Low-cost hardware implementations for discrete-time spiking neural networks

Horacio Rostro-Gonzalez, Jose Hugo Barron-Zambrano, Cesar Torres-Huitzil, Bernard Girau

Low-cost hardware implementations for discrete-time spiking neural networks

Horacio Rostro-Gonzalez[1], Jose H. Barron-Zambrano[2], Cesar Torres-Huitzil[2] and Bernard Girau[3]

[1]NeuroMathComp project team (INRIA, ENS Paris, UNSA LJAD), Sophia Antipolis, France

[2]Cinvestav Tamaulipas, Information Technology Laboratory, Victoria, Mexico

[3]LORIA/University Nancy 1, Cortex Group, Campus Scientifique, Vandoeuvre-les-Nancy, France

E-mail: hrostro@sophia.inria.fr · jhbarronz@tamps.cinvestav.mx · ctorres@tamps.cinvestav.mx · Bernard.Girau@loria.fr

1 ABSTRACT

In this paper, both GPU (*Graphing Processing Unit*) based and FPGA (*Field Programmable Gate Array*) based hardware implementations for a discrete-time spiking neuron model are presented. This generalized model is highly adapted for large scale neural network implementations, since its dynamics are entirely represented by a spike train (binary code). This means that at microscopic scale the membrane potentials have a one-to-one correspondence with the spike train, in the asymptotic dynamics. This model also permit us to reproduce complex spiking dynamics such as those obtained with general Integrate-and-Fire (gIF) models. The FPGA design has been coded in Handel-C and VHDL and has been based on a fixed-point reconfigurable architecture, while the GPU spiking neuron kernel has been coded using C++ and CUDA.

Numerical verifications are provided.

KEYWORDS

Spiking neuron models · FPGA · GPU

2 Introduction

The aims of this work is to present two low-cost alternatives in the field of neural networks hardware implementations. Hardware technologies have some capabilities to reproduce realistic neuron models (or biologically plausible neuron models, focusing in spiking neuron models [3]) and in other cases to reproduce large scale neural networks [13, 6]. In this direction analog hardware implementations [15, 12, 14] have demonstrated to be a powerful tool to reproduce complex models such as the Hodking-Huxley model [16]. This kind of implementations also offer us real time processing (contrary to digital implementations, which normally run much faster than in the real word), it because in the design process the electronic components are chosen with the adequate values. However analog implementations induce certain problems such as, design time, development costs and non re-programability , it last crucial, since models evolve constantly in the scientific world. On the other hand digital implementations based on a FPGA have limited power to reproduce realistic models, because real values can not be directly handled by a FPGA due to its internal structure, which is based in logic blocks. However architectures based on FPGA present important advantages, such as a faster time-to-market, low non-recurring engineering costs, design time, and the most important reprogrammability, which permit us to redefine our architecture when the model has evolved. In [9, 10] authors show that FPGA is widely used in large scale neural implementations, though with simplified neuron models. Here using generalized discrete-time spiking neuron models allow us to reproduce complex dynamics even on such restricted hardware.

In recent years GPUs [5] have been frequently used to solve complex mathematical problems. GPU use an heterogeneous programming scheme, which allows us to use different programming language in order to simulate the related mathematical model. This means that a GPU programming language can easily interact with other programming languages permitting us to have a programming scheme more flexible and optimal. The programmable GPU has evolved into a highly parallel, multithreaded, multi-core processor with tremendous computational power and very high memory bandwidth far beyond the graphical applications they have been designed for. More specifically, the GPU is especially well-suited to address problems that can be expressed as data-parallel computations, the same program being executed on many data elements in parallel. In this direction, Spiking Neural Networks could take profit of this powerful technology, since a Spiking Neuron Model could be coded as a GPU kernel and reproduced several times to build a neural network, therefore executed in parallel. In order to implement kernels into a GPU, NVIDIA has introduced a general-purpose parallel computing architecture, called CUDA, which is a new parallel programming language.

Related work

Some previous works on hardware implementations for spiking neurons has been proposed, a well-crafted GPU-based implementation for large scale spiking neural networks is presented in [11]. In this work the authors combines powerful programming languages to simulate the well-known Izikevich's simple spiking neuron model faster than a CPU, here the number of

neurons and synaptic connections is not a real problem since the simulation is running on a PC. Another interesting work is presented in [8], which offer us a survey on artificial neural networks in hardware. Existing SNN simulators such as NEST, PCSIM have demonstrated a parallel version that runs on simple clusters accelerating the computation time, others like SpiNNaker [4] deploys an application specific parallel processor interconnected by a network-on-chip, resulting in an approach that combines the performance an ease of programmability for realizing SNNs. Our work offers a new spiking neuron model, which is suitable to be implemented in large-scale neural networks on dedicated hardware. Another important point compared with previous published works is that we offer a solution for both technology GPU and FPGA.

The rest of the paper is organized as follows. Section 3 introduce the neuron models: the discrete-time spiking neuron model and the analog-spiking neuron model. In section 4, optimal spiking neural networks hardware implementations (FPGA-based reconfigurable architecture and GPU-based kernel) are presented, simulation parameters are also defined here. In section 5 experimental results are presented. Finally, section 6 presents the concluding remarks and some perspectives.

3 The discrete-time spiking neuron model

The following equations describe a discrete-time spiking neuron model (see [1] for a rigorous mathematical study of this model), where synaptic transmission has delays. This model has been derived from the gIF model, and forms an efficient basis of spiking neural networks [7]. The neural network has N neurons with a fully-connected topology (it could be change depending on the application). The dynamics of the membrane potential V and the firing state Z are given by the following discrete-time equations respectively:

$$V_i[k] = \gamma V_i[k-1](1 - Z_i[k-1]) + \\ + \sum_{j=1}^{N} \sum_{d=1}^{D} W_{ijd} Z_j[k-d] + I_i[k] \qquad (1)$$

$$Z_i[k] = \begin{cases} 1 & \text{if } V_i[k] \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

The equation (1) give us the behavior of the membrane potential for each neuron of index i in the network at time k. If the membrane potential reaches a firing threshold $\theta = 1$ the neuron i fires a spike, thus $Z_i[k] = 1$, otherwise $Z_i[k] = 0$. The constant $\gamma \in [0, 1]$ is the leak factor, D is the delay and $I_i[k]$ is the external current for neuron i at time k. The synaptic weights W_{ijd} are mapped into a postsynaptic response profile at different delays, $d \in D$ (Figure 1).

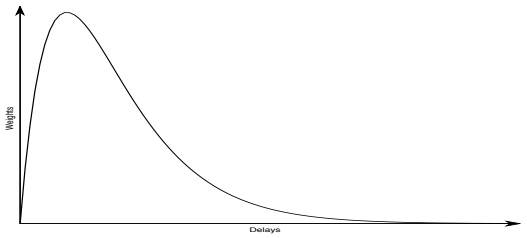

Figure 1: An alpha profile $\alpha(t) = H(t)\frac{t}{\tau}e^{-\frac{t}{\tau}}$, which represents the synaptic weights mapped at different delays.

This corresponds to "current" synapses but it has been shown in [2] that it efficiently approximates "conductance" based synapses.

A step further, the following equations can describe a new formalism, which is an extension of Equation (1) and permits us to consider more realistic spiking neuron models, since the neuron is seen as both an analog unit and/or a spiking unit. In the former case, the non-linearity is defined by an analog function i.e. sigmoid profile. This allows us for instance to consider 2D neuron models. The equation writes:

$$V_i[k] = \gamma_i[k] V_i[k-1] \rho(V_i[k-1]) + \\ + \sum_{j=1}^{N} \sum_{d=1}^{D} W_{ijd}\, \sigma\left(V_j[k-d]\right) + I_i[k] \qquad (2)$$

where $\sigma(V_i[k]) \in [0, 1]$ defines the non-linear transform (firing rate, spiking, etc.) and $\rho(V_i[k]) \in 0, 1$ is the reset mechanism. Constants have been previously defined.

In the next section we present a description of the different hardware implementations for the discrete-time spiking neural model of equation (1). While the extension to equation (2) is straight-forward, only look-up tables have to be added.

4 Implementations

We propose three different hardware implementations for the discrete-time spiking neuron model presented in section 3, where two of them use Hardware Description Languages (HDLs), Handel-C and VHDL to map the model onto a FPGA. A fixed-point arithmetic has been used to design the reconfigurable architecture for the proposed model. In the third case, we propose a GPU-based implementation, where the data handling is easier than in HDL ones, since GPU Kernel can use all data types.

Parameters used in the implementations

The following neural network parameters have been defined to test the hardware implementations.

For these implementations we use a leak rate $\gamma = 0.98$, a constant external current $I = 0.2$, a maximum inter-neural delay $D = 2$, size Network $N = 100$ with a fully-connected topology (it is 10000 synapses, each modeled with D parameters), where 80% of total connections are excitatory and the rest 20% are inhibitory.

Synaptic weights W_{ijd}, are randomly chosen from a truncated normal distribution, which is defined as follow:

Suppose $X \sim \mathcal{N}(\mu, \sigma^2)$ has a normal distribution and lies within the interval $X \in (a, b)$, $-\infty \leq a < b \leq \infty$. Then X conditional $a < X < b$ has a truncation normal distribution with probability density function

$$f(x; \mu, \sigma, a, b) = \frac{\frac{1}{\sigma}\phi(\frac{x-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})},$$

where $a = -1$, $b = 1$, $\mu = 0$, $\sigma^2 = 0.2$, $\phi(\cdot)$ is the probability function of the standard normal distribution and $\Phi(\cdot)$ its cumulative distribution function.

Fixed-point arithmetic

Now that networks parameters have been defined, we have to manage the fact that real values can not be mapped directly onto a FPGA, since it is composed of logic blocks, which use binary logic to perform complex combinational functions. In order to create a bridge between software and hardware, fixed-point and floating-point arithmetic are proposed. In this work we focus on a 16 bits fixed-point arithmetic, since floating point demands more computational resources.

The defined parameters are described by a 16 bits representation, where 4 bits are used by the integer part and the 12 bits remaining are considered for the fractional part. If we want a bigger neural network we can adjust the architecture to 6 bits for the integer part and 10 for the fractional one, losing some precision in the results.

In terms of implementations, the way to pass the networks parameters to a 16 bits fixed-point representation relies on the next equation:

$$\text{Fixed-point data} = (\text{int})(\text{real value} * 2^f)$$

where f represents the number of fractional bits.

4.1 A GPU based implementation for a Spiking Neural Network

In this section we present a GPU-based implementation for a spiking neural network, where GPU programming is based in heterogeneous programming, mixing different programming languages, C/C++ and CUDA. Sequential code is computed on the CPU via C/C++, and parallel code is computed on the GPU via CUDA. In order to differentiate between the code that will be executed on the CPU from there that will be executed on a GPU, we refers them as *host* and *device* respectively.

GPU implementation

The GPU based implementation can be described by the next steps:

1. **Initialization** consists in a sequential process where γ, I, N, D and W are defined, we also reserve the space memory to allocate V and Z in the *host*.
2. **Allocating kernel memory** here the *device* reserves memory space for V, Z and W. The memory space for V and Z is smaller than in the *host*, since we only need to know the value at the last time $k-1$ for V, as for Z, since only the last D spiking dynamics from the actual time k are necessary to compute the new membrane potential $V[k]$. In the case of the weights W, the space memory is the same than in the *host*, since we have a fully-connected network and all the synaptic connections must be computed.
3. **Invoking the GPU kernel**, at the moment to invoke our GPU kernel, which compute the discrete-time spiking neural network, we need to define two parameters, which variate depending on the graphic card that will be used to compute the implemented model. The parameters are: the number of blocks and the number of threads per block. In this work the parameters have been chosen from the size of the network (100 spiking neurons) and the PC graphic card (NVIDIA Quadro NVS 160 M). The architecture of this graphic card is designed to support 256 threads per block, this means that we need only 1 block to compute all the proposed spiking neural network.
4. **The GPU kernel** has been designed to run under a parallel scheme. In this sense the kernel has been divided in three parts that permit us to have optimal performance. In the main kernel we define the equation (1), where the connectivity is estimated by another kernel, which manage the behavior of the synaptic connections between the neuron of index i with the other neurons in the network at the last D times. Finally in the third kernel, we compare the membrane potential of all neurons with the threshold $\theta = 1$ determining the firing state of the network (spike train).
5. **Loading results in the *host***, finally the spike train Z and the membrane potentials V are read from the kernel and saved in the *host* in order to be analyzed or displayed.

FPGA hardware implementations are described in the next section.

4.2 A FPGA implementation for a discrete-time spiking neural network using Hadel-C

Handel-C is a high level programming language, which permits to pass rapidly from a software to a hardware implementation, especially if it has been coded in C or C++, since both languages are similar. As for VHDL programming we need to define the number of bits used to represent the data. Handel-C proposes also a set of libraries such as fixed-point and floating point libraries that can be used to perform complex combinational functions but that in some cases can not be synthesizables onto the FPGA. The main problem in Handel-C is that we have no control on its internal design process, since Handel-C is a high level programming language. However parallelism and synchronization are possibles in Handel-C. In terms of simulation, before to design a complex VHDL architecture, it is interesting to test the models with this software.

Handel-C implementation

The Handel-C implementations can be described in three simple steps:

1. Handel-C is used here to perform a HDL code to design a reconfigurable architecture for a spiking neural network. Defined optimized functions are not used in this implementation, since they present problems at the synthesis process. The architecture use a 16 bits fixed-point arithmetic, defining 4 bits for integer part and the 12 bits resting for the fractional one. Bits repartition has been though to guaranty high precision and to avoid a wrong spike estimation.

2. The next step is to simulate the implemented code with the fixed parameters. If the simulation throws some errors we need to correct them, otherwise we can go to the synthesis of the network into the FPGA.

3. Finally the spiking neural network is synthesized onto a Xilinx Spartan-3 FPGA.

4.3 A FPGA implementation for a discrete-time spiking neural network network using VHDL

A reconfigurable architecture for a FPGA based spiking network implementation is presented, with three main modules: the input module, the spiking neuron module and the output module. An auxiliary module is also used for general control in the network.

Input Module

The input data model consists of two internal blocks: a Gaussian Random Number Generator and a RAM memory. In the first block synaptic connection weights are estimated from a truncated normal distribution between -1 and 1 and then represented in a 16 bits fixed point arithmetic. Other way to generate the synaptic connection weights is via C/C++, where are directly mapped into a 16 bits fixed-point arithmetic and stored in a file, which further are loaded in the network. Finally weights, constants (γ and I) and initial conditions ($Z[D]$) are stored into a RAM memory.

Neuron Module

The neuron model architectures for equation 1 has been designed using combinational logic and made use of 16 bits fixed-point arithmetic to represent variables and constants, yielding a maximum performance.

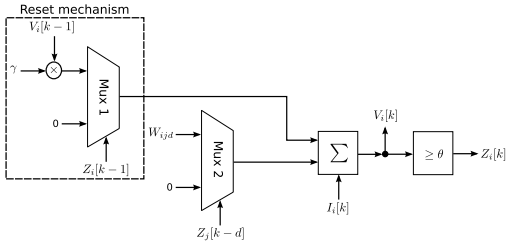

Figure 2: Block diagram of individual discrete-time spiking neuron

In the figure 2 we present the internal structure of a discrete-time spiking neuron model described by equation (1). Observing the diagram we can identify two multiplexers (Mux 1 and Mux 2) each one of them with only two inputs and both controlled by the state in Z, one adder ($\sum$) and one voltage comparator ($\geq \theta$); Mux 1 selects between the state "zero" (reset) if $Z_i[k-1] = 1$ and $\gamma * V_i[k-1]$ if $Z_i[k-1] = 0$. Mux 2 selects between the state "zero" (there are not effects of W on the neuron i at time $k-d$) and the contribution of all synapses ij at the time $k-d$. Here it is important to make a remark: all synaptic weights are charged at the same time (combinational process). The next step is to sum all, Mux1 output, Mux2 ouput and external current, the result is stored in $V_i[k]$ and compared with a firing threshold ($\theta = 1$). $V_i[k]$ is not a vector but only a variable which stores the last value of the membrane potential, once $V_i[k]$ has been compared with the threshold it is sent to the input by a flip-flop element and a binary value is assigned to $Z_i[k]$, 1 if the threshold has been reached, otherwise 0.

Further in order to implement the model described by the equation (2) we use a look-up-table to characterize the sigmoid function $\sigma(V_j[k-d])$ and to minimize the computational costs.

Output Module

The output module has two tasks. On one hand, it saves in a file the firing state of each neuron at

time k. On the other hand, it sends at the same time the $N \times D$ firing states from the actual time to each neuron (which is necessary since we are considering delays). The general control idxCtrl manages the control of this module and of the Neuron Module.

idxCtrl Module

The idxCtrl module is a kind of master clock, it is activated by a signal sent from the input module since all the synaptic weights previously stored into the RAM block are sent to each spiking neuron. Once the signal is received, it sends the address where the output of each neuron will be stored into the vector Z.

5 Results

Our main contribution is the implementation of a reconfigurable architecture (Figure 3) for a neural network with discrete-time spiking neurons. The internal components have a parallel design accelerating the neural processing. This scheme permits us to emulate large scale spiking neural networks, since the spiking dynamics is described by a spike train (binary code). In addition the precision analysis applied to the discrete-time spiking neuron model guaranty us a good approximation in the reproduction of spiking dynamics like gIF models. This is crucial since a wrong spike estimation could change all the neural network dynamics.

In this work we have a fully-connected network topology, where 80% of connections are excitatory and 20% inhibitory. We have chosen this topology in order to evaluate the limitation of this implementation, since the number of connections increase exponentially when the network size is increased. Further depending on the application different topologies can be applied, leading to a smaller area mapping.

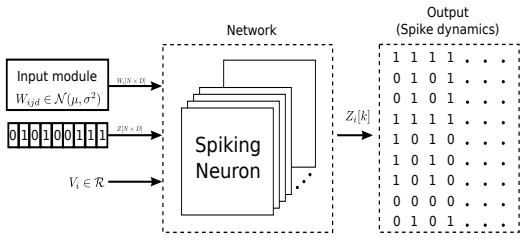

Figure 3: Reconfigurable architecture overview

Numerical Results

In this section we present some numerical results. In table 1 we show the processing time that takes each implementation to simulate the defined spiking neural network. Figure 4 shows a raster plot (spiking dynamics), where the same raster has been calculated from software and hardware implementations. Obviously if we make a zoom on a specific neuron, we find that the membrane potential in both hardware and software implementations are not the same, since a 16 bits fixed-point arithmetic has been employed in the design of the hardware architecture. In this sense figure 5 shows the membrane potential for neuron 10, where the blue line corresponds to the membrane potential estimated by software (C/C++), but also by hardware based on GPU. The red line corresponds to the membrane potential estimated with the implementations based on FPGA.

C/C++	C++ and CUDA	Handel-C	VHDL
0.231 s	69 ms	50 ms	50 ms

Table 1: : Processing time for software and hardware implementations.

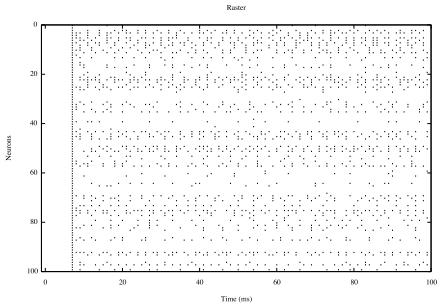

Figure 4: A raster plot (spike train). The same raster plot has been generated by software and hardware implementations.

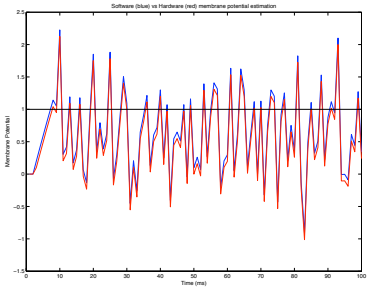

Figure 5: The membrane potential for a specific neuron i.e. neuron 10. The blue line corresponds to the membrane potential estimated on a PC and the red line corresponds to the membrane potential estimated on a FPGA.

6 Conclusions and Perspectives

In summary this paper proposes two spiking neuron models, which are well-suitable for large scale hardware implementations, these models have been implemented in 4 different languages, 2 of them in HDL (Handel-C andVHDL), which have been synthesized in a Xilinx

Spartan 3 FPGA. Other hardware implementation are based on GPU, which has been executed in a PC with a NVIDIA Quadro NVS 160 M card, plus another implementation in C/C++ not described here but used to have a reference point in a software programming language (C/C++) in order to verify the results obtained in hardware implementations.

The number of spiking neurons that we can mapping into the FPGA may seen relatively small because this was just a feasibility study. Actually there exists several FPGA with different capabilities, thus easily yielding machine computations. The key point is propose spiking neuron models that can be used in large scale neural network implementations, since are described for simple functions with parallel calculations and a reconfigurable architecture able to be mapped onto any FPGA. In the case of the GPU implementation the model has characteristics of fully parallel computing since the state of the network is updated at the same time for all neurons.

The hardware implementations proposed in this work exploit the obvious parallel nature of spiking neural networks and, can easily be extended to more complex models such as those with Synaptic Timing Dependent Plasticity (STDP), which is the next step of the present work.

Acknowledgements

Partially supported by the CorTex-Mex project and the SEP and the CONACYT of Mexico.

Thanks to Thierry Vieville, Bruno Cessac and J.C. Vasquez for scientific discussions.

References

[1] B. Cessac. A discrete time neural network model with spiking neurons. rigorous results on the spontaneous dynamics. *J. Math. Biol.*, 56(3):311–345, 2008.

[2] B. Cessac and T. Vieville. On dynamics of integrate-and-fire neural networks with conductance based synapses. *Front. Comput. Neurosci.*, 2, 2008.

[3] Wulfram Gerstner and Werner Kistler, editors. *Spiking Neuron Models*. Morgan Kaufmann, 2010.

[4] M.M. Khan, D.R. Lester, L.A. Plana, A. Rast, X. Jin, E. Painkras, and S.B. Furber. Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor. In *IEEE international joint conference on neural networks*, pages 2849–2856, 2008.

[5] David B. Kirk and Wen mei W. Hwu, editors. *Programming Massivelly Parallel Processors: A hands-on Approach*. MIT Press, 2003.

[6] N. Lewis and S. Renaud. Spiking neural networks in silico: from single neurons to large scale networks. In *Fourth International Multi-Conference on Systems, Signals and Devices*, 2007.

[7] Wolfgang Maass and Christopher M. Bishop, editors. *Pulsed Neural Networks*. MIT Press, 2003.

[8] Janardan Misra and Indranil Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, 2010.

[9] M.J.Pearson, C.Melhuish, M.Nibouche A.G.Pipe, I.Gilhesphy, and B.Mitchinson. Design and fpga implementation of a embedded real-time biologically plausible spiking neural network processor. In *Proc. Int. Field Programmable Logic Appl. (FPL)*, 2005.

[10] M.J.Pearson, A.G. Pipe, B. Mitchinson, K. Gurney, C.Melhuish, I.Gilhesphy, and M.Nibouche. Implementing spiking neural networks for real-time signal-processing and control applications: A model-validated fpga approach. *IEEE Transactions on Neural Networks*, 18:1472–1487, 2007.

[11] J. Moorkanikara Nageswaran, J. L. Krichmar N. Dutt, A. Nicolau, and A. V. Veindenbaum. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks*, 22:791–800, 2009.

[12] S. Renaud, J. Tomas, Y. Bornat, A. Daouzli, and S. Saïghi. Neuromimetic ics with analog cores: an alternative for simulating spiking neural networks. In *Proceedings of the IEEE 2007 InternationaI Symposium on Circuits And Systems ISCAS*, 2007.

[13] Kenneth L. Rice, Mohammad A. Bhuiyan, Christopher N. Vutsinas Tarek M. Taha, and Melissa C. Smith. Fpga implementation of izikevich spiking neural networks for character recognition. In *International Conference on Reconfigurable Computing and FPGAs*, 2009.

[14] J. Schemmel, J. Fieres, and K. Meier. Realizing biological spiking network models in a configurable wafer-scale hardware system. In *IEEE International Joint Conference on Neural Networks IJCNN*, 2008.

[15] J. Tomas, Y. Bornat, S. Saighi, T. Levi, and S. Renaud. Design of a modular and mixed neuromimetic asic. In *Proceedings of the 13th IEEE International Conference on Electronics, Circuits and Systems ICECS*, 2006.

[16] Q. Zou, Y. Bornat, J. Tomas, S. Renaud, and A. Destexhe. Real-time simulations of networks of hodgkin-huxley neurons using analog circuits. *Neurocomputing*, 69:1137–1140, 2006.