International Journal of Electronics

Hardware event-driven simulation engine for spiking neural networks

Online Publication Date: 01 May 2007
To cite this Article: Agís, R., Ros, E., Díaz, J., Carrillo, R. and Ortigosa, E. M. , (2007) 'Hardware event-driven simulation engine for spiking neural networks', International Journal of Electronics, 94:5, 469 - 480
To link to this article: DOI: 10.1080/00207210701308625
URL: http://dx.doi.org/10.1080/00207210701308625

International Journal of Electronics,
Vol. 94, No. 5, May 2007, 469–480

Hardware event-driven simulation engine for spiking neural networks

R. AGÍS*, E. ROS, J. DÍAZ, R. CARRILLO and E. M. ORTIGOSA

Department of Architecture and Computers Technology
of the University of Granada, Spain

(*Received 16 January 2006; in final form 30 November 2006*)

The efficient simulation of spiking neural networks (SNN) remains an open challenge. Current SNN computing engines are still far away from simulating systems of millions of neurons efficiently. This contribution describes a computing scheme that takes full advantage of the massive parallel processing resources available at FPGA devices. The computing engine adopts an event-driven simulation scheme and an efficient next-event-to-go searching method to achieve high performance. We have designed a pipelined datapath, in order to compute several events in parallel avoiding idle computing resources. The system is able to compute approximately 2.5 million spikes per second. The whole computing machine is composed only of an FPGA device and five external memory SRAM chips. Therefore, the presented approach is of high interest for simulation experiments that require embedded simulation engines (for instance, in robotic experiments with autonomous agents).

Keywords: Event-driven; Pipelined datapath; FPGA device; SNN computing engines

1. Introduction

The simulation of biologically plausible neural networks is a challenging task. Several properties of the biological nervous systems must be taken into consideration, in order to build up an efficient computing scheme.

- Biological neural networks consist of massive parallel computing resources organized in very densely connected topologies.
- Most of the information exchanged among the different computing elements (neurons) is encoded in spikes.
- The firing rate of biological neurons is low (with maximum rates of approximately 100 Hz). Therefore, the global activity depends on the network size and the average neuron firing rate.

On the other hand, the current technology has very different characteristics that can be exploited by adopting proper computing schemes.

*Corresponding author. Email: ragis@atc.ugr.es

- In general, digital circuits are able to work at very high clock rates (MHz or even GHz).
- The physical circuit connectivity is limited to 2-D patterns that consume large device resources. Therefore, densely connected topologies cannot be implemented directly on VLSI technology; however, it is impossible to adopt multiplexing techniques on shared buses.

The simulation of spiking neural networks using standard integration methods on conventional computational architectures (single- or multiprocessor platforms) is inefficient (Jahnke *et al.* 1997). This has motivated the implementation of specific hardware platforms to perform neural integration (Schoenauer *et al.* 2002, Mehrtash *et al.* 2003, Ros *et al.* 2006a). The approaches described in Schoenauer *et al.* (2002) and Mehrtash *et al.* (2003) implement a specific type of spike response model (SRM) (Eckhorn *et al.* 1989). We can find in Ros *et al.* (2006a) a description of a time-driven hybrid hardware and a software simulation scheme for networks of SRM neurons. Nevertheless, the spiking neuron model described in Ros *et al.* (2006a) incorporates additional biophysically inspired features, such as spike-driven synapses modelled as conductances, and it targets real-time simulations using time-slicing techniques. The neuron model of this contribution is the same as in Ros *et al.* (2006a), but the platform presented here adopts an event-driven scheme entirely simulated in hardware. Furthermore, the event handling scheme is radically different to any previous approach, since it is based on disordered lists that are efficiently accessed through a parallel searching engine that takes full advantage of the parallel resources of FPGA devices.

Although there are hardware approaches that implement efficient time-driven simulation schemes (Graas *et al.* 2004, Glackin *et al.* 2005, Ros *et al.* 2005, 2006a) in FPGA, the features enumerated above have motivated the development of event-driven processing schemes. This computing scheme is usually implemented in software (Delorme *et al.* 1999, Mattia and Del Guidice 2000, Delorme and Thorpe 2003, Makino 2003, Reutimann *et al.* 2003, Ros *et al.* 2006b), but has also been adopted in hardware approaches (Schoenauer *et al.* 2002, Mehrtash *et al.* 2003).

In this work, we present a specific purpose computing architecture to efficiently simulate spiking neural networks by adopting an event-driven scheme. The common approach is based on a queue of events ordered chronologically (Schoenauer *et al.* 2002, Mehrtash *et al.* 2003, Glacking *et al.* 2005, Ros *et al.* 2006b). In this case, the goal is to reduce the number of accesses required for the correct insertion of a new spike in its correct position. Specific purpose event-driven processing architectures (Schoenauer *et al.* 2002, Mehrtash *et al.* 2003) are based on chronologically ordered lists. Contrary to this approach, we use a disordered event list. Our processing scheme searches for the next-event-to-go before each computing loop. For this purpose, we implement a parallel searching strategy that takes full advantage of the parallel processing resources available in FPGA devices.

In event-driven simulation schemes, the neuron state variables are updated when it fires or receives a spike. Therefore, the simulation engine is able to jump from one spike to the next one. In this way, all the activity (spikes) of a network is queued in chronological order and processed sequentially. This scheme is very appropriate for sequential computing platforms. All the events need to be processed in a chronological order; this scheme is hardly parallelizable, since the events computed on different processing nodes can affect directly the event-list by producing new events or invalidating old ones. Therefore, events can only be processed in parallel adopting a speculative attitude and allow stepping backwards, if inconsistencies (interdependencies) are detected during the simulation. We implement a pipelined processing structure to further accelerating the simulator, but this requires the consideration of inter-spike dependency risks.

In this work, we focus on processing speed as the main performance indicator. Therefore, the goal is to design a system that is able to process the maximum number of spikes per second.

2. Description of the computing scheme

The computing scheme is illustrated in figure 1. The event list is stored on embedded memory resources, in order to facilitate the insertion and searching processes.

On the other hand, the neural state variables and the network topology are stored on external memory SRAM. The computing scheme description and the searching process are discussed in § 2.2.

2.1 *Scalable next-event selection: pick-up strategy*

In order to facilitate the insertion processes, we use a disordered event list. In this case, every time we need to extract an event, we search for the one with a minimum time label. We implement a parallel searching strategy taking full advantage of the parallel computing resources of the FPGA devices.

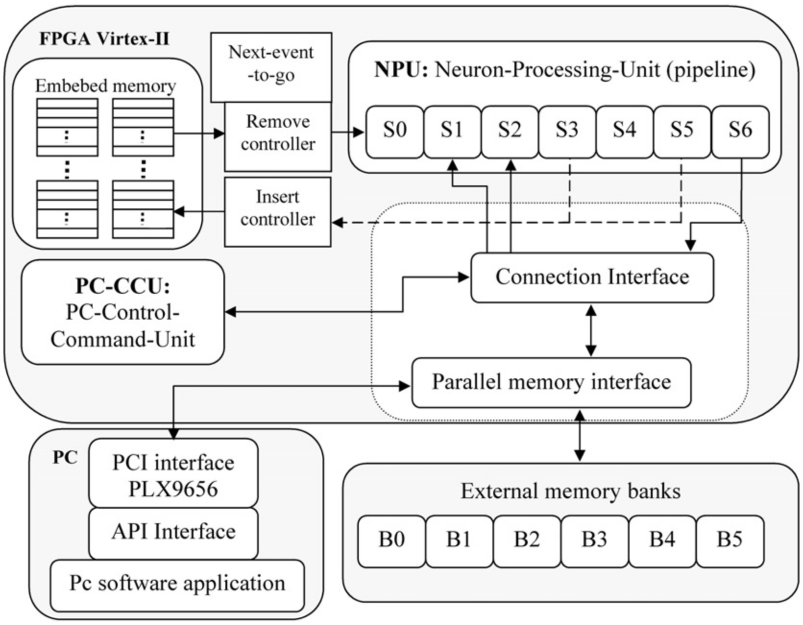

Figure 1. Computing architecture schematic.

Each event is characterized by four fields: time label, synapse identifier, source neuron, and target neuron. We distribute the storage of the time labels in different dual port embedded memory blocks (EMBs) of 512×32 bits to allow parallel access to a high number of elements. We implement parallel comparator circuits of 4 and 8 elements each. We use distributed memory buffers to segment the searching process in several comparator stages in a micro-pipelined structure (as shown in figure 2).

The reconfigurability of the FPGA facilitates the modification of the amount of memory resources allocated for specific tasks. Therefore, depending on the global network activity, it may be convenient to use more or less embedded memory blocks for the time labels of the spikes.

We have implemented a pipelined searching structure to efficiently handle event lists of up to 2^{14} spikes. The events are distributed in 128 dual port EMBs, in order to allow reading 256 elements in parallel to fill a buffer implemented in distributed memory. This allows these 256 elements to access 32 comparator circuits of 8 elements each, producing 32 candidates that are stored in the second buffer (on distributed memory). These 32 elements access 4 comparator circuits of 8 elements each, producing 4 candidates that are stored in the third buffer. Finally, a single comparator of 4 elements provides the selected element out of the primary 256 candidates. After this is done, this winning event is stored as the one with the minimum time label. This scheme is further expanded sequentially in the following manner: the next event that goes out of this pipelined searching structure is compared with the one stored previously Last-mim. In this way, with this last sequential comparator cycle, we are able to manage event lists of up to 2^{14} elements consuming up to 69 clock cycles. Note that in order to pipeline this processing datapath, we need to store in buffers (distributed memory) not only the time labels, but also an index to identify the original spike (in the embedded memory block that is being processed).

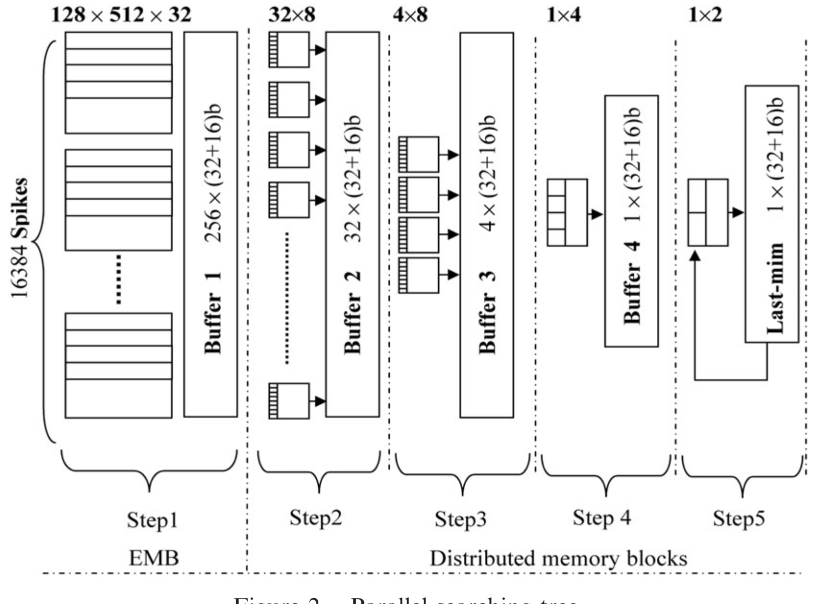

Figure 2. Parallel searching tree.

In fact, this scheme can be further scaled up by using external memory banks to manage larger event lists, at the cost of reducing significantly the searching speed when all the input memory resources are saturated

$$N_{\text{clk\_cycles\_search}} = 4 + \left\lceil \frac{T_{\text{EMB}}}{256} \right\rceil \tag{1}$$

The number of clock cycles required in this searching structure ($N_{\text{clk\_cycles\_search}}$) is given by expression (1). T_{EMB} denotes the number of spikes stored in all the EMBs used by the searching module, and $\lceil\,\rceil$ denotes the function that produces the first integer above the considered real number. The offset of 4 is given by the number of cycles consumed while filling the pipeline structure.

Note that, after the next-event-to-go is found, all the data fields of this event need to be retrieved, which also consumes another 5 cycles.

2.2 *Memory resources for the event list*

A high number of events addressable in the same clock cycle accelerate the whole parallel searching datapath. After the next-event-to-go is selected, as commented above additionally, some further clock cycles are needed due to the different data fields (d_f) retrieval related with this event. Depending on the pipelined structure and processing scheme, the sequential access to the data fields can degrade the performance. Summarizing, two factors become important to achieve high performance during the searching and retrieval of the next-event-to-go:

1. maximum parallelism on the next time label searching;
2. maximum parallelism on the next-event-to-go data fields reading.

There are different strategies that can be adopted as discussed in the following subsections.

2.2.1 Complete events on external memory resources. We can store the events on external memory SRAM circuits. Nevertheless, in this case, the parallel access to time labels of different events is restricted. For instance, a computing platform with M_{SRAM} supporting SRAM chips (of 32 word length) accessible in parallel can retrieve up to M_{SRAM} events in each access. In this way, the number of clock cycles ($N_{\text{clk\_cycles\_search}}$) required to search the next-event-to-go is given by expression (2). Both terms of this equation are related to the searching process. In this case, $N_{\text{clk\_cycles\_search}}$ depends linearly on the number of events N_E, an offset that relies on the number of pipelined stages of the searching tree C, and the last term is related to the time consumed to retrieve the data fields of the next-event-to-go that are also stored on external memory

$$N_{\text{clk\_cycles\_search}} = \frac{N_E}{M_{\text{SRAM}}} + C + \frac{d_f}{M_{\text{SRAM}}} \tag{2}$$

However, the parallel access to events is limited to M_{SRAM}. This scheme is very scalable to a high number of events. Finally, the fact that access to external memory is also constrained by the characteristics of the SRAM chips (latency, maximum data throughput, etc) has to be taken into account.

2.2.2 Complete events on embedded memory blocks. In order to allow a higher level of parallelism in the event searching process, we can store the events on embedded memory blocks. This is the strategy that we have adopted for our experiments. In this case, the level of parallelism will depend on the number of EMBs dedicated to this issue. If we implement dual port memory banks and we distribute the events in a balanced way among the available EMBs, we can access in parallel $P_a = 2M_{\mathrm{EMB}}$ data ports (where M_{EMB} is the number of dedicated EMBs). If each event consists of d_f data fields (time label, synapse identifier, source neuron, target neuron, etc) of 32 bits, and we store all of them in EMBs, in a Virtex II device, we would be able to store N_E events, being $N_E = M_{\mathrm{EMB}}\, 512/d_f$. With this approach, we achieve a considerable access parallelism to time labels of different events, but we consume valuable EMB resources to store further data (the other data fields of the events that are not needed during the searching process in which the parallel access is critical). This strategy will limit the scalability of the system in terms of workload (maximum size of the event list) but maximizes the parallel access (P_a) during the searching process. The problem is that once the next-event-to-go is selected, the rest of the data fields need to be retrieved sequentially (spending a few more clock cycles at the end of the searching process). The number of cycles required for searching the next-event-to-go is shown in expression (3)

$$N_{\mathrm{clk\_cycles\_search}} = \frac{N_E}{2 \cdot M_{\mathrm{EMB}}} + C + d_f \tag{3}$$

2.2.3 Time labels on specific EMBs. We can separate the data fields of the events by storing the time labels on a few dedicated EMBs and the other data fields on some other EMBs. This can be seen as storing the event data fields in long word width memory resources. But in this case, the access parallelism for a given number of events is constrained, since some of the EMBs are not accessed in the searching process. This approach dedicates specific access ports to the EMBs with time labels (this communication bus is used during the searching process) and other different ports to access the other data fields (this bus is used to retrieve the data fields of the next-event-to-go). In this case, the number of clock cycles, consumed in the searching process, depends linearly on the number of events (N_E) and the data fields (d_f), as expressed in equation (4)

$$N_{\mathrm{clk\_cycles\_search}} = \frac{N_E \cdot d_f}{2 \cdot M_{\mathrm{EMB}}} + C + 1 \tag{4}$$

2.2.4 Event fields on interleaved embedded memory. Another option is to conveniently interlace the events on the different EMBs in a way that during the searching process only the time labels are accessed. But once the next-event-to-go is selected, the rest of the data fields of the events are retrieved in a single access cycle. This approach maximizes the number of parallel ports for the searching process, and also optimizes the access to the rest of the data fields of the selected event, as indicated in expression (5). Interleaved memory architecture is a useful technique used in vector processors to exploit their parallelism, which is amply discussed in Briggs and Davidson (1977).

$$N_{\text{clk\_cycles\_search}} = \frac{N_E}{2 \cdot M_{\text{EMB}}} + C + 1 \tag{5}$$

2.2.5 Some event fields stored on specific embedded memory blocks. The events' time labels are stored in a balanced way in EMBs, to maximize the access parallelism ($P_a = 2M_{\text{EMB}}$). The other data fields of the events are stored in external memory resources. This is not very limiting, since only a single event (next-event-to-go) will be completely retrieved per processing iteration. In this way, we also maximize the size of the event list that can be efficiently managed using EMBs ($N_E = M_{\text{EMB}} \cdot 512$). If we use external memory with 2 cycles of time delay, the performance will be the same as in expression (5), but adding 2 more clock cycles (external memory latency).

2.3 *Pipelined event-processing datapath*

The computing strategy is outlined in the block diagram of figure 3. The different stages are the following: (S0) the next-event-to-go is searched (this is done through a parallel searching tree, as described in the previous section); (S1) access to memory is gained in order to retrieve the source neuron state variables and the connection characteristics; (S2) the target neuron state is loaded; (S3) the next spike (if there remains any spike of the output connection tree to be processed) of this source neuron connection tree is inserted into the event list; (S4) the target neuron state is updated (including learning); (S5) the axon-hillock is processed (spike firing decision), and (S6) the target neuron state and connection weight is stored (updated in the learning module).

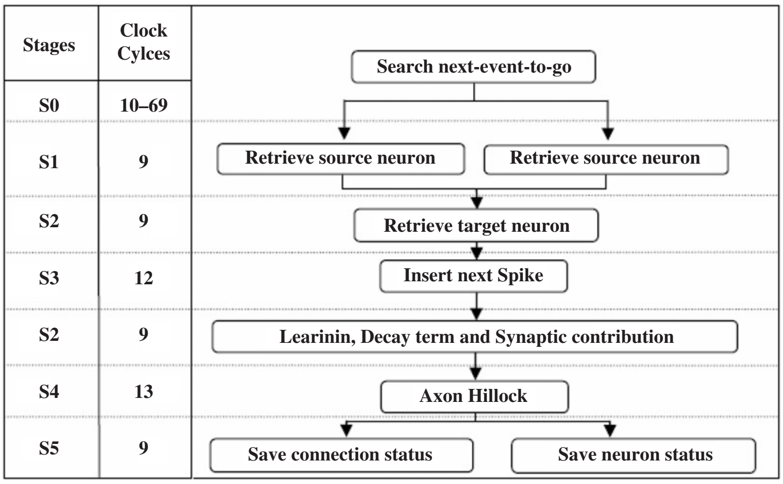

Figure 3. Pipeline datapath. Seven stages in the datapath.

When a neuron fires, it produces multiple spikes that will reach different target neurons according to the network topology. Each connection is characterized by a weight and a synaptic delay. In order to restrict the number of spike insertions, we consult the output connection tree in each computation cycle (ordered according to the synaptic delays) of a neuron that has fired, and we insert just the next-event according to the synaptic delay. This keeps the event list at a manageable size.

We have implemented a pipelined event-processing datapath consisting of the 7 stages outlined in figure 3. Note that for lists of up to 2048 events, the next-event searching structure consumes less than 13 clock cycles, therefore not degrading the global processing performance. All the processing stages are quite balanced, being the spike insertion process that requires 13 clock cycles the limiting one. This leads to a performance of more than 2.5 million spikes per second with a system clock rate of 33 MHz (provided that the event list size is shorter than 2048 elements).

The clock cycles consumption of each of the stages of the pipelined datapath is included in figure 3. The stage S0 consumes between 10 and 69 cycles, due to the fact that the number of clock cycles depends on the size of the event list. But even with the optimized stage, without a global coarse pipeline, we obtain a data throughput between 464 000 and 272 000 spikes per second. This has motivated the pipelined processing structure that allows performances between 2.5 and 0.478 million spikes per second. We need to take into account that depending on the network topology, we will have up to 1% of performance degradation, due to inter-spike risks. This occurs when S5 (Axon Hillock) inserts a new event in the event list with a time label, which is below the one of the last spike that entered the pipeline structure. In this case, it is necessary to reset the whole datapath to keep the chronological processing order. The performance values on figure 5 have been obtained with a circuit running at 33 MHz. We can estimate the performance degradation for a specific neural system. For instance, a network of 100 neurons, densely connected through an all-to-all topology and an average latency of $3\,\text{ms} \pm 2\,\text{ms}$ (standard deviation of a Gaussian distribution). Each neuron fires at an average rate of 10 Hz. In this case, seven events can be processed in parallel in the pipelined datapath. The pipeline needs to be restarted when an event enters into the datapath and its time label is shorter than the time label of another event being processed in another stage of the computing architecture. That means that the event that arrives occurred before an event that is currently being processed, and therefore may affect the network state. Since events need to be processed in a chronological order, the pipeline needs to be restarted. In this case, the performance degradation is below 0.185%. This degradation grows with the average latency and the standard deviation of this value in the networks, as this is what generates inter-spike risks.

2.4 *Neural model*

The described general architecture is valid for multiple neuron models. In fact, the neural state computation is a single processing stage that can be seen as a black box.

The only restriction is that the neural model allows the neural state variables to be updated discontinuously. Currently, we are using the proposed platform to test bio-inspired robotic control experiments (Boucheny *et al.* 2005) with the neural model illustrated in table 1.

Table 1. Neural Model Characteristics. V_x denotes the membrane potential and W the connection weight. The weight is uploaded according to the first expression. The connection between cells K and J turns stronger or weaker, depending on the inter-spike time between the events produced by both neurons.

Stage	Feature	Clock cycles	Expression
S4	Learning of synapse (i)	3	$\Delta W_i^{t_k} = 27 \cdot (t_k^i - t_j^i) \cdot e^{((t_j^i - t_k^i)/10)}$
	Passive decay	4	$V_x = V_x - \left[\frac{V_{\max}}{\tau} \cdot (t_k - t_j)\right]$
	Synaptic contribution	2	$V_x = V_x + W_i$
S5	Axon Hillock (spike insertion)	13	$\text{Out} = \begin{cases} 1 & \text{if} \quad V_x \geq V_{\text{threshold}} \\ 0 & \text{if} \quad V_x \geq V_{\text{threshold}} \end{cases}$

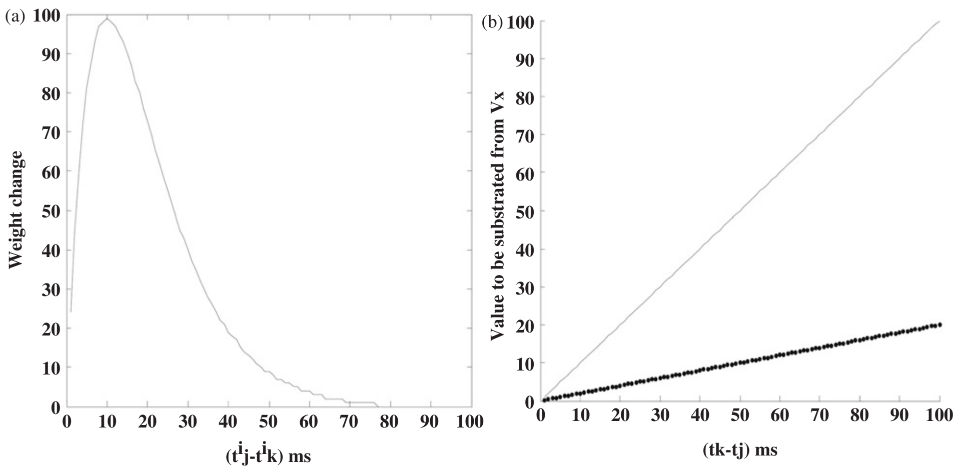

Figure 4. Neural model: (a) spike-time dependent plasticity and (b) membrane potential passive decay term.

Figure 4 illustrates the neural model behavior described in table 1. Figure 4(a) represents the spike time dependent plasticity (STDP) expressed in the first equation of table 1. S4 estimates the weight change through a pre-calculated look-up-table obtaining an average error of 0.47%. Figure 4(b) represents the passive decay term; it plots the time dependent term that needs to be subtracted to the membrane potential according to the second equation of table 1. We particularize different neural models with specific values of τ (a Purkinje cell model with a stronger passive decay term represented in the upper trace and a granular cell model with weaker passive decay term in the lower trace).

3. Simulation performance and hardware resources

The complete processing datapath consumes 74 clock cycles (provided that the event list has less than 2048 elements). But by using a pipeline processing strategy, we process one spike every 13 clock cycles (provided that the interspike risks do not affect significantly the system performance as has been estimated in § 2.3). Therefore, with a system clock frequency of 33 MHz, the achieved performance is approximately 2.5 million spikes per second. It is difficult to compare the performance with other approaches, since each of them uses different neural models. Currently, one of the most efficient event-driven software versions (Ros *et al.* 2006b) is able to compute up to 0.8 Mspikes/second using an AMD processor at 2.8 GHz. It is significant to note that, through the design of a specific purpose datapath working at a clock rate about 2 orders of magnitude lower than conventional computers; we are able to outperform in more than a factor of 2 the processing performance. Other simpler spiking neurons simulators are able to process (Delorme *et al.* 1999, Delorme and Thorpe 2003) but only including simplified neural models network topologies.

It is also remarkable that the exploration of other neural models (even of a higher complexity) would not significantly degrade the system performance, if the computation can be done in less than 13 independent steps or split in several pipelined processing stages.

The data throughput (D_t) follows expression (6), which is independent from the network size and includes a degradation term (A_{risks}) dependent on the inter-spike risks. This factor will not be significant in realistic networks in which spikes of output connection trees will be almost consecutively processed

$$D_t = \frac{f_{\text{cll}}}{A_{\text{risks}} + \max[13, N_{\text{clk\_cycles\_search}}]} \tag{6}$$

The performance rigorously follows the characterization expression outlined above. The surface in figure 5 has been done using a network topology (all-to-all connectivity with short synaptic delays). In this case, the inter-spike risks do not significantly affect the system performance. As can be seen, the performance does not depend on the network size, only on the global activity achieving a maximum performance of 2.5 millions spikes per second. The hardware resources consumption is summarized in table 2.

4. Discussion

The main innovation of the presented approach is the efficient use of the parallel computing resources of FPGA devices for an event-driven processing scheme. We have adopted a strategy that handles efficiently disordered event lists, which is a completely novel approach in the framework of event-driven spiking neural network simulation. We have used extensively parallel computing in the next-event-to-go searching structure that has been implemented with a finely pipelined searching tree.

The whole computing scheme is also implemented in a coarse pipelined datapath of 7 stages. Here, we need to handle inter-spike risks. However, as estimated in § 2.3, they are not significant in realistic networks in which spikes of specific output connection trees will be processed almost consecutively.

Although comparisons between different event driven approaches are difficult, since different authors adopt different neural models and computing strategies, the presented approach exhibits very promising performance results. It outperforms in more than a factor of two a similar approach implemented in software (Ros *et al.* 2006b). This is very important, taking also into account that the presented

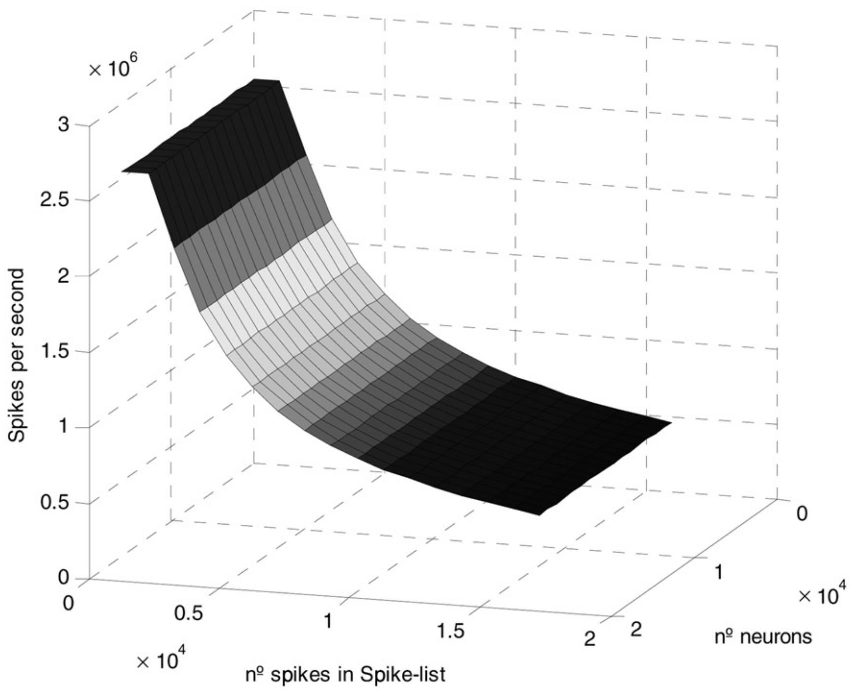

Figure 5. Performance vs. global network activity and network size.

Table 2. Hardware resources consumption. Design compiled on a Virtex II 6000 of Xilinx.

Pipeline state	#System gates	EMBs	#Clock cycles
S0	4 823 495	64	10–69
S1	2450	–	9
S2	919	–	9
S3	1172	–	12
S4	1955	–	9
S5	1182	–	13
S6	1451	–	9

computing scheme can embed more complex neural models without significantly degrading the system performance (provided that the computing stages are designed with less than 13 clock cycles). In this sense, we call this approach scalable in the neural model complexity.

Another important point is that, since the described computing platform is very general and can be easily adapted for different neural models, it becomes of interest in the framework of massive simulations and real-time experiments (for instance, in robotic experiments learning with sensory-motor integration schemes).

In § 2.2, we have discussed different memory strategies to efficiently implement the event-list. Two main aspects are considered, i.e., accessing time and scalability: we can conclude that storing the events on embedded memory blocks maximizes the parallel access. Therefore, this represents the most powerful choice for an event list of a moderate size (several thousands of events), keeping in mind that the restricted number of embedded memory blocks limits its scalability.

Acknowledgements

This work has been supported by the Spanish national grant DEPROVI (DPI-2004-07032) and the EU project SENSOPAC (IST-028056).

References

C. Boucheny, R. Carrillo, E. Ros and O.J.-M.D. Coenen, "Real-time spiking neural network: an adaptive cerebellar model", *LNCS*, 3512, pp. 136–144, 2005.

F.A. Briggs and E.S. Davidson, "Organization of semiconductor memories for parallel-pipelined processors", *IEEE Transaction Computer*, C-26, pp. 162–169, 1977.

A. Delorme and S. Thorpe, "SpikeNET: An event-driven simulation package for modelling large networks of spiking neurons", *Network: Computation in Neural Systems*, 14, pp. 613–627, 2003.

A. Delorme, J. Gautrais, R. van Rullen and S. Thorpe, "SpikeNET: a simulator for modelling large networks of integrate and fire neurons", *Neurocomputing*, 26–27, pp. 989–996, 1999.

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", *Proc. ICNN I*, 3512/2005, pp. 723–720, 1989.

B. Glackin, T.M. McGinnity, L.P. Maguire, Q.X. Wu and A. Belatreche, "A novel approach for the implementation of large scale spiking neural networks on FPGA hardware", *LNCS*, 3512/2005, pp. 552–563, 2005.

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

A. Jahnke, T. Schoenauer, U. Roth, K. Mohraz and H. Klar, "Simulation of spiking neural networks on different hardware platforms", *LNCS*, 1327, pp. 1187–1192, 1997.

T. Makino, "A discrete-event neural network simulator for general neuron models", *Neural Computer & Application*, 11, pp. 210–223, 2003.

M. Mattia and P. Del Guidice, "Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses", *Neural Computation*, 12, pp. 2305–2329, 2000.

N. Mehrtash, D. Jung, H.H. Hellmich, T. Schoenauer, V.T. Lu and H. Klar, "Synaptic plasticity in spiking neural networks (SP^{2}INN): a system approach", *IEEE Transaction Neural Networks*, 14, pp. 980–992, 2003.

J. Reutimann, M. Guigliano and S. Fusi, "Event-driven simulation of spiking neurons with stochastic dynamics", *Neural Computation*, 15, pp. 811–830, 2003.

E. Ros, E.M. Ortigosa, R. Agis, R. Carrillo, A. Prieto and M. Arnold, "Spiking neurons computing platform", *LNCS*, 3512, pp. 471–478, 2005.

E. Ros, E.M. Ortigosa, R. Agis, R. Carrillo and M. Arnold, "Real-time computing platform for spiking neurons (RT-Spike)", *IEEE transactions on Neural Networks*, 17, pp. 1050–1063, 2006a.

E. Ros, R. Carrillo, E.M. Ortigosa, B. Barbour and R. Agís, "Event-driven simulation scheme for spiking neural models based on characterization look-up tables", *Neural Computation*, 18, pp. 2959–2993, 2006b.

T. Schoenauer, S. Atasoy, N. Mehrtash and H. Klar, "NeuroPipe-Chip: a digital neuro-processor for spiking neural networks", *IEEE Trans. Neural Networks*, 13, pp. 205–213, 2002.