiking Mem.*; (v) the synaptic weights *Weights Mem.*; (vi) and the parameter memory *Parameter Mem.* to store the Izhikevich parameters $a - d$ per neuron. In order to support an arbitrary interconnect structure among the SNs, the *Weights Mem.* stores N^2 synaptic weights. By changing the synaptic weights stored in the *Weights Mem.*, various SNN configurations can be realized. The memory units shown in Fig. 2 are implemented using the FPGA's block RAM (BRAM) resources. The BRAMs are specialized on-chip storage resources on an FPGA, where their maximum width and depth varies among different device families. For example, the Xilinx Artix-7 FPGAs can store up to 36 kB of data in BRAMs, with the BRAMs' width being programmable up to 72 bits.

In order to process *PDeg* neuron computations per clock cycle, all memory units are partitioned in such a way that there are no conflicting data accesses. This avoids implementing a queue for managing memory read/write requests. Each row of the designed partitioned memory modules stores the indices of *PDeg* SNs, and thus, $N/PDeg$ rows are required to store all neurons' indices. The neuron's index stored at row i and column j of the partitioned memory module is given as $i + (j \times N/\ PDeg)$. The synaptic weight memory has $N^2/\ PDeg$ rows and each row stores *PDeg* weights, each of which is denoted with a unique identifier $\lambda\rho$, representing the synaptic weight from neuron λ to neuron ρ. Moreover, a bitwise OR operation is performed on the read data from the SN output memory. When all of the *PDeg* bits in a row of the spiking output memory are zeros, unnecessary accumulations from the neurons represented by that row to all other post-synaptic neighboring neurons are avoided, which in turn saves a significant number of clock cycles, memory read operations, and redundant accumulations.

3.2 Neuron input accumulation

Due to the generalized architecture, each neuron can have a large fanout of N, where N denotes the number of total neurons supported by the target device. The synaptic input accumulation thus poses the largest computational bottleneck of the proposed design. Figure 3 shows the datapath of the *Input Accumulator* module. The network inputs *Net. Input* for the current emulation time are passed to the accumulation register R through the 2-to-1 multiplexers. The *iRST* control signal enables the accumulation registers to be enabled, thus resetting the accumulated input to the first *Net. Input* value. The synaptic weights *Syn. Weight* and *Spiking Output* are read from the *Weights Mem.* and *Spiking Mem.*, respectively. If a *Spiking Output* bit is high, then the register is enabled, and the synaptic weight from that source neuron is added to the net input of the neuron.

3.3 Neuron state computation

The module *Neuron State Updater* performs the main SNN computations described by the neuron state Eqs. (1) – (3). Figure 4 shows the high-level block diagram of the neuron state updater. The datapath consists of two pipelined units, the membrane voltage updater *MVU* and the recovery variable updater *RVU*, which are responsible for computing the neuron state Eqs. (1) and (2–3), respectively. The datapath of the MVU unit, shown in Fig. 5, consists of the voltage updater units (VUUs), pipeline latency compensation shift registers *PLC Shift Registers*, and ΔV scaling units. The seven-stage pipelined VUUs compute the changes to the neuron voltage values, denoted as ΔV. While accurate representation of 0.04 in the fixed-point numerical format would require a relatively large number of fractional bits, we instead utilize a shift-and-add approximation as $\frac{1}{32}+\frac{1}{128}+\frac{1}{1024}=0.040039\approx 0.04$. The rest of Eq. (1), in which the input value v is added to the scaled ΔV, is performed by the ΔV scaling units. The inputs to the ΔV scaling units include the change in voltage for each neuron, the original voltage input to the neuron, and the time scale. The original input value v is shifted

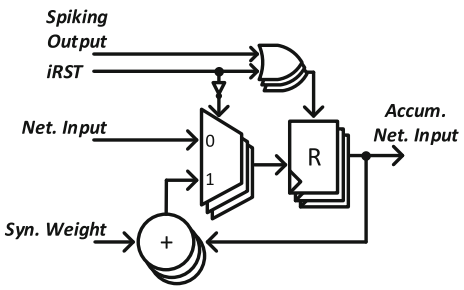

Fig. 3 The datapath of the input accumulator module

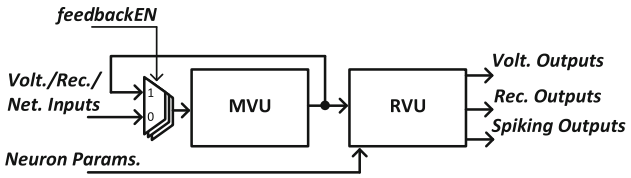

Fig. 4 The block diagram of the neuron state updater

through the *PLC Shift Registers*. The time scale is implemented using the arithmetic right shift of input values by $-1\times(\log_2 t)$ bit positions, where t denotes the resolution of the time step used in Eq. (1). For $t = 0.0625$ ms, this corresponds to a 16-bit arithmetic right shift.

The number of iterations used by the VUUs is equal to the shift position used by the ΔV scaling units. The multiplexers at the inputs to the VUUs and the *PLC Shift Registers* can select either the input ports to read voltage, recovery, and network inputs for the VUUs, or to accept the outputs of the VUUs for subsequent iterations. The select line *feedbackEN* asserted by the control unit is responsible for enabling the feedback into the VUUs. For cases in which the number of inputs streaming to the VUUs exceeds the pipeline depth of VUUs, additional output shift registers are added to the output of the ΔV scaling units and to the output of the *PLC Shift Registers*. The number of inputs that will propagate through the VUUs is equal to the number of rows in the voltage, recovery variable, and input memories, which is *N*/*PDeg*.

Figure 6 shows the datapath of the designed *RVU Pipeline* module. It consists of the *Recovery Voltage Units* (RVUs), pipeline latency compensation shift registers *PLC Shift Registers*, a bank of comparators, a bank of resettable adders, and output resetting/spiking multiplexers. The four-stage pipelined RVU computes the output recovery variable, as given in Equation (2). The outputs *Volt. Outputs* and *Rec. Outputs* of the MVU are passed to the *Recovery Pipeline* unit, shown as *MVU Outputs*. The desired spiking dynamics of neurons, denoted by the Izhikevich parameters $a - d$, are passed through the input port *Neuron Params.*. The *PLC Shift Registers* are used to compensate for the latency of the RVU and to propagate the parameters c and d, and voltage v values. If the shifted *Volt. Inputs* values are greater than or equal to the value of the spike threshold in Eq. (3), a spike is emitted. The output of the comparators are used as the select lines of the multiplexers, which will either pass the original *Volt. Inputs* values to the output if a spike did not occur, or they will pass the shifted reset value c, given the occurrence of a spike. Similarly, the output of the RVUs, the updated recovery variables, will be passed to the output port *Rec. Outputs* if a spike did not occur; otherwise, the updated recovery variables will be added to the shifted parameters d for resetting the recovery variables, as given in Eq. (3).

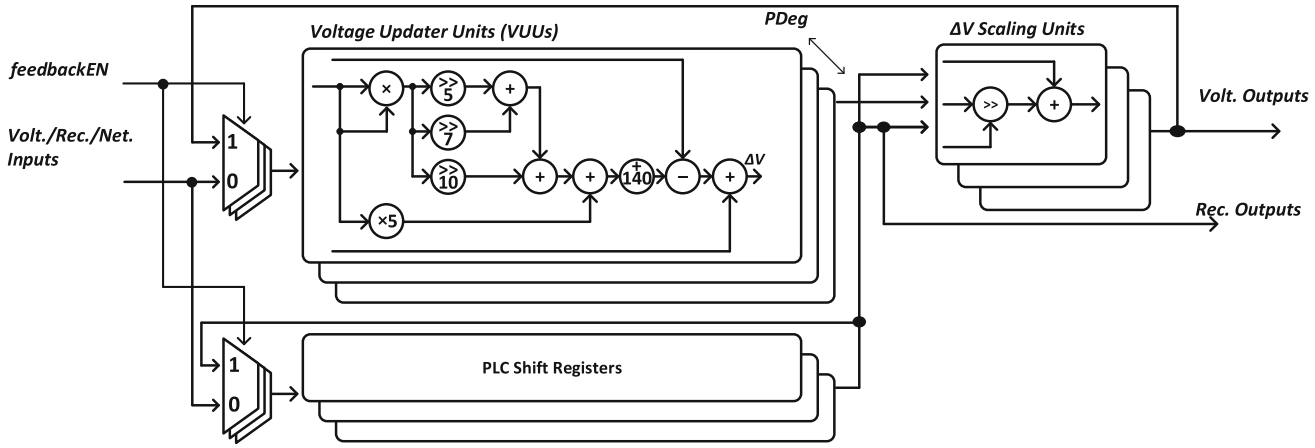

Fig. 5 The block diagram of the membrane voltage updater (MVU)

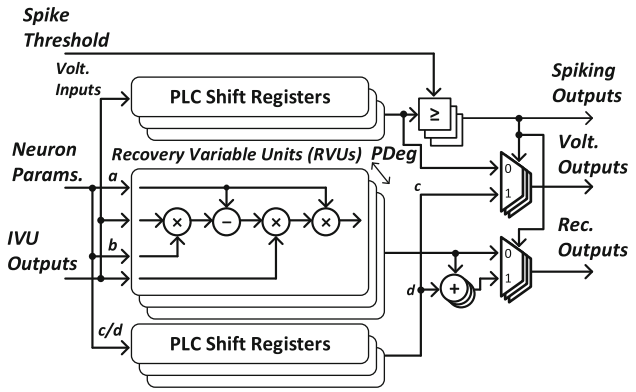

Fig. 6 The block diagram of the Recovery Variable Updater (RVU)

The operations of the designed architecture can be briefly explained by the following example. For simplicity, assume three RS neurons, each of which has a synaptic weight of 10 mV to the other two neurons. Also, assume that the current values for the membrane voltage and the recovery variables are $v = [-45, -45, -23]$ and $u = [-13, 12, 10]$, respectively, and $t = 0.5$. Finally, assume that the next external inputs to each neuron are equal to $I = [5, 0, 12]$. First, the neuron input accumulation unit will read input data from the *Input Mem.*. Initially, the accumulated network inputs for each neuron are equal to the values read from the memory, i.e., $I_{\text{acc}} = [5, 0, 12]$. Next, the *Spiking Mem.* is read and its output value is 100, which indicates that the first neuron emitted a spike during the last millisecond of operation. Thus, the synaptic weight value of 10 mV is added to the accumulated network input of neurons '2' and '3' and hence, $I_{\text{acc}} = [5, 10, 22]$. Once the external inputs and synaptic weights are accumulated, the *Neuron State Updater* computes the updated values of $v' = [-25, -52, 98]$ and $u' = [0.15, -0.45, 0.19]$. Since the membrane potential of neuron '3' exceeds the threshold of 35 mV, the membrane potential and the recovery variable for neuron '3' is updated as $v' = [-25, -52, -65]$ and $u' = [0.15, -0.45, 8.19]$. The spiking outputs of the neurons, 001, are then written into the *Spiking Mem.*, which indicates that neuron '3' has fired a spike. This process is repeated for each millisecond of the SNN operation.

4 SNN emulation and implementation results

The brain's neural activity is recorded using either non-invasive or invasive techniques, the latter of which generally provides greater spatial resolutions and decoding accuracy. The raw recorded signals are then amplified and digitized using an analog-to-digital converter (ADC), and filtered to the frequency bands of interest. In order to provide real-time interfacing between biological neurons and a SNN, we must verify that the SNN can operate predefined time constraints.

4.1 Real-time SNN emulation

A typical sampling frequency of 10 kHz [35] requires the SNN processing to be completed within 0.1 ms for real-time operation. Therefore, the number of iterations of Equation (1) must be either 8 ($t = 0.125$ ms) or 16 ($t = 0.0625$ ms). Figure 7a shows the maximum clock cycle latency for the input accumulation when all 1000 neurons in the network have fired. Figure 7b shows the number of spiking neurons for a SNN with 1000 fully-connected neurons. One can see that the maximum spiking rate is only 5.1% of all neurons in the network and hence, it is unlikely that all neurons of the network will fire simultaneously. The average spiking rate is even significantly

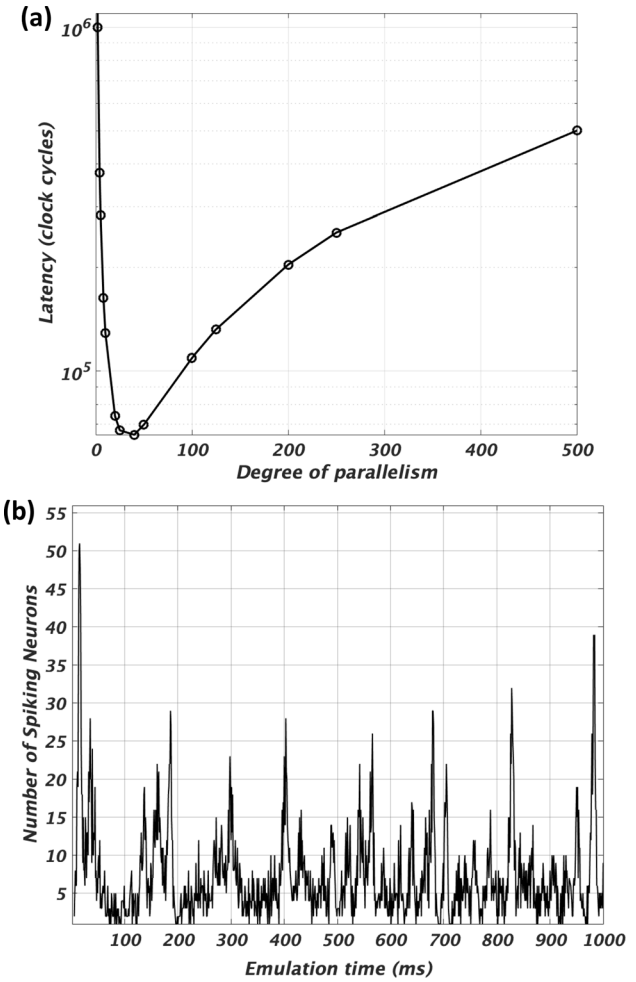

Fig. 7 **a** The maximum clock latency of the input accumulation module for varying degrees of parallelism, and **b** the number of spiking neurons for a SNN with 1000 fully-connected neurons over one second of emulation

smaller, at 0.7413%. The accumulation latency can be given as:

$$((2N/\text{PDeg})^2 + 20) + SG(N + (\text{PDeg}^2 - \text{PDeg})), \qquad (4)$$

where SG denotes the number of SN groups in which at least one SN fired. The maximum number of SN groups $N/PDeg$ gives a maximum latency of 129, 020 clock cycles for a network with $N = 1000$ and $PDeg = 10$.

Figure 8 shows the latency, based on the number of clock cycles, for updating the neurons' v and u values, with the time resolutions $t = 0.125$ ms and $t = 0.0625$ ms, for a SNN with $N = 1000$ neurons over varying degrees of parallelism. The number of clock cycles required to complete the neuron state updating process can be given as $\left(\frac{N}{\text{PDeg}}(V_{\text{Iter}} + 1)\right) + 7$, where $V_{\text{Iter}} = 1/t$ denotes the number of iterations required by the membrane voltage updater for the desired time resolution t. Because the designed architecture utilizes shifting operation to implement the

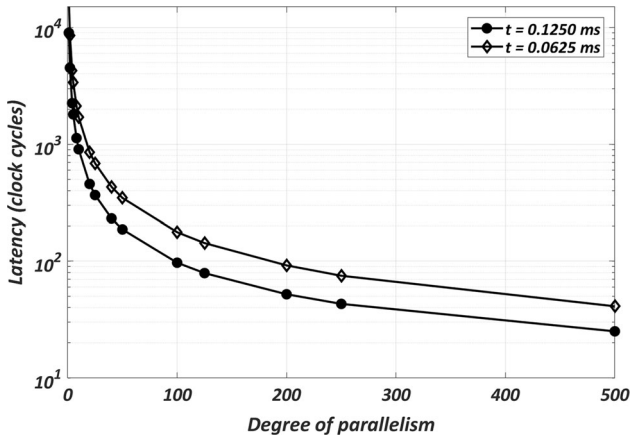

Fig. 8 The clock cycle budget for computing the neuron state equations with varying degrees of parallelism for a network with $N = 1000$ neurons over 1000 emulation time steps

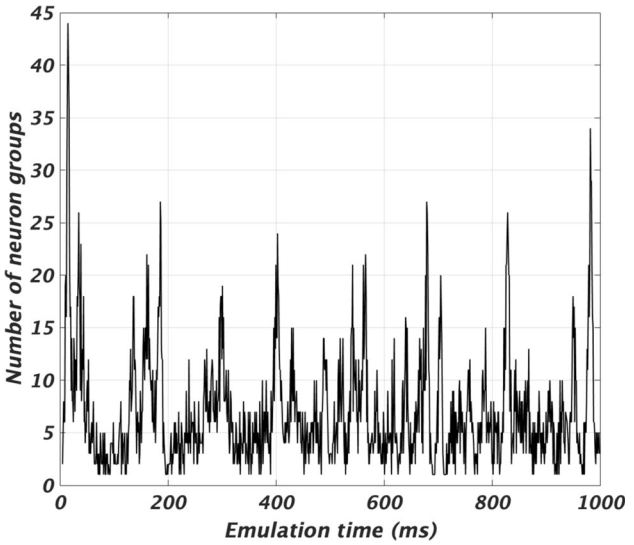

Fig. 9 The number of SN groups for $N = 1000$ neurons, $PDeg = 10$, over 1000 emulation time steps

required time scaling in Equation (1), the time resolution t must be a negative integer power of 2 and hence, the degree of parallelism $PDeg$ must be an even divisor of N. According to Fig. 8, FPGAs that can accommodate storage and computational resources for about one-fifth of the number of neurons N in a SNN yield approximately two degrees of magnitude lower accumulation latency than emulating a SNN with a single processing element (e.g., traditional in-order execution with $PDeg = 1$). This is also the case for the input accumulation process. To quantify the time available for the real-time operation (i.e., completing one iteration of computations within one millisecond), we can determine the number of SN groups to estimate the maximum clock cycle budget of the input

accumulation module for each SN group. As shown in Fig. 9, for $N = 1000$ and $PDeg = 10$, the maximum number of SN groups is 44, which corresponds to an input accumulation clock cycle budget of 67, 980. This clock cycle budget can be added to the clock latency of the MVU, which is 1707 for $V_{\text{Iter}} = 16$. As a result, the total clock cycle budget for the real-time SNN operation is 69, 687.

4.2 Hardware implementation results

We have implemented the designed generalized reconfigurable hardware architecture for a SNN with 1000 fully-connected neurons with $PDeg = 10$ on a Xilinx Artix-7 XC7A200T FPGA. It utilizes 12700 (9.49%) lookup tables (LUTs), 11139 (4.16%) flip-flops (FFs), 14.50 (3.97%) block RAMs (BRAMs), and 110 (14.86%) dedicated multiplication units (DSP48s). This network size is comparable to those of the other single FPGA implementations, for a fair comparison. The iterative computation of the membrane potentials via the MVU with the folded hardware architecture requires a relatively small number of configurable FPGA resources. The clock cycle budget for the real-time operation of the implemented SNN can be determined as $92\text{MHz} \times 1\text{ms} = 92000$ cycles, where 92 MHz is the operating frequency of the SNN implemented on the FPGA and one millisecond is required for completing the SNN computations before the next set of neural samples are available based on a 10 kHz sampling rate. The inference speed of the network depends on the degree of parallelism, the time step resolution, and the number of emulated spiking neurons. For example, given an FPGA operating frequency of 92 MHz, degree of parallelism $PDeg = 10$, emulating 1000 spiking neurons with a 0.5 ms time step resolution, each computation time step takes 0.49 ms. Note that the size of the networks that can be implemented is directly limited by the number of computational resources and memory elements available on the target device.

4.3 Design verification

To test the designed and implemented SNN architecture, we have developed a custom MATLAB-based graphical user interface (GUI), shown in Fig. 10. The GUI interacts with the FPGA through a serial port. The baud-rate and the parameters of the synthesized and implemented SNN on the FPGA device are specified under the *FPGA Settings* panel. The *Options* panel allows the user to randomly generate network parameters and synaptic weights, load previously generated network parameters and synaptic weights from a file, view the network parameters and synaptic weights currently loaded onto the FPGA, and upload the network parameters and synaptic weights onto

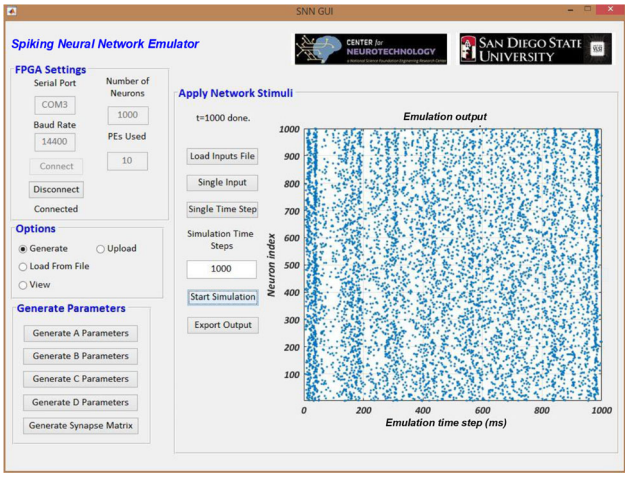

Fig. 10 The developed graphical user interface for emulating and testing SNNs on FPGAs

the FPGA. The SNN is trained using our custom-developed scripts in Python. The neuron parameters $a - d$ and the synaptic matrix values derived by the training are then loaded onto the SNN hardware through the GUI for functional verification. Once the connection to the FPGA is established, the GUI displays the *Apply Network Stimuli* panel, which allows the user to apply network inputs to the SNN hardware implemented on the FPGA. The scatter plot *Emulation output* then shows the spiking activity of each neuron for each emulation time step. As shown in Fig. 10, the emulation output of the implemented SNN utilizing 1000 fully-connected neurons with $PDeg = 10$. A neuron has a connection to another neuron in the network through a randomly initialized weight value. Weight values of zero imply that there is no connection between a pre- and post-synaptic pair of neurons. In this specific model, neurons have no self-recurrent connections. Although the SNs are randomly coupled and no synaptic training was employed, the occasional dark vertical lines indicate synchronized firing events across the majority of neurons. This behavior is akin to the mammalian neo-cortex behavior during an awake state and is an indication that the Izhikevich model reproduces the behavior of biological neurons and their spiking dynamics.

In order to evaluate the SNN and Izhikevich model for a practical application, we evaluate our design using the MNIST digit recognition dataset [36]. The dataset consists of 28×28 pixel grayscale handwritten digits with values between 0 and 255. The images are converted into poisson-firing spike trains following the approach in [37]. The firing rate of a pixel is proportional to the pixel intensity, with a maximum firing rate of 350 Hz. Each set of image spike trains are presented for 50 milliseconds, with another 50 millisecond of no input to allow the network to settle before presenting the next image. The weights from input

spike trains to spiking neurons are derived through spike-time dependent plasticity (STDP) using a supervised training scheme described in [38]. The weight change rule is given by:

$$\Delta w_{ij}(t) = \mu \cdot \xi_j(t) \sum_{\hat{t}=t-\epsilon}^{t} s_i(\hat{t}), \tag{5}$$

where $w_{ij}(t)$ denotes the weight from input i to neuron j, μ denotes the learning rate, $\xi_j(t)$ denotes the STDP-based learning rule for neuron j, ϵ denotes the STDP window in milliseconds, and $\sum_{\hat{t}=t-\epsilon}^{t} s_i(\hat{t})$ denotes the number of spikes fired by input i during the time interval $[t-\epsilon,\ t]$. The learning rule $\xi_j(t)$ is given by:

$$\xi_j(t) = \begin{cases} +1 & z_j(t) = 1, r_j \neq 1 \in [t-\epsilon, t], \\ -1 & z_j(t) = 0, r_j = 1 \in [t-\epsilon, t], \\ 0 & \text{otherwise}, \end{cases}$$

where $z_j(t)$ denotes the target spike train for neuron j and r_j denotes whether neuron j has fired a spike within the STDP window. The above equations can thus be interpreted as local STDP-based rules, whereby neurons that should fire have their input weights strengthened and neurons that should not fire have their weights weakened. If a neuron fires within the STDP time window ϵ, the neuron is performing as desired and no weight change is required.

While the STDP-based learning rules are suitable for on-chip learning, our proposed custom architecture is not intended to support it. On-chip training needs additional hardware, such as storing a history of spiking activity over a STDP window. Also, for training the synaptic weights from external spiking activity, an additional memory unit would be required to store input spiking history. Combining STDP-based learning rules with stimuli-dependent teaching signal requires an additional memory unit to define which neuron populations should respond to specific stimuli.

In our network, the 100 spiking neurons are divided into ten groups of equal size, where each group is assigned to an input digit class. The target spike trains have a firing rate of β Hz. The spike trains are active during the first 50 milliseconds of image spike train presentation. Table 1 gives the neuron model and learning parameters used during our experiments. We use a subset of the MNIST dataset consisting of 60000 images, with 42000 used for training and 18000 used for testing. The training set is presented to the network six times and weights are tuned with the learning rule given in Eq. (5). After training, the network's prediction is given by the digit group that fires the most spikes.

Even though the input digits are represented using Poisson-distributed spike trains, the weight values shown in Fig. 11 demonstrate that the trained synaptic weights form digit-specific representations of the inputs. For example, it can be seen that the weights for the digit group 0 and 3 resemble handwritten 0 s and 3 s. The confusion matrix of the network's classification is shown in Fig. 12. It is noted that the network is able to accurately classify the handwritten digits with the classification accuracy of 89.8%. Several previously published works have used supervised spike-based methods for training SNNs using the MNIST dataset. A restricted boltzmann machine (RBM) was realized using the leaky I &F neurons in [39] and achieved a classification accuracy of 91.9%. In [40], a spiking convolutional network was implemented using the leaky I &F neurons and achieved a 91.3% classification accuracy. In [41], the Izhikevich neurons were used to implement two layers of visual system neurons (V1 and V2) followed by a classification layer. In total, the network architecture consists of over 71,000 neurons with over 130 million synapses and achieved a classification accuracy of 91.6%. In [42], a two-layer network of stimuli-selective leaky I &F neurons were used to make predictions regarding the particular stimulus presented, and achieved a 96.5% classification accuracy. Unfortunately, the state-of-the-art SNN-based MNIST classifiers offer a lower

Table 1 The neuron model and learning parameters for Izhikevich-based SNN digit recognition

Neuron parameters		Learning parameters	
Parameter	Value	Parameter	Value
a	0.02	μ	0.0001
b	−0.1	ϵ	10 ms
c	− 55	β	300 Hz
d	6		

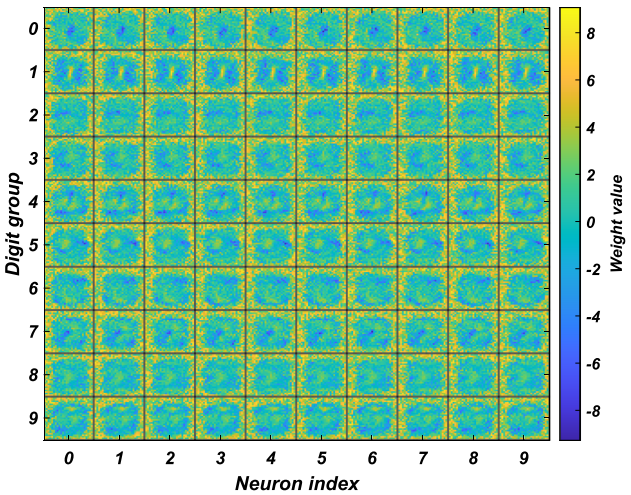

Fig. 11 Weight matrices of the trained Izhikevich SNN

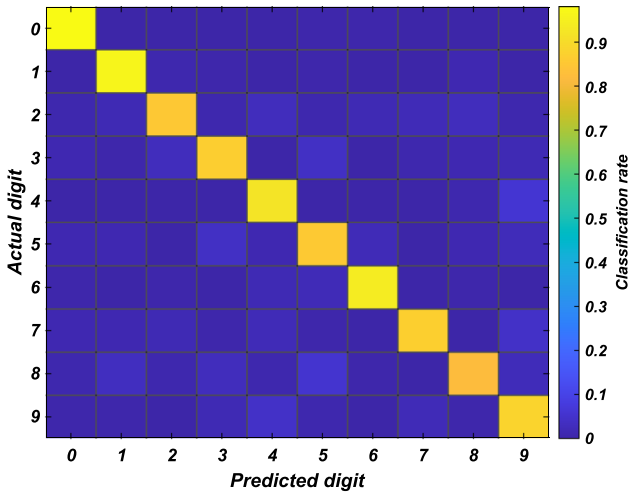

Fig. 12 Confusion matrix for the SNN-based MNIST digit recognition task

classification accuracy compared to the non-spiking convolutional neural network models [43]. The accuracy might be improved by employing more layers, increasing the amount of abstract features that can be learned, incorporating conductance-based dynamics into the synaptic weights, and employing additional pre-processing layers that perform edge-detecting [42] and orientation-detecting [40, 41].

After obtaining input weights, trained weights can be sent to the network by performing the weighted sums of spike inputs with their respective weight parameter. For the MNIST digit recognition task, for a given time step, the weights from inputs that spike at time t can be accumulated into a single external input I, based on Eq. (1), thus providing weighted spike train input to the network.

4.4 Comparison

The implementation of a hardware dedicated to emulating SNNs has been of interest in recent years. There are two overarching implementation platforms in use today: (i) programmable devices, including FPGAs and CPUs; and (ii) ASICs. Below, we discuss several state of the art implementations for the two implementation platforms.

4.4.1 Programmable devices

FPGAs offer extensive flexibility to emulate SNNs with various network topologies and spiking behaviors. Table 2 gives the characteristics and FPGA implementation results of various state-of-the-art Izhikevich-based SNN hardware architectures. The SpiNNaker project [24] utilizes general-purpose processors to implement SNNs using a distributed computing approach, where a processing node incorporates 18 ARM968 processors and a combination of processing nodes are used concurrently for simulating SNNs. SNNs are first simulated in software using the PyNN [44] or Nengo [45] frameworks, and then mapped to the SpiNNaker nodes. The distributed computing approach is ideal for simulating large-scale SNNs. While utilizing an array of programmable processors offers a high degree of flexibility, the time resolution scales down to only one ms, which makes it impractical for real-time operation. Moreover, while up to 1000 SNs can be simulated per node, the level of connectivity among neurons is not realistic, with a fan-out of only 100 outputs per neuron. The work in [21] presents NeuroFlow, a scalable distributed computing SNN emulation platform using FPGA-based processors. A NeuroFlow system using six FPGAs can simulate SNNs of up to 600K neurons. The time resolution of one millisecond can be achieved for SNNs of up to 400K neurons. While it is reasonable to compare NeuroFlow and SpiNNaker platforms, as both are targeting large-scale SNNs, our design primarily focuses on a generalized hardware architecture for moderately-sized SNNs with arbitrary interconnect structures and with programmable time resolutions for real-time operation.

Dedicated reconfigurable hardware realization of SNNs has also received interest [18–23], many of which have focused on accelerating the emulation of large-scale SNNs. The work in [18] presents an FPGA-based implementation of an Izhikevich-based SNN, which supports a fully-connected network of 1024 neurons with over one million synapses. The design uses 16 dedicated synaptic accumulation units, each of which can perform the accumulation of four synapses. The network operations are performed using either single- or double-precision floating-point units and their design operates at 133 MHz with the reported timing resolution of 10μs. The FPGA implementation of fully-connected SNNs in [19] supports relatively small networks with only 117 neurons, operating at 84 MHz, while achieving a one millisecond time resolution. The work in [20] presents an Izhikevich-based SNN implementation on a Kintex-7 FPGA, capable of simulating over 1000 neurons with a time resolution of 0.1 ms. The work in [22] implements HH-based neurons in a set of tightly integrated cores, supporting up to 500 neurons each, on a Kintex-7 FPGA. Unfortunately, the total memory utilization in terms of number of BRAMs was not reported, which may explain their relatively low register and LUT usage. The work in [23] also presents a core-based approach for implementing HH-based neurons on an Artix-7 FPGA. Rather than supporting a fully-connected network, their work focuses on randomly connected networks, where the connectivity among neurons is generated off-line on a workstation and encoded into an initial seed vector. The seed vector is then used to generate the connectivity matrix on-the-fly instead

Table 2 The characteristics and FPGA implementation results for various SNN realizations

Work	Neuron model	FPGA device	Number of neurons	Synapses per neuron	Time resolution (ms)	Regs. (%)	LUTs (%)	BRAMs (%)	DSP48s (%)	Freq. (MHz)
[18]	IZH	Virtex-5	1024	1024	0.01	32420 (15)	19397 (9)	264 (81.5)	16 (8.3)	133
[19]	IZH	Virtex-4	117	117	1	970 (2)	1598 (3.3)	8 (2.5)	1 (0.19)	84
[20]	IZH	Virtex-6	1440	1440	0.1	48502 (16)	55884 (37)	392 (94)	408 (53)	–
[22]	HH	Kintex-7	500	500	0.00002	2360 (0.57)	5551 (2.7)	–	28 (3.33)	100
[23]	HH	Artix-7	4096	4096	0.0078	25430 (9)	46045 (34)	20 (5.4)	280 (37)	58.8
Ours	IZH	Artix-7	1000	1000	0.0625 – 0.5	11139 (4.1)	12700 (9.49)	14.5 (3.9)	110 (14.8)	92

of storing the full-connectivity on the FPGA. As given in Table 2, our SNN hardware architecture is among the most resource efficient designs when implementing similarly sized SNNs, yet being able to realize a range of time resolutions and hence, provides a viable FPGA-based real-time emulation platform. Utilizing a relatively small percentage of on-chip reconfigurable resources along with scalability of the designed hardware architecture makes it suitable for emulating moderately-sized SNNs on a single FPGA.

While the implemented design is more area efficient than the state-of-the-art FPGA-based SNNs, its general-purpose architecture also offers several important design trade-offs. In addition to supporting a "variable" number of neurons (in the order of tens of thousands), it also allows an efficient realization of an "arbitrary" interconnect structure between neurons in the network, which normally requires a relatively large synaptic weight memory. Current FPGA-based SNNs manage the size of the weight memory by only realizing a particular subset of interconnections for a specific application. Another tradeoff is that the designed and implemented SNN architecture supports a wide range of temporal resolutions for added flexibility, while current SNNs designs would support optimized computation of membrane voltages using a fixed time resolution.

4.4.2 ASIC realizations

ASIC realizations of SNNs include mixed-signal neuromorphic chips [25, 46, 47], which mimic biologically-plausible electrical behavior of neurons [48], and programmable digital SNN implementations [28–30]. The former designs generally offer lower power consumption and require smaller silicon area [49], whereas the latter designs can leverage the programmability of processors for increased flexibility. To compare our proposed generalized SNN hardware architecture with the state-of-the-art ASIC realizations, our designed SNN architecture is implemented in a standard 32-nm CMOS technology. The CMOS layout of the implemented ASIC is shown in Fig. 13, which is estimated to occupy 3.6 mm^2 of silicon area. Post-layout synthesis and simulation results show that the ASIC chip will dissipate 3.6 mW from a 1.16 V supply while operating at 34.7 MHz frequency.

Table 3 gives the characteristics and ASIC implementation results of various SNN designs. The work in [25] presents a mixed-signal SNN implementation in a standard 180-nm CMOS technology, supporting up to 256 neurons with 16K synapses. The spiking neurons are realized based on the AdExp neuron model [51]. The digital neuron implementation, often referred to as silicon neurons, mimic the behavior of the AdExp neuron with analog circuits. While the AdExp neuron model can exhibit various spiking

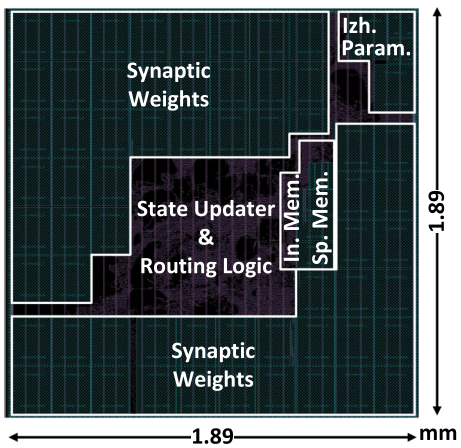

Fig. 13 Layout of the implemented SNN hardware architecture in a standard 32-nm CMOS process

Table 3 The characteristics and ASIC implementation results for various SNN realizations

Work	[25]	[26]	[27]	[28]	[29]	[30]	Ours
Circuit	Mixed-Signal	Mixed-Signal	Digital	Digital	Digital	Digital	Digital
CMOS process (nm)	180	180	45	28	28	14	32
No. spiking dynamics	20[1]	20[1]	3	20	11	6	20
Neurons per core	256	256	256	256	256	1024	256
Synapses per core	16K	128K	64K	64K	64K	1 M – 114K	64K
Weight storage	12-bit CAM	Capacitor	1-bit SRAM	3-bit SRAM + 1-bit SRAM	1-bit SRAM	(1-9)-bit SRAM	16-bit SRAM + Izh. Param.
Voltage (V)	1.3 – 1.8	1.8	0.53 – 1.0	0.55 – 1.0	0.7 – 1.05	0.5 – 1.25	1.16
Die area per core (mm^2)	9.6	51.4	4.2	0.086	0.095	0.4	3.6
Power consumption (mW)	0.2 - 2.7	4	0.5	0.4	64	4	3.6
Scaled area per core (mm^2)[2]	0.24	1.28	2.9	0.12	0.13	–	3.6
Scaled power consumption (mW)	0.17-−0.85	1.25	0.62-−1.88	0.50-−1.95	73–312	–	3.6

[1]The AdExp neuron has not yet been proven to exhibit all 20 spiking dynamics

[2]Scaled to a 32-nm CMOS process utilizing a 1.16V supply voltage following the scheme in [50]

dynamics, it has not yet been shown to reproduce all of the 20 spiking behaviors modeled by Izhikevich's description [52]. The processing units store the address of the post-synaptic spike in a 10-bit content addressable memory (CAM) and the parameters for the desired dynamics are stored in a 2-bit SRAM. The SNN presented in [26] also implements the AdExp model with silicon neurons in a standard 180-nm CMOS technology. The synaptic weights are stored in on-chip capacitors in the array of synapses. The work in [27] implements a SNN consisting of 256 neurons, based on the I &F neuron model, and 64K synapses in a standard 45-nm CMOS technology. As opposed to the previous approaches in which a neuron's operations are performed using analog circuits, the computations in [27] are performed digitally. The binary synaptic weights are stored in an array of SRAM cells within the cross-bar structure of synapses. The work in [28] presents a digital 256-neuron, 64K-synapse neuromorphic processor. It is implemented in a 28-nm CMOS technology and employs the time-multiplexing technique to emulate the cross-bar structure of synapses. IBM's TrueNorth [29] and Intel's Loihi [30] are also two digital implementations of SNNs. TrueNorth is a multi-core programmable processor implemented in a 28-nm CMOS technology, where each core contains 256 spiking neurons and 64K synapses. Loihi is also a multi-core programmable processor, implemented in a 14-nm CMOS technology, where each core supports 1024 neurons with up to one million synapses.

While commercial SNN processors and distributed computing solutions in [21] and [24] focus on very large-scale SNNs, our architecture and those in [25–28] are single-chip realizations for a moderate network size. Even though the operating frequency of our design is relatively slow, it is sufficient for real-time emulation of SNNs. For a fair comparison, we have scaled the silicon area and power consumption to a 32−nm CMOS process utilizing a 1.16-V supply voltage as presented in [50]. We can see that the digital implementations presented in [28–30] occupy smaller silicon areas than ours, since our design naturally requires more storage elements to define the spiking dynamics on a per-neuron basis. The Izhikevich parameters a and b are stored in 12-bit SRAM units, the c values are stored in 32-bit SRAM units, and the d values are stored in 16-bit SRAM units. While the discussed SNN designs utilize a cross-bar synaptic structure for communication among neurons, the accumulation of synaptic weights in our design is performed in a hybrid sequential-parallel approach with an adjustable degree of parallelism. The synaptic weights are stored in 16-bit SRAM units and for a realistic level of interconnection, the synaptic weight memory may occupy a relatively large silicon area. Note that while our generalized design allows storing the Izhikevich parameters for each individual neuron, to substantially reduce the memory requirement, the Izhikevich parameters $a - d$ can be constrained to specific values for a group or all of the neurons in the employed SNN. Similarly, in [16], seven types of neurons are specified, each with a predefined set of parameter values for $a - d$. Our

approach differs in that we aim to provide a robust and flexible platform for emulating diverse SNNs.

While it is shown that our digital SNN architecture consumes acceptable power, recent analog implementations offer a potential alternative candidate. As discussed in Sect. 3, one of the challenges for the efficient realization of spiking neural networks (SNNs) is the process of accumulating synaptic weights. Neuromorphic architectures employ crossbar interconnects to accumulate weighted inputs through current flow. Also, instead of storing weight parameters in SRAM cells, they utilize memristor-based memory, which has a higher density and lower power consumption [53]. They are also suitable passive elements for modeling neuronal synapses with biological learning rules, such as spike-time dependent plasticity [54–57]. Therefore, in terms of area, power consumption, as well as scalability, the neuromorphic architecture could offer a viable alternative. However, while analog processing provides higher speed than digital processing, unfortunately, noise and inaccuracies accumulate in analog systems. Digital systems can reliably process data and tolerate noise and inaccuracies during neural signal processing. For brain-computer interfaces where the speed of neural signal recording and processing is far less than the speed of digital integrated circuits, low-power realization of spiking neural networks in the digital domain is a feasible alternative candidate.

5 Conclusion

This article presented an area- and power-efficient scalable reconfigurable hardware architecture for real-time emulation of spiking neural networks (SNNs). We presented the characteristics and implementation results of our designed generalized SNN hardware architecture on both field-programmable gate arrays (FPGAs) and on application-specific integrated circuit (ASIC). Compared to the similarly-sized SNNs implemented on a single FPGA, our design is capable of emulating moderately-sized SNNs in real time while utilizing significantly fewer reconfigurable resources. We also verified the performance of the proposed SNN on the MNIST digit recognition task and showed that it can accurately classify handwritten digits with 89% accuracy. Compared to the state-of-the-art ASIC realizations of SNNs, our design consumes less power utilizing a hybrid parallel-sequential scheme. Moreover, as opposed to the state-of-the-art ASIC realizations, our design's power consumption and real-time processing are not affected by the spiking rate of the emulated SNN. Our design supports various spiking rates based on the number of neurons and allows an adjustable degree of parallelism and time step resolution. The silicon area and low power consumption of the realized generalized SNN hardware architecture make it suitable for single-chip SNN realizations while emulating SNNs of various sizes and interconnect topologies in real-time.

Acknowledgements This work was supported by the Center for Neurotechnology (CNT), a National Science Foundation (NSF) Engineering Research Center (EEC-1028725) and the NSF Award #2007131.

Availability of data and material The MNIST datasets used during this study are available at http://yann.lecun.com/exdb/mnist/.

Declarations

Conflict of interest The authors have no competing interests to declare.

References

1. National Spinal Cord Injury Statistical Center, Facts and figures at a glance, f Alabama at Birmingham, (2016)
2. Yin X-X, Sun L, Fu Y, Lu R, Zhang Y (2022) U-net-based medical image segmentation. J Healthcare Eng 15:2022
3. Lea C, Vidal R, Reiter A, Hager GD (2016) Temporal convolutional networks: a unified approach to action segmentation, in : European Conference on Computer vision
4. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN , Kaiserl, Polosukhin I (2017) Attention is all you need, Adv Neural Inform Process Syst, 30
5. Raganato A, Tiedemann J (2018) An analysis of encoder representations in transformer-based machine translation, in: EMNLP Workshop BlackboxNLP: analyzing and interpreting neural networks for NLP
6. Maass W (1997) Networks of spiking neurons: the third generation of neural network models. Neural Netw 10(9):1659–1671
7. Bonifazi P et al (2013) In vitro large-scale experimental and theoretical studies for the realization of bi-directional brain-prostheses. Front Neural Circuit 7:40
8. Nawrot A, Pistohl T, Schrader S, Hehl U, Rodriguez V, Aertsen A (2003) Embedding living neurons into simulated neural networks, in: IEEE Conference on Neural Engineering, pp. 229–232
9. Zbrzeski A et al (2016) Bio-inspired controller on an FPGA applied to closed-loop diaphragmatic stimulation. Front Neurosci 10:275
10. Mosbacher Y, Khoyratee F, Goldin M, Kanner S, Malakai Y, Silva M, Grassia F, Simon YB, Cortes J, Barzilai A et al (2020) Toward neuroprosthetic real-time communication from in silico to biological neuronal network via patterned optogenetic stimulation. Sci Rep 10(1):1–16
11. Buccelli S, Bornat Y, Colombi I, Ambroise M, Martines L, Pasquale V, Bisio M, Tessadori J, Nowak P, Grassia F et al (2019) A neuromorphic prosthesis to restore communication in neuronal networks. IScience 19:402–414
12. Berger T et al (2011) A cortical neural prosthesis for restoring and enhancing memory. J Neural Eng 8(4):046089
13. Berger T et al (2012) A hippocampal cognitive prosthesis: multi-input, multi-output nonlinear modeling and vlsi implementation. IEEE Transact Neural Syst Rehabilitat Eng 20(2):198–211
14. Abbott LF (1999) Lapicque's introduction of the integrate-and-fire model neuron. Brain Res Bullet 50(5):303–304

15. Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117(4):500–544
16. Izhikevich EM (2003) Simple model of spiking neurons. IEEE Transact Neural Netw 14(6):1569–1572
17. Izhikevich EM (2004) Which model to use for cortical spiking neurons? IEEE Transact Neural Netw 15(5):1063–1070
18. Luk W, Thomas D (2009)FPGA accelerated simulation of biologically plausible spiking neural networks, in: IEEE Symposium on field programmable custom computing machines, pp. 45–52
19. Ambroise M, Levi T, Bornat Y, Saighi S (2013) Biorealistic spiking neural network on FPGA, in: IEEE information sciences and systems, pp. 1–6
20. Pani D et al (2017) An FPGA platform for real-time simulation of spiking neuronal networks. Front Neurosci 11(90):1–13
21. Cheung K, Schultz SR, Luk W (2016) NeuroFlow: a general purpose spiking neural network simulation platform using customizable processors. Front Neurosci 9(516):1–15
22. Khoyratee F, Grassia F, Saïghi S, Levi T (2019) Optimized real-time biomimetic neural network on FPGA for bio-hybridization. Front Neurosci 13:377
23. Akbarzadeh-Sherbaf K, Abdoli B, Safari S, Vahabie A-H (2018) A scalable FPGA architecture for randomly connected networks of hodgkin-huxley neurons. Front Neurosci 12:698
24. Furber SB, Galluppi F, Temple S, Plana LA (2014) The spinnaker project. Proceed IEEE 102(5):652–665
25. Moradi S, Qiao N, Stefanini F, Indiveri G (2017) A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). IEEE Transact Biomed Circuits Syst 12(1):106–122
26. Qiao N, Mostafa H, Corradi F, Osswald M, Stefanini F, Sumislawska D, Indiveri G (2015) A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. Front Neurosci 9:141
27. Seo JS, *et al.* (2011) A 45 nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons, in: IEEE Custom Integrated Circuits Conference
28. Frenkel C et al (2019) A 0.086-mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos. IEEE Transact Biomed Circuits Syst 13(1):145–158
29. Akopyan F et al (2015) Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. IEEE Transact Comput-Aided Des Integrat Circuits Syst 34(10):1537–1557
30. Davies M et al (2018) Loihi: A neuromorphic manycore processor with on-chip learning. IEEE Micro 38(1):82–99
31. Pei J, Deng L, Song S, Zhao M, Zhang Y, Wu S, Wang G, Zou Z, Wu Z, He W et al (2019) Towards artificial general intelligence with hybrid tianjic chip architecture. Nature 572(7767):106–111
32. Wells RB (2005) "Cortical neurons and circuits: a tutorial introduction [online]. [cit. 2008-02-05]," URL: http://www.mrc.uidaho.edu/~rwells/techdocs/CorticalNeuronsandCircuits.pdfd
33. Parhi KK, Wang C-Y, Brown AP (1992) Synthesis of control circuits in folded pipelined dsp architectures. IEEE J Solid-State Circuit 27(1):29–43
34. Kocaturk M, Gulcur HO, Canbeyli R (2015) Toward building hybrid biological/in silico neural networks for motor neuroprosthetic control. Front Neurorobot 9:8
35. Tsai D, Sawyer D, Bradd A, Yuste R, Shepard KL (2017) A very large-scale microelectrode array for cellular-resolution electrophysiology. Nat Commun 8(1):1802
36. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proceed IEEE 86(11):2278–2324
37. Diehl PU, Cook M (2015) Unsupervised learning of digit recognition using spike-timing-dependent plasticity. Front Comput Neurosci 9:99
38. Tavanaei A, Maida A (2019) Bp-stdp: Approximating backpropagation using spike timing dependent plasticity. Neurocomputing 330:39–47
39. Neftci E, Das S, Pedroni B, Kreutz-Delgado K, Cauwenberghs G (2014) Event-driven contrastive divergence for spiking neuromorphic systems. Front Neurosci 7:272
40. Zhao B, Ding R, Chen S, Linares-Barranco B, Tang H (2014) Feedforward categorization on aer motion events using cortex-like features in a spiking neural network. IEEE Transact Neural Netw Learn Syst 26(9):1963–1978
41. Beyeler M, Dutt ND, Krichmar JL (2013) Categorization and decision-making in a neurobiologically plausible spiking network using a stdp-like learning rule. Neural Netw 48:109–124
42. Brader JM, Senn W, Fusi S (2007) Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. Neural Comput 19(11):2881–2912
43. An S, Lee M, Park S, Yang H, So J (2020) "An ensemble of simple convolutional neural network models for mnist digit recognition," arXiv preprint arXiv:2008.10400
44. Davison AP, Brüderle D, Eppler JM, Kremkow J, Muller E, Pecevski D, Perrinet L, Yger P (2009) Pynn: a common interface for neuronal network simulators. Front Neuroinform 2:11
45. Stewart TC, Tripp B, Eliasmith C (2009) Python scripting in the nengo simulator. Front Neuroinform 3:7
46. Hofstötter C, *et al.* (2005) The cerebellum chip: an analog VLSI implementation of a cerebellar model of classical conditioning, in: Advances in neural information processing systems, pp. 577–584
47. Park J, Ha S, Yu T, Neftci E, Cauwenberghs G (2014) A 65k-neuron 73-mevents/s 22-pj/event asynchronous micro-pipelined integrate-and-fire array transceiver,in: IEEE BioCAS Proceedings, pp. 675–678
48. Mead C (1989) Analog VLSI and neutral systems, NASA STI/Recon Tech Rep A, 90
49. Joubert A, Belhadj B, Temam O, Héliot R (2012) Hardware spiking neurons design: Analog or digital? in: IEEE International Joint Conference on Neural Networks, pp. 1–5
50. Stillmaker A, Xiao Z, Baas B (2011)Toward more accurate scaling estimates of cmos circuits from 180 nm to 22 nm,VLSI Computation Lab, ECE Department, University of California, Davis, Tech. Rep. ECE-VCL-2011-4, 4, p. m8
51. Brette R, Gerstner W (2005) Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. J Neurophysiol 94(5):3637–3642
52. Naud R, Marcille N, Clopath C, Gerstner W (2008) Firing patterns in the adaptive exponential integrate-and-fire model. Biol Cybernet 99(4–5):335
53. Pal S, Gupta V, Ki WH, Islam A (2019) Design and development of memristor-based rram. IET Circuits, Dev Syst 13(4):548–557
54. Chu M, Kim B, Park S, Hwang H, Jeon M, Lee BH, Lee B-G (2014) Neuromorphic hardware system for visual pattern recognition with memristor array and cmos neuron. IEEE Transact Ind Electron 62(4):2410–2419
55. Shukla A, Ganguly U (2018) An on-chip trainable and the clockless spiking neural network with 1r memristive synapses. IEEE Transact Biomed Circuits Syst 12(4):884–893
56. Chen B, Yang H, Zhuge F, Li Y, Chang T-C, He Y-H, Yang W, Xu N, Miao X-S (2019) Optimal tuning of memristor conductance variation in spiking neural networks for online unsupervised learning. IEEE Transact Electron Dev 66(6):2844–2849
57. Zheng N, Mazumder P (2018) Learning in memristor crossbar-based spiking neural networks through modulation of weight-

dependent spike-timing-dependent plasticity. IEEE Transact Nanotechnol 17(3):520–532