FPGA-based Biophysically-Meaningful Modeling of Olivocerebellar Neurons

Georgios Smaragdos[1], Sebastian Isaza[2], Martijn Van Eijk[3], Ioannis Sourdis[4] and Christos Strydis[1]

[1]Dept. of Neuroscience, Erasmus Medical Center, Rotterdam, The Netherlands
[2]Dept. de Ingeniería Electrónica, Universidad de Antioquia UdeA, Medellín, Colombia
[3]Faculty of Electrical Engineering, Mathematics & Computer Science, Delft University of Technology, Delft, The Netherlands
[4]Dept. of Computer Science & Engineering, Chalmers University of Technology, Gothenburg, Sweden
{g.smaragdos, c.strydis}@erasmusmc.nl
sisaza@udea.edu.co M.F.vanEijk@student.tudelft.nl sourdis@chalmers.se

ABSTRACT

The Inferior-Olivary nucleus (ION) is a well-charted region of the brain, heavily associated with sensorimotor control of the body. It comprises ION cells with unique properties which facilitate sensory processing and motor-learning skills. Various simulation models of ION-cell networks have been written in an attempt to unravel their mysteries. However, simulations become rapidly intractable when biophysically plausible models and meaningful network sizes ($\geq$100 cells) are modeled. To overcome this problem, in this work we port a highly detailed ION cell network model, originally coded in Matlab, onto an FPGA chip. It was first converted to ANSI C code and extensively profiled. It was, then, translated to HLS C code for the Xilinx Vivado toolflow and various algorithmic and arithmetic optimizations were applied. The design was implemented in a Virtex 7 (XC7VX485T) device and can simulate a 96-cell network at real-time speed, yielding a speedup of $\times$700 compared to the original Matlab code and $\times$12.5 compared to the reference C implementation running on a Intel Xeon 2.66GHz machine with 20GB RAM. For a 1,056-cell network (non-real-time), an FPGA speedup of $\times$45 against the C code can be achieved, demonstrating the design's usefulness in accelerating neuroscience research. Limited by the available on-chip memory, the FPGA can maximally support a 14,400-cell network (non-real-time) with online parameter configurability for cell state and network size. The maximum throughput of the FPGA ION-network accelerator can reach 2.13 GFLOPS.

Categories and Subject Descriptors

J.3 [**LIFE AND MEDICAL SCIENCES**]: [Biology and genetics]

General Terms

Design, Performance, Experimentation

Keywords

Spiking Neural Networks; Computational Neuroscience; Inferior Olive; Cerebellum; Hodgkin Huxley

1. INTRODUCTION

Artificial Neural Networks (ANNs) have been successfully used in robotics and artificial-intelligence applications on numerous occasions in the past [7]. Contrary to the typical, Von-Neumann-based computing model, a biologically-inspired ANN does not execute explicit sequential instructions to solve its computational problems. In such a network, functions across nodes (or neurons) are evaluated concurrently and the relation between the input and output of the ANN is determined by the network topology and method of interconnectivity. This topology can also be dynamically adaptive, subject to the ongoing neural computations, thus mimicking biological behavior known as plasticity.

Advances in neuroscience and greater understanding of the brain have gradually led to the creation of mathematical models of neurons and whole networks that do not simply mimic biological behavior in an abstract way but simulate it with significant detail. Such an example is the Spiking Neural Network (SNN). Here, information is not just transferred by the firing rate of each neuron in the network as is the case in classical ANNs but by the transfer of spikes [19]. Due to the SNNs' ability to model additional neuron characteristics and adapt them according to spike-train amplitude, frequency and precise arrival times, SNNs can have greater computational and predictive power than Artificial-Neural-Networks (ANNs) [14]. This, alongside with the advances of technology in computer science and engineering, has opened the possibility of implementing larger-scale NNs that have the ability to more accurately simulate brain behavior.

The United-States-National-Academy of Engineers has classified brain simulation as one of the Grand Engineering Challenges [16]. Biologically accurate brain simulation is a highly relevant topic for *neuroscience* for a number of reasons: *(1) Accelerated brain research:* Neuroscientists plan to gain greater understanding of brain behavior by simulations based on biologically accurate models. Depending on the complexity of the model, it can provide insights ranging from single-cell behavior to network dynamics of whole brain regions without having to perform in-vivo experiments. *(2) Brain rescue:* If brain functions can be simulated accurately enough and in real-time, this can lead (a) Medium-turn, to seamless brain-rescue devices with various medical and other applications; and (b) long-term, to robotic prosthetics and implants for restoring lost brain functionality in patients. *(3) Advanced A.I.:* ANNs have already been successfully used in this field even though they have not even remotely reached the computational capacity of biological systems. It is believed that greater understanding of biological systems and their richer computational dynamics can lead to more advanced, bio-inspired, artificial-intelligence (AI) models for autonomous and robotic applications. *(4) New computer-architecture paradigms:* Alternatives to the typical Von-Neumann architectures can be very useful for massively parallel applications and could potentially provide defect-tolerant systems emulating the brain's adaptability.

The main challenge in building complex, biologically accurate SNNs lies largely in the computational and communication load of the network simulations. Furthermore, biological NNs execute these computations with massive parallelism, something that conventional CPU-based execution cannot cope with very well. As a result, the neuron-population size and interconnectivity are quite low when running on PCs (with models implemented in MATLAB or neuromodeling languages such as NEURON and GENESIS). This greatly impedes the efficiency of brain research in relation to the goals of brain simulation stated above.

A good alternative would be the execution of neuron models in *GPUs*. GPUs can exploit application parallelism better and, thus, can be more efficient in running neuron models. Yet, in the case of complex models or very large-scale networks, they may not be able to provide real-time performance (i.e. to produce results for every network cell within the time allotted by the model simulation step) due to the high rates of data exchange between the neurons [8]. Moreover, GPUs are inefficient in terms of energy and power.

Another alternative would be the use of *supercomputers*. Although these systems can emulate the behavior and parallelism of biological networks with sufficient speed, the sheer size and complexity of these solutions makes them useful only for behavioral simulations. Supercomputer deployments require immense space, implementation, maintenance and energy costs while lacking any kind of mobility.

Mixed-signal VLSI is another option for simulating SNNs. Such designs achieve adequate simulation speeds while simulating the biological systems more accurately since they model neurons through analog signals, just like their biological counterparts. However, mixed-signal VLSI designs are much more difficult to implement, lack flexibility and often suffer from problems typical in analog design; for instance, accuracy issues and reduced reproducibility of results due to transistor mismatching, leakage currents, crosstalk etc. [3].

Implementing the neural network in parallel, digital hardware can efficiently match the parallelism of biological models and provide real-time or hyper-real-time performance useful for simulations, prosthetics and robotics applications. While ASIC design is certainly an option, it is expensive, time-consuming and – most importantly – inflexible: An ASIC cannot be altered after fabrication, yet model changes often required in *fitting* novel neuron models would require a new development cycle, just like mixed-signal VLSI.

Most of these issues can be tackled through the use of FPGAs. FPGAs, although slower than ASICs, still provide enough performance for real-time and hyper-real-time neuron simulations, exploiting the inherent parallelism of hardware. Besides requiring a lot less energy and, in some cases, less space than most of the above solutions for the same computational power, the reconfiguration property of FPGAs provides the flexibility of modifying brain models on demand. This flexibility is substantially enhanced by the use of high-level synthesis tools which speed up the development process; manually developing the hardware description every time a new brain model is released or an existing one is re-calibrated would be impractical and time-consuming. Besides, dynamic reconfigurability can provide a way to emulate the plasticity of biological neural networks in ways other solutions cannot.

In this paper we present an FPGA-accelerated application for a specific, biophysically-meaningful NN model, using single-precision floating-point (FP) arithmetic computations. The application models the network of one of the essential areas for the function of the cerebellum, the Inferior Olive. Concisely, the contributions of this work are:

- The analysis of the Inferior Olive model to recognize characteristics and possible performance bottlenecks.
- The optimization of the original model algorithm achieving $\times 2$ performance improvement in hardware.
- The design, implementation and validation of the FPGA-based accelerator, achieving $\times 45$ speedup compared to C and 96 neurons simulated at real-time.
- The performance evaluation of the designed accelerator and evaluation of precision error, issued by design optimizations, to guarantee preservation of the biological behavior reflected in the original model.

The rest of the paper is organized as follows: Section 2 presents background information necessary to tackle the neuromodeling problem at hand. Section 3 gives an overview of related works in the field. Section 4 covers the description of the Inferior Olive Model and presents application profiling results. Section 5 describes the FPGA-based Inferior Olive design and architecture. Section 6 presents the area, performance and error evaluation, while also presenting a comparison to other FPGA-based SNN applications. Finally, Section 7 concludes this paper.

2. BACKGROUND

In this section, background information on the biological structures of interest and the modeling approaches thereof will be presented. This information will help to better grasp the implementation and evaluation phases of this work.

2.1 The Biological Neuron

Neurons are electrochemically excitable cells[1] that process and transmit signals in the brain. The biological neuron comprises in general (although, in truth is a much more complicated system) three parts (called compartments in neuromodeling jargon): The *Dendrites*, the *Soma* and the *Axon*. The dendritic compartment represents the cell input stage. The dendrites pick up electrochemical stimuli from other cells and transfer them to the soma. In turn, the soma processes the stimuli and translates them into a cell *membrane potential*, which evokes a cell response called an *action potential* or, simply, a *spike*. This response is transferred through the axon, which is the cell output stage, to

[1]We will use the terms *neuron* and *(neural) cell* interchangeably throughout the paper.

other cells. An electrochemical connection between two cells (axon-dendrite) is referred to as a *synapse*.

2.2 The Cerebellum and the Inferior Olive

The cerebellum is one of the most complex and tightly packed regions in the brain, playing an important role in sensorimotor control. It does not initiate movement but influences the sensorimotor region in order to precisely coordinate the body's activities and motor learning skills. It also plays an important role in the sensing of rhythm, enabling the handling of concepts such as music or harmony.

The Olivocerebellar circuitry – of which the Inferior Olivary Nucleus (ION) is a part – is a relatively well-charted region of the cerebellum [6]. The ION (comprising the so-called ION cells) provides one of the two main inputs to the cerebellar system: the climbing fibers. ION cells are also interconnected by purely electrical connections between their dendrites, called *gap junctions*, considered to be important for the synchronization of activity within the nucleus and, thus, greatly influencing movement and motor learning.

2.3 Neural Modeling

There is a number of mathematical models describing the behavior of spiking-neuron compartments and neuron networks [13]. The more complex the model, the better it can emulate biological dynamics and can more accurately represent its biological counterpart. The simplest models are the Integrate-and-Fire (IaF) which model very basic spiking behavior. An IaF model essentially implements a spike-integrator function. When the membrane potential surpasses a certain threshold the neuron fires an output spike.

A more advance neuron model is the Izhikevich [12] neuron. With only 2 equations and 4 parameters the model can recreate all main behavioral neuron patterns in terms of input/output behavior. But if the goal is to simulate and study the neuronal behavior in greater detail (like details of the ionic and chemical channels of the cell) or create multi-compartmental models, one has to use **biophysically-meaningful** models, like conductance-based models.

Conductance-based models are complex and dynamic representations of neurons that can even be biophysically meaningful; that is, models that can accurately model the internal mechanisms and state of a cell. The most important such model is the so-called Hodgkin-Huxley (HH) model dating back in 1952 [11]. The model uses 4 equations and dozens of parameters to describe in detail the electrical activity in the cell. This activity results from the combination of external changes in the membrane potential and in the internal concentrations of the main chemical components involved in the transmission of neural-signals: calcium, potassium and sodium. However, this high modeling accuracy comes at the cost of complexity. HH models tend to be orders of magnitude more computationally intensive compared to other types, making efficient simulation a challenge. As explained in the next sections, in this work we accelerate an extended HH model for Inferior-Olive neurons.

3. RELATED WORK

In the past, a number of designs have been proposed for the implementation of neuron and network models in FPGAs. In this section, we present some of the most notable past works in the field.

Shayani et al. [18, 17] proposed a neuron model using a Quadratic IaF model [10, 9] which enabled simulating extra dendritic and axonal properties. The system supported on-line network-topology adaptation. Each neuron had 16 synapses and the maximum network size was 161 neurons. The simulation ran at $\times 4210$ faster than real-time on a Virtex 5 FPGA. The authors estimated that they could simulate a little above 1000 neurons at real-time, provided that they utilized the whole FPGA chip.

Two of the most notable implementations using Izhikevich neurons are the designs proposed by Cheung et al. [5, 4] and by Moore et al. (Bluehive [15]). Each approach proposed an FPGA architecture for very large-scale SNNs which is event-driven for optimizing the network traffic and the assorted memory-bandwidth needs. This optimization is based on the fact that new neuron states do not need to be calculated at every single time step but only in the instances that a new input arrives at the neuron, as otherwise the neuron state would not change. To improve on their initial memory-bandwidth requirements [4], Cheung et al. replaced the FPGA board with a Maxeler Dataflow Machine [5]. This FPGA-based device features state-of-the-art memory systems that increase the bandwidth capabilities greatly compared to simple FPGA boards. As a result, the size of the implemented network achieved was 64K neurons, each having about 1000 synaptic connections to neighboring neurons. In the Bluehive device, Moore et al. took another approach: They used external DDR2 RAMs and built custom-made SATA-to-PCI connections for stacking FPGA devices for facilitating large SNN simulations. Only a small portion of data was stored on-board the FPGAs. In a Stratix IV FPGA, the authors simulated 64K Izhikevich neurons with 64M simple synapses at real-time performance.

These works (as most others in the field) have incorporated fixed-point arithmetic to implement the computation of their neuron models. Zhang et al. [20] have proposed a somatic and dendritic HH-accelerator processor using dedicated FP units. The FP units were custom-designed to provide better accuracy, resource usage and performance. The 32-bit FP arithmetic used in the model produced a neuroprocessor architecture which met a real-time-performance demand. The architecture also included advanced synapses. The system could simulate 4 complete cells (synapse, dendrite, soma) at real-time speed.

Beuler et al. [2] proposed a system for studying firing synchronization of NNs through gap junctions. They used a custom-made simplified model of a HH neuron. The main hardware component was the NepteronCore, also using 32-bit FP arithmetic. The system achieved real-time simulation speeds and included a simple GUI for parameter setup. The maximum size of the network was 400 neurons.

4. APPLICATION DESCRIPTION

The application studied in this paper models the behavior of the ION neurons as an important module of the olivocerebellar circuit. We have chosen this state-of-the-art model as a first step towards building a high-performance olivocrebellar simulation platform. The model not only divides the cells in multiple compartments but also creates a network onto which neurons are interconnected. Although the single-cell model is described first, the rest of the paper deals with the cell-network model, also described in the coming sections.

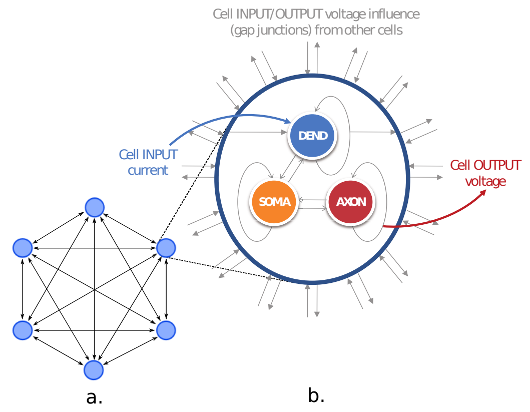

Cell INPUT/OUTPUT voltage influence (gap junctions) from other cells
Cell INPUT current
DEND
SOMA
AXON
Cell OUTPUT voltage
a.
b.

Figure 1: Illustration of (a) a 6-cell network-model example and (b) the 3-compartmental model of a single ION cell.

4.1 The ION-cell model

The ION cell model we have implemented has been originally developed by de Gruijl et al. [1]. The model is an extension of the original HH-based model [11] and divides the neuron into three computational compartments – closely resembling their biological counterparts – as shown in Figure 1(b). For every compartment, a few chemical channels are present in the model so as to contribute to the total compartment potential. Every compartment has a state that holds the electrochemical variables and, on every simulation step, the state is updated based on: i) the previous state, ii) the other compartments' previous state, iii) the other neuron's previous state and iv) the externally evoked input.

The computational model operates in a fashion that allows *concurrent* execution of the three compartments. The model is calibrated to produce every per-neuron output value with a 50 μsec time step. This means that, in order to support real-time simulations, all neurons are required to compute one iteration of compartmental calculations within 50 μsec. Due to the realistic electrochemical variables handled by the model, most of the computations require FP arithmetic.

4.2 The ION-network model

Figure 1(a) illustrates the network-model architecture with an example size of 6 cells. Every cell receives, through the dendritic compartment, the influence of all other cells in the network, thus modeling the massive biological gap junctions present in the Inferior Olive. For the analysis in the coming sections, we divide the dendritic compartment further into two parts to separate the gap-junction modeling from that of the pure dendritic computations. As illustrated in Figure 1(b), the dendrites in every cell also receive an externally evoked input current while the axonal voltage of all cells is considered the external output. The system works in lock-step computing discrete output values (with a 50 μsec time step) that, when aggregated in time, contribute to form the electrical waveform response of the system. The ION network must be synchronized in order to guarantee the correct exchange of cell state data when multiple cells and compartments are being computed simultaneously.

4.3 C-code profiling

The ION-network model was initially available to us in Matlab but was re-written in C for various analysis purposes and so as to be used in a High-Level Synthesis (HLS) tool. Next, we present profiling results that give insights on the distribution of computations performed in the model.

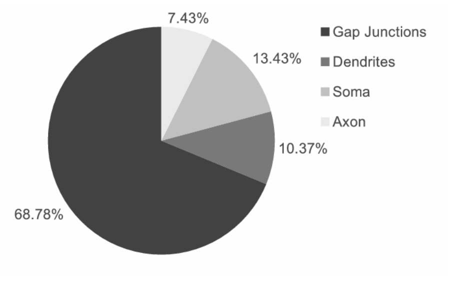


Figure 2: Software profiling of the computation time taken by the various compartments in the ION-cell model.

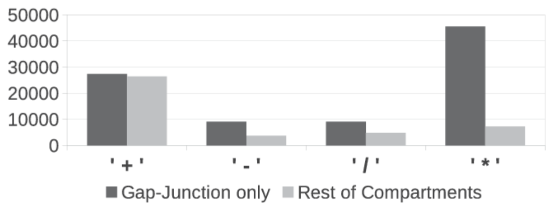


Figure 3: Software profiling of the arithmetic operations in the model for a 96-cell, fully interconnected network.

Figure 2 shows the execution-time distribution of the various model compartments (program functions). The numbers presented were obtained with GProf on an Intel Core i7 3770 (3.4 GHz, 8GB RAM, Ubuntu 12.04 OS). The gap-junction computations are shown separately from the dendrites. This division has been made in order to stress the fact that, in this fully connected ION-network model, gap-junctions computations take up most of the execution time. The reason is that they need to be repeated as many times as the network size for every individual neuron being simulated. Moreover, it should be noted that the total amount of gap-junction computations in the network grows exponentially with the number of cells. Next in terms of computational load is the Soma compartment due to the multiple and complex electrochemical channels that are modeled. Last is the Axon compartment because it only models two such channels.

Figure 3 shows the profiling of the arithmetic operations performed in the model, for one iteration of the whole ION-network. We differentiate the operations belonging to the gap-junction compartment from the rest again, to show its importance. Results show that (FP) multiplications performed in the gap-junction compartment dominate the distribution. Finally, we can see that the gap junctions contain the largest fraction of operations for all operation types and will, therefore, consume more FPGA resources.

5. FPGA-BASED OLIVOCEREBELLAR NEUROMODELING

After profiling the C code based on the Matlab model, we used it as the basis for generating the proposed hardware solution using the Vivado HLS tool. The resulting hardware accelerator simulates the behavior of multiple ION-cells *step by step* based on the aforementioned model. The hardware accelerator is designed to work alongside a softcore or host CPU that controls the total number of simulation steps and handles the I/O of the accelerator. The CPU feeds the accelerator with initialization data (initial state) and with evoked-current inputs (external stimuli of neurons) and outputs the result of the computations at every simulation step. Output data can be stored in on-board memory (e.g., SD cards) or sent to an off-board PC host.

Both, neuron states and evoked inputs – required at every simulation step – are stored in on-chip BRAMs, so as to avoid incurring off-chip latency. The performance benefit of using on-chip storage is substantial compared to going off-chip, especially for complex models such as ours, which require handling large amounts of data to represent the network state. On the other hand, this creates a constraint on the maximum network size that can be simulated, which depends on the storage capacity of the FPGA BRAMs.

The remainder of this section offers the details of our FPGA-based approach and the optimizations performed to improve the performance and area efficiency of our design.

5.1 Overview of the hardware design

The general block diagram of the proposed system can be seen in Figure 4. The actual execution is performed at the "ION Network" component, which consists of multiple identical parallel neuron-processing modules, each modeling the dendrite, soma and axon parts of a single ION cell. Our design further includes a set of BRAMs for storing the evoked inputs to the neurons as well as their state, which is updated after each simulation step. The execution flow of the ION network is controlled by a – local to the accelerator – kernel control unit. Our actual implementation of the ION network consists of eight hardware neuron-processing modules, which are able to simulate eight ION-cells in parallel.

The accelerator was designed to give run-time control over a number of simulation parameters, providing flexibility and the ability for more complex experiments. During execution, each neuron state parameter can be modified. Interconnectivity density is also adjustable during simulation.

Next, we describe the functionality of our FPGA-based accelerator. First, the neurons in the network are initialized with data streamed from the CPU to the FPGA. The initialization data are either produced by the CPU itself or read by on- or off-board resources. This introduces a delay (discussed in Section 6) which is however paid only once at the ION-network simulation onset.

After initialization, the actual execution of the network simulation is performed. Each simulation step begins with storing new evoked inputs of the neurons in BRAM, representing the network external input vector. Following the storing of this vector, the kernel-control copies to dedicated BRAM banks part of the other cells' state (the dendritic voltages) needed for computing the gap-junction effect. Each hardware neuron-processing module has a separate dual-port, BRAM bank to store its respective gap-junction data. By making this design choice, we improve the memory bandwidth during the gap-junction processing and allow the HLS-tool scheduling techniques to maximize parallelism. This would not be possible if both compartment and gap-junction logic shared the same memory banks.

With all input data ready, the next state of each neuron is computed. Each hardware neuron-processing module

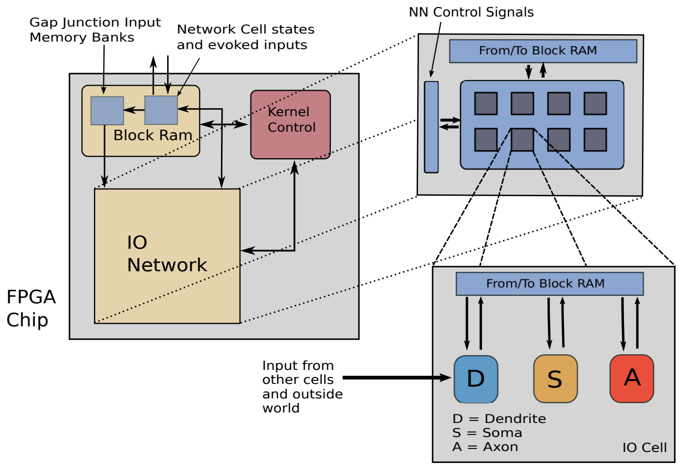


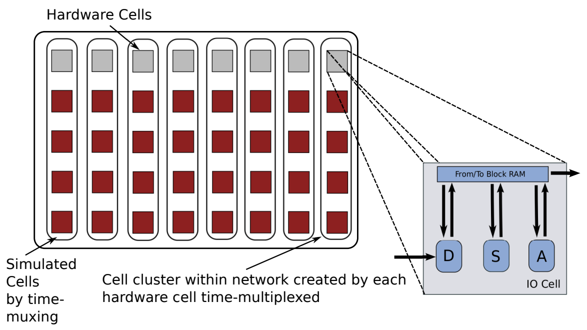
Figure 4: Block diagram of the Olivocerebellar neuromodeling hardware design.

executes in parallel. It is worth noting that all three compartments (dendrite, soma, axon) within a neuron module could execute in parallel, as they have no dependencies with each other. In practice, the axon and soma execute sequentially (soma first, axon second) to save on resources, while the dendrite compartment executes concurrently with the axon and the soma. That is due to the execution time of the dendrite which is longer than that of the axon and soma compartments combined. The final phase of the execution involves storing the newly produced ION-cell states in the BRAMs, to be used in the next simulation step, and streaming the output values needed by the experiment outside the accelerator. For our test cases, this output is the axon voltage of each neuron for every simulation step, representing the ION-network response. Presumably, the output could be any part of the neuron states, depending on the requirements of the neuroscientific experiment.

5.2 Time-multiplexing execution

For the accelerator to achieve real-time performance, each simulation step must be completed within the same time window of 50 μsec. Obviously, such a "real-time" constraint does not have a counterpart in biological neurons (that feature continuous function). It is imposed by our ION network simulator which is a self-contained, fixed-timestep, transient simulator – similar to most HH-based simulators – with a constant step $\Delta t = 50\mu sec$ in our case. Respecting this time-step duration is essential for generating biologically-plausible signals that can be interfaced to living tissue.

Of course, our hardware neuron network (and hardware modules in general) runs significantly faster than the real-time constraint at hand. We exploit this *latency slack* by using our hardware resources more efficiently and maximizing the number of simulated neurons by *time-multiplexing* of hardware blocks. More precisely, we use the same hardware neuron-processing module multiple times within a simulation step to compute states of different simulated neurons. As illustrated in Figure 5, each hardware neuron-processing module evaluates multiple simulated neurons that together comprise the total simulated network. By online adjusting the number of simulated cells each hardware neuron is simulating (i.e. the time-multiplexing factor), the net-



Hardware Cells
From/To Block RAM
D
S
A
IO Cell
Simulated Cells by time-muxing
Cell cluster within network created by each hardware cell time-multiplexed

Figure 5: Time multiplexing of hardware neurons.

work size can be altered without re-synthesizing the hardware kernel, even during the simulation, if experiments indeed require it (for instance, to emulate synaptic plasticity). This is achieved by storing different input vectors and cell states for each simulated neuron evaluated in each hardware neuron-processing module. However, the input vectors and cell states need to be stored in the BRAM; this ultimately means that the maximum network size shall be *constrained by the amount of available of on-chip memory.* The BRAMs are statically allocated before synthesis to support the maximum number of possible simulated cells at runtime.

5.3 HLS C-Code Optimizations

A number of optimizations for increasing the efficiency and performance of the hardware design were implemented in the C code, motivated by code inspection and the profiling information presented in Section 4.3. According to profiling results of the reference C code, the most computationally intensive compartment in the model is the dendrite, more specifically, the gap-junction computations. These are responsible for accumulating the influence of all other neurons in the network and include complex arithmetic operations such as FP exponents and divisions performed for every other cell state, as shown in Listing 1. In such an all-to-all interconnected network, the amount of gap-junction computations increases *exponentially* with the network size.

Listing 1: Original gap-junction code.

```
for (i=0; i<ION_N_INPUT; i++) {
  V = prevVdend - neighVdend[i];
  f = 0.8 * exp(-1 * V * V/100) + 0.2;
  Ic = Ic + (CONDUCTANCE * f * V);
}

return Ic ;
```

Without changing the actual functionality described in Listing 1, we rewrote and simplified the gap-junction code. As shown in Listing 2, we removed from the for-loop any operations that are common for all iterations, thus reducing the required computations substantially. More precisely, the mathematical expression implemented by the code in Listing 1 was modified as shown in Formula (1). In other words, we removed computations simulating the total gap-junction influence (Ic) from the accumulation loop, saving three multiplications and one addition per for-loop iteration. In the optimized code, the gap junctions accumulate only the input parameters of Ic and computes the total influence only once, after the accumulation has been completed. This modification yielded a notable increase in the network size supported by our design for real-time simulations.

Design	Area	Real-Time Network Size	One Cell Latency
Baseline	99% of LUTs	48 cells	603 cycles
Opt1	99% of LUTs	84 cells	347 cycles
Opt2	96% of LUTs	96 cells	333 cycles
Opt3	91% of LUTs	96 cells	323 cycles

Table 1: Synthesis Estimation for each optimization case with Vivado HLS 2013.2 for a Virtex 707 evaluation board. Opt1: Gap-junction calculations' optimizations. Opt2: Division-by-constant replacement in dendritic compartment. Opt3: Division-by-constant replacement in all 3 compartments.

Listing 2: Optimized gap-junction code.

```
for (i=0; i<ION_N_INPUT; i++) {
  V = prevVdend - neighVdend[i];
  f_new = V * exp(-1 * V * V/100);
  F_acc =+ f_new;
  V_acc =+ V;
}

Ic = CONDUCTANCE * (0.8*F_acc + 0.2*V_acc);
    return Ic ;
```

$$\begin{aligned}
Ic &= \sum_{i=0}^{i=N-1} (C * f_i * V_i) \\
&= C * \sum_{i=0}^{i=N-1} (0.8 * \exp(-1 * V_i * \frac{V_i}{100}) + 0.2) * V_i) \\
&= C * (0.8 * \sum_{i=0}^{i=N-1} [V_i * \exp(-1 * V_i * \frac{V_i}{100})] + 0.2 * \sum_{i=0}^{i=N-1} *V_i) \\
&= C * (0.8 * \sum_{i=0}^{i=N-1} f\_new_i + 0.2 \sum_{i=0}^{i=N-1} *V_i) \\
&\text{where } C \text{ is the CONDUCTANCE and } N \text{ the ION\_N\_INPUT.}
\end{aligned} \tag{1}$$

A second modification in the original code that helped increase both performance and area efficiency was the replacement of any division-by-constant with an arithmetic equivalent (but less computationally intensive) multiplication-by-constant (e.g. $\frac{A}{100} \Leftrightarrow A * 0.01$). In computer arithmetic, the above modifications can introduce precision error in the computations performed; in the evaluation section we measure the effect of our optimization in the quality of the simulations. As shown next, the ION-model computations have a large number of divisions-by-constant operations the replacement of which can influence both area and performance without introducing a significant precision error that would affect correct model behavior. This optimization had to be performed manually as the HLS tool does not support it automatically so as to avoid introducing potential precision error without the developer's consent.

In Table 1, we can see the performance and area benefits for the application, for each code modification. Opt1 denotes the gap-junction code modifications. The other two optimizations refer to the replacement of divisions-by-constant with multiplications-by-constant. Opt 2 replaces divisions only in the slowest part of the model (dendrite compartment), while Opt3 in the entire code. We initially attempted replacing the divisions only in the dendrite, since our main concern is performance while making sure that the arithmetic error would not be significant. As Opt1 and Opt2 had

Design	Speed-up
C Code – Double Floats	×58.64
C Code – Single Floats	×60.82
FPGA Accelerator	×731.23

Table 2: Speed-up of C implementations and the FPGA-based accelerator compared to original Matlab code for simulating a real-time network.

only favored the dendrite/gap-junction compartments, Opt3 was eventually also deemed useful as the balance changed and it lead to an extra performance benefit.

Overall, these modifications achieved an almost 50% decrease in single-neuron execution latency, doubling the maximum network size able to be simulated at real-time speed. There is also some area improvement which is not substantial due to the fact that both multiplications and divisions use in – most cases – the same number of DSP slices.

6. EVALUATION

We evaluate, next, the performance and area cost of our proposed approach and measure its speedup compared to a software implementation. Moreover, we estimate the precision error after our modifications and, finally, discuss the efficiency of our approach compared to other related works.

6.1 Experimental methodology

The development of the Inferior-Olive design was performed using the Xilinx Vivado High Level Synthesis Tool (HLS v2013.2). The tool gives the ability to describe hardware IPs using a subset of ANSI C and then automatically handles production of the IP control logic, hardware scheduling of the operations and translation of the described design in SystemC, VHDL or Verilog code. Vivado HLS also supports algorithm validation using the C code, as well as integration with RTL simulators for validation of the produced HDL code. The tool actually provides the RTL simulation with the correct input vector according to C test-benches. This allows for explicit RTL hardware validation with testbenches simulating the complete CPU/IP system operation.

The ION-network design was translated to VHDL code using HLS and validated using QuestaSim 10.1 in RTL. Our testbench highlighted the basic behavior of the ION-model. All neurons are initialized with identical states, and left without any outside stimuli, remain synchronized with their axon voltage values oscillating. After 20,000 simulation steps, evoked current signals are issued to all neurons for 500 simulation steps. The ION neurons respond to these stimuli by producing a complex spike as seen in Figure 9(a) before returning back to their oscillating steady state. The testbench simulates 6 seconds of real brain time, taking 120,000 simulation steps to complete.

6.2 Experimental Results

The accelerator achieves *real-time execution* for a 96-neuron network with 100% (full) interconnection ratio at an operating frequency of 100 MHz[2] using a Virtex 7 XC7VX485T FPGA. In Table 2 we can see a performance comparison of the C code and the hardware accelerator against the original Matlab implementation; both the C-code and Matlab model

[2]The operating frequency is limited by the Xilinx IP blocks used in the design.

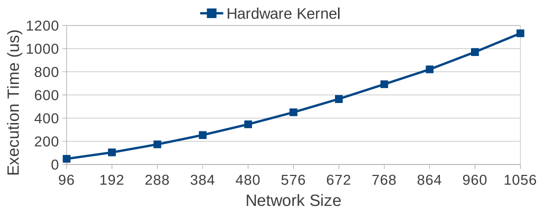


Figure 6: Accelerator step execution time for different network sizes.

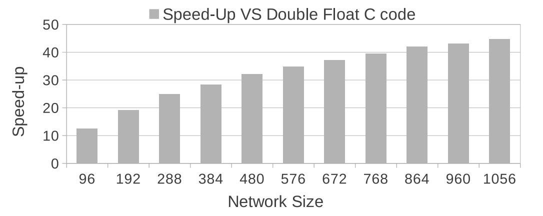


Figure 7: Accelerator performance comparison to double-FP C implementation.

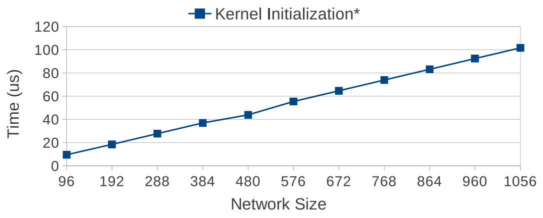


Figure 8: Initialization delay for different network sizes.

run on a Xeon 2.66GHz machine with 20GB RAM. The double-FP C implementation is about ×58 faster that Matlab, while the use of single-FP arithmetic gives a speedup of almost ×61. The FPGA ION-network kernel achieves an impressive ×731 speedup compared to the Matlab version and ×12.5 compared to the C implementation.

The on-chip memory (BRAM) resources available allow for *maximally* simulating a 14,440-cell network (non-real-time). Figure 6 plots the execution time of our designs for different network sizes. It can be observed that the execution time scales with the network size slightly worse than linearly due to the gap-junction computations which increase exponentially with the network size for an all-to-all interconnected network. However, this is still significantly better than execution-time trends in software. This point is better illustrated in Figure 7 which plots the FPGA-based speedup compared to the double-FP C implementation. As the network size increases above 96 cells, our FPGA-based simulation becomes slower than real-time, however it achieves an increasingly better speedup compared to the C implementation. This shows that the increasing gap-junction computations scale more gracefully in our parallel FPGA-based solution than in software. For a network of 96 cells, the speedup is about ×12.5 compared to the C code implementation and goes up to ×45 for a network size of 1,056 neurons.

Finally, the initialization delay also increases for higher network sizes, but in a linear fashion (Figure 8). It reaches a little over 100 μsec for a 1,056-neuron simulation. It should be noted that this time becomes proportionally smaller and even negligible for longer simulation times. Naturally, it

Area Component	Utilization	% of Available
LUTs	251485	83%
BRAMs	804	78%
FF	162217	27%
DSPs	1600	57%

Table 3: Area utilization for the Virtex 707 evaluation board.

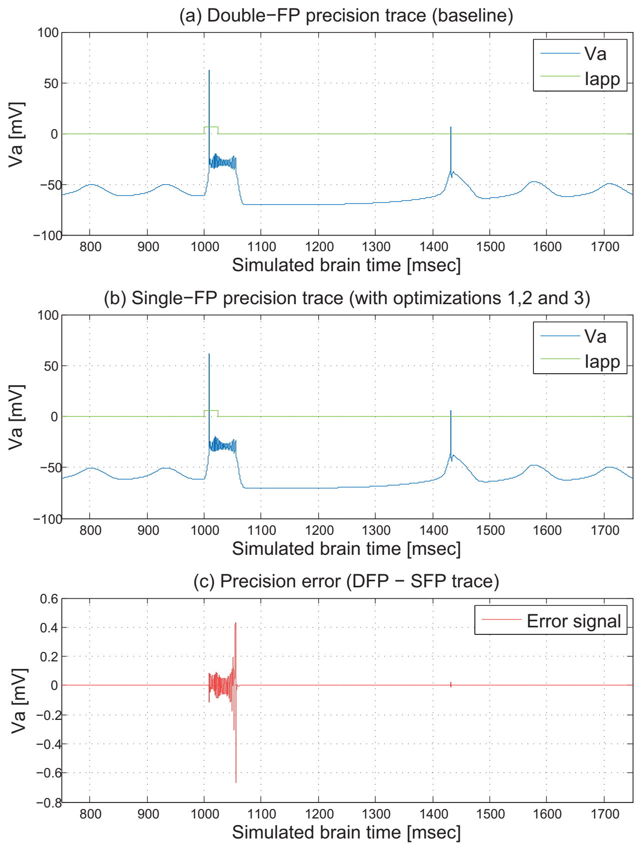


Figure 9: Graphical comparison of numerical-precision error. Externally evoked input current (Iapp) in green, axonal voltage in blue and error signal in red (Va). (a) Reference trace in double-FP precision. (b) The same trace generated with single-FP precision and all three code optimizations. (c) The error signal (i.e. difference) between the two traces. Observe the amplitude units of the error.

also represents the time penalty incurred for re-initializing the cell-states at runtime.

Place-&-Route area results are retrieved using Vivado IDE 2013.2 (Table 3). Our accelerator has been designed to utilize the maximum of the FPGA resources; in practice, it uses 83% of available LUT logic, 78% of BRAMs, 27% of Flip-Flops and 57% of the available DSPs on the FPGA chip.

6.3 Error Estimation

As previously mentioned, the original ION-network model performed all computations with double-FP precision. The main reason was that its modelers (as so many peers in the neuromodeling field) have arbitrarily opted for double-FP precision since this is the highest intrinsically supported precision in many modern programming languages (here: Matlab). However, early in our design effort, we realized that double-FP precision would tax the FPGA with such high performance and area costs that no significant acceleration of the application could be achieved. We, therefore, resorted to switching to single-FP precision calculations for the hardware version of the ION-network model.

To make such a decision, we had to first make sure that single-FP precision would be sufficient for the application at hand. Due to the fact that correct, “reference” brain-simulation traces do not exist (in fact, this is one of the goals of model simulators like the one we are porting in this work), only empirical metrics of correctness can be given by neuroscientists at the moment. That is, over a practically infinite amount of simulation time – for instance 1 day of real-time brain simulation (amounting to approx. 1.73 billion simulation steps) – the double-FP and single-FP simulation traces should not exhibit any biophysically different results.

Multiple tests have been run. As a simple illustration, in Figure 9 both the dynamic (complex spike) and steady-state (subthreshold oscillations) behavior of a single ION cell between 700 and 2000 msec of a simulation trace have been captured. Figures 9(a) and (b) illustrate runs with double-FP and single-FP precision, respectively. In the single-FP case are also included the 3 code optimizations discussed in Section 5.3 which contribute an additional precision error. In Figure 9(c), the error signal (i.e. difference of the two signals) is plotted over the same simulation period. Analysis of the error reveals that there is *no phase error*. A *very low amplitude error* is observed which ranges from 0.0%, at cell resting state (when most internal cell variables change rapidly), to about 2.1%, at cell firing state. Such a low error signal does not affect the simulator functionality, especially since the model itself cannot guarantee such high accuracy to the real biological system. In conclusion, computations in single-FP precision along with the 3 performed optimizations are considered to *not* compromise the ION-network simulation correctness and are permanently adopted. It is, of course, conceivable that a more constrained numerical range could also be used (i.e. fixed-point precision), but extensive precision analysis of the mathematical model would be required to identify such an (integer) range with certainty.

6.4 Comparison to Related Work

We discuss next the efficiency of our design and related SNN FPGA-based approaches and attempt to analyze and compare them. A direct comparison is not possible as different works consider different neuron-models with radically different characteristics, which potentially change completely the requirements of each design. Moreover, despite its complexity, each model type has its own merits for neuroscience and potential usefulness in applications. Depending on application constraints or the subject of simulation experiments different models can be of use.

Table 4 summarizes related FPGA-based brain-modeling works. Designs using the Quadratic IaF model such as the work in [18, 17] can implement spike latencies, activity-dependent thresholds and bistability properties of resting and tonic[3] spiking. This model requires only 7 FP operations per 1ms for each neuron making it useful for the exploration of large neuron integrator networks. The Izhikevich neuron is a more advanced model, which emulates all

[3]Neuron fires continuously while receiving stimuli.

Design	[18, 17]	[4][5]	[15]	[20]	[2]	ION Design
Model	Quadratic IaF	Izhikevich	Izhikevich	HH	simplified HH	Extended HH
Time Step (ms)	1	1	1	-	0.1	0.05
Real-Time Network Size	1000	64000	64000	4	400	96
Arithmetic Precision	Fixed Point	Fixed Point	Fixed Point	Floating Point	Floating Point	Floating Point
Operations per Neuron in 1ms	>7	>13	>13	>1200	<1200	22200
Neuron Model OPs * Net. Size (MFLOPS)	>7*	>832*	>832*	>4.8	<480	2131.2
Interconnectivity Density	1% (10 per neuron)	1.5% (1000 per neuron)	1.5% (1000 per neuron)	100%	100%	100%
Speed-up vs. CPU	-	-	x162 (C code) 4-FPGA System	x12 (C Code)	-	x12.5 (C Coce) x731.23 (Matlab)
FPGA Chip	Virtex 5 XC5VL330T	Virtex 6 SX475T Maxeler Machine	Stratix IV 230	Spartan 3 XC3SD1800a	Virtex 4	Virtex 7 XC7VX485T
Device Capacity (LUTs/ALMs)	207360 6-input LUTs	297600 6-input LUTs	91200×4 ALMs 2 ALM≈4 6-in LUT	33280 4-input LUTs	-	303600 6-input LUTs
Performance density (FLOPS/LUT)**	34*	2796*	1140*	576	-	7019

* Fixed-point operations ** 6-input LUTs

Table 4: Overview of FPGA SNN Implementations on achievable real-time network sizes. CPU Speed-up for [15] is compared to a Xeon 2.80GHz/48GB RAM, for [20] compared to a Pentium 4 3GHz/3GB RAM and for the ION design to a Xeon 2.66GHz/20GB RAM.

known input/output behavior in a spiking neuron. Izhikevich models are useful for researching the dynamics of large neural networks. Their simplicity also allows for the implementation of very large network sizes in FPGA devices, as described in [5] and [15], achieving sizes already significant to actual brain subsystems (tens of thousands of neurons). A little more costly than the IaF, each Izhikevich neuron model requires 13 operations per 1ms. On the other hand, biophysically-meaningful models such as the HH models used in [20, 2], are one to two orders of magnitude more costly in terms of operations, which is one of the main reasons why real-time network sizes achievable in such designs are much smaller. Another interesting observation is that HH models have a much shorter simulation timestep (tens of μseconds) compared to Quadratic IaF and Izhikevich models (1 msec), which increases their complexity and their accuracy. The ION-model used in the present work has a simulation step of 50 μsec, $\times 2$-$\times 20$ shorter than other models, making it have the tightest real-time constraint among the related works reported.

Especially for the ION-cell model considered in this paper, the complexity is even greater than the other HH models. The design accurately models 3 compartments and the gap junctions that account for more than 22,000 FP operations per 1 msec, about $\times 19$ more than the second most complex related model. Moreover, the use of simpler models to increase efficiency is not an option when modeling the Inferior Olive. Izhikevich and IaF models only have two basic output responses for their neuron: resting and firing states. The biological behavior of the Inferior Olive requires greater resolution, since neurons are constantly oscillating even in their resting state and have the property to synchronize. Such behavior could not be simulated with such simpler models.

A strategy that improved performance in some of the related works, however not applicable in the ION-model, is the event driven execution, e.g., in [5] and [15]. Due to the fact that the response of the Inferior Olive neurons oscillates, for the network to retain the ability to synchronize, the neuron needs to compute compartment states and transmit its data through the network connections in every simulation step; consequently event driven simulation is not appropriate.

Another important performance advantage that the designs of simpler models have is the use of fixed-point arithmetic. In such models, precision errors can be insignificant for correct behavior. That is not self-evident in HH models such as the one dealt with here, as they are much more sensitive to both amplitude and phase precision errors. For this reason our design needs to deal with the complexity and cost of FP operations.

Although the above approaches are radically different, in Table 4 we attempt to quantify their complexity and evaluate their efficiency. We take into account the amount of computations per neuron in 1 msec and the network size to estimate the performance of each work in FP operations per second (FLOPS) and properly marking those that use fixed-point. It must be noted that estimations for the computing capabilities of each design are based on data presented in [13] and cannot account for the computations due to the extra custom-made characteristics in the network models of each design, as we do not have this information available. We assume that the majority of the computations come from the simulation of the main neuron model. Our design achieves 2,131.2 MFLOPS, when matching the real-time constraints, while [2] achieves 480 MFLOPS; in both cases computations are mostly due to the complexity of the models. Then, [5] and [15] support 832 million fixed-point operations per second mostly due to the size of the simulated networks. It is worth noting that the highest connectivity (in absolute numbers) is provided by [5] and [15], connecting 1,000 neurons all-to-all, however, the amount of data exchanged is expected to be significantly lower (and less frequent) compared to [2] and compared to our design, which connect 400 and 96 neurons, respectively. Taking into account the area resources used in each work, we define a metric for performance density and measure operations per second per unit area (LUT). Our design has the highest performance density, with second best being at least $\times 2.5$ lower (without taking into account the difference between fixed- and floating-point) [5]; the higher number of DSP-slices in our FPGA device (2,800 vs. 2,015) is however in our advantage.

Finally, interesting conclusions can be derived when comparing the speedup of each approach over software implementations. Compared to a CPU, [2] achieves a $\times 12$ speedup, while the Bluehive device reaches an impressive $\times 162$ using, however, four FPGA devices. Our design achieves a $\times 731.23$ speedup compared to the original Matlab code and $\times 12.5$ compared to the double-FP C code. This speedup reaches almost $\times 45$ for higher network sizes.

7. CONCLUSIONS

We presented an efficient FPGA design for a biophysically-meaningful model of the Inferior Olive, an important part of the olivocerebellar subsystem in the brain responsible for motor coordination and learning. Through a detailed analysis of the application and optimization of the original algorithm, our ION-model design achieves real-time performance as well as sufficient speedup for use in neuroscience experiments. Our FPGA accelerator managed to simulate a network of 96 ION-neurons in real-time being more than $\times 700$ faster than the original Matlab model and $\times 12.5$ faster than the C implementation. The speedup can reach $\times 45$ for a 1,056-cell network, showing substantially better scalability with increasing network sizes compared to software. Although our accelerator implements an ION-network which is $\times 19$ more computationally intensive and has $\times 2$-$\times 20$ tighter real-time constraints compared to related models, it achieves at least $\times 2.5$ better performance density supporting 2.13 GFLOPS with a single FPGA device. The empirical precision-error analysis revealed that using our optimizations and single-FP arithmetic creates a very slim amplitude error and no phase errors, preserving the correct biological behavior while benefiting in performance. Our design, implemented in a Virtex 7 XC7VX485T FPGA, can maximally support a 14,400-cell network with online parameter configurability for neuron state and network size.

8. ACKNOWLEDGMENTS

This work has been partially funded by the DeSyRe project (EC 7th Framework Programme, Grant Agr. Nr. 287611). We are also grateful to Drs. S. Koekkoek, J. de Gruijl, R. van Leuken and M. Negrello for all their help.

9. REFERENCES

[1] P. Bazzigaluppi, J. R. De Gruijl, R. S. Van Der Giessen, S. Khosrovani, C. I. De Zeeuw, and M. T. G. De Jeu. Olivary subthreshold oscillations and burst activity revisited. *Frontiers in Neural Circuits*, 6(91), 2012.

[2] M. Beuler, A. Tchaptchet, W. Bonath, S. Postnova, and H. A. Braun. Real-Time Simulations of Synchronization in a Conductance-Based Neuronal Network with a Digital FPGA Hardware-Core. In *Artificial Neural Networks and Machine Learning – ICANN 2012*, September 2012.

[3] D. Brüderle. PyNN and the FACETS Hardware. `www.neuralensemble.org/media/slides/CodeJam2\_Bruederle\_FacetsHardware.pdf`, [Online; accessed 18-December-2013] 2008.

[4] K. Cheung, S. R. Schultz, and P. H. W. Leong. A Parallel Spiking Neural Network Simulator. In *Int. Conf. on FPT*, pages 47–254, Dec. 2009.

[5] K. Cheung, S. R. Schultz, and W. Luk. A large-scale spiking neural network accelerator for FPGA systems. In *Int. conf. on Artificial Neural Networks and Machine Learning*, ICANN'12, pages 113–120, 2012.

[6] C.I. De Zeeuw, F.E. Hoebeek , L.W.J. Bosman, M. Schonewille, L. Witter, and S.K. Koekkoek. Spatiotemporal firing patterns in the cerebellum. *Nat Rev Neurosci*, 12(6):327–344, jun 2011.

[7] H. de Garis, M. Korkin, and G. Fehr. The CAM-Brain Machine CBM: An FPGA Based Tool for Evolving a 75 Million Neuron Artificial Brain to Control a Lifesized Kitten Robot. *Auton. Robots*, 10(3):235–249, May 2001.

[8] H. Du Nguyen. GPU-based simulation of brain neuron models. Master's thesis, Delft Technical University, Aug. 2013.

[9] G. Ermentrout and N. Kopell. Parabolic Bursting in an Excitable System Coupled With a Slow Oscillation. *SIAM J on Applied Mathematics*, 46:233–253, 1986.

[10] G. B. Ermentrout. Type I membranes, phase resetting curves, and synchrony. *Neural Computation*, 83:979–1001, 1996.

[11] A. L. Hodgkin and A. F. Huxley. Quantitative description of membrane current and application to conduction and excitation in nerve. *Journal Physiology*, 117:500–544, 1954.

[12] E. Izhikevich. Simple Model of Spiking Neurons. *IEEE Trans. on Neural Networks*, 14(6), 2003.

[13] E. Izhikevich. Which Model to Use for Cortical Spiking Neurons? *IEEE Trans on Neural Net.*, 15(5), 2004.

[14] W. Maass. Noisy Spiking Neurons with Temporal Coding have more Computational Power than Sigmoidal Neurons. In *Neural Information Processing Systems*, pages 211–217, 1996.

[15] S. W. Moore, P. J. Fox, S. J. Marsh, A. T. Markettos, and A. Mujumdar. Bluehive — A Field-Programable Custom Computing Machine for Extreme-Scale Real-Time Neural Network Simulation. In *IEEE Int. Symp. on FCCM*, pages 133–140, 2012.

[16] National Academy of Engineering (nae.edu). Grand Challenges for Engineering, 2010.

[17] H. Shayani, P. Bentley, and A. M. Tyrrell. A Cellular Structure for Online Routing of Digital Spiking Neuron Axons and Dendrites on FPGAs. In *ICES '08, Int. Conf. on Evolvable Systems: From Biology to Hardware*, pages 273–284, 2008.

[18] H. Shayani, P. Bentley, and A. M. Tyrrell. Hardware Implementation of a Bio-plausible Neuron Model for Evolution and Growth of Spiking Neural Networks on FPGA. In *NASA/ESA Conf. on Adaptive Hardware and Systems*, pages 236–243, June 2008.

[19] G. Wulfram and W. Werner. *Spiking Neuron Models.* Cambridge University Press, 2002.

[20] Y. Zhang, J. P. McGeehan, E. M. Regan, S. Kelly, and J. L. Nunez-Yanez. Biophysically Accurate Floating Point Neuroprocessors for Reconfigurable Logic. *IEEE Trans on Computers*, 62(3):599–608, march 2013.