Real-Time Computing Platform for Spiking Neurons (RT-Spike)

Eduardo Ros, Eva M. Ortigosa, Rodrigo Agís, Richard Carrillo, and Michael Arnold

***Abstract*—A computing platform is described for simulating arbitrary networks of spiking neurons in real time. A hybrid computing scheme is adopted that uses both software and hardware components to manage the tradeoff between flexibility and computational power; the neuron model is implemented in hardware and the network model and the learning are implemented in software. The incremental transition of the software components into hardware is supported. We focus on a spike response model (SRM) for a neuron where the synapses are modeled as input-driven conductances. The temporal dynamics of the synaptic integration process are modeled with a synaptic time constant that results in a gradual injection of charge. This type of model is computationally expensive and is not easily amenable to existing software-based event-driven approaches. As an alternative we have designed an efficient time-based computing architecture in hardware, where the different stages of the neuron model are processed in parallel. Further improvements occur by computing multiple neurons in parallel using multiple processing units. This design is tested using reconfigurable hardware and its scalability and performance evaluated. Our overall goal is to investigate biologically realistic models for the real-time control of robots operating within closed action-perception loops, and so we evaluate the performance of the system on simulating a model of the cerebellum where the emulation of the temporal dynamics of the synaptic integration process is important.**

***Index Terms*—Field-programmable gate arrays, pipeline processing, real time system, spiking neural network hardware.**

I. Introduction

A number of studies have investigated the use of event-driven computation schemes for the efficient simulation of spiking neurons [8], [31], [36]. The computation in these schemes, the integration of presynaptic spikes and the generation of postsynaptic spikes, occurs only at the arrival of each presynaptic spike; for efficiency each presynaptic spike is generally processed in a single computational step. This requires that two pieces of information can be fully determined at the time that the synapse is activated. First is the effect of the presynaptic spike on the membrane potential of postsynaptic neurons, i.e., the synapse's postsynaptic current profile (PSC). Second is the probability that the neuron fires in response to the presynaptic spike. This imposes strong restrictions on the types of neurons that can be simulated while still retaining the efficiency of the event-driven approach, principally that the synaptic time constant is sufficiently small that the PSC reaches its maximal value within one computational time-step. This is necessary to be able to accurately predict the occurrence of the postsynaptic spikes. Voltage-dependent synapses [for example, driven by N-Methyl-D-Aspartate (NMDA) receptors] are also problematic as the PSC depends on the membrane potential.

We are interested in synchronization phenomena within neural populations, for which important features are the temporal dynamics of the synapses [9], [11], [12]. We focus on the spike response model (SRM) for neurons [15] where a synaptic time constant accounts for *noninstantaneous synaptic dynamics*, i.e., for the gradual injection of charge in the membrane potential due to a presynaptic spike. We also focus on conductance-based synaptic interactions, even though it has been pointed out that equivalent current networks can be built at the cost of a different connectivity [25], [26]. We are interested in neurophysiological studies [3], [14], [35] which investigate the gradual injection of charge in conductance-based synaptic interactions for cerebellar function [3], [14], [20]. A neural model including gradual injection of charge is difficult to simulate with standard event-driven approaches. Nevertheless, it is worthwhile to mention that table-based approaches [37] can be used—however, their memory costs are high. Other event-driven schemes can be extended to include this feature at a higher computational load since this requires managing "uncertain events" [31]. Event-driven approaches represent a valid choice when simulating spiking neural networks on single-processor platforms. However, although these approaches take full advantage of the sparse spiking activity of pulsed neural networks, they are difficult to parallelize and therefore more suitable for single processor platforms. The study of how to implement parallel computing schemes for the simulation of parallel networks better suits a time-driven approach, which offers the additional bonus of being extensible to more biologically detailed and complex neural and synaptic models.

The standard integration approach is computationally inefficient when using conventional computational architectures (single- or multiprocessor platforms) [22]. Because of this, some authors have addressed the implementation of specific hardware platforms to perform the neural integration [17], [23], [32], [40], [41]. These studies implement a specific type of SRM according to the neuron model proposed by Eckhorn *et al.* [10]. Here we present a hybrid hardware and software simulation scheme for networks of SRM neurons. The novelty

Manuscript received November 11, 2004; revised November 10, 2005. This work was supported by the European Union under Projects SpikeFORCE (IST-2001-35271) and SENSOPAC (IST-028056) and by National Spanish Grant DPI 2004-07032.

E. Ros, E. M. Ortigosa, R. Agís, and R. Carrillo are with the Department of Computer Architecture and Technology, University of Granada, Granada E-18071, Spain (e-mail: eduardo@atc.ugr.es; eva@atc.ugr.es; ragis@atc.ugr.es; rcarrillo@atc.ugr.es).

M. Arnold is with CNL, Salk Institute of Biological Studies, La Jolla, CA 92037 USA (e-mail: mikea@salk.es).

Digital Object Identifier 10.1109/TNN.2006.875980

TABLE I
IIR TEMPORAL FILTERS AS DEFINED BY EXPONENTIALLY DECAYING FUNCTIONS TOGETHER WITH THEIR APPROXIMATIONS. *Output* IS 1 WHEN THE NEURON FIRES A SPIKE AND 0 OTHERWISE

Exponential functions	Approximated functions
$S_{exc}(t)=\begin{cases} 0 & , \ t < t_0 \\ S_{exc} \cdot e^{-(t-t_0)/\tau_{exc}} & , \ t \geq t_0 \end{cases}$	$S_{exc}(t)= A_{exc} \cdot w_{exc} + B_{exc} \cdot S_{exc}(t-1)$
$S_{inh}(t)=\begin{cases} 0 & , \ t < t_0 \\ S_{inh} \cdot e^{-(t-t_0)/\tau_{inh}} & , \ t \geq t_0 \end{cases}$	$S_{inh}(t)= A_{inh} \cdot w_{inh} + B_{inh} \cdot S_{inh}(t-1)$
$G(t)=\begin{cases} 0 & , \ t < t_0 \\ G \cdot e^{-(t-t_0)/\tau_{refrac}} & , \ t \geq t_0 \end{cases}$	$G(t)= 1-g(t)$ where $g(t)= A_{refrac} \cdot Output + B_{refrac} \cdot g(t-1)$

of our approach is in the implementation of a spiking neuron model i) that incorporates additional biophysically inspired features such as spike-driven synapses modelled as conductances, ii) whose complexity can be easily scaled to include further features as required by the application, and iii) that can be used to run simulations in real time. Our aim is not to build a general purpose neural accelerator for simply speeding up simulations, but to build hardware for real-time experiments. To this end, we have defined a well-structured hardware/software communication protocol imposing strong restrictions in order to achieve response times on the order of milliseconds.

Spiking neural networks are characterized by two different features: 1) their activity and communication is based on spikes (this motivates the simulation by event-driven schemes on single unit processing platforms) and 2) their massive parallelism (this motivates the exploration of parallel computing schemes as the one presented in this contribution). The parallelization of time-driven simulation schemes is possible, since the connection latencies allow the parallel computation of different neurons on a restricted number of processing units using time-slicing techniques. The presented contribution focuses on a parallel simulation platform whose performance increases with the number of processing units. This is of interest in the framework of current technology that allows the exploration of parallel computing schemes. The performance of the presented platform is difficult to compare with event-driven approaches. Furthermore, the ultimate goal of this contribution is not to directly outperform these other approaches but rather to explore parallel computing avenues (even on single chips) that are becoming possible due to the continuously increasing number of processing resources on single devices.

A simulation requires the definition of a network topology (connection weights and transmission delays) and the cell characteristics (temporal dynamics of its synaptic modules, membrane capacitance, resting potential, and time constant), which can be different for each neuron. The PSC is modeled as a filter driven by a Dirac delta function centered at the time of the occurrence of the presynaptic spike (see the expressions in Table I). The current neuron model includes noninstantaneous synaptic coupling to capture y-aminobutyric acid GABAergic and A-Amino-5–hydroxy-3–methyl-isoxazole propionic acid (AMPA) mediated temporal dynamics. We plan to incorporate further features in the neuron model, such as NMDA channels, synaptic short-term dynamics [43], alternative/complementary firing mechanisms, and intrinsic resonant properties [21]. Our approach is based on reconfigurable hardware, i.e., field-programmable gate arrays (FPGA). This reprogrammable technology facilitates the easy modification of the neural model computed by the hardware platform. We present a pipelined computing scheme where the stages are computed in parallel. New computational features can be added as new circuits working in parallel, so that increasingly the complexity of the model does not increase the time taken to simulate a single neuron. This is possible by taking advantage of the inherent parallel computing resources of FPGA devices. The FPGA technology that has been used in this approach embeds different resources: i) general-purpose logic resources that can be customized to design high performance data paths (as described in Section IV-A) and ii) on-chip memory resources that can be configured to allow parallel access to different data sets (as described in Section IV-B).

In the following sections, we define the division between the hardware and the software components, a detailed description of the hardware implementation of the neural model and its parallelization, the utilization of hardware resources, and the performance of the system. Our motivating objective is to investigate biologically realistic models for the control of robots operating within closed perception-action loops, which requires a real-time computing platform with an integration time-step of 100 μs or less. Hence we present initial results for the real-time implementation of a model of the cerebellum applied to a simple tracking task. Real-time processing at this time scale restricts the size of the network that can be simulated (see Sections VI and VII). The computing platform can also be used to accelerate the simulation of larger scale networks if the real-time restriction is relaxed.

II. COMPUTING SCHEME

We have developed a hybrid software–hardware computing platform. The hardware component consists of an add-on board providing memory resources and an FPGA device that works as a reconfigurable neuroprocessor. The software component

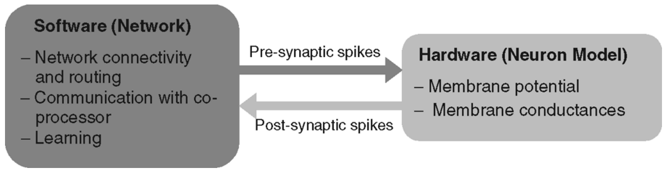


Fig. 1. Software/hardware computing platform. The software simulates the routing of spikes within the network and the learning. The hardware computes the membrane potential and conductances of the neural model. An event-driven communication scheme is adopted to connect both components.

runs on the host computer and the communication between the two is via the PCI bus. Currently the hardware component is restricted to computing the evolution of single-neuron state variables. The software component is responsible for maintaining the network connectivity, for routing spikes between neurons, and for learning (defined as the plasticity of maximal synaptic efficacies). Communication between the hardware and software components is restricted to spike events; the software component sends packets of addressed presynaptic spikes to the hardware and receives back packets of generated postsynaptic spikes (see Fig. 1). The neural states are stored in the coprocessor board and do not need to be communicated at each epoch.

The communication between the software and hardware components is through packets of spike events. The software component generates presynaptic events and the hardware component generates postsynaptic events.

- *Presynaptic events:* Each presynaptic event is a structure $I(t_s, A_{neu}, w_{exc}, w_{inh})$, where t_s is the time of the event relative to the current epoch given as an integer between zero and (N_t-1), A_{neu} is the address of the postsynaptic or target neuron, and (w_{exc}, w_{inh}) are the synaptic efficacies.
- *Postsynaptic events:* Each postsynaptic event is a structure $\text{Output}(t_s, A_{neu})$, where t_s is the time of the event relative to the current epoch given as an integer between one and N_t, and A_{neu} is the address of the postsynaptic or generating neuron.

In the current implementation, each neuron can have two types of synapses: one that is excitatory and one that is inhibitory. This will be extended in future implementations, where the presynaptic event structure will have (w_{exc}, w_{inh}) replaced with $(w, type)$, allowing a model with a variety of distinct synaptic types such as AMPA, NMDA, GABAa, or GABAb.

The communication is aggregated into *epochs* to reduce the fixed costs of communicating over the PCI-bus. Each epoch consists of N_t simulation time cycles or time steps (typically 20 time steps of 100 μs). The communication between the coprocessor board and the host computer occurs between epochs. The presynaptic events for the next epoch are transferred from the host computer to the coprocessor board, and the postsynaptic spikes produced in the previous epoch are transferred back to the host computer. In each epoch the number of neural states updates to be computed is $N_{\text{comp}} = N_t \cdot N_{\text{neurons}}$, where N_{neurons} is the number of neurons being simulated.

To minimize the time that hardware coprocessor spends waiting, it is desirable to overlap the computation and communication within the software component. As the host is generally a shared memory multiprocessor machine, an obvious choice is to use threads and shared memory, with a separate thread for 1) routing of postsynaptic spikes through to presynaptic events, 2) communicating with the hardware coprocessor, and 3) any learning that is required. To organize the parallelization, we choose a variant of the bulk synchronous parallel programming model (BSP) [19]. A BSP approach breaks a simulation into epochs, where each epoch is divided in to three ordered phases: 1) simultaneous local computation within each processing unit, 2) communication of data between the processing units, and 3) a barrier synchronization which makes all data transfers visible. The aggregation that is present in the communication between the software and hardware components is extended through to the communication within the software component between the threads. The last two steps of the BSP model are combined for a threaded implementation. Each thread maintains a personal copy of any read-write shared data, the management of which occurs at the barrier (see Fig. 2).

Currently, the communication and computation within the hardware coprocessor are sequential. Concurrent communication and computation within the software component introduces a latency into the routing of a spike between two neurons that is twice the epoch period [see Fig. 2(b)]. This latency must be less than the minimum synaptic delay between two neurons. The communication between the host and the coprocessor board is done via the on-board memory banks (see Fig. 3). The FPGA device reads events from these memory banks into the on-chip memory resources and then writes back the generated postsynaptic events. This parallelization scheme does not constrain the type of learning rule that can be implemented.

We have implemented a software emulator of the hardware platform to be able to test the network configurations in computers without the hardware accelerator. This software component emulates the hardware at low level. It uses the same computation and communication primitives, and so the results obtained by this software component should be the same as those obtained using the hardware board. The simulation system passes the iterative computation of the neural states either to the hardware component or to the software emulator. The software emulation tool is used to validate the integer arithmetic in the hardware platform. During the design stage, it is critical to choose valid bit-widths for the different variables so as to avoid significant differences between the results obtained by a simulation using floating-point arithmetic in a conventional computer and those obtained by the hardware platform using integer arithmetic.

III. Neuron Model

The chosen neural model consists of terms to calculate the membrane potential together with a spike generation mechanism [15]. For a neuron with excitatory and inhibitory synapses, the subthreshold membrane potential V_x is given by

$$V_x = V_x - \frac{1}{\tau_r}(V_x - V_{\text{resting}}) + G \cdot S_{exc} \cdot (U_{\max} - V_x) - G \cdot S_{inh} \cdot (V_x - U_{\min}). \quad (1)$$

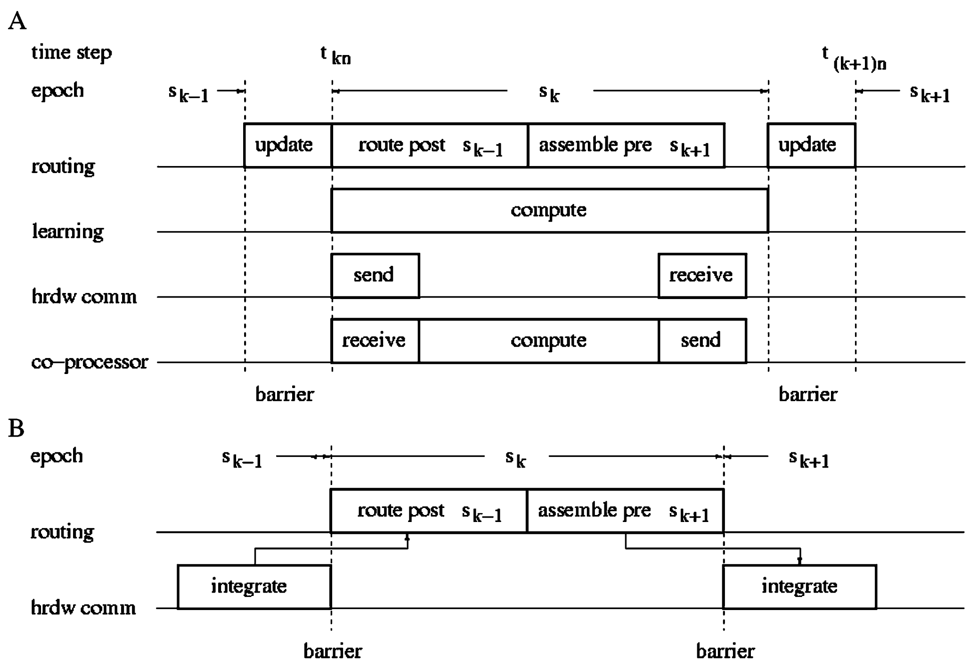


Fig. 2. Parallelization within the software component. (a) A multithreaded BSP-based implementation allocates separate threads (top three time-lines) for routing, learning, and communication with the coprocessor. The bottom time-line represents the hardware coprocessor itself. Communication between threads is aggregated into epochs s_k (consisting of n time steps), and occurs at the barrier where each thread's copy of the shared data is updated. The barrier represents an explicit synchronization that occurs every epoch. Communication and computation are overlapped within the overall system but are sequential within the coprocessor. (b) Overlapping communication and computation introduces a latency in routing postsynaptic spikes (from epoch $\mathrm{s}_{\mathrm{k}-1}$) through to presynaptic spikes (for epoch $\mathrm{s}_{\mathrm{k}+1}$). This latency is equal to twice the epoch period and must be less than the minimum synaptic delay.

τ_r is the time constant of the resting term, S_{exc} and S_{inh} are the synaptic contributions, and G is a gain term that modulates the refractory period. $U_{\max}$ and $U_{\min}$ are the maximum and minimum values of the membrane potential; i.e., the reversal potentials for the excitatory and inhibitory synapses. V_{resting} is the membrane resting potential. This follows SRM in which the synapses are modelled as spike-driven conductance terms with synaptic efficacies that depend on the time elapsed since the received presynaptic spike (emulating the receptor mediated injection of charge).

The neuron is implemented using a processing unit (see Fig. 4) with the following components.

- Two input infinite impulse response (IIR) spike response filters that emulate noninstantaneous synaptic coupling with different time constants (see Table I and Fig. 4): τ_{inh} for inhibitory GABAergic synapses and τ_{exc} for excitatory AMPA synapses.
- An integrator register that stores the postsynaptic potential, which is computed at each time step using (1).
- One output IIR filter with the time constant τ_{refract} that controls the postfiring refractory period through the gain term G.
- One spike generation mechanism that generates spikes according to the membrane potential. Our approach includes a simple threshold crossing firing mechanism.

The dynamic terms S_{exc}, S_{inh}, and G are given as IIR temporal filters with time constants τ_{exc}, τ_{inh}, and τ_{refract}. They are modelled using the exponentially decreasing functions given in the left column of Table I and implemented using the approximations given in the right column.

IV. Parallel Computing Strategies

A sequential implementation of the neuron model requires 17 time steps for all the computations. This is optimized by adopting a design strategy that efficiently exploits the parallel computing resources of FPGA devices.

A. Pipeline Computing

Implementing algorithmic parallelism, or pipelining, is a frequently used technique in hardware design to reduce the number of time steps needed to perform complex operations. In order to exploit the inherent parallelism of the FPGA devices, we have designed a pipelined computing structure with $N_{\text{stages}} = 5$ stages (S_1 to S_5; see Fig. 5) that iteratively updates the neural state variables. These stages are as follows.

1) *State fetch:* This stage retrieves the neural state for each neuron from the neural state table [stored in the embedded memory blocks (EMBs)]. This stage also takes the cell input, which can be an input spike from the presynaptic event table (stored in the EMB) or zero in the absence of presynaptic events.
2) *Neural computation I:* Calculate the synaptic gain terms using the IIR filters (S_{exc} and S_{inh}).
3) *Neural computation II:* Calculate the membrane potential (V_x)

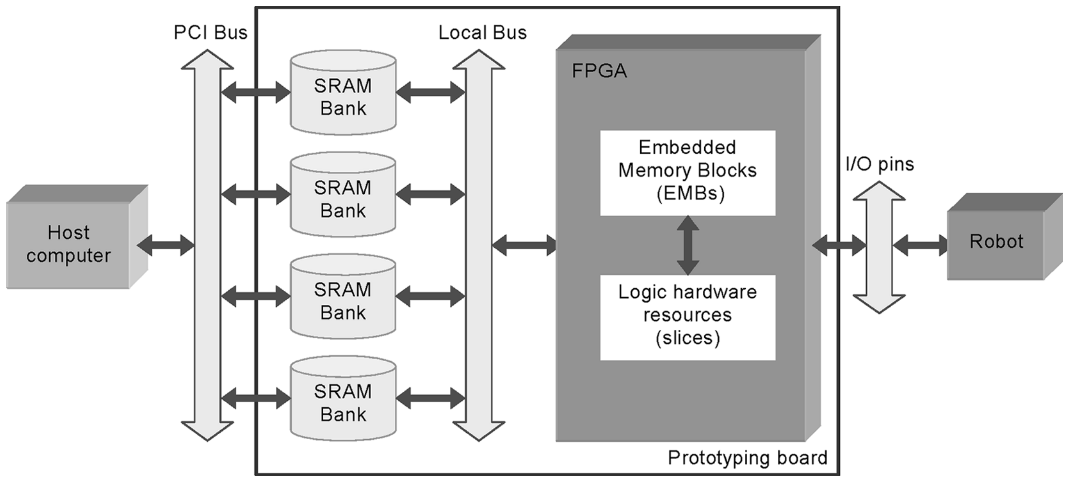


Fig. 3. Software/hardware communication scheme. We use on-board memory SRAM banks to interface the software and hardware components. All the neuron variables (membrane potentials and conductances) are permanently stored in the on-board memory banks. The presynaptic events and postsynaptic events are stored in a memory bank by the software and hardware components respectively. We use on-chip embedded memory resources as buffers to enable the parallel processing of multiple circuits on the chip.

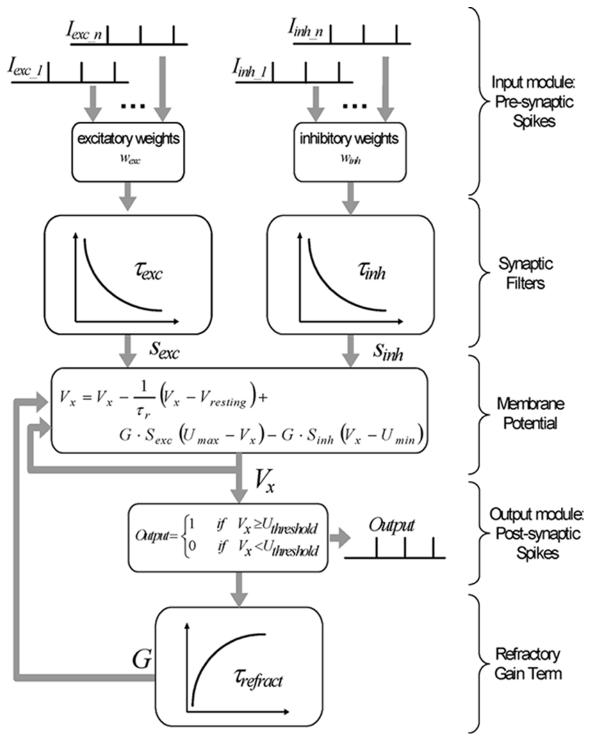

I_{exc_n}
I_{exc_1}
I_{inh_n}
I_{inh_1}
excitatory weights w_{exc}
inhibitory weights w_{inh}
τ_{exc}
τ_{inh}
S_{exc}
S_{inh}
$V_x = V_x - \frac{1}{\tau_r}(V_x - V_{resting}) + G \cdot S_{exc}(U_{max} - V_x) - G \cdot S_{inh}(V_x - U_{min})$
V_x
$Output = \begin{cases} 1 & if\ V_x \geq U_{threshold} \\ 0 & if\ V_x < U_{threshold} \end{cases}$
Output
G
$\tau_{refract}$
Input module: Pre-synaptic Spikes
Synaptic Filters
Membrane Potential
Output module: Post-synaptic Spikes
Refractory Gain Term

Fig. 4. Processing unit scheme. The presynaptic events are summed up in software to reduce the communication costs. The computations indicated in this diagram can be pipelined as described in the next section.

4) *Neural computation III:* Calculate the resting and refractory components (G) and generate a postsynaptic spike if required.
5) *State write-back:* Store the output spike and the neural state in the tables.

It is worthwhile to note that further neural features such as NMDA channels, firing threshold oscillations, etc., can be processed in this pipeline structure in specific stages. Since all the stages are computed in parallel, we could include new neural features without degrading significantly the computation speed by processing them in extra pipelined stages (provided that the maximum number of cycles per stage is kept low). An illustrative example of how to implement a more complex model in a pipeline computing structure designed in FPGA is given in [16].

B. Scalability

Besides the internal parallelism of each data path, the computing platform uses several processing units (PUs) in the FPGA that run in parallel. We have adopted a scalable computing scheme by splitting the embedded memory resources into customized blocks assigned to different computing units. In this way, all the processing units can access to their input and write their output data in parallel as illustrated in Fig. 6.

Besides the N_{PU} processing units, there is another circuit that works in parallel: the *presynaptic event grabber* (PEG) (Fig. 6). At each simulation time step or cycle, the processing units compute new neural states for the presynaptic events and neuron states stored in the embedded memory blocks. At the same time, the PEG loads the events for the next cycle from the on-board external memory (SRAM) into the embedded memory. This is distinct to the communication that occurs with the host every epoch.

To make the processing scheme scalable, each processing unit has its own dedicated EMBs. In this way, the processing units can in parallel access their input data, produce output spikes, and update neuron states. A double buffer of EMBs is required (termed a and b in Fig. 6). While the PEG writes the presynaptic events for the next simulation cycle into a, the processing units use the data for the current cycle from b and write the new neuron states for the next cycle into a. In the next simulation time cycle of the epoch, the tables a and b are switched. This is repeated after each cycle until the end of the epoch.

The adopted event-driven communication scheme allows distributing the simulation engine into several chips properly addressing them within the communication protocol. This multiple-chip scheme enhances the scalability possibilities of the platform beyond the capabilities of a single device. Other authors [7], [48] have already explored similar schemes.

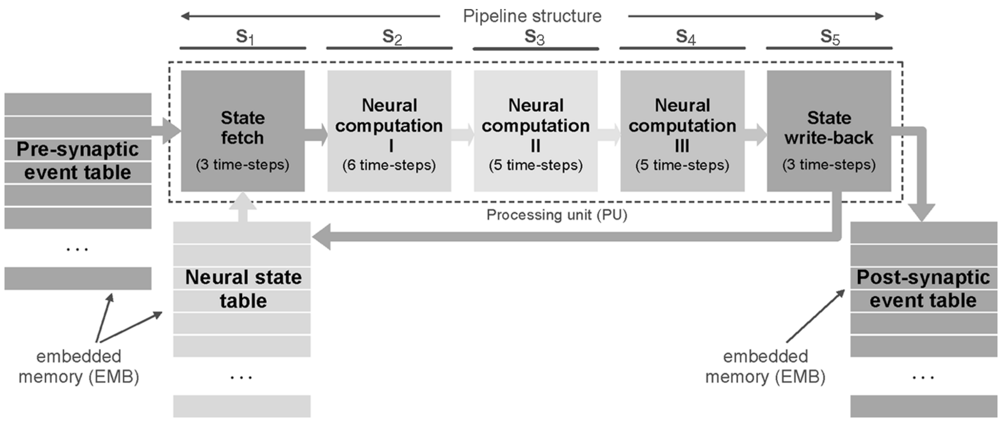

Pipeline structure
S1
S2
S3
S4
S5
State fetch (3 time-steps)
Neural computation I (6 time-steps)
Neural computation II (5 time-steps)
Neural computation III (5 time-steps)
State write-back (3 time-steps)
Processing unit (PU)
Pre-synaptic event table
Neural state table
Post-synaptic event table
embedded memory (EMB)
embedded memory (EMB)

Fig. 5. Pipeline structure. We define five pipeline stages. The computing cycles of each stage are indicated in the figure to show which ones are limiting the whole computation speed.

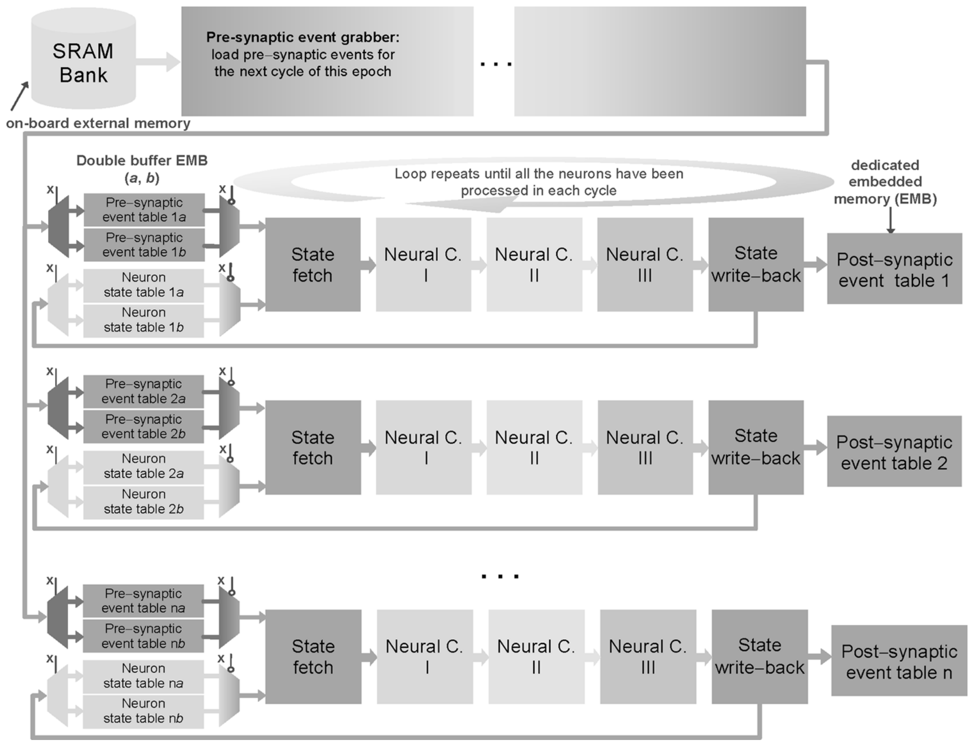

SRAM Bank
on-board external memory
Pre-synaptic event grabber: load pre–synaptic events for the next cycle of this epoch
Double buffer EMB (a, b)
Loop repeats until all the neurons have been processed in each cycle
dedicated embedded memory (EMB)
Pre–synaptic event table 1a
Pre–synaptic event table 1b
Neuron state table 1a
Neuron state table 1b
State fetch
Neural C. I
Neural C. II
Neural C. III
State write–back
Post–synaptic event table 1
Pre–synaptic event table 2a
Pre–synaptic event table 2b
Neuron state table 2a
Neuron state table 2b
Post–synaptic event table 2
Pre–synaptic event table na
Pre–synaptic event table nb
Neuron state table na
Neuron state table nb
Post–synaptic event table n

Fig. 6. Parallelism at the level of processing units. The presynaptic event grabber (PEG) and the processing units run in parallel. The cycles consumed by the PEG depend on the pre/postsynaptic events while the cycles consumed by the processing units depend on the number of simulated neurons (see Fig. 11). While the PEG grabber is using input buffers a, the processing units are using input buffers b.

V. Computational Resources Consumption

The computational load described in Section III has been distributed into different pipelined stages to allow parallel processing along the datapath. The computational resources of each of these stages are summarized in Table II. We have used the RC1000 board of Celoxica [5] as prototyping platform to evaluate the computation scheme. This is a PCI board with four banks of SRAM memory on board (2 MB each) and a Xilinx device with 2 million gates (Virtex-2000E) [49]. Table II summarizes the hardware resources required for the whole computing system with a single processing unit. The resource utilizations given in Fig. 7 are estimates extracted from partial compilations of the circuit. When the system is compiled as a whole, the allocation of resources is optimized and there are hardware resources that end up being shared across stages.

Table II shows that a complete single processing unit consumes less than 25% of the hardware resources (slices), of the device. We can implement up to four processing units running in parallel on a single device as indicated in Figs. 8 and 9. Fig. 7 shows that stages S_1 and S_5 consume similar resources since they are related with the data retrieval and output storage of the pipeline structure. S_2 is the most expensive stage since it includes two IIR filters computed in parallel (with their gain terms). These multiplications require high precision (14 bits for the input terms) to make the filters stable; hence this stage includes two fixed-point multipliers of 14 bits. Furthermore, this stage needs to store the temporal states of the filters (two per neuron), which requires significant memory resources. S_3 uses the variable states of the other stages (reutilizing their memory resources) and requires only additive computations. S_4 includes

TABLE II
COMPUTATIONAL RESOURCES OF A COMPLETE PIPELINED PROCESSING UNIT. TOTAL AMOUNT OF RESOURCES USED AND AS A PERCENTAGE OF THE XILINX DEVICE XCV2000-E. THIS DEVICE INCLUDES ONLY GENERAL-PURPOSE PROCESSING CIRCUITRY (SLICES) AND EMBEDDED MEMORY RESOURCES (160 BLOCKS OF 4 Kbits EACH). THIS PROCESSING UNIT IS ABLE TO COMPUTE SIX NEURONS IN PARALLEL USING A PIPELINE PROCESSING SCHEME

# PUs	# embedded memory blocks	% embedded memory blocks	# slices	% slices	# Total equivalent gates	# time steps
1	80	50	4581	23.86	1389247	6

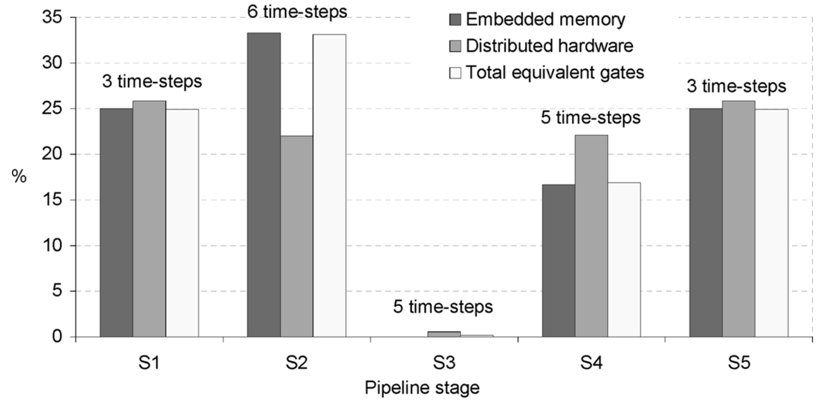


Fig. 7. Computational resources of each pipeline stage. Each bar represents the resources percentage of a single processing unit.

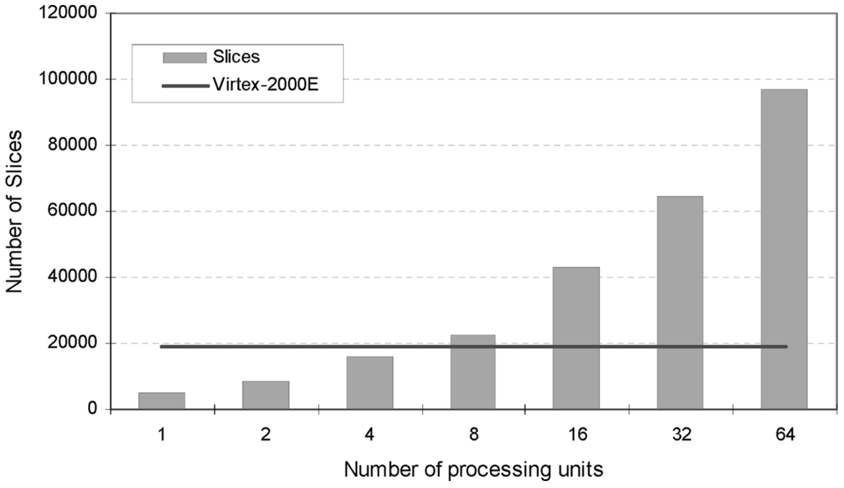


Fig. 8. Hardware resources consumed as the number of slices. Consumption grows with the number of processing units running in parallel on a single FPGA device. We are able to allocate four processing units on the Virtex-2000E.

the output spike generation, the refractory gain term and the resting term (which is done by shifting registers). S_4 requires half of the memory resources of S_2 because in this stage only the refractory state of each neuron needs to be stored.

The number of time steps consumed in each pipeline stage is shown in Fig. 5 to indicate those stages that constrain the effective speed of computation. Since all the stages work in parallel synchronously, the data throughput is limited by the longest stage (S_2 in this case). The longest pipeline stage consumes $\mathrm{NTS}_{lps} = 6\ \mathrm{time\ steps}$ (Fig. 5); each processing unit produces an updated neuron state every NTS_{lps} steps. There is a latency (L) of 30 time steps in computing a neuron state $(L = N_{\mathrm{stages}} \cdot \mathrm{NTS}_{lps})$. If F_{clk} is the clock frequency, the data throughput or number of neural state updates per second is given by

$$R_{\mathrm{neuron}} = \frac{F_{\mathrm{clk}}}{\mathrm{NTS}_{lps}}. \qquad (2)$$

For a circuit design with $\mathrm{NTS}_{lps} = 6$ time steps and $F_{\mathrm{clk}} =$ 25 MHz, then $R_{\mathrm{neuron}} = 4.16 \cdot 10^6$ updates/s.

The computational resources to compute a network of 1024 neurons are detailed in Figs. 8 and 9. The hardware resources increase with the number of processing units. The number of slices increases approximately in a linear fashion (Fig. 8). Memory resources increase slightly up until eight processing units, after which there is a rapid increase (Fig. 9). Up until eight processing units, the memory resources are efficiently used, while for higher numbers of processing units, the memory resources are suboptimally managed because the number of neurons to be simulated in each processing unit is small, given a total network size of 1024 neurons. Using the FPGA of the RC1000 prototyping platform, we have implemented and tested the system with 1024 neurons and are able to place up to four cores on the chip.

The on-chip resources do not depend on the number of neurons simulated or the number of input/output spikes. This is so

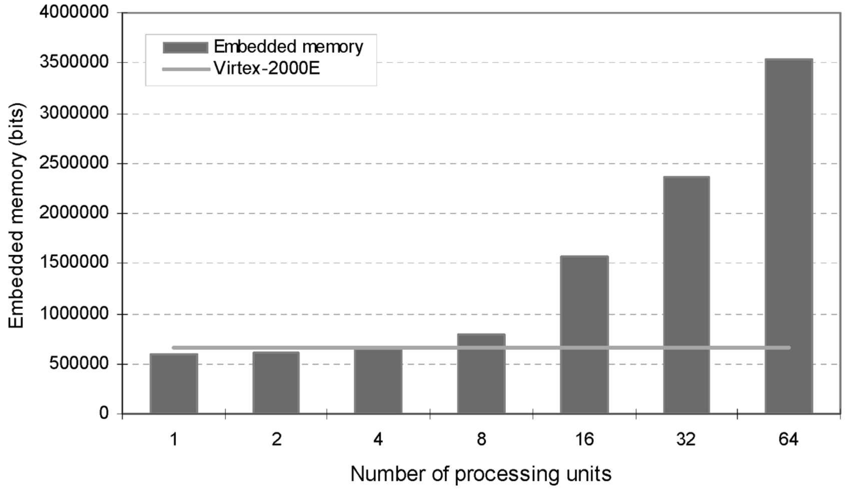


Fig. 9. Embedded memory resources consumed. For fewer than four processing units, the memory resources required barely increase, because the EMBs are fully used. For more than eight processing units, the increment is much more significant because the EMBs are being suboptimally utilized for this network size.

because they are stored into on-board memory resources and they are transferred to the FPGA in packets. Nevertheless, the process that communicates with this on-board memory (*presynaptic event grabber* in Fig. 6) has a cost in time that depends on the number of input/output events that are transferred. This is illustrated in the next section in Fig. 11.

VI. Computing Performance

In Section II, we described the computing scheme based on epochs of N_t time cycles. This computing strategy reduces the communications over the PCI-bus. The time taken by the hardware to compute each epoch t_{epoch} is given by

$$t_{\text{epoch}} = \sum_{i=0}^{N_t-1} t^i_{\text{TimeCycle}} = \frac{1}{F_{\text{clk}}} \sum_{i=0}^{N_t-1} \text{NTS}^i_{\text{TimeCycle}} \tag{3}$$

where $t^i_{\text{TimeCycle}}$ is the computing time of the time cycle $i (i = 0 \ldots N_t - 1)$, $\text{NTS}^i_{\text{TimeCycle}}$ is the number of time steps to compute time cycle i, and F_{clk} is the clock frequency. The presynaptic event grabber and the processing units are working in parallel (Fig. 6) and the slowest limits the effective computing speed. $\text{NTS}^i_{\text{TimeCycle}}$ is given by

$$\begin{aligned} &\text{NTS}^i_{\text{TimeCycle}} \\ &\quad = \text{MAX}\left(\text{NTS}^i_{\text{PEG}}, \text{NTS}^i_{\text{processing-units}}\right) \\ &\text{NTS}^i_{\text{TimeCycle}} \\ &\quad = \text{MAX}\left(1 + \left(N^i_{\text{presynaptic}} \cdot \text{NTS}_{\text{Load-pre}}\right),\right. \\ &\qquad\qquad \left. L + \left(\frac{N_{\text{neurons}} \cdot \text{NTS}_{lps}}{N_{PU}}\right)\right). \end{aligned} \tag{4}$$

On one hand, the time taken by the PEG is variable, as it depends on the number of presynaptic events received for that cycle $(N^i_{\text{presynaptic}})$ and the number of time steps needed to take an event from the external memory to the embedded memory $(\text{NTS}_{\text{Load-pre}})$. On the other hand, the time taken by the processing units is fixed and depends on the number of neurons (N_{neurons}), the number of processing units (N_{PU}), the number of time steps of the longest stage of the pipeline structure (NTS_{lps}), and the latency $(L = N_{\text{stages}} \cdot \text{NTS}_{lps})$. The computing time of each epoch t_{epoch} can be given as

$$\begin{aligned} t_{\text{epoch}} = \frac{1}{F_{\text{clk}}} \sum_{i=0}^{N_t-1} \text{MAX}\big(1 + &\left(N^i_{\text{presynaptic}} \cdot \text{NTS}_{\text{Load-pre}}\right), \\ &L + \left(\frac{N_{\text{neurons}} \cdot \text{NTS}_{lps}}{N_{PU}}\right)\big). \end{aligned} \tag{5}$$

Assuming that the processing units take longer than the presynaptic event grabber, and choosing the following configuration—$N_t = 20$ cycles, $N_{\text{neurons}} = 1024$ neurons, $\text{NTS}_{lps} = 6$ time steps, and $F_{\text{clk}} = 25$ MHz—then the time taken by the hardware using a single processing unit to compute one epoch is $t_{\text{epoch}} \approx 4.94$ ms. The total time to compute an epoch is greater, including the time to communicate the pre- and postsynaptic events and the time to share data between and synchronize the threads in the software component.

A suitable integration step for biologically plausible neurons based on the chosen neuron model is 100 μs. For a simulation to be in real time, an epoch of 20 time cycles must be computed in less than 2 ms. To achieve this we use several computing processing units in parallel. The performance results in Fig. 10 show how the computing time decreases with the number of processing units (given for 1024 neurons, 20 time cycles per epoch, an integration time step of 100 μs, and variable numbers of presynaptic events). The results show that the design is scalable and the processing inherently parallel when the number of presynaptic events is not too high. If the number of presynaptic events is too high, then the speed is limited by the PEG. If all the neurons receive presynaptic events in each cycle of the epoch (the worst case scenario in Fig. 10), then the PEG limits the data throughput and the evaluation time is almost independent of the number of processing units. If no presynaptic events are received in an epoch (the best case scenario in Fig. 10), then the speed of the processing units limits the data throughput, and the design is perfectly scalable. Note that the best case curve follows exactly an exponential function in which the computing time is divided by two as the number of processing units is doubled.

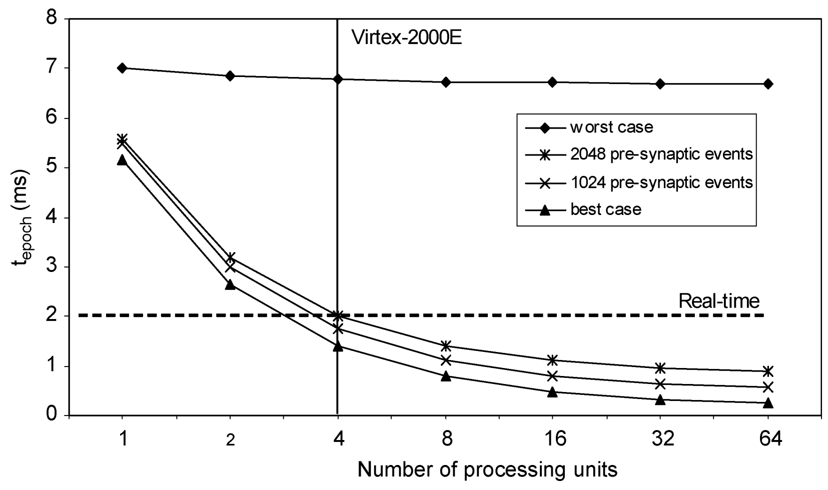


Fig. 10. Scalability study. The cost per epoch is given for 1024 neurons, 20 cycles/epoch, computed at 25 MHz, with varying number of presynaptic events (worst case has 20 480 presynaptic events, best case has zero presynaptic events). The cost omits the transmission time through the 100 MB/s PCI bus using DMA (0.2 ms for 1024 presynaptic events and 0.3 ms for 2048 presynaptic events).

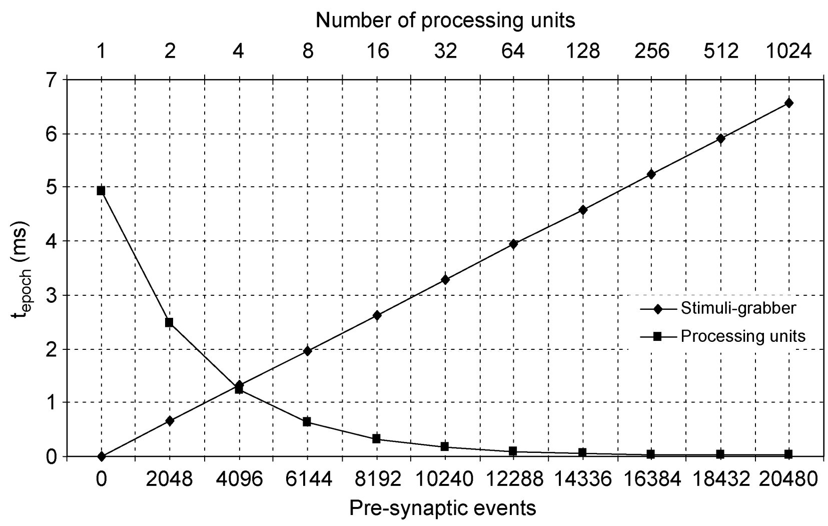


Fig. 11. Scalability study. Given for 1024 neurons, 20 cycles/epoch, and 25 MHz. The cost (omitting transmission time over the PCI bus) depends on the maximum of the cost for the presynaptic event grabber and cost for the processing units, as these processes run in parallel. The optimal configuration is with four processing units handling up to 4096 presynaptic events in real time.

Further illustration is given in Fig. 11, which plots the cost in time for each of the two parallel processes. The cost of the presynaptic event grabber depends on the number of events (bottom axis). The cost of the processing units depends on the number of processing units running in parallel (top axis). This plot illustrates that the coprocessor board can compute 1024 neurons in real time when receiving fewer than 6144 presynaptic events per epoch (one epoch is 20 time-steps of 100 μs of simulated time—for real time, each epoch needs to be computed in less than 2 ms). The optimal configuration is with four processing units on the chip processing 4096 presynaptic events per epoch.

The costs given in Figs. 10 and 11 omit the transmission time through the PCI bus [100 MB/s of measured bandwidth using direct memory access (DMA)]. The output events transmission time is almost insignificant (1024 output events take about 0.1 ms). The maximum number of output events per epoch is limited to the number of simulated neurons. On the other hand, the communication time for the input events is significant (0.3 ms for 2048 presynaptic events and 0.6 ms for 4096). These transmission times need to be taken into account when determining the true capacity of the hardware coprocessor.

VII. Real-Time Cerebellar Simulations

The performance of the system is dependent on the network connectivity and the load given as the number of presynaptic events per epoch. It is therefore important to test its performance within a biologically realistic context. For this we simulate a model of the olivary-cerebellar system applied to a simple tracking task (visual smooth pursuit). Movement in one dimension is modelled using two olivary-cerebellar microcircuits, each representing movement in opposite directions. Details of the number of cells, their connectivities and their parameters are given in Table III. Full details of the model and its application to visual smooth pursuit are given in [1]. Of significance to this discussion are the following points. First, the number of connections to each neuron varies greatly. The cerebellum

TABLE III
TYPES AND PARAMETERS FOR THE OLIVARY-CEREBELLAR MODEL

Cell type	Activity	Number	Fan-in	Fan-out	$\tau_{resting}$ (ms)	τ_{exc} (ms)	τ_{inh} (ms)
Granule	0-100 Hz	242-1010	5	10	10	2	2
Purkinje	30-120 Hz	2	#granule	2	20	5	5
Interneuron	5- 60 Hz	4	#granule	2	20	5	5
Golgi	5- 60 Hz	4	#granule	#granule	20	5	5
Nuclear	30 -70 Hz	2	23	2	10	2	2
Inferior Olive	0-10 Hz	2	2	1	70	5	5
Mossy fibres	20-80 Hz	21		#granule+6			
Feedback	20-80 Hz	2		1			

does not represent a heterogeneous network of neurons, but contains both extremely low and extremely high patterns of connectivity. Each granule cell receives fewer than ten inputs, while each Purkinje cell receives hundreds of thousands. Secondly, the number of presynaptic events per epoch (and the computational load) is widely distributed in the simulations as shown in Fig. 12(a). The cells in the inferior olive fire at around 1 Hz, those in the granular layer appear to be mostly silent with short bursts up to 100 Hz, and the Purkinje cells may average around 80 Hz. Thirdly, the cerebellum is an example of a brain system that needs to be modelled in real time. The cerebellum is involved in the fine control and coordination of timed movement (amongst much else) but its exact role in the brain is unknown. The simplified and artificial version of the smooth pursuit problem used for this demonstration of the hardware can be easily simulated and is not a difficult task for the cerebellum to learn. Providing the realistic and sufficiently rich context that will be needed to elucidate cerebellar function implies being able to study cerebellar models operating in real time in the real world.

All of the neurons are calculated in the coprocessor, except for the mossy fiber and feedback inputs. They encode the information about the smooth pursuit task, such as eye position and movement as well as the retinal slip or movement of the target on the retina. These analog sensory signals are translated into spike trains, and the spike trains outputted from the cerebellar nuclei are translated back into analog signals using software. The model was built to explore the role of climbing fiber-modulated plasticity in the cerebellar cortex and nuclei in cerebellar learning. It uses a spike timing dependent Hebbian algorithm for plasticity at the parallel fiber to Purkinje cell synapses, at the mossy fiber to cerebellar nuclear cell synapses, and at the mossy fiber to granule cell synapses. The performance, however, is measured after the learning has stabilized and the plasticity rule is inactivated, to ensure that it is bound by the performance of the FPGA coprocessor. The details of the learning are given in [1]. The learning is significant to this paper only in that it configures the model to have a realistic activity profile. The speedup with the hardware coprocessor over a software coprocessor is given in Fig. 12(b). The total number of cells in the model is varied by changing the number of granule cells.

Besides robot control, cerebellar-like control schemes are being applied also for other real-world applications [27].

VIII. DISCUSSION

We have described a hybrid hardware–software platform for the realtime simulation of spiking neurons based on the spike response neural model including a gradual injection of charge and synapses modelled as input-driven conductances. This computation scheme is without any of the restrictions on the model imposed by current event-driven schemes. The hardware coprocessor is based on reconfigurable hardware (FPGA), and we have adopted a computing strategy that makes exhaustive use of the processing parallelism resources of the FPGA device. In particular, we have designed a pipeline processing scheme with a data throughput of six clock cycles. Furthermore, we have segmented the memory resources to make the processing system scalable. In this way, several processing units can run efficiently in parallel (avoiding a data access bottleneck).

Different hardware and software accelerators for simulating spiking neural networks simulations are difficult to compare since the time resolution of the integration, their objectives, and the neuron models are all different. Compared to other specific computing platforms [17], [23], [32], [40], [41], our approach addresses the simulation of smaller scale networks but with a higher time resolution (100 μs). The primary difference is that we are interested in real-time (as opposed to simply faster) simulations, as the motivation of this paper is to investigate real-world closed-loop sensorimotor systems (robotic platforms) where real-time processing is critical. Other specific hardware neural implementations based on analog VLSI technology addressed applications related with dynamic clamping [29], [30]. These approaches aim real-time interactions between hardware-based neural models and biological neurons, therefore building artificial and biological hybrid networks in vitro for specific tissue characterization studies. The time

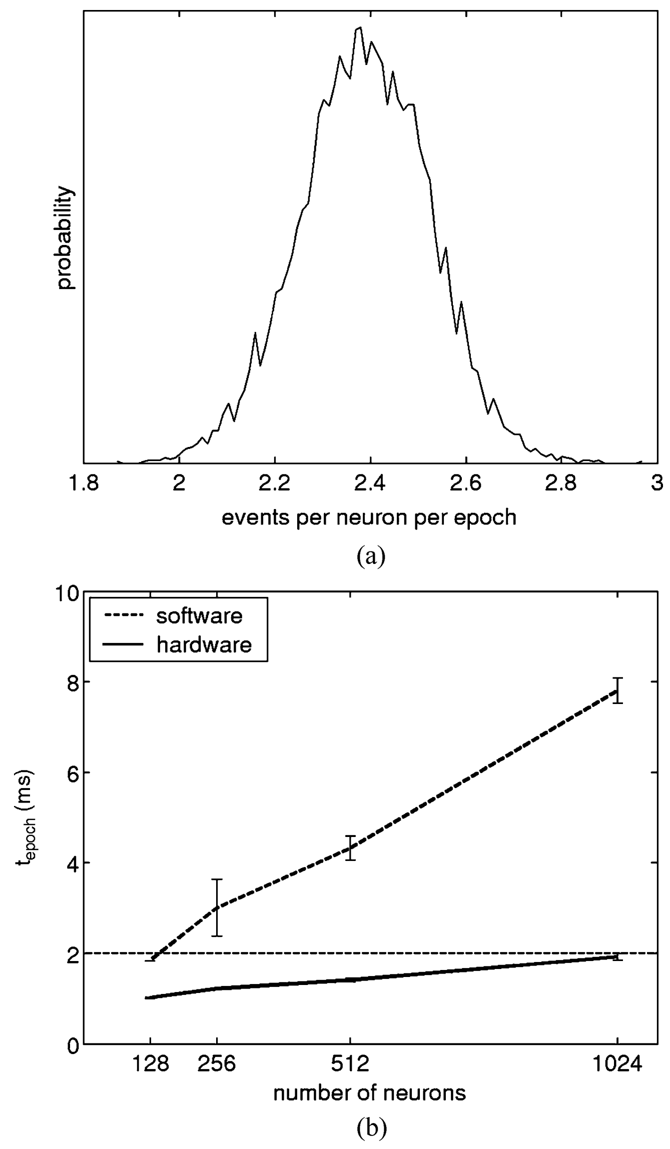


Fig. 12. Performance on cerebellar model. (a) The number of presynaptic spikes per neuron per epoch for the cerebellar model with 242 granule cells applied to a tracking task. (b) The speedup of the hardware coprocessor over a software coprocessor with respect to the total number of neurons.

restrictions of that application domain are very exigent, but the network size and topology flexibility are not so important. Our simulation platform has been designed for robotic applications and sensorimotor studies. Here the capability of simulating different network topologies of small and medium size is critical. Furthermore, the possibility of easily testing different networks for specific processing tasks makes FPGA technology the best choice for our approach.

Our software/hardware hybrid computing scheme enables the simulation of heterogeneous networks, since the topology is defined and simulated in the software components. As described in Section II (Fig. 2), three processes run concurrently in the host. The routing component consults network topology and translates postsynaptic spikes into presynaptic spikes incorporating the connection weight and delay. Another module dedicated to the communication with the hardware board takes into account these connection delays and accumulates coincident presynaptic spikes addressing the same target neuron in order to utilize the limited software–hardware communication bandwidth. This scheme allows the simulation of neural networks with distance depending spike transmission delays which is of specific importance when simulating the neural structure in the olivary-cerebellar system. For example, the parallel fibers to Purkinje cell synapses have specific delay distributions and the conductance speed for the climbing fibers is adjusted for the axonal length to ensure that all fibers have a fixed delay [44].

We have chosen a neuron model with synapses modelled as input-driven conductances. This increases the computational complexity and requires multiplication circuits that are a limited resource on the FPGA. Nevertheless, in our current pipeline processing architecture, this feature is not limiting the global computation speed (i.e., the time to compute a single neuron), which is bound by the input noninstantaneous synaptic coupling (S_2 in Fig. 5). This synaptic model (input-driven conductance) is an important feature to study different aspects of cerebellar processing [2], [4], [33]. Network mechanisms such as neuronal gain modulation at the granular layer [33] modelled by means of a shunting inhibition require cell models with synapses modelled as variable conductances.

The cell model includes the gradual injection of charge in the synaptic modules. This feature allows the study of synchronization processes. How synchronization arises from specific topology [12], [28], [50] instead of intrinsic cell synchronization mechanisms is an important topic. Several studies have demonstrated that synchronization between reciprocally connected inhibitory neurons is facilitated by their finite connections delays and the gradual injection of charge at the synapses [13], [47], although other mechanisms, such as gap junctions, seem to play a complementary role [24]. This finite connection delay separates the generation of the action potential in one of the cells from the peak of the synaptic inhibition in the paired cell [28].

FPGA technology has experienced great advances in recent years and is continuously evolving. We have described the circuits with a high-level hardware description language [45], which facilitates the efficient management of the memory and computational resources. The design is modular and can be easily customized with different levels of parallelism.

With the current prototyping platform (RC1000) we can implement up to four processing units and compute in real time (with an integration time step of 100 μs) up to 1024 neurons receiving 4096 presynaptic events. But we have studied the scalability of the system and the design can be easily adapted to more powerful devices for real time computing of larger neural systems. We have also tested the computing scheme using a RC2000 [5], where we can implement 18 processing units on a single FPGA (of 6000 million gates) being able to compute 90 neurons in parallel (due to the five-stage pipeline processing scheme). With this prototyping board we obtain a speed up of three with respect to the RC1000, depending on the network size. Both of the tested platforms are general purpose prototyping boards. Higher improvements (we would estimate speedups of one order of magnitude) are expected with custom boards (currently under design) incorporating higher memory resources and communication bandwidth.

With the current approach the average network activity is critical, i.e., if the number of presynaptic events received is too high the "computing" time increases. With a short integration time step (100 μs in the experiments of Section VII) the activity per time step will be low. In the cerebellar model, the average number of presynaptic spikes per neuron per 2 ms time step is 2.4 (see also [4] and [6]). In fact, the presented approach uses an iterative computing strategy in the FPGA and an event-driven communication scheme between the host and the coprocessing board. This iterative computing mode allows the exploration of specific issues difficult to track within an event-driven computing scheme. Among these issues are different waveforms of neurotransmitters release probability functions and short-term plasticity, which we plan to address in future work. On the other hand, the event-driven communication scheme takes full advantage of the limited bandwidth. The computing scheme is fully scalable, even though the analysis of the consumption of resources is based on a specific prototyping platform and a reduced number of neurons. There are two main factors that limit the scalability to very large scale simulations: i) the software/hardware communication bandwidth (that motivates the future use of PCI-X boards) and ii) the limited computing parallelism (that increases with more powerful FPGA devices, provided that the on-chip computing scheme is scalable).

The goal of the approach in this paper is the real-time simulation of small- and medium-scale networks. This imposes restrictions on the simulation time resolution and the number of presynaptic events. In the current computation scheme, the maximum number of neurons that can be simulated depends on the on-board available memory. We are currently developing circuit modules to work with DDRAM circuits. This allows one to scale the network size up to $64 \cdot 10^6$ neurons using 1 GB on-board memory, while the number of synaptic connections would be limited by the host computer memory resources. The speedup gained with such a platform compared with a fully software simulation has been illustrated with small-scale networks in Fig. 12. Parallelization of large-scale simulations on clusters of computers with embedded acceleration boards is under study but estimating the speedup in this scenario is speculative, as it depends on different factors such as the network topology, host loads, and internode communication rates. An advantage of the design is that the different elements of the computation, such as the learning, the routing of spikes, and the computation of metrics, can be moved to the hardware coprocessor as they mature. This frees the host CPUs to handle the communication between nodes in a distributed memory system. Currently, the performance bottleneck in the system is imposed by the use of a general-purpose prototyping board (RC1000). Having tested the feasibility and applicability of the presented computing scheme, we have started the design of a purpose-specific FPGA-based board. It includes a state-of-the-art FPGA device as the main computing element, increased on-board memory resources and enhanced PCI-based communication bandwidth.

The comparison of the proposed computing platform with event-driven simulation schemes is not easy, since the adopted neuron model implements mechanisms for the gradual injection necessary for defining synapses with different time constants. This feature is difficult to incorporate into an event-driven simulation schemes although some approaches are exploring this possibility with different restrictions [37], [40].

The presented software/hardware approach provides high flexibility to explore different learning strategies at a system level. As commented in Section IV, the neuron model can be scaled up to higher levels of complexity, incorporating features such as voltage dependent active channels mediated by NMDA receptors, synaptic short-term dynamics, etc. Further circuits can be added to compute these features in parallel. The pipeline-processing scheme provides a significant benefit in that extra computing stages, for example, plasticity, can be added without increasing the per neuron computation time. One problem is that the routing and learning are currently performed on the host computer, which limits the size and the complexity of the model. Using our current software framework and a standard dual-processor workstation, Hebbian plasticity in any more than 5% of the synapses will result in loss of performance. This emphasizes the importance of embedding the learning and the routing of spikes onto the FPGA, exploiting its inherent computing parallel resources. It is not straightforward since it requires the network topology to be stored and managed onboard [18], [32]. This will be future work, once we have built a coprocessor board with sufficiently large memory resources.

Real-time simulations have a long-term significance for neurobiological models in general and cerebellar models in particular. Models of networks away from the sensory and motor periphery and with more than a handful of cells are inevitably hugely under constrained. One solution is to use heterogeneous models that combine constraints at multiple levels of abstraction. Another approach is to evaluate the model within a larger model of a semicomplete biological system, where the modelling is informed by the utility and the behavior of the overall system and where the interactions between the components impose constraints. Real-time modelling is perhaps not important for the study of much behavior in this context, for example, reasoning. The cerebellum, however, has critical roles in the dynamic integration of sensorimotor signals necessary for finding motor coordination, and studying this issue with real agents requires a real-time model. The choice of a neuron with synapses modelled as variable conductances was made to accommodate related work. In this, detailed biophysical conductance-based models are constructed based on the electrophysiology of the different types of cells in the cerebellum [34], [39], [42], [46]. These models are too expensive to investigate large-scale network properties, and so simplified conductance-based models are being derived that will run on the described hardware platform. The goal is to take detailed biophysical characterizations and to identify key functional properties of each cell at the network and systems level.

Our long-term motivation for embedding routing and learning onto the hardware platform is to investigate biological sensorimotor integration and control systems operating in the real world. This work is one step towards a neuromorphic sensorimotor system, i.e., the goal of embedding a complete spiking neural system including all the sensor and actuator subsystems into a stand-alone robotic platform.

REFERENCES

[1] M. Arnold, "Feedback learning in the olivary-cerebellar system," Ph.D. dissertation, Dept. Comp. Sci., Univ. Sydney, Sydney, Australia, 2001.

[2] M. Berends, R. Maex, and E. De Schutter, "A detailed three-dimensional model of the cerebellar granular layer," *Neurocomput.*, vol. 58–60, pp. 587–592, 2004.

[3] M. Bezzi, T. Nieus, O. J.-M. Coenen, and E. D'Angelo, "An integrate-and-fire model of a cerebellar granule cell," *Neurocomput.*, vol. 58–60, pp. 593–598, 2004.

[4] C. Boucheny, R. Carrillo, E. Ros, and O. J.-M. D. Coenen, "Real-time spiking neural network: An adaptive cerebellar model," *Lecture Notes Comput. Sci.*, vol. 3512, pp. 136–144, 2005.

[5] Celoxica, Hardware Platforms 2001–2004 [Online]. Available: http://www.celoxica.com

[6] O. J.-M. D. Coenen, M. Arnold, T. Sejnowski, and M. Jabri, "Parallel fibre coding in the cerebellum for life-long learning," *Auton. Rob.*, vol. 11, no. 3, pp. 291–297, 2001.

[7] V. Dante, P. Del Giudice, and A. M. Whatley, "Hardware and software interfacing to address-event based neuromorphic systems," *Neuromorph. Eng.*, vol. 2, no. 1, pp. 5–6, 2005.

[8] A. Delorme and S. Thorpe, "SpikeNET: An event-driven simulation package for modelling large networks of spiking neurons," *Network Comput. Neural Syst.*, vol. 14, pp. 613–627, 2003.

[9] R. Eckhorn, R. Bauer, W. Jordan, M. Brosh, W. Kruse, M. Munk, and H. J. Reitboeck, "Coherent oscillations: A mechanism of feature linking in the visual cortex?," *Biol. Cyber.*, vol. 60, pp. 121–130, 1988.

[10] R. Eckhorn, H. J. Reitboeck, M. Arndt, and P. Dicke, "Feature linking via stimulus evoked oscillations: Experimental results from cat visual cortex and functional implication from a network model," in *Proc. ICNN I*, 1989, pp. 723–720.

[11] ——, "Feature linking via synchronization among distributed assemblies: Simulations of results from cat visual cortex," *Neural Comput.*, vol. 2, pp. 293–307, 1990.

[12] R. Eckhorn, A. M. Gail, A. Bruns, A. Gabriel, B. Al-Shaikhli, and M. Saam, "Different types of signal coupling in the visual cortex related to neural mechanisms of associative processing and perception," *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1039–1052, Sep. 2004.

[13] U. Ernst, K. Pawelzik, and T. Geisel, "Synchronization induced by temporal delays in pulse-coupled oscillations," *Phys. Rev. Lett.*, vol. 74, pp. 1570–1573, 1995.

[14] D. Gall, F. Prestori, E. Sola, A. D'Errico, C. Roussel, L. Forti, P. Rossi, and E. D'Angelo, "Intracellular calcium regulation by burst discharge determines bidirectional long-term synaptic plasticity at the cerebellum input stage," *J. Neurosci.*, vol. 25, pp. 4813–4822, 2005.

[15] W. Gerstner and W. Kistler, *Spiking Neuron Models*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[16] E. L. Graas, E. A. Brown, and R. H. Lee, "An FPGA-based approach to high-speed simulation of conductance-based neuron models," *Neuroinf.*, vol. 2, pp. 417–435, 2004.

[17] G. Hartmann, G. Frank, M. Schaefer, and C. Wolff, "SPIKE128K—An accelerator for dynamic simulation of large pulse-coded networks," in *MicroNeuro'97*, 1997, pp. 130–139.

[18] H. H. Hellmich and H. Klar, "SEE: a concept for an FPGA based emulation engine for spiking neurons with adaptive weights," in *5th WSEAS Int. Conf. Neural Networks Applications (NNA '04)*, Udine, Italy, 2004, pp. 930–935.

[19] J. Hill, W. McColl, D. Stefanescu, M. Goudreau, K. Lang, S. Rao, T. Suel, T. Tsantilas, and R. Bisseling, "BSPlib: The BSP programming library," *Parallel Comput.*, vol. 24, no. 14, pp. 1947–1980, 1998.

[20] M. Ito, *The Cerebellum and Neural Control*. New York: Raven, 1984.

[21] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1063–1070, Sep. 2004.

[22] A. Jahnke, T. Schoenauer, U. Roth, K. Mohraz, and H. Klar, "Simulation of spiking neural networks on different hardware platforms," in *ICANN97*, 1997, vol. 1327, Lecture Notes Comput. Sci., pp. 1187–1192.

[23] A. Janke, U. Roth, and H. Klar, "A SIMD/dataflow architecture for a neurocomputer for spike-processing neural networks (NESPINN)," in *Proc. MicroNeuro'96*, 1996, pp. 232–237.

[24] N. Kopell and B. Ermentrout, "Chemical and electrical synapses perform complementary roles in the synchronization on interneuronal networks," *Proc. Nat. Acad. Sci. USA*, vol. 101, no. 43, pp. 15482–15487, 2004.

[25] G. La Camera, W. Seen, and S. Fusi, "Equivalent networks of conductance- and current-driven neurons," *Lecture Notes Comput. Sci.*, vol. 2714, pp. 449–452, 2003.

[26] ——, "Comparison between networks of conductance- and current-driven neurons: stationary spike rates and subthreshold depolarization," *Neurocomput.*, vol. 58–60, pp. 253–258, 2004.

[27] C. M. Lin and Y. F. Peng, "Missile guidance law design using adaptive cerebellar model articulation controller," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 636–644, May 2005.

[28] R. Maex and E. De Schutter, "Resonant synchronization in heterogeneous networks of inhibitory neurons," *J. Neurosci.*, vol. 23, no. 33, pp. 10503–10514, 2003.

[29] S. Le Masson, A. Laflaquiere, T. Bal, and G. Le Masson, "Analog circuits for modeling biological neural networks: Design and applications," *IEEE Trans. Biomed. Eng.*, vol. 46, no. 6, pp. 638–645, Jun. 1999.

[30] G. Le Masson, S. R. Le Masson, D. Debay, and T. Bal, "Feedback inhibition controls spike transfer in hybrid thalamic circuits," *Nature*, vol. 417, pp. 854–858, 2002.

[31] M. Mattia and P. Del Giudice, "Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses," *Neural Comput.*, vol. 12, pp. 2305–2329, 2000.

[32] N. Mehrtash, D. Jung, H. H. Hellmich, T. Schoenauer, V. T. Lu, and H. Klar, "Synaptic plasticity in spiking neural networks ($\mathrm{SP^2INN}$): A system approach," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 980–992, Sep. 2003.

[33] S. J. Mitchell and R. A. Silver, "Shunting inhibition modulates neuronal gain during synaptic excitation," *Neuron*, vol. 38, no. 3, pp. 433–445, 2003.

[34] Z. Nusser, S. Cull-Candy, and M. Farrant, "Differences in synaptic GABA(A) receptor number underlie variation in GABA mini amplitude," *Neuron*, vol. 19, no. 3, pp. 697–709, 1997.

[35] D. Philipona and O. J.-M. D. Coenen, "Model of granular layer encoding in the cerebellum," *Neurocomput.*, vol. 58–60, pp. 575–580, 2004.

[36] J. Reutimann, M. Giugliano, and S. Fusi, "Event-driven simulation of spiking neurons with stochastic dynamics," *Neural Comput.*, vol. 15, pp. 811–830, 2003.

[37] E. Ros, R. R. Carrillo, E. M. Ortigosa, B. Barbour, and R. Agís, "Event-driven simulation scheme for spiking neural networks using look-up tables to characterize neuronal dynamics," *Neural Comput.*, 2006, accepted for publication.

[38] R. R. Carrillo, E. Ros, B. Barbour, C. Boucheny, and O. Coenen, "Event-driven simulation of neural population synchronization facilitated by electrical coupling," *Biosyst.*, 2006, accepted for publication.

[39] D. J. Rossi and M. Hamann, "Spillover-mediated transmission at inhibitory synapses promoted by high affinity alpha6 subunit GABA(A) receptors and glomerular geometry," *Neuron*, vol. 20, no. 4, pp. 783–795, 1998.

[40] T. Schoenauer, S. Atasoy, N. Mehrtash, and H. Klar, "NeuroPipe-chip: A digital neuro-processor for spiking neural networks," *IEEE Trans. Neural Netw.*, vol. 13, no. 1, pp. 205–213, Jan. 2002.

[41] M. Shaefer, T. Schoenauer, C. Wolff, G. Hartmann, H. Klar, and U. Rueckert, "Simulation of spiking neural networks—Architectures and implementations," *Neurocomput.*, vol. 48, pp. 647–679, 2002.

[42] R. A. Silver, D. Colquhoun, S. G. Cull-Candy, and B. Edmonds, "Deactivation and desensitization of non-NMDA receptors in patches and the time course of EPSCs in rat cerebellar granule cells," *J. Physiol.*, vol. 493, no. 1, pp. 167–173, 1996.

[43] L. S. Smith and D. S. Fraser, "Robust sound onset detection using leaky integrate-and-dire neurons with depressing synapses," *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1125–1134, Sep. 2004.

[44] I. Sugihara, E. J. Lang, and R. Llinas, "Unifrom olivocerebllar conduction time underlies Purkinje cell complex spike synchronicity in the rat cerebellum," *J. Physiol.*, vol. 470, pp. 243–271, 1993.

[45] Celoxica, Technical library [Online]. Available: http://www.celoxica.com/techlib/default.asp

[46] S. Tia, J. K. Wang, N. Kotchabhakdi, and S. Vicini, "Developmental changes of inhibitory synaptic currents in cerebellar granule neurons: Role of GABA (A) receptor alpha 6 subunit," *J. Neurosci.*, vol. 16, no. 11, pp. 3630–3640, 1996.

[47] C. van Vreeswjik, L. F. Abbott, and G. B. Ermentrout, "When inhibition not excitation synchronizes neural firing," *J. Comput. Neurosci.*, vol. 1, pp. 313–321, 1994.

[48] R. J. Vogelstein, U. Mallik, and G. Cauwenberghs, "Beyond event-driven communication: dynamically-reconfigurable spiking neural systems," *Neuromorph. Eng.*, vol. 1, no. 1, pp. 1, 9, 2004.

[49] Xilinx, FPGA devices 1994–2003 [Online]. Available: http://www.xilinx.com

[50] B. Yu and L. Zhang, "Pulse-coupled neural networks for contour and motion matchings," *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1186–1201, Sep. 2004.

Eduardo Ros received the Ph.D. degree from the University of Granada, Granada, Spain, in 1997.

He is currently an Associate Professor in the Department of Computer Architecture and Technology at the same university. There, he is responsible for two European projects related with bioinspired processing schemes. His research interests include simulation of biologically plausible processing schemes, biomedical signal processing, hardware implementation of digital circuits for real-time processing in embedded systems, and computer vision.

Eva M. Ortigosa received the Ph.D. degree in computer engineering from the University of Málaga, Málaga, Spain, in 2002.

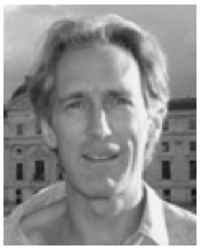
She was with the Computer Architecture Department, University of Málaga, from 1996 to 2002. Since then, she has been with the Computer Architecture and Computer Technology Department, University of Granada, Granada, Spain. Her current areas of research interest are in the fields of neuromorphic engineering, FPGA implementation of neural-like systems, spiking neuron models and synchronization processes, bioinspired processing systems, and neural networks.

Rodrigo Agís received the M.S. degree in computer science from the University of Granada, Granada, Spain, in 2003.

He is currently a Research Assistant with the Department of Computer Architecture and Technology at the same university. He has long experience designing control outlines for robots. He is also interested in robot mechatronics and real-time processing architectures. His current research work is related to the design of processing schemes and bioinspired circuits, reconfigurable hardware, and implementation of digital circuits for real-time processing in embedded systems.

Richard Carrillo received the M.S. degree in computer science from the University of Granada, Granada, Spain, in 2002, where he is currently working toward the Ph.D. degree in computer engineering.

His work involves developing efficient computation engines for the simulation of large biological neural networks in real time. Among his research interests are spiking neural networks and the study of perception-action loops in biological systems.



Michael Arnold received the Ph.D. degree in computer science from the University of Sydney, Sydney, Australia, in 2002.

He is currently a Postdoctoral Fellow with the Computational Neurobiology Laboratory, Salk Institute for Biological Studies, San Diego, CA, and is a Partner with Altjira SA, Switzerland. His research interests include development and learning in autonomous sensorimotor systems, simulation of biological systems, and software for modeling complex systems.