Digital Spiking Neuron Cells for Real-Time Reconfigurable Learning Networks

Haipeng Lin, Amir Zjajo, Rene van Leuken
Circuits and Systems Group
Delft University of Technology
Delft, The Netherlands

***Abstract*—The high level of realism of spiking neuron networks and their complexity require a substantial computational resources limiting the size of the realized networks. Consequently, the main challenge in building complex and biologically-accurate spiking neuron network is largely set by the high computational and data transfer demands. In this paper, we implement several efficient models of the spiking neurons with characteristics such as axon conduction delays and spike timing-dependent plasticity. Experimental results indicate that the proposed real-time data-flow learning network architecture allows the capacity of over 2800 (depending on the model complexity) biophysically accurate neurons in a single FPGA device.**

Keywords—Digital spiking neuron cells, neuron network, learning network, real-time data-flow architecture.

I. INTRODUCTION

The spiking neural networks SNNs (SNNs) [1]-[2] replicate the dynamic behaviors and information processing mechanisms of a biological neural system [3], and exhibit temporal pattern processing [4] and fault-tolerant capabilities [5]. Subsequently, the main challenge in designing complex and biologically-accurate SNNs is primarily set by the high data transfer and computational demands. Nevertheless, it is only through large-scale networks and/or real-time simulation that biological dynamics for specific experiments e.g. brain-machine interfaces, can suitably be modeled. Execution of such networks on CPUs with generic programming suites or neuro-modeling-specific languages, however, require a excessive amount of time to complete. Eventhough slower than custom-made ASICs, field-programmable gate arrays (FPGAs) are, due to the inherent high-parallelism, capable of providing sufficient performance for real-time and even hyperreal-time neuron network simulations. In addition, via (partial) reconfigurations of the hardware, various network topologies as well as numerous neuron models, (e.g. Izhikevich [6]-[7], integrate and fire (IaF) [8] model and its extensions such as the leaky IaF, IaF-or-Burst [9], quadratic IaF [10], or Hodgkin-Huxley (HH) [11]-[12], simplified Hodgkin-Huxley [13], *extended* Hodgkin-Huxley [14]), can be simulated. This flexibility is substantially enhanced by the use of high-level synthesis tools, which speed up the development process.

In this paper, we implement several models of the spiking neurons representing different trade-offs between the biophysical accuracy and computation complexity. The models with characteristics such as axon conduction delays, gap junctions [14], spike timing-dependent plasticity (STDP) [15]-[16], electrochemical state descriptions [17], etc. are implemented in a real-time data-flow learning network. The input is localized for each neuron cell, i.e., multiple packets can be transmitted to the specific neuron cells at the same cycle. Concurrently, the parameters are also localized for each cell. Additionally, the implemented system offers configurable on- and off-chip communication latencies as well as neuron calculation latencies.

This research was supported in part by the European Union and the Dutch government, as part of the CATRENE program under Heterogeneous INCEPTION project.

II. SPIKING NEURON CELLS

A. Hodgking-Huxly Neuron Cell Model

The model based on experimental findings in [17] (Fig. 1) implements a neuron with three distinct compartments: the axon, the soma, and the dendrite. Operationally, the neuron network needs to compute and communicate simulated neuron cell responses to their neighbors and the axon. We run both operation concurrently, and devise separate hardware architectures for computation (based on the multi-compartmental *extended* HH [14]), and communication requirement. We refer to a neuron computation unit as a physical cell (*PhC*)[1]. Within a *PhC*, topology-independent *Axon+Soma* (in cycles) computation, and the topology-dependent dendrite computation operate in parallel [18].

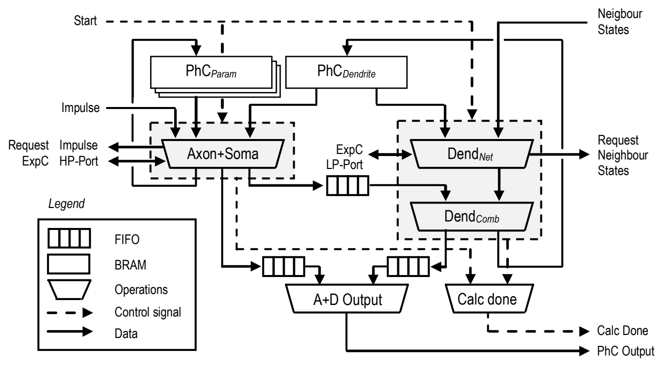

Fig. 1: Dataflow of an extended Hodgkin-Huxley model [18].

[1] The computation units are called physical cells (*PhCs*) to recall that they are physically implemented in hardware.

B. Integrate and Fire Neuron Cell Model

In general, due to the efficient computation, the integrate and fire model is applicable for simulation of large amount of neural cells in the network if the biophysically meaningful features of the neuron cell are not an issue. A data flow of integrate and fire neuron cell model is shown in Fig. 2. The inputs are in the form of the current, consisting of the spikes from the neighbors, and the external stimuli. All the inputs are added together to derive a total current. This current is used in the potential function to calculate a new membrane potential V_{mem} together with parameters and current membrane potential, and one parameter is updated in the memory of parameter. Afterwards, this new $V_{mem(new)}$ is compared with the threshold potential. If the V_{mem} exceeds the threshold, a new spike will be generated, and the V_{mem} and one parameter will be reset and stored in the local memory, respectively. Additionally, the V_{mem} will be written back to its local memory, waiting for the next cycle. Finally, the spike is transmitted to the neighbors

C. Izhikevich Neuron Cell Model

The Izhikevich model is usually used to examine the patterns of the spike trains. In the proposed system, the implementation of the Izhikevich model can be subdivided into three parts: axonal conduction delay, STDP, and spike generation.

Axonal Conduction Delay: The axonal conduction delays can vary greatly (depending on the axonal velocity and distance, respectively), and can be as large as 44 ms and as small as 0.1 ms [19]. The implementation of axonal conduction delay includes three basic functions: f_{delay} (defines the exact delay time for each arriving spike), f_{check} (checks whether spikes arrive within the predefined time step), and $f_{current}$ (generates the current). The dataflow is shown in Fig. 3. At each step, firstly all inputs from neighbors are compared. The delay information is stored in a local memory, which records the delay time for each neighbor. If the spikes are accepted, the f_{delay} attach delay information to these spikes, and store them in a buffer. The spike generation and packets arrival occur at the same cycle. Accordingly, an additional delay is placed in the module to enable readout of the spikes at real-time.

We assume that the spikes from neighboring neurons depend on the pre-synapse weights; consequently, the (delayed) spike buffer only need to record the real arriving time for each pre-synaptic neuron.

STDP: The implementation of STDP [15] refers to several important variables, e.g. long-term potentiation (LTP) [20], long-term depression (LTD) [21], pre-synapse weight, pre-synapse derivatives, post-synapse weight, and post-synapse derivatives. We add a global memory in the general structure of network to dynamically update STDP parameters. When a single (Izhikevich) neuron cell receives the spike from a specific pre-synaptic neighbor, the pre-synapse derivative will be depressed by the variable LTD. Next, the new parameter value of STDP is updated in the global memory, so that the corresponding pre-synaptic neuron is informed immediately of the change of its post-synapse derivative (the synaptic weight of the pre-neuron equals the synaptic weight of the post-neuron). In case a spike is generated in the neuron itself, the variables LTD and LTP are reset. If no spikes occur, all variables will decay according to the STDP rule.

Spike Generation: In this independent module, the new membrane potential is calculated based on the external current I_{app}, pre-synaptic weight, and delays. If new membrane potential is lower than threshold, no spike is generated, and the parameters of STDP decay proportionally to the f_{decay}. Consequently, the membrane potential and parameters are reset to the predefined constant values, and the new data are fed back to the corresponding local memories.

D. The System Overview

In the implemented architecture (Fig. 4) [22], the neuron cells are connected with decreasing probability the further they are apart [1]. The cells placed near to each other in the network are connected within the (neighbor) cluster. Each cluster consists of several individual computation units, i.e. physical cells, which are positioned around a shared memory for storing all the communication data needed by the *PhC*s. In the clusters, configurable routing tables define how *PhC*s are arranged within the neuron network.

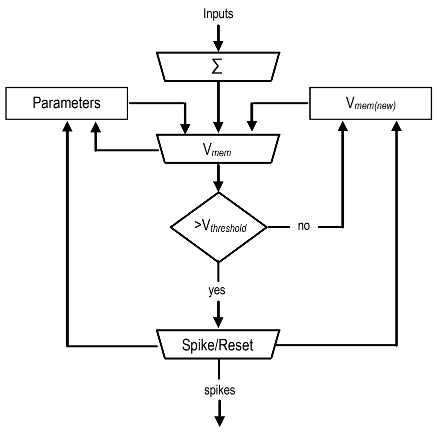

Fig. 2: Dataflow of an integrate and fire model.

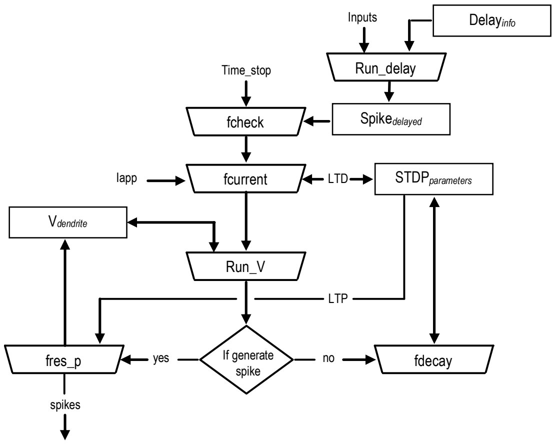

Fig. 3: Dataflow of an Izhikevich model

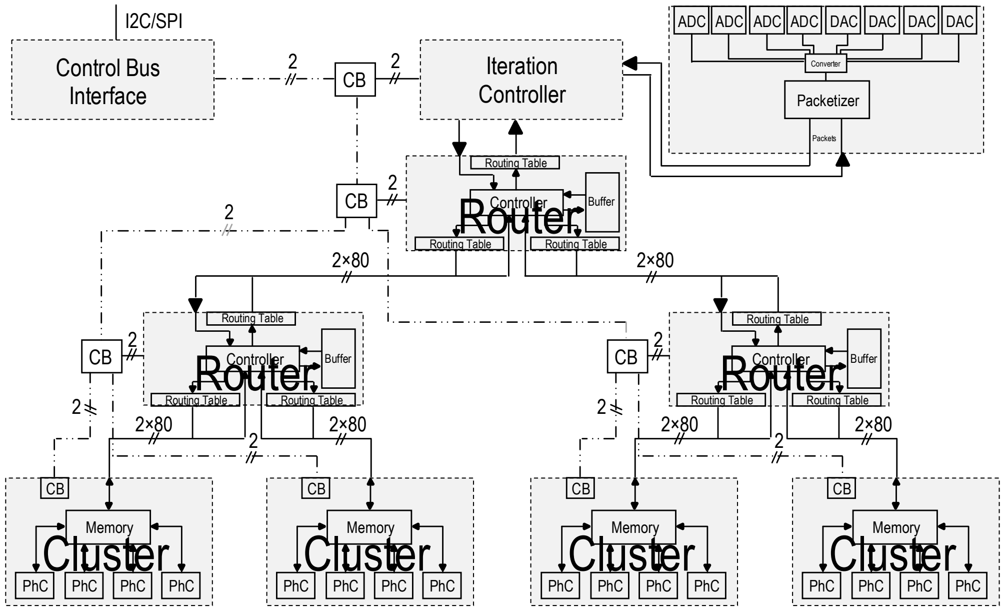

Fig. 4: The system overview. The computing elements (the *PhCs*) are grouped inside a cluster to make communication between neighboring cells fast. These clusters are connected in a tree topology NoC. The router fan-out in this case is 2, and can be changed according to the requirements of the implementation. The same holds true for the number of *PhCs* in any cluster [22].

By attaching each cluster to a binary tree network, responses between *PhC*s are shared. Furthermore, through the top node of the tree network a current impulse can be applied to all *PhC*s, and all output results of the neuron network streamed.

Input and Output: The interface connecting to external environment only focus on the value of signal and data format; the corresponding digital signal is assigned with identification number and transmitted to destination. In order to handle multiple external stimuli simultaneously, the interface is capable of localizing input for each neuron cell. In each simulated time cycle, the interface can deal with multiple external sources and send these packets one by one to destinations. Routers in the network utilize the look-up routing tables to forward the receiving packets. The cluster containing destination of neuron cell accept the packet and store it in shared memory; the corresponding neuron cell can read it before calculation.

Parameters: Sets of parameters are locally stored in the memory of each (independent) neuron module. Consequently, the system can be efficiently set to specific configuration without rewriting configuration file each time. Additionally, since neuron cell read or update parameters on each cycle, localization of storing parameters significantly reduce latency, in particular a communication latency. For each neuron cell, the parameters are set individually (a set of parameters represents various states of neuron cell and different neuron cells have different values of the parameters). The parameters are initialized with random function to generate unique values of parameters within each neuron cell. The individual value of parameters generated by functions is in a specific range, whose max and min values are pre-defined in configuration file.

Network Scalability: Several FIFO buffers are designed in the router to handle congestion. Each input or output port has a corresponding write- or read-buffer. Once write buffers are full, the following packets will be stored in the delayed buffer first. As soon as the write buffer is empty, the packets in the delayed buffer will be picked up and forwarded to destinations via write buffer. In order to simplify the design, all packets are defined with same size and the same priority in the system. Hence, there is no need to split packets to several flits. The number of the neuron cells, the clusters and number of time-shared cells in one *PhC* are pre-defined in configuration file. Once system starts, it firstly initializes the structure of network based on parameters of the system. A recursive function is designed to generate new branch from root node, until the number of leaf nodes is equal or larger than the pre-defined parameter.

*Synchronization Between Cluster*s: Communications within a cluster is designed to be much faster than communications among clusters. Hence, a cluster with inter-cluster communications completes a calculation slower. We synchronize the clusters with a special packet routed to the root of the tree, where it is counted, i.e. when done packets have arrived at the root.

Adjustments to the Network to Scale over Multiple FPGAs: As the hardware resources are limited on any FPGA, the maximum number of neuron cells in the network is always limited. To overcome this constraint, the proposed architecture is designed in such a way that the network can be ported over multiple FPGAs with ease. Since the communication frequency decreases closer to the root of the network tree, multiple FPGAs can be connected at the highest level without significant impact on performance.

However, while applying the tree topology on multi-FPGA systems and only adding another tree layer would offer easy extendibility, the limited connection options of each FPGA (and need for an additional FPGA for routing between the FPGAs containing the clusters), severely restrict their capabilities. Consequently, since neighboring FPGAs communicate the most, the FPGAs are connected in a ring topology, which is less complex in terms of topology generation and administration of the routing tables [22].

III. Model Configuration

To find an optimal design, first the limit is given on the total number of implementable physical cells $Total_{PhC}$ in FPGA, based on the available critical resources (1). The maximal number of clusters φ in the system depends on required accuracy. After dividing $Total_{PhC}$ by φ, the amount of physical cells per cluster (*PPC*) can be determined as

$$Total_{PhC} < \frac{\#CriticalResourse}{\#Resourse_{,} PhC} \quad (1)$$

$$PPC = [\#Total_{PhC} \times \frac{1}{\varphi}], \text{ where } \varphi \leq 5 \quad (2)$$

The latency of physical cells cannot exceed the real-time constraint[2]. For each neuron, the latency cycle C_{neuron} consists of two parts: calculation cycle C_{cal}, and communication cycle C_{com}

$$C_{neuron} = C_{cal} + C_{com} \quad (3)$$

The calculation cycle designates the time that neuron cell needs to generate the response, performs the calculations with the parameters, and update the results in the memory. The sending of the results to corresponding neighbors via the network is calculated in the communication cycles. The latency of the physical cell C_{PhC} is

$$C_{PhC} = TSF \times C_{cal} + C_{com} \quad (4)$$

where TSF is the time sharing factor for each physical cell. To improve the efficiency of system the C_{PhC} aims to be close to the real-time constraint, and consequently, larger number of the neuron cells can compute the responses within a given system period T_{system} (6). From (4), an upper bound of the time-share-factor can be calculated by considering the latency of *PhC*, the number of calculation cycles, and the communication cycles

$$C_{system} = \frac{T_{system}}{CLK_{period}} \quad (5)$$

$$C_{PhC} \leq C_{real-time} \leq C_{system}, \ where\ C_{real-time} = 50\mu s \quad (6)$$

$$TSF \leq \left\lfloor \frac{C_{PhC} - C_{com}}{C_{cal}} \right\rfloor \quad (7)$$

As a result, the total number or neurons implemented in the system is derived by

$$Total_{neurons} = \varphi \times PCC \times TSF \quad (8)$$

[2] The 'real-time' constraint is the maximum number of cell states that can be computed within the model (with a step time of 50 µs given by [14]).

The buffer depth is defined based on the layer where the corresponding router is placed in the tree network. Taking into considerations the number of physical cells per cluster and time-share factor, the maximal amount of packets in each cluster is

$$Num_{PhC,cluster} = N \times PPC \times TSF \quad (9)$$

where *N* is the maximal number of the connections to the neighbors. By adding the number of clusters in the design, the low bound on the buffer depth is given as

$$Depth_{downstream,ith} \geq 2^{i} \times N \times PPC \times TSF + Num_{other} \quad (10)$$

$$Depth_{upstream,ith} \geq 2^{i+1} \times N \times PPC \times TSF \quad (11)$$

IV. Experimental Results

All simulations are completed with cycle-accurate SystemC, including all computation and communication latencies, both on- or off-chip. To simulate the system behavior three different connection schemes are utilized; all-to-all connections (all cells are connected to all other cells), normal-distributed distance-based connections (probability based connections weighted by the distance between cells), and neighbor based connections (connects every cell to every neighboring cells, thus, resulting in 8 connections per cell).

The configuration file contains all the relevant parameters of the system and can be easily modified allowing exploration of different fan-out values, different cell communication schemes, etc. After the design is configured with the chosen accuracy (32/64-bit), it is synthesized through the Vivado HLS tool to generate VHDL code, and test bench files. To assess the proposed design, the synthesizable VHDL code is compiled with ModelSim, and the simulated axon voltages are compared to the reference C model [23]. Since the number of computational cycles is fixed (within a physical cell cluster) for a given topology, the hardware designs closest to real-time timing constraint are scaled-up by increasing the number of physical cell clusters in the tree network [18]. However, without routing tables in the tree network, all resulting potentials are sent in an all-2-all type fashion.

The neuron spiking properties are governed by the specific parameter sets: these properties have well-defined role in defining explicit brain functions, e.g. the cortical neurons with tonic bursting contribute to the gamma-frequency oscillations in the brain [24]. Fig. 5 and Fig. 6 illustrate several types of the neuron behavior in the simulated system. Similar patterns are found with biological test [25]. Most neurons are quiescent but can fire spikes when stimulated. Fig. 5a shows the typical firing of the integrate and fire model. When the pulses of the current are injected at the input, the neuron fire a train of spikes, the process called tonic spiking [26] (Fig. 5b). If such neurons fire continuously, it indicates that persistent input is offered to the neurons. A specific neuron could fire only a single spike at the onset of the input, and could subsequently stay quiescent, i.e. a response called phasic spiking illustrated in Fig. 5c. Specific neurons fire periodic bursts of spikes when stimulated, as shown in Fig 6a.

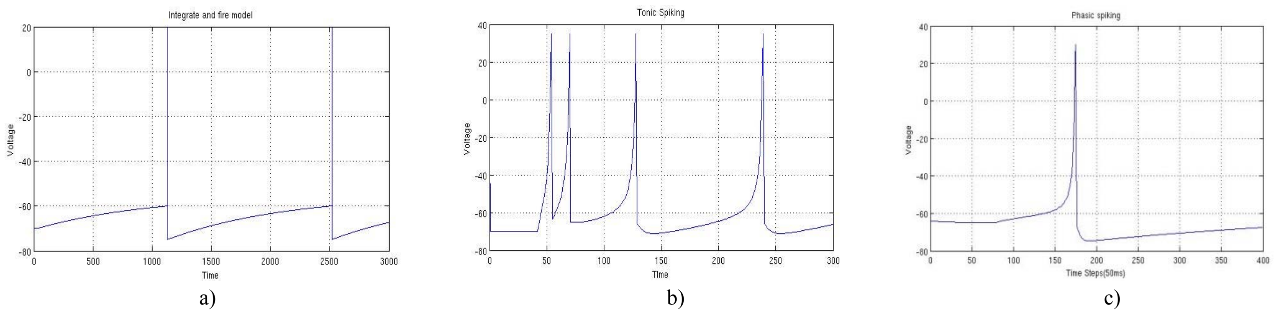

Fig. 5: SystemC simulation of the neuro-computational properties of spiking neurons: a) integrate and fire model, b) tonic spiking in the Izhikevich model, c) phasic spiking in the Izhikevich model.

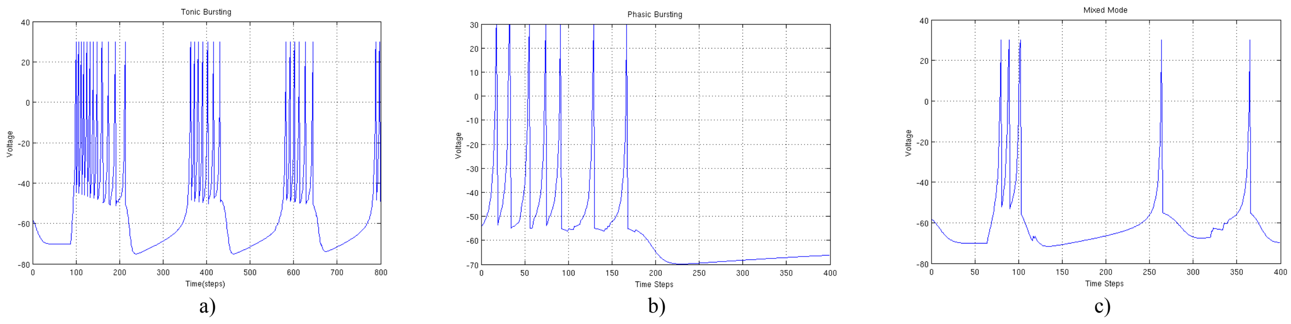

Fig. 6: SystemC simulation of the neuro-computational properties of spiking neurons in the Izhikevich model: a) tonic bursting, b) phasic bursting, c) mixed-type of spiking activity.

Model	*Cluster*	*PhC*	*TSF*	*BRAM %*	*DSP %*	*FF %*	*LUT %*	*Neurons*
Hodgkin-Huxley [23]	*NA*	*8*	*12*	*78*	*57*	*27*	*83*	*96*
Hodgkin-Huxley [18]	*18*	*2*	*33*	*23.6*	*35*	*27.5*	*90*	*1188*
Izhikevich	*5*	*8*	*70*	*38*	*22*	*25*	*89*	*2800*
Integrate and fire	*5*	*8*	*75*	*23*	*20*	*16*	*54*	*3000*

Table I - Hardware utilization of the most important components of the system on a Xilinx Virtex 7 XC7VX550 FPGA board. Routers sizes are generated by synthesizing a SystemC model of a router using Vivado HLS 2013.4.

Similar to the phasic spiking, the neurons can show phasic bursting behavior, as in Fig. 6b, which is needed to transmit saliency of the input, to overcome the synaptic transmission failure and reduce neuronal noise [27], or can be used for selective communication between neurons [28]. Intrinsically bursting excitatory neurons [29] depicted in Fig. 6c can exhibit a mixed type of spiking activity.

Table I lists hardware utilization of different spiking neuron models in the system. The system is implemented on the Virtex 7- XC7VX55 FPGA device. The hardware utilization is sub-divided into four different resource types, including look-up tables (LUT), flip-flops (FF), digital signal processors (DSP), and block memories (BRAM); smaller components, like the synchronization circuits, are omitted for clarity. FPGA resources can accommodate 1188 Hodgkin-Huxley type neuron cells, and approximately 2800 and 3000 Izhikevich, and integrate and fire type neural cells, respectively. The minimum simulation interval to achieve a realistic representation of the neuron-cell behavior is determined as in [14]. Consequently, since each neuron cell can be re-used multiple times within the real-time boundary (and to benefit from the high level of parallelism and performance of the FPGA), the neuron cells are time-multiplexed.

All results reported are for real-time simulations with double floating point precision for the most biologically accurate representation of a neuron cell behavior.

V. Conclusions

A real-time reconfigurable learning neuron networks are not only limited by run-time configurability, the re-synthesis of the system, and the interconnect between the neurons, but mainly with amount of neurons which can be placed on the chip. In this paper, we implement several models of the spiking neurons with axon conduction delays and spike timing-dependent plasticity (STDP) in a real-time data-flow learning network. The system implemented on the Virtex 7 - XC7VX55 device can accommodate 1188 Hodgkin-Huxley type neuron cells, and approximately 2800 and 3000 Izhikevich, and integrate and fire type cells, respectively. A tree-based communication bus is utilized since it offer run-time configurability, e.g. the user-enabled configuration of the connectivity between cell, the calculation parameters adaptability, etc. Consequently, the system does not need to be re-synthesized just to experiment with a different connectivity between cells. The cells are grouped around a shared memory in clusters to allow instantaneous communication.

REFERENCES

[1] W. Gerstner, W.M. Kistler, Spiking neuron models: single neurons, populations, plasticity, *Cambridge University Press*, 2002.

[2] W. Maass, "Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons", in M. Mozer, et al. (ed.), *Neural Information Processing Systems*, MIT press, pp. 211-217, 1997.

[3] S. Ghosh-Dastidar, H. Adeli, "Spiking neural networks," *International Journal of Neural Systems*, vol. 19, no. 4, pp. 295-308, 2009.

[4] Q. Yu, R. Yan, H. Tang, K.C. Tan, H. Li, "A spiking neural network system for robust sequence recognition," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 3, pp. 621-635, 2016.

[5] H. Shayani, P. Bentley, M. Tyrrell, "A cellular structure for online routing of FPGAs," in *Evolvable Systems: From Biology to Hardware*, Springer Berlin-Heidelbarg, pp. 273-284, 2008.

[6] E.M. Izhikevich, "Which model to use for cortical spiking neurons?", *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063-1070, 2004.

[7] K. Cheung, S.R. Schultz, W. Luk, "A large-scale spiking neural network accelerator for FPGA systems", *International Conference on Artificial Neural Networks and Machine Learning*, pp. 113-120, 2012.

[8] H. Shayani, P.J. Bentley, A.M. Tyrrell. "Hardware implementation of a bio-plausible neuron model for evolution and growth of spiking neural networks on FPGA", *NASA/ESA Conference on Adaptive Hardware and Syst.*, pp. 236-243, 2008.

[9] G. Smith, C. Cox, S. Sherman, J. Rinzel, "Fourier analysis of sinusoidally driven thalamocortical relay neurons and a minimal integrate-and-fire-or-burst Model," *Neurophysiology*, vol. 83, pp. 588-610, 2000.

[10] G.B. Ermentrout, "Type I membranes, phase resetting curves, and synchrony," *Neural Computation*, vol. 83, pp. 979-1001, 1996.

[11] A.L. Hodgkin, A.F.Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve", *Journal of Physiology*, vol. 117, no. 4, pp. 500-544, 1952.

[12] Y. Zhang, *et al.*, "Biophysically accurate floating point neuroprocessors for reconfigurable logic", *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 599-608, 2013.

[13] M. Beuler, *et al.*, "Real-time simulations of synchronization in a conductance-based neuronal network with a digital FPGA hardware-core", *International Conference on Artificial Neural Networks and Machine Learning*, pp. 97-104, 2012.

[14] M. van Eijk, C. Galuzzi, A. Zjajo, G. Smaragdos, C. Strydis, R. van Leuken, "ESL design of customizable real-time neuron networks", *IEEE International Biomedical Circuits and Systems Conference*, pp. 671-674, 2014.

[15] W. Gerstner, R. Kempter, J.L. van Hemmen, H. Wagner, "A neuronal learning rule for sub-milisecond temporal coding", *Nature*, vol. 383, no. 6595, pp. 76-81, 1996.

[16] S.J. Thorpe, "Spike-based imagge processing: can we reproduce biological vision in hardware", in A. Fusiello, et al. (ed.), *Computer Vision*, Springer Berlin-Heidelbarg, pp. 516-521, 2012.

[17] J.R. de Gruijl, *et al.*, "Climbing fiber burst size and olivary subthreshold oscillations in a network setting", *PLoS Computational Biology*, vol. 8, no. 12, pp. 1-10, 2012.

[18] G.J. Christiaanse, A. Zjajo, C. Galuzzi, R. van Leuken, "A real-time hybrid neuron network for highly parallel cognitive systems", *International Conference of the IEEE Engineering in Medicine and Biology Society,* pp. 792-795, 2016.

[19] E.M. Izhikevich, "Polychronization: computation with spikes," *Neural Computation*, vol. 18, pp. 245-282, 2006

[20] E. Pastalkova, P. Serran, D. Pinkhasova, E. Wallace, A. Fenton, T. Sacktor, "Storage of spatial information by the maintance mechanism of LTP," *Science*, vol. 31, no. 5790, pp. 1141-1144, 2006.

[21] P.V Massey, Z.I. Bashir, "Long-term depression: multiple forms and implications for brain function," *Trends in Neuroscience*, vol. 30, no. 4, pp. 176-184, 2007.

[22] J. Hofmann, A. Zjajo, C. Galuzzi, R. van Leuken, "Multi-chip dataflow architecture for massive scale biophysically accurate neuron simulation", *International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5829-5832, 2016.

[23] G. Smaragdos, S. Isaza, M. F. van Eijk *et al.*, "Fpga-based biophysically-meaningful modeling of olivocerebellar neurons," *IEEE International Symposium on Field-programmable Gate Arrays*, pp. 89-98, 2014.

[24] C.M. Gray, D.A. McCormick, "Chattering cells: Superficial pyramidal neurons contributing to the generation of synchronous oscillations in the visual cortex," *Science*, vol. 274, no. 5284, pp. 109-113, 1996.

[25] N. Schweighofer, *et al.*, "Electrophysiological properties of inferior olive neurons: a compartmental model", *Journal of Neurophysiology*, vol. 82, no. 2, pp. 804-817, 1999.

[26] J.R. Gibson, M. Belerlein, B.W. Connors, "Two networks of electrically coupled inhibitory neurons in neocortex," *Nature*, vol. 402, pp. 75-79, 1999.

[27] J. Lisman, "Bursts as a unit of neural information: Making unreliable synapses reliable," *Trends in Neuroscience*, vol. 20, pp. 38-43, 1997.

[28] E.M. Izhikevich, N.S. Desai, E.C. Walcott, F.C. Hoppensteadt, "Bursts as a unit of neural information: Selective communication via resonance," *Trends in Neuroscience*, vol. 26, pp. 161-167, 2003.

[29] B.W. Connors, M.J. Gutnick, "Intrinsic firing patterns of diverse neocortical neurons," *Trends in Neuroscience*, vol. 13, pp. 99-104, 1990.