A Real-Time Reconfigurable Multichip Architecture for Large-Scale Biophysically Accurate Neuron Simulation

Amir Zjajo, *Member, IEEE*, Jaco Hofmann, Gerrit Jan Christiaanse, Martijn van Eijk, Georgios Smaragdos, Christos Strydis, Alexander de Graaf, Carlo Galuzzi, *Member, IEEE*, and Rene van Leuken, *Member, IEEE*

***Abstract*—Simulation of brain neurons in real-time using biophysically meaningful models is a prerequisite for comprehensive understanding of how neurons process information and communicate with each other, in effect efficiently complementing in-vivo experiments. State-of-the-art neuron simulators are, however, capable of simulating at most few tens/hundreds of biophysically accurate neurons in real-time due to the exponential growth in the interneuron communication costs with the number of simulated neurons. In this paper, we propose a real-time, reconfigurable, multichip system architecture based on localized communication, which effectively reduces the communication cost to a linear growth. All parts of the system are generated automatically, based on the neuron connectivity scheme. Experimental results indicate that the proposed system architecture allows the capacity of over 3000 to 19 200 (depending on the connectivity scheme) biophysically accurate neurons over multiple chips.**

***Index Terms*—Biophysically accurate neuron simulation, multichip data-flow architecture, neuron network.**

I. Introduction

CONTINIOUS neuroscientific progress gradually led to the realization of mathematical models of the neuron cells and their intricate networks [1], [2]; realistic models, which simulate biological behavior with a large level of accuracy, as in the case of spiking neural networks (SNNs) [1], [3]. In SNNs, propagated information is not just encoded by the firing rate of each neuron in the network, as in artificial neural networks (ANNs), e.g., perceptron [4], but, in addition, by amplitude, spike-train patterns, and the transfer rate. The high level of realism of SNNs and more significant computational and analytic capabilities in comparison with ANNs, however, limit the size of the realized networks. Consequently, the main challenge in building complex and biophysically-accurate SNNs is largely posed by the high computational and data transfer demands.

In addition, biological NNs are characterized by co-localized memory and calculations, and execute computations with high degree of parallelism, for what conventional, von Neumann CPU-based execution is not very well suitable. Due to their inherent high-level of parallelism, reconfigurable hardware, such as field-programmable gate arrays (FPGAs), are capable of providing sufficient performance for real-time and even hyperreal-time simulations of these collective and distributed networks. Furthermore, the reconfiguration property of FPGA provides the flexibility to modify the network topologies and the brain models on demand, (e.g., Izhikevich [2], [5], [6], integrate and fire (IaF) [7] model (and its extensions such as the *leaky* IaF, IaF-or-Burst [8], *quadratic* IaF [9]), Hodgkin-Huxley (HH) [10]–[12], *simplified* Hodgkin-Huxley [13], *extended* Hodgkin-Huxley [14]).

Small-scale special purpose systems, such as ROLLS [15] intended for cortical-like computational modules, implement 256 IaF neurons. In [16], the system containing several tens of thousands of leaky IaF neuron cells are implemented on Virtex-7 FPGA platform. In [17], analog-based Neurogrid system replicates neurons as an electrical system [18]. With 16 Neurocores, i.e., the computation elements of the Neurogrid, the system is able to simulate over 1 million quadratic IaF neurons with billions of synapses. TrueNorth in [19], [20] contains 4096 cores, totaling 1 million programmable digital IaF spiking neurons and 256 million configurable synapses. Although significantly larger number of neurons can be simulated when compared to the FPGA solutions, the Neurogrid and TrueNorth platforms are not as flexible concerning model changes, nor the neuron behavior can be as easily observed. The models are not analyzed as applications in general, but only as implementations on the particular platform. Additionally, no neuroscientific-experiment instances with biological plausibility were considered. Large

Manuscript received August 3, 2017; revised October 12, 2017; accepted November 11, 2017. Date of publication January 26, 2018; date of current version March 22, 2018. This work was supported in part by the European Union and the Dutch government, as part of the CATRENE program under Heterogeneous INCEPTION project. This paper was recommended by Associate Editor D. Ham. *(Corresponding author: Amir Zjajo.)*

A. Zjajo, G. J. Christiaanse, M. van Eijk, A. de Graaf, and R. van Leuken are with the Circuits and Systems Group, Delft University of Technology, Delft 2628 CD, The Netherlands (e-mail: amir.zjajo@ieee.org; jan.christiaanse@tudelft.nl; m.vaneijk@tudelft.nl; a.degraaf@tudelft.nl; r.vanleuken@tudelft.nl).

J. Hofmann was with the Circuits and Systems Group, Delft University of Technology, Delft 2628 CD, The Netherlands. He is now with the Embedded Systems Group, Darmstadt University of Technology, Darmstadt D-64289, Germany (e-mail: jaco.hofmann@tudarmstadt.de).

G. Smaragdos and C. Strydis are with the Department of Neuroscience, Erasmus Medical Center, Rotterdam 3015 GE, The Netherlands (e-mail: g.smaragdos@erasmus.nl; c.strydis@erasmus.nl).

C. Galuzzi was with the Circuits and Systems Group, Delft University of Technology, Delft 2628 CD, The Netherlands. He is now with the BMI Research Group, Maastricht University, Maastricht 6211 LN, The Netherlands (e-mail: c.galuzzi@maastricht.nl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TBCAS.2017.2780287

scale experiments are performed with large number of general purpose processors as in the SpiNNaker project [21] as well, where over one million low power ARM cores are connected by a fast mesh based interconnect link. Subsequently, the largest SpiNNaker system is able to simulate over one billion neuron cells of Izhikevich type [2].

However, for electrochemically accurate neuron modeling, which is a focus of our study, the conductance-based multi-compartment Hodgkin-Huxley model [10] is required. Biophysically accurate models of biological systems, such as the ones using the Hodgkin-Huxley formalism, are comprised mostly of a set of computationally challenging differential equations often implementing an oscillatory behavior. If the interconnectivity between oscillating neurons is also modelled (e.g., gap junctions, input integrators, synapses), the cells become coupled oscillators. Consequently, all neuron states need to be completely updated at each simulation step to retain correct functionality. As a result, cycle-accurate, transient simulator is necessary. The above difficulties in associated HH models and multi-compartmental models with complex connections, in conjunction with biophysically plausible neuron network sizes, pose significant challenges especially when using conventional computing machines.

The HH model incorporates the membrane potential, and includes the concentration of various chemicals, inside the neuron, in the calculations to represent its behavior. The computational complexity of conductance-based models is orders-of-magnitude higher than IaF models, posing a significant challenge for their efficient simulation. For HH models, GPU implementations have been shown to be less efficient compared to reconfigurable hardware solutions [22], [23], even though providing notable speedups [24].

In [12], a simplified version of the HH model is used in an FPGA based simulator that is able to simulate 400 physiologically realistic neurons on a Virtex-4 FPGA device. Until recently, most HH models accelerated in reconfigurable platforms, due to their efficiency, employed fixed-point arithmetic. However, limited accuracy of fixed-point representation results in a faulty representation of neural spike location, altering the functional behavior of the neuron [25]. The system in [11] simulates a biophysically accurate representation of the neuron using floating-point arithmetic, and the HH model. The cost of the biophysical accuracy is a low network size; the largest system proposed contains 4 neurons. In our previous work [14], we could simulate 48 *extended* HH neuron model cells (highly biophysically-accurate model [26]) with floating point arithmetic on a Virtex 7 FPGA platform. In [23], a similar system is refined to include up to 96 neurons.

In this paper, we propose an efficient multi-chip dataflow architecture for the *extended* HH neuron cell and subsequent interconnected network [27], which exploits data locality and minimize network communications over one or multiple FPGA devices. The proposed system provides several key aspects compared to existing approaches:

i) Close to linear growth in the communication cost: with proposed data localization scheme and the resulting linear growth in communication cost, 31× to over 200× more neurons could be simulated in comparison to the state-of-the-art designs, which are limited by the exponential growth in the communication cost.
ii) The extendibility of the system over multiple chips to build more accurate systems: the system maintains linear growth up to 8 FPGA devices.
iii) The use of double floating-point arithmetic for the most biologically accurate cell behavior simulation, and an easy implementation of various neuron network topologies, cell communication schemes, as well as models and kinds of cells.
iv) A high run-time configurability, which reduces the need for resynthesizing the system. Additionally, adaption of routing tables, and changes to the calculation parameters are also possible. In this way, the system reduces the time required for experiments with biophysically accurate neurons.
v) A powerful simulator designed for high precision spiking neuron network simulations, but flexible enough to be used for smaller neural networks. The simulator features configurable on- and off-chip communication latencies as well as neuron calculation latencies. All parts of the system are generated automatically based on the neuron interconnection scheme in use. The simulator allows exploration of different system configurations, e.g., the interconnection scheme between the neurons, the intracellular concentration of different chemical compounds (ions), which affect how action potentials are initiated and propagate.

The paper is organized as follows: In Section II, the cell abstract model and the hardware design configuration are described. Section III focuses on the system design from a high-level perspective, including adjustments to the network to scale over multiple FPGAs, synchronization between clusters and connectivity, and structure generation. Section IV discusses implementation details including clusters, routers, and the control bus for run-time configuration. In Section V, simulations and hardware utilization for different scenarios including network and multi-chip performance are presented. Finally, Section VI provides a summary and the main conclusions.

II. The Inferior Olivary Nucleus Cell

A. Abstract Model Description

The neuron cells considered in this paper are located in the inferior-olivary nucleus (ION). The ION is an especially well choreographed part of the brain [28], [29]. The *extended* (by gap junctions) HH model based on experimental findings in [29] (Fig. 1) implements a neuron with three distinct compartments: the dendrite, the soma and the axon. The gap junctions are part of the dendritic compartment; consequently, the dendritic compartment receives the extra input coming from the inter-neuron connection. The gap junctions (which differ from typical synapses in that they are purely electrical) are associated with important aspects of cell behaviour as they are not just simple connections; rather, they involve significant and intricate electrical processes, which is reflected in their modelling details.

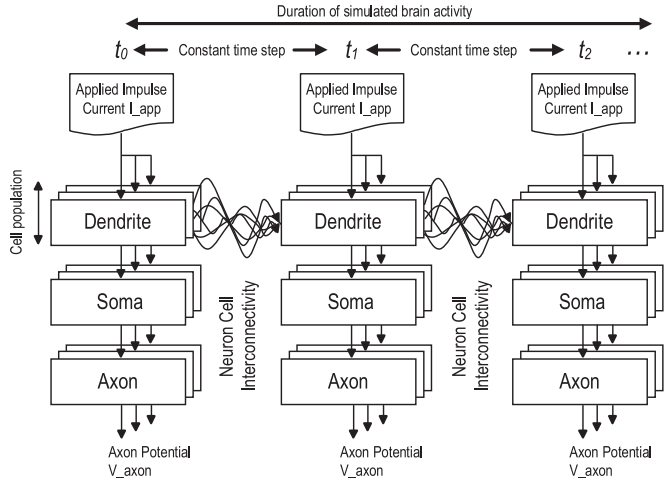


Fig. 1. Inferior olive neuron model [30].

TABLE I
NEURON REQUIREMENTS PER SIMULATION STEP

Computation	**FP operations per neuron**
Gap Junction	12 per connection
Cell Compartment	859
I/O and storage	**FP variables per neuron**
Neuron States	19
Evoked Input	1
Connectivity Vector	1 per connection
Neuron Conductances	20
Axon Output	1 (Axon voltage)
Compartmental Task	**% of FP ops for 96 cells**
Soma	13
Dendrite	10
Axon	8
Gap Junction	69

Every compartment includes biophysical attributes, i.e., state parameters denoting its electrochemical state, and computation is performed in all three compartments (and within each gap junctions connection itself). For the calculation of a single parameter, one exponential function and several multiplications and divisions need to be carried out. Additionally, for realistic signal representation the use of floating-point (FP) arithmetic is essential [25]. The total number of FP operations required for simulating a single step of a single neuron cell (including a single gap junction) is 871 (Table I). In an n-cell network (n_c), if each neuron maintains a constant number of connections λ to neighboring cells, the complexity of overall gap junction computation cost increase as $\mathrm{O}_{gj}(n_c \times n_c \times \xi)$, where ξ is the connectivity density [30]. The worst-case interconnectivity scenario occurs when $\xi = 1$, i.e., all-to-all neuron connection, resulting in $\mathrm{O}_{gj}(n_c^2)$ complexity. All remaining, non-gap junction computation increases linearly $\mathrm{O}_{cell}(n_c)$ since the rest of the application is of purely dataflow nature [30]. The neuron model defines effectively a transient simulator through computing discrete output axon values in time steps which, when integrated in time, recreate the output response of the axon. The three compartments and gap junctions are evaluated/updated concurrently at each simulation step. The model is calibrated with a simulation time step of 50 μsec, where this simulation step also defines the real-time behaviour of the whole network. Simulations steps are identical to each other in terms of operations performed.

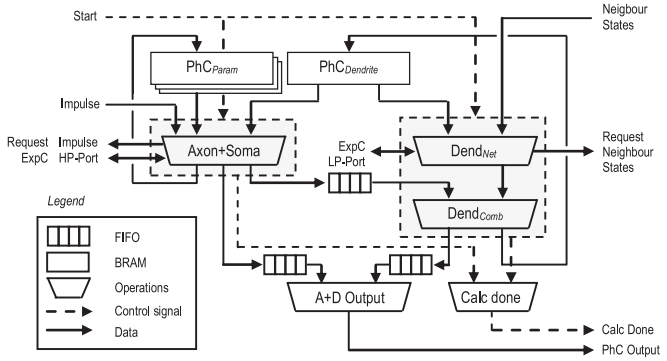


Fig. 2. Dataflow of a *PhC*. The dashed box on the left is the Axon/Soma calculation. The one on the right is the dendrite calculation. Cell states are stored in the BRAM (memory). All in- and output data signals are connected to FIFO buffers (not shown) [31].

B. The ION Cell Design Configuration

Operationally, the neuron network needs to compute and communicate simulated ION 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), and communication (Section III). We refer to a neuron computation unit as a physical cell (*PhC*).[1]. Within a *PhC*, the topology-dependent (i.e., incorporating the neighbors coupling) dendrite calculation (η_{dend}), and topology-independent *Axon+Soma* (η_{a+s}) calculation run in parallel (Fig. 2) [31]. The total amount of cycles each *PhC* requires (η_{PhC}) is

$$\eta_{PhC} = \max(\eta_{dend}, \eta_{a\,+\,s}) \tag{1}$$

The *Axon+Soma* computational unit computes the axon and soma state, and updates a set of cell parameters, based on the current cell compartment states and cell parameters. Internally, the dendrite calculation is dependent on the result of the *Axon+Soma* calculation to calculate the new dendrite state. Externally, both calculations use the same exponent co-processor (*ExpC*). The exponent operations, compared to standard operations, require relatively more resources and cycles to complete.

To reduce the required amount of resources without adverse effect on the calculation latency, we utilize a single exponent instance over multiple neuron calculations [31] in a Kahn process network [32]. As the *Axon+Soma* calculation has a

[1]The computation units are called physical cells (*PhCs*) to recall that they are physically implemented in hardware, and that the outputs of their computations mimic the actual inferior-olivary nucleus (ION)-cell behavior. See [29] for additional information.

longer critical path (and is topology independent), it is scheduled with a higher priority. Each calculation within a cluster is synchronized, resulting in *Axon* potentials being calculated at predictable times. The dendrite calculation is, however, not synchronized, giving it more flexibility over the exponent co-processor. By keeping the critical path within the dendrite calculation to a minimum, and by allowing it to start processing new network neighbors before the current exponent is known, each simulated cell can quickly be scaled up to allow more connections within the neuron network. The exponent co-processor is, thus, constantly being given new values to calculate, with high priority tasks arriving after a deterministic amount of cycles.

The Axon+Soma Calculation Unit Configuration: Exponent operands within the *Axon+Soma* calculation unit consists of an addition and multiplication carried out by two constants (χ, β) on a single cell state (compartmental potential):

$$e^{\phi}, \phi = (State + \chi) \times \beta \tag{2}$$

The controller for the application specific co-processor (ASC) insures that the correct state potential, and operand constants (χ, β) are stored in the registers at the correct time. Multiple *PhC*s are scheduled around a single *ExpC* to reduce the required resources. Consequently, the ASC is adjusted to receive multiple cell states from multiple sources, and to send a source address with the exponent operand as (addressed) output. The calculation can be subdivided in two segments, fetch and schedule, respectively. Before any calculations can be carried out, the hardware necessities to *fetch* the required cell states and parameters from the local memory, while also sending out a request to the memory controller to receive a (non-zero) impulse value. With the fetched values the *Axon+Soma* calculation unit can partly offload its calculations to the ASC, while starting calculations that are independent of the results from the exponent calculations. The η_{a+s} is determined by the number of *PhC*s that share a co-processor, and how the axon and soma calculations are scheduled

$$\eta_{a\,+\,s} = \eta_{a+s(base)} + \gamma \times (\nu - 1) - \theta(\nu) \tag{3}$$

where $\eta_{a+s(base)}$ is the base latency, i.e., one *ExpC* is connected to only one *PhC*, γ is the number of exponent calculations required by the *Axon+Soma* calculation unit, ν is the maximum number of *PhC*s sharing a *ExpC*, and θ is the overlapping factor.

The Dendrite Calculation Unit Configuration: The *dendrite* calculation unit computes the new dendrite compartmental state based on the current dendrite state, the neighboring dendrite states, and finally an intermittent response generated by the *Axon+Soma* calculation unit. $\mathrm{Dend_{Net}}$ computes the coupling effect of neighboring cells in the neuron network; its input is determined by the current dendrite state that is *fetched* from the BRAM memory, and the neighboring dendrite states requested from the memory controller. $\mathrm{Dend_{Net}}$ is scheduled around the exponent operation and is split into two parts: computations that are dependent of the exponent result, an those that are not. $\mathrm{Dend_{Comb}}$ combines the intermittent response received from the *Axon+Soma* calculation unit and results from the $\mathrm{Dend_{Net}}$ to generate a new dendrite state potential. The resulting dendrite state from this operation is then locally updated and communicated to the memory controller. The dendrite compartmental computation latency η_{dend}[2] can be written as function of

$$\begin{aligned} \eta_{dend} &= \max(\eta_{din}, \eta_{a\,+\,s}) + \tau \\ \eta_{din} &= \delta(\varphi) + \max(\eta_{block}, \alpha \times N_D) + \omega \times N_D \end{aligned} \tag{4}$$

where α is the amount of cycles that takes place before each (low priority) exponent calculation, N_D is the number of dendrites, δ is the start-up delay partly dependent on the amount of dendrite calculations that share a memory core, and φ is the grouping factor. A blocking time η_{block} is involved if the exponent calculation is being blocked by another task after all α calculations are performed, and ω is the number of cycles after the result of the exponent calculation is known.

The Exponent Core: We schedule the exponent operands through a *read scheduler*, i.e., the scheduler that feeds exponent operands onto a single channel (vector) with an (additional) address.[3] The vector is fragmented over the architecture that calculates the exponent, and a shift register that keeps track of where the current calculation (address) is in the pipeline. When the exponent is calculated, a valid address is presented at the end of the shift register, signaling a write back to a specific (addressed) output FIFO.

C. The ION Cell Cluster Controller

The neuron cells are connected with decreasing probability the further they are apart [1]. The individual computation units, i.e., physical cells that are in a close proximity to each other are placed within a confinement of a (neighbor) cluster. The amount of clusters κ implemented in the FPGA is based on the critical resources and is determined as $\kappa = PhC_{tot}/\varphi$. The cluster controller relates new values to the calculation architecture when requested, and store and route their responses. Each cluster controller is designed around several parallel running hardware architectures, that are synchronized by FIFO's. In Fig. 3, an example is given of a cluster controller with two connected *PhCs*.

The *init* controller receives a coded set of initialization parameters (e.g., the cluster identification number, a local routing table, initial parameters for the local *PhCs* and the dendrite states of all cells) through the initialization channel. The write controller communicates with the *PhCs* by request. If the *PhCs* are not initialized and a request arrives, the write controller transmit the initialization parameters in a pre-defined order starting with the dendrite state. During the simulation, if a dendrite request is received, the write controller looks up (i.e., with the help of the local routing table) which dendrite states of the neighboring cells it should send. Consequently, the write controller sequentially sends each dendrite state addressed on the columns of that row. If an applied current request is received, the local address

[2]Within the dendrite calculation, $\mathrm{Dend_{Comb}}$ combines the 2 results after τ cycles.

[3]The high priority is already addressed by the ASC. Low priority inputs are only passed as operands, and are given an address based on which FIFO they are read from.

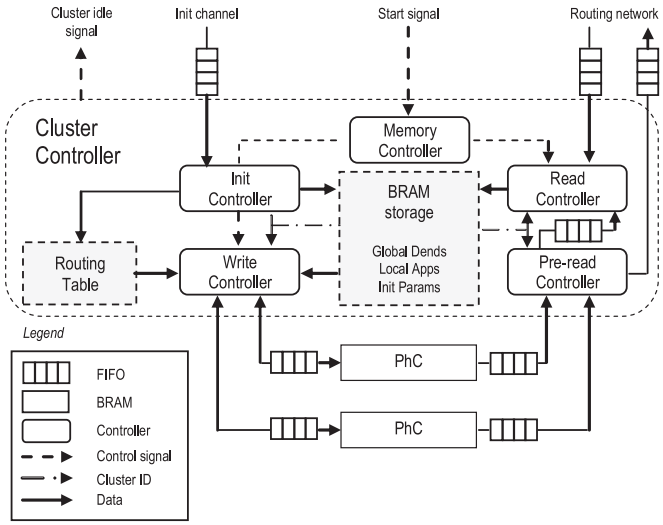


Fig. 3. A diagram showing how the controllers are housed within the cluster controller; the cluster done logic is excluded from view. Dataflow of a *PhC*. The dashed box on the left is the Axon/Soma calculation.

is used to get the applied current from the BRAM storage. The neighboring cell addresses are placed column-wise in the routing table, while each row represent where a local cell is located in the IO topology. After a cell response is generated, the pre-read controller determines the global address of the cell within the neural network.

If the cell response is an axon value, the signal is sent to the routing network; if it is a dendrite response, the new value is duplicated and sent both to the routing network as to the read controller through an internal FIFO for storage. The read controller authorize storage of the applied ION currents and dendrite responses in the BRAM, i.e., the read controller determines which ionic currents will be admitted by calculating the relevant global address range based on the cluster identification number and the amount of cells that are connected to the cluster controller. The dendrite states and applied currents are stored in the BRAM in two parts; current and next state memory. At the start of each simulation-round a start signal is received by memory controller and the bit is issued that indicates that the next state memory block is now current state and vice versa. This prevents memory being overwritten by the read controller before it can be sent to the *PhC*. The cluster controller falls into an idle state when the connected *PhCs* have finished calculations and all newly generated results have been stored and/or sent to the routing network.

III. Multi-Chip Dataflow Architecture

Neural connectivity have been previously implemented through shared bus networks [14], however, bandwidth restrictions limit the scalability of such approaches. Alternatively, local buses between adjacent neurons arranged in one-dimensional [19] or two-dimensional [21] grid network have been proposed for increased routing flexibility. However, one of the most notable features of the real brain networks is their high degree of clustering, with nodes (neurons) connecting preferentially to others in their local neighbourhood [33]. The large density of local connections in brain networks may have several functional and evolutionary benefits, such as enhanced communication speeds, and minimal wiring and metabolic costs. To scale communication linearly with neuron count we emulate the cortex's hierarchically branching wiring patterns.

In the clusters, configurable routing tables define how *PhC*s are arranged within the neuron network. By attaching each cluster to a binary tree network, responses between *PhC*s are shared (Fig. 4) [27]. 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. The design can be tuned using 4 parameters: the number of clusters, the number of *PhC*s, the amount of shared exponent coprocessors within a cluster, and the time sharing factor for each *PhC*.

A. Localize Communication Between Clusters

The data from other cells is read seldom in the *PhC* [1], [14]. Consequently, a single cell does not require memory access each clock cycle, allowing for a shared memory design with time-shared instead of parallel memory access. The main advantage is that the common case of close communication is still optimal. The number of *PhC* around one shared memory is limited by placement and wire length constraints of the FPGA technology in use.

B. Connecting Clusters: Routers

In the proposed architecture, each router has 2 to n children, and each child can be either a cluster or another router. The clusters transmit only two types of data, i.e., dendritic and axon hillock potentials. While cell dendrite potentials are shared among all IONs, axon hillock potentials are only given as an output. Consequently, the router is designed with the following rules: *i*) in a balanced tree network each router is connected to one bi-directional upstream and two bi-directional downstream channels, *ii*) new dendrite potential values can arrive through any channel and are passed along the other two channels, and *iii*) new axon hillock potential values only arrive through one of the two down-stream channels and are then transmitted to the upstream channel.

The data produced by each cell in the network and the cell identification number and are combined in a packet. Based on a static routing table (which reflects the way the cells communicate), each router decides in which direction i.e., to which cells, the packet has to be forwarded to. Within the proposed design, each router (Fig. 5) is connected to 3 channels and is implemented around a single core (Router Logic) together with a (FIFO) buffer. The channels consist of an input and output FIFO, forming a bi-directional channel. The router logic reads every channel in a round robin type fashion. If a new packet is present in one of the channels it is read (and based on the rule set), the packet is transmitted to one or two of the other channels.

However, due to hardware limitations a channel might fill up before it is emptied (read). Since no packet is allowed to be

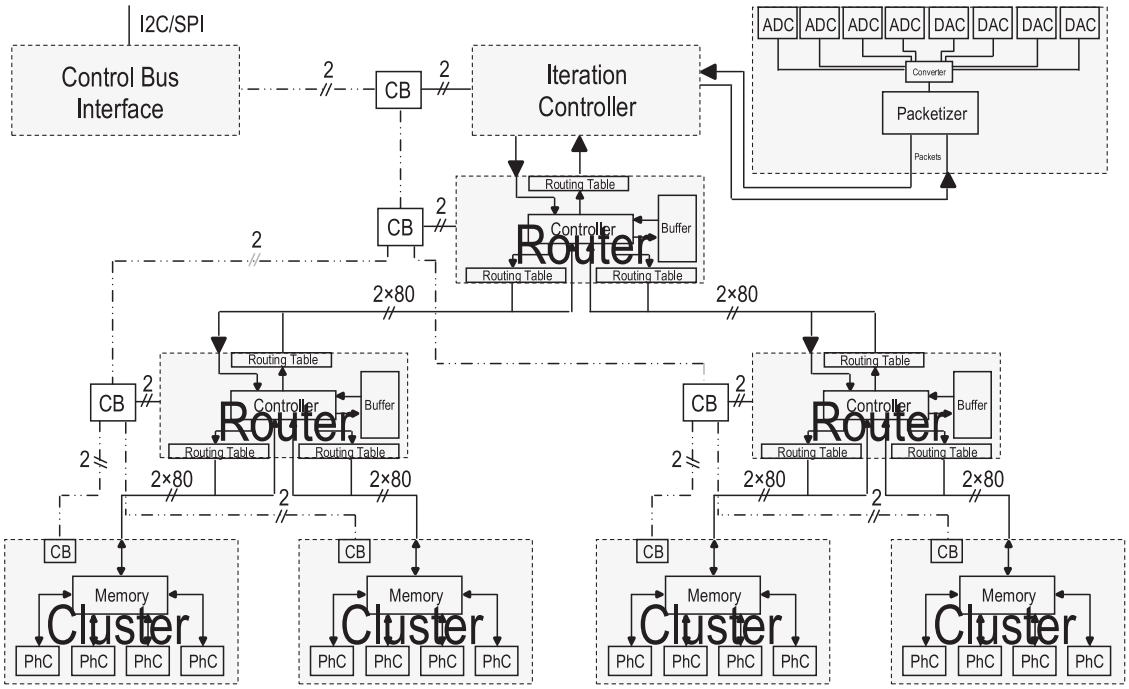


Fig. 4. The system overview. The computing elements (the *PhCs*) are grouped inside a cluster to make communication between neighboring cells fast. These clusters are connected in a tree topology NoC. The router fan-out in this case is 2, and can be changed according to the requirements of the implementation. The same holds true for the number of *PhCs* in any cluster [27].

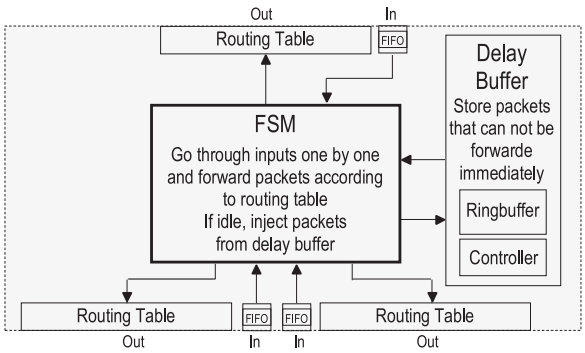


Fig. 5. The router diagram [27].

dropped, packets that cannot be forwarded right away, i.e., when the receiving buffer is full, are stored for delayed delivery.

The width of this delayed buffer is $b_p + [\log_2(n_o)]$ bit, where b_p is the amount of bits for a packet, and n_o is the amount of outputs of the router. By designing the router around a small finite state machine (FSM), each symbol can be passed to 1 or 2 channels every 2 clock cycles. To avoid cases in which the router continuously try to deliver delayed packets to full routers, new packets always have precedence over the delayed ones. Since packet forwarding is not aware of the complete network connectivity, the components are efficient, and with limited overhead.

C. The Control Bus for Run-Time Configuration

Due to the high number of components, the most commonly used bus systems, e.g., Wishbone or Serial Peripheral Interface (SPI), are not applicable. In addition, these buses require a significant number of wires to address every component in the system. Consequently, we designed a custom-made bus, which follows the tree structure of the NoC.

By traversing the tree, all components can be addressed, from the routers down to the clusters and to each *PhC*. Importance of throughput is reduced, since all configuration is set while the system is paused. The amount of data transmitted via the bus is low with the largest transmissions being parameter changes of a cell. A bus command comprises of two parts, i.e., the address and the payload. Correspondingly, the bus first opens up a connection to a specific component using the address, and then forwards the payload to the component. Each component in the system, e.g., a router or a cluster, has an attached control bus router that can either forward the bus signal to any of its children, or forward the bus signal to the attached component.

D. Adjustments to the Network to Scale Over Multiple FPGAs

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.

Although adding another tree layer promises easy extendibility, the limited connection possibilities of each FPGA, and need for an extra FPGA for routing between the FPGAs containing the clusters, however, restrict their use. Consequently, as most communications occur between neighboring FPGAs, the FPGAs are connected in a ring based topology (Fig. 6), which is less complex in terms of topology generation and administration of the routing tables.

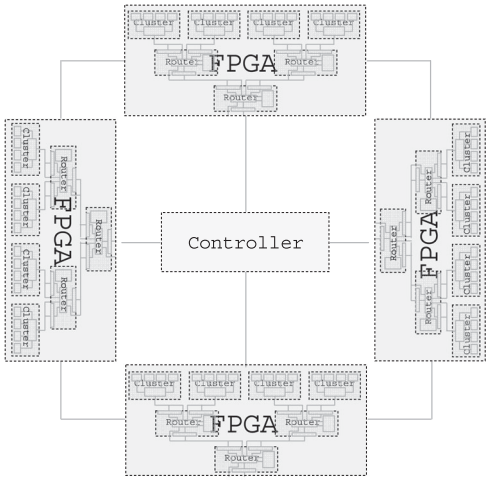


Fig. 6. The single FPGA implementations are connected using a ring topology network. The FPGA are synchronized via a central controller.

To synchronize the communication between the clusters, one of the FPGAs contains a controller that handles all the synchronization packets. In large systems this could impact the time needed to complete the iteration. To prevent this, we use one of the FPGAs as a master. Consequently, the signal does not have to cross multiple stages, the run time is constant for any number of cells, and signal can finish iteration immediately. The master FPGA, in turn, issues the new round signal when adequate.

IV. Experimental Results

The system is automatically generated using a human-readable configuration file, which includes all relevant parameters of the system and can be easily modified allowing exploration of different cell communication schemes, several fan-out values, etc. The control interface includes initialization, setting of I_{app}, direct memory access (DMA) (scatter mode) and Ethernet user datagram protocol (UDP) transfer from the FPGA to the PC, and interrupt support. After the design is configured with the desired accuracy (32/64-bit), it is synthesized through the Vivado HLS tool to generate VHDL code, and test bench files.

The multi-FPGA system experimental setup is illustrated in Fig. 7, while FPGA resources utilization is shown in Fig. 8. In Table II, the hardware utilization for double/single floating-point precision for the main components of the system are shown in terms of flip-flops (FF), time sharing factor (TSF), block RAM (BRAM), and look-up tables (LUT); smaller components, like the synchronization circuits, are omitted for clarity. All results noted are for the 8-way connectivity inferior-olivary network model. The minimum time interval to achieve a realistic representation of the neuron-cell behavior is determined as in [29]. For comparison purposes, Table III lists hardware utilization of different spiking neuron models, i.e., Xilinx Virtex 7 XC7VX550 FPGA can accommodate 1188 32b-accurate Hodgkin-Huxley type neuron cells, and approximately 2800 and 3000 Izhikevich, and integrate and fire type neural cells, respectively.

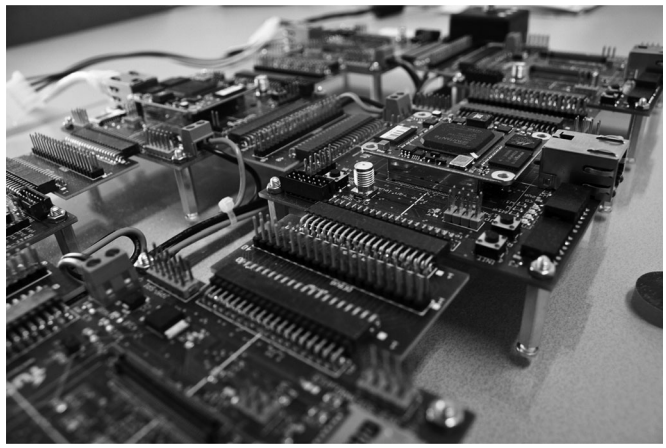


Fig. 7. Multi-FPGA system experimental setup.

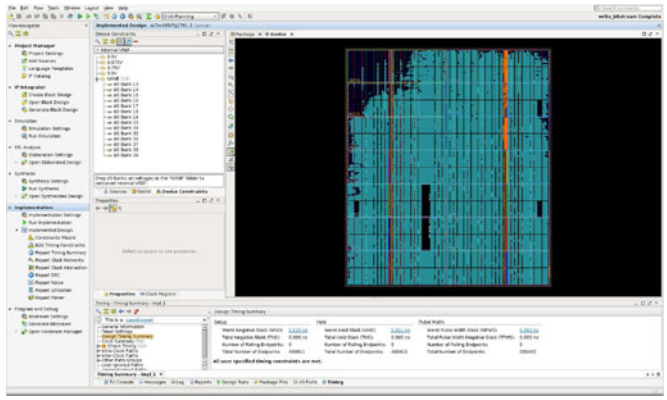


Fig. 8. FPGA resources utilization.

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 [34]. In the *extended* HH model [29] a compartment is added to model the axon hillock of the cell and enable the model to generate axonal bursts of sodium spikes. The model includes a high- and low-threshold calcium current, calcium-dependent potassium current, and a potassium and sodium current. All compartments also have a passive leak current. Most neurons are quiescent but can fire spikes when stimulated. When the pulses of the current are injected at the input, the neuron fire a train of spikes, the process called tonic spiking [35]. 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. Specific neurons fire periodic bursts of spikes when stimulated. Similar to the phasic spiking, the modelled neurons can show phasic bursting behavior, which is needed to transmit saliency of the input, to overcome the synaptic transmission failure and reduce neuronal noise [36], or can be used for selective communication between neurons [37]. Intrinsically bursting excitatory neurons [38] can exhibit a mixed type of spiking activity.

TABLE II
IMPLEMENTABLE CELLS ON A FPGA WITH CRITICAL RESOURCES UNDERLINED

Implementation						Resources (Absolute)				Resources (Total Utilization)				Results	
FPGA	Accuracy	Clusters	PhC	TSF	ExpC	LUT	FF	DSP	BRAM	LUT	FF	DSP	BRAM	SimC[b]	Cost/SimC[c]
[14][a]	64b	NA	8	6	NA	240 k	209 k	1384	42	69.4%	30.2%	48.1%	1.8%	48	$ 144
Virtex 7	64b	9	2	23	1	324 k	202 k	1215	233	93.7%	29.2%	42.2%	19.7%	414	$ 15.5
Virtex 6	64b	1	7	20	2	124 k	77 k	634	122	82.9%	25.8%	82.6%	14.7%	140	$ 20.1
Spartan 6	64b	1	1	8	1	23 k	21 k	113	27	25.5%	11.6%	62.8%	10.1%	8	$ 29.6
[23][d]	32b	NA	8	12	NA	251 k	162 k	1600	804	83%	27%	57%	78%	96	$ 51.2
Virtex 7	32b	18	2	33	1	311 k	190 k	1008	557	90%	27.5%	35%	23.6%	1188	$ 5.4
Virtex 6	32b	4	4	29	2	128 k	85 k	480	192	85.4%	28.5%	62.5%	23.1%	464	$ 6.1
Spartan 6	32b	1	4	18	2	36 k	23 k	152	33	39.9%	12.8%	84.4%	12.3%	72	$ 3.3

[a]Only estimates are given in the previous design, built on the same Xilinx Virtex 7 XC7VX550 FPGA board as the current design.
[b]We refer to each (neuron) node in the neuron network as a Simulated Cell (*SimC*), whereas we refer to the hardware used to simulate the cells as a *PhC*.
[c]Reference costs were taken from [40].
[d]Extended HH model, 22.2 k operations per neuron in 1 ms, 100% interconnectivity density, 2131.2 MFLOPS.

TABLE III
HARDWARE UTILIZATION OF THE MOST IMPORTANT COMPONENTS OF THE SYSTEM ON A XILINX VIRTEX XC7VX550 FPGA BOARD

Model	Cluster	PhC	TSF	BRAM%	DSP%	FF%	LUT%	Neurons
Hodgkin-Huxley	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

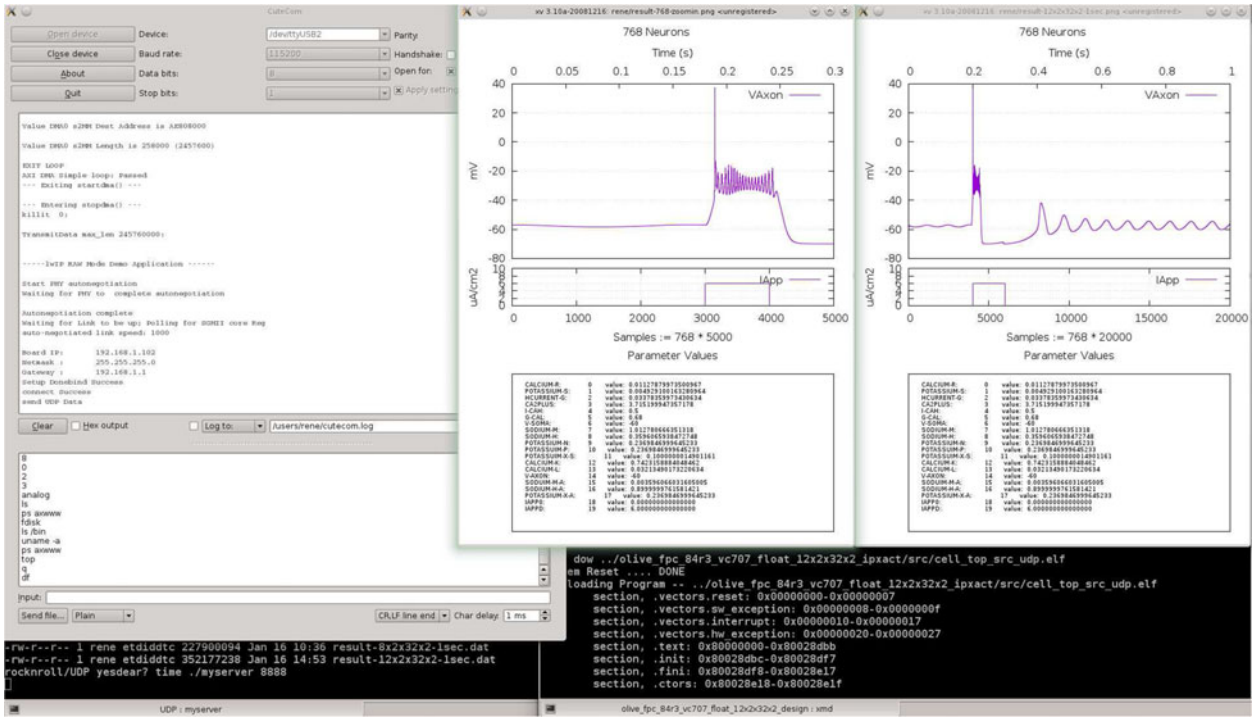


Fig. 9. Neuron network running on the Xilinx Virtex 7 XC7VX550 FPGA: output waveform of the inferior olive cell model in a network configuration, axon potential vs. brain-simulation time.

A visualization of the ION axon potential V_{axon} run on the FPGA for 1s brain simulation time (with associated neuron parameters) in a biophysically accurate neuron network consisting of 768 *extended* Hodgkin-Huxley neuron cells is shown in Fig. 9. Here, an impulse of -1 mA/cm^2 is applied after 0.19 s for a duration of 100 ms with a resting neural network surface current of 0.5 mA/cm^2. Similar patterns are found with biological test [39]. Simulating identical network settings in SystemC require 59 minutes of *cpu*-time on a openSUSE 13.1 (x86_64) system with Intel Xeon CPUs E5-1620 3.5 GHz processor and 32 GB of memory. Consequently, a FPGA ported design yields >3500 $\times$ speed-up (performed in a real-time). The hardware results are

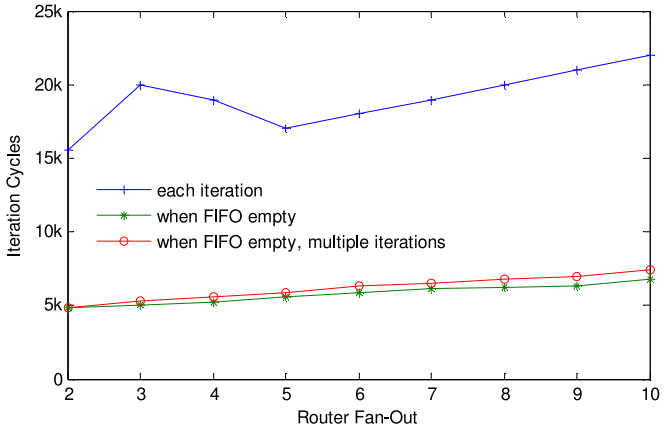


Fig. 10. Three packet injection schemes. Injecting the packets after each round through all input FIFO results in the slowest iteration times. Injecting only when all FIFO are empty for one round or for multiple rounds achieves comparable performance. Injecting after one round of input inactivity results in the best overall performance.

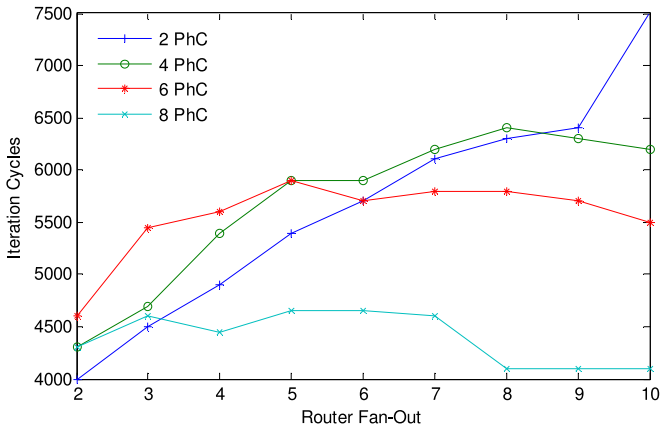


Fig. 11. Comparison of iteration performance for different cluster sizes. Small routers perform better than large routers - the best performing configuration consists of small clusters and small routers at two *PhCs* per cluster and a fan-out of two.

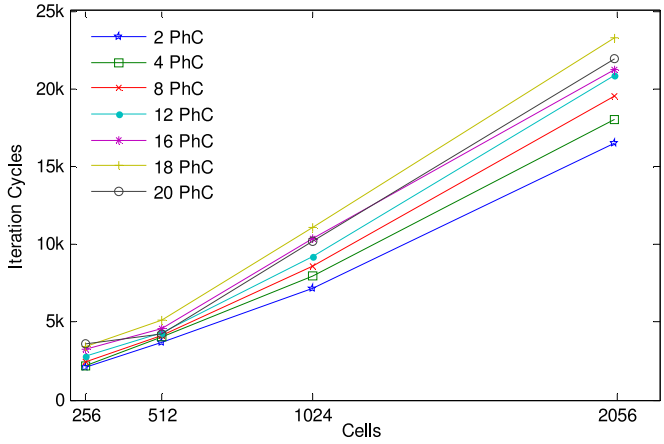


Fig. 12. Comparison of systems with different cluster sizes; the routers are kept constant at fan-out of two; clusters with two, four and eight *PhCs* achieve the shortest iteration times, while larger clusters are generally slower.

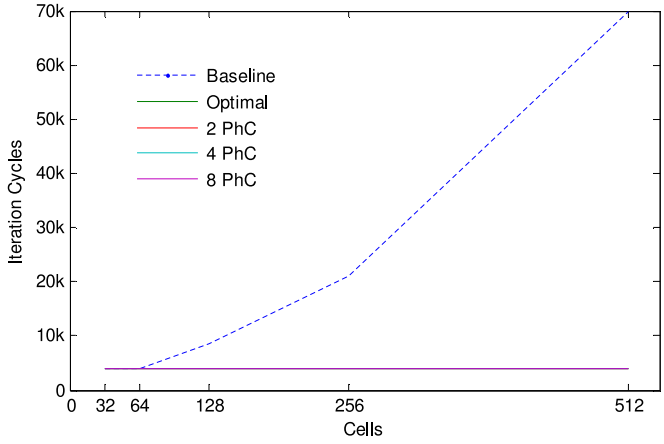


Fig. 13. Comparison between different cluster sizes for different amounts of cells. The neighbor connection scheme is used and each cell calculation takes 534 cycles. The systems use a router fan-out of two. The baseline design is from [14]; all presented configurations of the new system meet the brain real time at 100 MHz. Furthermore, all of them scale *linearly* with the number of cells. The baseline on the other hand scales *exponentially*.

compared to a golden-reference file, containing the expected values of the simulation. Observed error is very low, less than $0.2 \times 10^{-5}\%$ for cell resting state (when most internal cell variables change rapidly), and at cell firing state, for both 32 and 64-bit configurations. *Design Space Exploration:* For the router performance, the time when delayed packets get injected into the network is essential: we compared three injection models (Fig. 10), i.e., packets are injected after each run through all input FIFO, packets are only injected if the FIFO have been empty for a complete round, and packets are injected if there has been no activity on the inputs for 10 rounds.

Injection after each iteration is not feasible and results in very long iteration times. Fig. 11 highlights the different fan-out in respect to cluster sizes. Small clusters with small routers and bigger clusters with large fan-out (>8) provide the best overall performance. The best overall performance is achieved by 2 *PhC* with a router fan-out of 2 with 4012 cycles. The cluster size choice is illustrated in Fig. 12. All routers are kept at a fan-out of 2 (i.e., optimal fan-out as shown in Fig. 8). Due to the fact that all the *PhC* of a cluster time-share a memory, the clusters with more *PhCs* perform worse, especially at a higher number of cells: the difference for 2048 cells between 2 *PhCs* clusters and 18 *PhC* clusters is 30%.

Single FPGA Performance: Comparison between different cluster sizes for different amounts of cells is illustrated in Fig. 13. The absolute difference in execution time for 32 to 512 cells is 354 cycles, and 191 cycles in the worst case (2 *PhCs* system). While the proposed system scales *linearly* with the number of cells in the system, the baseline [14] scales *exponentially*. Considering an average increase in run-time of 120 cycles for the twice the amount of cells, the maximum number of *PhC* on a single chip can be estimated as $\psi = (c_n - c_o)/120$, where c_n is the number of iteration cycles available for one iteration, i.e., at 100 MHz within the brain real-time boundary[4] it leads to 5000

[4]The maximum number of cell states that can be computed within the model (in the case of the evaluated, high-detail inferior-olive model, the simulation time step is 50 μs [29].

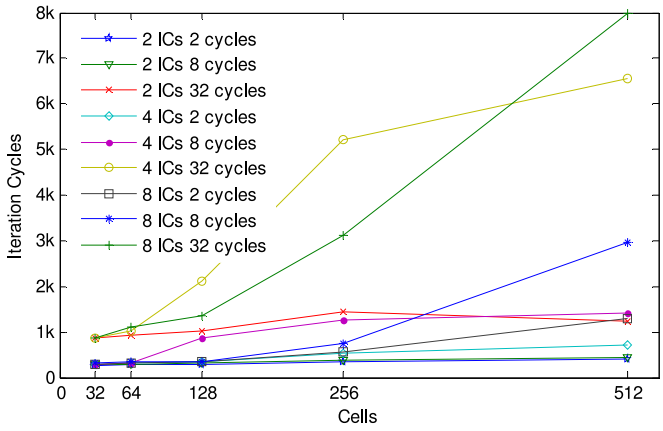


Fig. 14. Comparison between different system configurations utilizing between one and eight chips; the multichip systems utilize the *packet* based synchronization method.

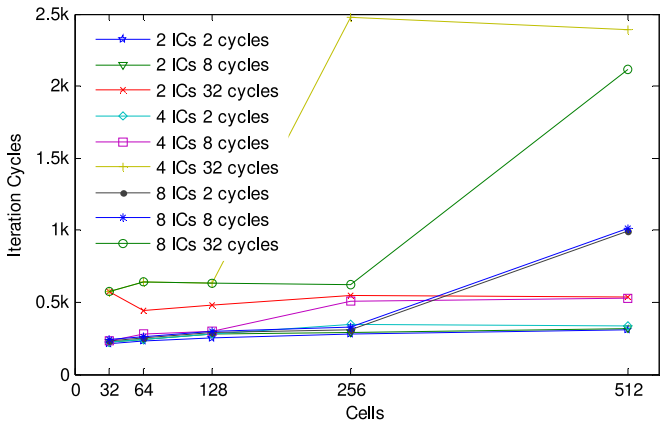


Fig. 15. Comparison between different system configurations utilizing between one and eight chips; multichip systems utilize the *dedicated wire* based synchronization method.

cycles, and c_o stands for the amount of cycles currently used, i.e., 4372 cycles at 512 cells. The value received, ψ, denotes the amount of times the number of cells in the system can be doubled: $\psi = 5.23$. Thus, the system supports more than 19200 cells on one chip for the neighbor connection scheme. For normal connection mode, the system requires 4597 cycles at 512 cells. The increase in iteration time for the twice the amount of cells is 155, leading to $\psi = 2.6$ doublings, and hence, more than 3000 cells on one chip.

Multi FPGA Performance: Comparisons are performed using both the dedicated wire, as well as the packet-based synchronization methods. In contrast to the packet-based approach, in a dedicated controller method that is connected with each chip, no packet has to cross multiple chip boundaries. For the normal connection scheme the dedicated controller performs slightly better than the packet based controller (Fig. 14). The dedicated controller system is 3.84 times faster than the packet based approach at 8 chips with very slow communication (Fig. 15). For faster connection speeds on the other hand the difference in communication time is in the order of 1%.

V. Conclusion

Current neuron simulators, which are precise enough to simulate neurons in a biophysically-meaningful way, are limited in amount of neurons to be placed on the chip, the interconnect between the neurons, run-time configurability and the re-synthesis of the system. In this paper, we propose a system that is able to bridge the gap between biophysical accuracy and large numbers of cells (19200 cells for neighbor connection mode and over 3000 cells in normal connection mode). The cells are grouped around a shared memory in clusters to allow for instantaneous communication. Clusters that are close communicate using only one hop in the network; clusters that are further away communicate less frequently and, consequently, the penalty for taking multiple hops is less severe. Added advantage is that the system can be extended over multiple chips without significant performance penalty. This combination of clusters and a tree topology network-on-chip allows for almost linear scaling of the system. To provide run-time configurability, a tree-based communication bus is used, which enables the user to configure the connectivity between cells and change the parameters of the calculations. As a result, re-synthesizing the whole system just to experiment with a different connectivity between cells is not required. The user has to enter the amount of neurons in the system as well as the desired connectivity scheme. From this information, all required routing tables and topologies are automatically generated, even for multi-chip systems. A porting the network to the FPGA yields at least several thousand simulation speed-up in comparison with SystemC simulation, with negligible loss of accuracy.

References

[1] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge Univ. Press, 2002.
[2] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE Trans. Neural Networks*, vol. 15, no. 5, pp. 1063–1070, Sep. 2004.
[3] W. Maass, "Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons," in *Neural Information Processing Systems*, M. Mozer *et al.*, ed. Cambridge, MA, USA: MIT Press, 1997, pp. 211–217.
[4] W. McColloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.
[5] K. Cheung, S. R. Schultz, and W. Luk, "A large-scale spiking neural network accelerator for FPGA systems," in *Proc. Int. Conf. Artif. Neural Netw. Mach. Learn.*, 2012, pp. 113–120.
[6] D. Pani *et al.*, "An FPGA platform for real-time simulation of spiking neuronal networks," *Frontiers Neurosci.*, vol. 11, pp. 1–13, 2017.
[7] H. Shayani, P. J. Bentley, and A. M. Tyrrell, "Hardware implementation of a bio-plausible neuron model for evolution and growth of spiking neural networks on FPGA," in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, 2008, pp. 236–243.
[8] G. Smith, C. Cox, S. Sherman, and 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.
[9] G. B. Ermentrout, "Type I membranes, phase resetting curves, and synchrony," *Neural Comput.*, vol. 83, pp. 979–1001, 1996.
[10] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.*, vol. 117, no. 4, pp. 500–544, 1952.
[11] Y. Zhang *et al.*, "Biophysically accurate floating point neuroprocessors for reconfigurable logic," *IEEE Trans. Comput.*, vol. 62, no. 3, pp. 599–608, 2013.
[12] S. Y. Bonabi *et al.*, "FPGA implementation of a biological neural network based on the Hodgkin-Huxley neuron model," *Frontiers Neurosci.*, vol. 8, pp. 1–12, 2014.

[13] M. Beuler *et al.*, "Real-time simulations of synchronization in a conductance-based neuronal network with a digital FPGA hardware-core," in *Proc. Int. Conf. Artif. Neural Netw. Mach. Learn.*, 2012, pp. 97–104.
[14] M. van Eijk, C. Galuzzi, A. Zjajo, G. Smaragdos, C. Strydis, and R. van Leuken, "ESL design of customizable real-time neuron networks," in *Proc. IEEE Int. Biomed. Circuits Syst. Conf.*, 2014, pp. 671–674.
[15] N. Qiao *et al.*, "A re-configurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128 k synapses," *Frontiers Neurosci.*, vol. 9, pp. 1–17, 2015.
[16] J. Luo, G. Coapes, T. Mak, T. Yamazaki, C. Tin, and P. Degenaar, "Real-time simulation of passage-of-time encoding in cerebellum using a scalable FPGA-based system," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 3, pp. 742–753, Jun. 2016.
[17] B. V. Benjamin *et al.* "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.
[18] A. Andreou and K. Boahen, "Synthetic neural circuits using current-domain signal representations," *J. Neural Comput.*, vol. 1, no. 4, pp. 489–501, 1989.
[19] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.
[20] B. U. Pedroni *et al.*, "Mapping generative models onto a network of digital spiking neurons," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 4, pp. 837–854, Aug. 2016.
[21] J. Navaridas *et al.*, "Understanding the interconnection network of SpiNNaker," in *Proc. Int. Conf. Supercomput.*, 2009, pp. 286–295.
[22] G. Smaragdos *et al.*, "Real-time olivary neuron simulations on dataflow computing machines," in *Supercomputing* (*Lecture Notes in Computer Science*), J. M. Kunkel, T. Ludwig, and H. W. Meuer eds. Cham, Switzerland: Springer, 2014, pp. 487–497.
[23] G. Smaragdos *et al.*, "FPGA-based biophysically-meaningful modeling of olivocerebellar neurons," in *Proc. Int. Symp. Field Programmable Gate Arrays*, 2014, pp. 89–98.
[24] H. D. Nguyen, Z. Al-Ars, G. Smaragdos, and C. Strydis, "Accelerating complex brain-model simulations on GPU platforms," in *Proc. IEEE Des., Autom., Test Europe Conf.*, 2015, pp. 974–979.
[25] Y. Zhang *et al.*, "A biophysically accurate floating point somatic neuroprocessor," in *Proc. IEEE Int. Conf. Filed Programmable Logic Appl.*, 2009, pp. 26–31.
[26] P. Bazzigaluppi *et al.*, "Olivary subthreshold oscillations and burst activity revisited," *Frontiers Neural Circuits*, vol. 6, no. 91, pp. 1–13, 2012.
[27] J. Hofmann, A. Zjajo, C. Galuzzi, and R. van Leuken, "Multi-chip dataflow architecture for massive scale biophysically accurate neuron simulation," in *Proc. Int. Conf. IEEE Eng. Med. Biol. Soc.*, 2016, pp. 5829–5832.
[28] C. I. De Zeeuw *et al.*, "Spatiotemporal firing patterns in the cerebellum," *Nature Rev. Neurosci.*, vol. 12, no. 6, pp. 327–344, 2011.
[29] J. R. de Gruijl *et al.*, "Climbing fiber burst size and olivary subthreshold oscillations in a network setting," *PLoS Comput. Biol.*, vol. 8, no. 12, pp. 1–10, 2012.
[30] G. Smaragdos *et al.*, "Performance analysis of accelerated biophysically-meaningful neuron simulations," in *Proc. Int. Symp. Perform. Anal. Syst. Software*, 2016, pp. 1–11.
[31] G. J. Christiaanse, A. Zjajo, C. Galuzzi, and R. van Leuken, "A real-time hybrid neuron network for highly parallel cognitive systems," in *Proc. Int. Conf. IEEE Eng. Med. Biol. Soc.*, 2016, pp. 792–795.
[32] G. Kahn, "The semantics of a simple language for parallel programming," in J. L. Rosenfeld ed.: Information Processing 74, *Proceedings of IFIP Congress 74*, 1974, pp. 471–475.
[33] C. Mehring, U. Hehl, M. Kubo, M. Diesmann, and A. Aertsen, "Activity dynamics and propagation of synchronous spiking in locally connected random networks," *Biol. Cybern.*, vol. 88, no. 5, pp. 395–408, 2003.
[34] C. M. Gray and 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.
[35] J. R. Gibson, M. Belerlein, and B. W. Connors, "Two networks of electrically coupled inhibitory neurons in neocortex," *Nature*, vol. 402, pp. 75–79, 1999.
[36] J. Lisman, "Bursts as a unit of neural information: Making unreliable synapses reliable," *Trends Neurosci.*, vol. 20, pp. 38–43, 1997.
[37] E. M. Izhikevich, N. S. Desai, E. C. Walcott, and F. C. Hoppensteadt, "Bursts as a unit of neural information: Selective communication via resonance," *Trends Neurosci.*, vol. 26, pp. 161–167, 2003.
[38] B. W. Connors and M. J. Gutnick, "Intrinsic firing patterns of diverse neocortical neurons," *Trends Neurosci.*, vol. 13, pp. 99–104, 1990.
[39] N. Schweighofer *et al.*, "Electrophysiological properties of inferior olive neurons: A compartmental model," *J. Neurophysiol.*, vol. 82, no. 2, pp. 804–817, 1999.
[40] AVNET. Avnet express. [Online]. Available: http://avnetexpress.avnet.com. Accessed on: Mar. 21, 2016.

Amir Zjajo received the M.Sc. and DIC degrees from Imperial College London, London, U.K., in 2000 and the Ph.D. degree from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 2010, all in electrical engineering.

In 2000, he joined Philips Research Laboratories as a member of the research staff with the Mixed-Signal Circuits and Systems Group. From 2006 until 2009, he was with Corporate Research of NXP Semiconductors as a Senior Research Scientist. In 2009, he joined Delft University of Technology, Delft, The Netherlands, as a Faculty Member with the Circuit and Systems Group. He has authored or coauthored more than 80 papers in referenced journals and conference proceedings, and holds more than ten U.S. patents or patents pending. He is the author of the books *Brain-Machine Interface: Circuits and Systems* (Springer, 2016), *Low-Voltage High-Resolution A/D Converters: Design and Calibration* (Springer, 2011, Chinese translation, China Machine Press, 2015), and *Stochastic Process Variations in Deep-Submicron CMOS: Circuits and Algorithms* (Springer, 2014). He cofounded Syntigent Technologies B.V. to commercialize bionic signal processing technology. His research interests include energy-efficient mixed-signal circuit and system design for health and mobile applications, data sense-making, sensor fusion, and neuromorphic electronic circuits for autonomous cognitive systems.

Dr. Zjajo serves as a member of Technical Program Committee of IEEE International Symposium on Quality Electronic Design, IEEE Design, Automation and Test in Europe Conference, IEEE International Symposium on Circuits and Systems, IEEE International Symposium on VLSI, IEEE International Symposium on Nanoelectronic and Information Systems, and IEEE International Conference on Embedded Computer Systems.

Jaco Hofmann received the B.Sc. degree in computer science from TU Darmstadt, Darmstadt, Germany, in 2012, and the M.Sc. degree in embedded systems from TU Delft, Delft, The Netherlands, in 2014. He is currently working toward the Ph.D. degree in computer science at TU Darmstadt. His research interests include application specific hardware accelerators, heterogeneous accelerator architectures, accessibility of FPGAs for researchers and software-defined networking.

Gerrit Jan Christiaanse received the M.Sc. degree in embedded systems from the Delft University of Technology, Delft, The Netherlands. His research interests include reconfigurable computing and multicore communication protocols.

Martijn van Eijk received the M.Sc. degree in computer engineering from the Delft University of Technology, Delft, The Netherlands. His research interests include blockchain-based electronic systems and crypto-currencies.

Georgios Smaragdos received the M.Sc. degree in computer engineering from the Delft University of Technology, Delft, The Netherlands. He is currently working toward the Ph.D. degree with the Department of Neuroscience, Erasmus Medical Center, Rotterdam, The Netherlands. His research interests include reconfigurable computing, fault-tolerant computing, and high-performance computational neuroscience.

Christos Strydis received the B.Sc. degree in electronics and computer engineering from the Technical University of Crete, Chania, Greece, in 2003, the M.Sc. degree (with hons.) in computer engineering with a minor in biomedical engineering in 2005, and the Ph.D. degree in computer engineering from the Delft University of Technology, Delft, The Netherlands, in 2011.

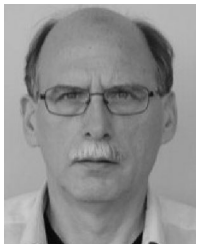
He is currently an Assistant Professor with the Department of Neuroscience, Erasmus Medical Center, Rotterdam, The Netherlands, and is also a Chief Engineer with Neurasmus BV, The Netherlands. He is the Head with the computer-engineering lab in the department and leads the Erasmus Brain Project effort. His research interests revolve around the topics of high-performance computational neuroscience and of next-generation implantable medical devices with a focus on implantable neuromodulators.

Dr. Strydis has acted as technical-program-committee member in various international conferences. He has also peer-reviewed, as well as authored or coauthored manuscripts in well-known international conferences and journals.

Alexander de Graaf received the M.Sc. degree in electrical engineering from the Delft University of Technology, Delft, The Netherlands, in 1983. Since 2010, he has been a Senior Design Engineer with the Delft University of Technology. His research interests include neuromorphic design, neural networks, analog/mixed signal simulation, and low-power design.

Carlo Galuzzi received the M.Sc. degree (cum laude) in mathematics from the Department of Mathematics, University of Milan (in Italian: Universita' degli Studi di Milano, "Statale"), Milan, Italy, in 2003, and the Ph.D. degree in computer engineering from the Delft University of Technology, Delft, The Netherlands, in 2009.

From 2009 till 2016, he was a Postdoctoral Researcher with the same university. Since 2016, he has been an Assistant Professor with Maastricht University, Maastricht, The Netherlands. His research interests include instruction-set architecture customizations, reconfigurable and parallel computing, brain modeling, hardware/software codesign, mathematical modeling, graph theory, and design space exploration.



Rene van Leuken received the M.Sc. and Ph.D. degrees in electrical engineering from the Delft University of Technology, Delft, The Netherlands, in 1983 and 1988, respectively.

He is currently a Professor with the Circuit and Systems Group, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology (TU Delft), The Netherlands.

He has authored or coauthored papers in all major journals, conferences and workshops proceedings, and has received several best paper awards over the years. His research interests include high-level digital system design, system design optimization, VLSI design, and high performance compute (DSP) engines. His major research activity is neuromorphic computing.

Dr. van Leuken has been involved in many major research and development projects: ESPRIT, FP6, FP7, JESSI, MEDEA, and recently in ENIAC/CATRENE, and ARTEMIS projects. He is member of the PATMOS steering committee and the DATE Technical Program Committee.