Building Blocks for Spikes Signals Processing

A. Jimenez-Fernandez, A. Linares-Barranco, R. Paz-Vicente, G. Jiménez, A. Civit

Abstract—**Neuromorphic engineers study models and implementations of systems that mimic neurons behavior in the brain. Neuro-inspired systems commonly use spikes to represent information. This representation has several advantages: its robustness to noise thanks to repetition, its continuous and analog information representation using digital pulses, its capacity of pre-processing during transmission time, …, Furthermore, spikes is an efficient way, found by nature, to codify, transmit and process information. In this paper we propose, design, and analyze neuro-inspired building blocks that can perform spike-based analog filters used in signal processing. We present a VHDL implementation for FPGA. Presented building blocks take advantages of the spike rate coded representation to perform a massively parallel processing without complex hardware units, like floating point arithmetic units, or a large memory. Those low requirements of hardware allow the integration of a high number of blocks inside a FPGA, allowing to process fully in parallel several spikes coded signals.**

I. INTRODUCTION

NATURE, thanks to species evolution, has found efficient solutions to solve environment adaptation problems for living beings. Bio-inspired systems try to understand real biological systems, extracting from them all possible advantages. As a subset of them, we can find the neuro-inspired systems, these systems face nowadays engineering problems, trying to solve them inspired in the way that the nervous system of living beings codifies and process information. Living beings brains allow them to interact dynamically with their environment, unlike robotics systems that need in best cases a controlled environment, being a great current challenge to improve robots adaptability and cognitive skills. The solution could be to replace progressively robots sensorial, control, and cognitive systems, by new neuromorphic systems, developing new processing architectures inspired in the way that biology performs these tasks. Neuromorphic systems provide a high level of parallelism, interconnectivity, and scalability; doing complex processing in real time, with a good relation between quality, speed and resource consumption. Neuromorphic engineers work in the study, design and development of neuro-inspired systems developed, like aVLSI chips for sensors [1][2], neuro-inspired processing, filtering or learning [3][4][5][6], neuro-inspired control pattern generators (CPG) [9], neuro-inspired robotics [8][17] and so on. Neuromorphic engineering community grows every year as it demonstrate the success of the Telluride and Capo Caccia Cognitive Neuromorphic workshops [9][10].

This work was supported by the Spanish grants SAMANTA II (TEC2006-11730-C03-02) and VULCANO (TEC2009-10639-C04-02), and by the Andalusia Council grants BrainSystems (P06-TIC-0 1417).

A.Jimenez- Fernandez, A. Linares-Barranco, R. Paz-Vicente, G. Jiménez, and A. Civit are with the Dept. of Computer Architecture and Technology, University of Seville, Seville, SPAIN (e-mail: mrivas@atc.us.es).

Spiking systems are neural models that mimic the neurons layers of the brain for processing purposes. Signals in spikes-domain are composed by short pulses in time, called spikes. Information is carried by spikes frequency or rate [11], following a Pulse Frequency Modulation (PFM) scheme, and also from other point of view, in the inter-spike-time (ISI) [5]. One important feature of spikes rate coded signals is that using this kind of modulation, spikes represents a continuous signal, not a stream of digital samples of a digital signal. In consequence, spikes rate coded information can be processed in a continuous way, without codifying information into discrete samples. Because of the simplicity of these models, spikes processing do not need a complex hardware to perform information processing, like multipliers. In consequence, this hardware can be replicated performing a massively parallel information processing [13][14].

One important consideration is how living beings have found an efficient way to transmit and represent information using spikes. Let's suppose the hypothesis that our eyes codify visual information using a 32-bit integer per cell representing the averaged value for a period of time, like conventional digital cameras. Then, a digital bus is needed to communicate each retina cell value. The number of buses needed in a brain would be impracticable, so a bus sharing will be necessary. However, when information is codified through spikes, one simple line is enough to communicate one cell activity. Then to communicate all the retina cells a set of lines will be required. This situation represents a problem for neuromorphic engineers when developing VLSI chips like a silicon retina since available pins of chips are limited. For example, a 128x128 retina will need 16.384 lines to communicate all the cells activity. The solution adopted by neuromorphic engineers consist in assign an identifier to each cell and use it to represent the cell that is producing the spike using a shared output digital bus. Furthermore, spike based representation of a signal allows continuous information flow in time, in contrast to discrete representation used by conventional digital vision systems, like camcorders. Thanks to these dedicated communication channels, system parallelism is increased because there is a virtual point to point connection between one sender and one receiver cell of different chips or systems that allow each cell to work independently. These cells (neurons) perform very simple computations, like on integrate and fire neuron model [12]. Although these operations are very simple, our brain has a huge number of neurons, with thousand of connections between them, allowing implementing a really complex processing of information, fully parallel and in real-time.

In the other hand, traditional computer and digital signal processing systems (DSP) are different to spike-based systems. DSP codify the information as digital words with different characteristics (lengths, representations types, accuracy,...) like integer, fixed point or floating point numbers. DSP also need complex arithmetic units to process information, and sometimes the complexity of these units doesn't allow to include several in a DSP. Inside in a DSP a reduced number of buses are shared and multiplexed in time between elements. Memory has the same problem, because it is a shared resource that requires overhead instructions for a correct managing. Then a DSP usually have to reuse in time arithmetic components, multiplex buses and memory access, and execute overhead instructions, these facts limits the parallelism level and decreases the efficiency of DSP useful number of operations performed. In the context of DSP most advanced devices, parallelism doesn't go beyond the instruction level parallelism (ILP) and single instruction – multiple data (SIMD) capabilities (10 operations in parallel)

Present work takes advantage from spike based representation for new implementations of basic signal processing operations, like low-pass and high-pass filters, Our aim is to adapt existing discrete computation models and algorithms available in DSP to hardware components that processes spike coded information. We have designed Spikes Signal Processing systems (SSP). SSP is massively parallel, appropriate for real-time processing, and allows to synthesize in an FPGA a huge number of processing elements thanks to models simplicity.

SSP will improve present neuromorphic systems with new functionalities (pre or post processing filters). We propose and analyze new architectures for SSP with equivalent behavior to analog frequency filters. SSP components have been designed as building blocks, allowing connecting them together in several ways to perform complex spikes processing. SSP building blocks presented in this paper have been written in VHDL, and simulated inside MATLAB with the addition of Xilinx System Generator.

In Section II we present a set of basic building blocks designed, as SSP primitive functions. In section III we combine basic blocks to design complex blocks that can perform spike based filters equivalent to analog filters. In section IV we present simulation results and discuss them. Finally, in section V we present the building blocks hardware implementation, and a performance study, compared to traditional DSP.

II. BASIC BUILDING BLOCKS FOR SSP

Like any complex digital or analog systems is divided or decomposed into simple components, we propose to develop a SSP using four basic blocks:

- A synthetic spikes generator for converting an analog signal into spikes, like an input or a reference.
- A spikes integrator and generator that works like an analog integrator or capacitor.
- A spike hold and fire, for adding or subtracting two spikes coded signals.
- A spike frequency divider, which reduces the frequency of incoming spikes stream by a constant value.

A. *Synthetic Spikes Generator (RB-SSG)*

A Synthetic Spikes Generator (SSG) will transform a digital word (SSG input) into a frequency rate of spikes (SSG output). This element is necessary in those scenarios where some reference signals (sensors or feedback signals) are not coded using the spike representation. There are several ways to implement a SSG as presented in [15]. A SSG should generate a synthetic spikes stream, whose frequency should be proportional to a constant ($k_{SpikesGen}$) and an input value (x), according to next equation:

$$SSG(x)_{SpikesRate} = k_{SpikesGen} * x \qquad (1)$$

Selected SSG implements the reverse bitwise method found detailed in [16] for synthetic AER events generation. Fig. 1 shows the internal components of the reverse bitwise SSG (RB-SSG) implemented. RB-SSG uses a continuous digital counter (figure top), whose output is reversed bitwise and compared with the input absolute value (*ABS(x)*). In the case that input absolute value is greater than reversed counter value, a new spike is fired (figure bottom). RB-SSG ensures a homogeneous spikes distribution along time, thanks to reversing bitwise counter output. Since a reference or an analog signal can be negative, is necessary to generate positive and negatives spikes. Therefore, we use a demultiplexor to select the right output spikes, where selection signal is the input sign, or *X(MSB)*. Finally, a clock frequency divider (figure left top) is included to adjust RB-SSG gain. This element will activate a clock enable (*CE*) signal, for dividing spikes generator clock frequency, according with a frequency divider signal (*genFD*). In consequence RB-SSG gain ($k_{BWSpikesGen}$) can be calculated as follows:

$$K_{BWSpikesGen} = \frac{F_{CLK}}{2^{N-1}(genFD+1)} \qquad (2)$$

Where *Fclk* represents system clock frequency, *N* the RB-SSG bits length, and *genFD* clock frequency divider value. These parameters can be modified in order to set up RB-SSG gain according with design requirements.

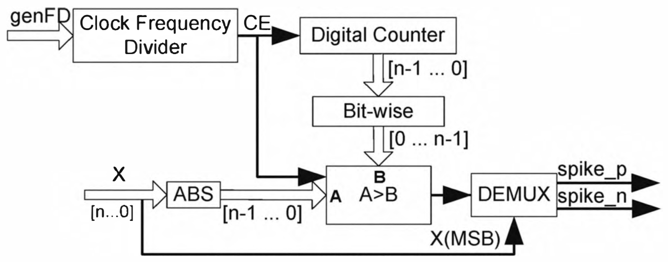

Fig. 1. Reverse Bitwise Synthetic Spikes Generator block diagram.

B. *Spikes Integrator & Generator (SI&G)*

Spikes Integrator & Generator (SI&G) is composed by a spikes counter for the integrator part, and by a RB-SSG, as showed in Fig. 2. Spikes counter is a digital counter that is increased by one when a positive spike is received, and its value is decreased by one with a negative spike. Counter output is the RB-SSG input. Therefore, new spikes generated have a frequency proportional to spikes count or spikes integration. The SI&G gain is set by RB-SSG parameters.

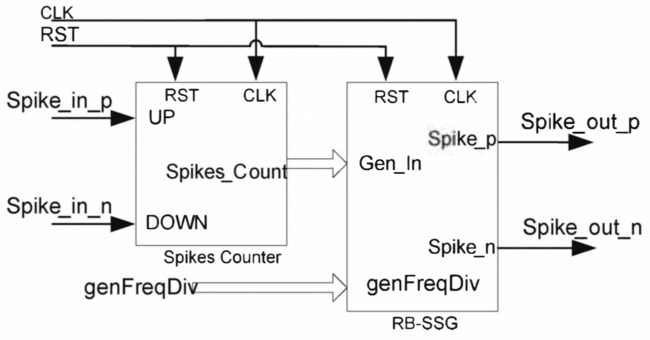

Fig. 2. Spikes Integrate & Generate block diagram.

With these considerations, SI&G spikes output frequency, $f_{I\&G}$, and SI&G gain, $k_{I\&G}$, can be expressed as:

$$f_{I\&G} = k_{I\&G} * \int f_{inputSpikes}\, dt = \frac{F_{CLK}}{2^{N-1}(genFD+1)} \int f_{inputSpikes}\, dt \tag{3}$$

Similarly to analog systems, we can calculate equivalent SI&G transfer function in "spikes-domain" using Laplace transform. SI&G transfer function is presented in (4), being equivalent to an ideal integrator with a gain of $k_{I\&G}$.

$$SI\&G(s) = \frac{F_{SI\&G}(s)}{F_{inputSpikes}(s)} = \frac{k_{I\&G}}{s} = \frac{F_{CLK}}{2^{N-1}(genFD+1)*s} \tag{4}$$

For SI&G testing purpose we have executed a set of simulations for different pairs of parameters (*N*: number of bits, *genFD*: generator freq divider): 13-0, 14-1, 16-0, and a constant rate of input spikes. Fig. 3 shows SI&G simulation results, at figure top is represented the frequency of input spikes in blue (a step signal), and three SI&G frequency output spikes couples for different parameters. These couples are composed by simulated reconstructed output (solid line) and theoretical response (discontinuous line). At figure bottom we can see SI&G inputs spikes in blue, and also SI&G output spikes in green. Input spikes represent a step signal, having a constant spike rate, SI&G output spikes have a constant linear frequency increasing when input spikes are positives, like a ramp, with a slope equivalent to $k_{I\&G}$, and also decreasing output spikes frequency with negative spikes, as expected from an ideal analog integrator with same features. Simulations also denoted a good accuracy with theoretical responses.

C. Spikes Hold & Fire (SH&F)

This block performs the subtraction between two spikes stream. SH&F will allow us to implement feedback in the SI&G block, obtaining new transfer functions and desingning more complex systems for SSP.

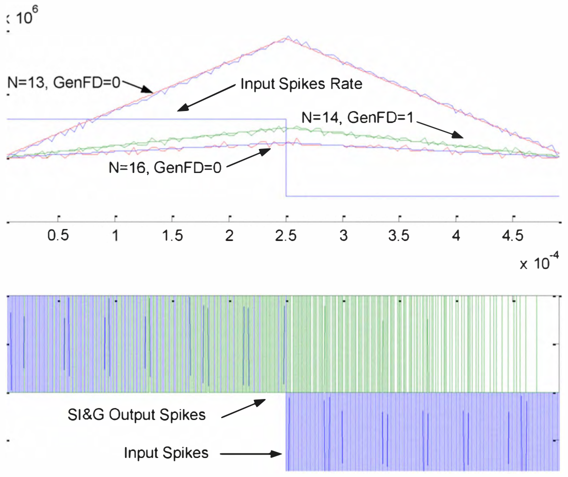

Fig. 3. SI&G simulation output spikes rate reconstructed, top, and input/output spikes, bottom.

Subtracts a spikes input signal (f_u) to another (f_Y), means to get a new spikes signal which spike rate ($f_{SH\&F}$) will be the difference between both inputs spikes rate:

$$f_{SH\&F} = f_U - f_Y \tag{5}$$

The procedure of the SH&F is to hold the incoming spikes a fixed period of time while monitoring the input evolution to decide output spikes. Fig. 4 shows an example of how SH&F can evolve from a positive spike. This block has two inputs U (positive input) and Y (negative input commonly used as feedback). Let's suppose that a positive U spike (U+) is received, figure left. U+ is held internally (state U+), figure center, doing nothing in the case that no spikes are received. When a new spike arrives, it behaves in one or other way according to spike input port and sign, figure right. On top, if SH&F receives a positive spike (U+), held spike is fired as a positive spike, and new one is held internally (U+). If a negative input spike is received in the port U (U-), or a positive spike in the port Y (Y+), held spike is cancelled and no output spike is produced. Finally, if a negative Y spike (Y-) is received (figure bottom), hold spike is sent and last one received is held with positive sign (U+). Similar SH&F behavior can be extended to any kind of input spikes (U-, Y+ and Y-) using the same logic: hold, cancel, and fire spikes according to input spikes ports and sign.

This block has been successfully used previously to design spike-based closed-loop control systems in mobile robots by authors [17][18]

A. Spikes Frequency Divider (SFD)

This block will divide the spike rate of an input signal by a constant. We can think in many ways to implement this basic block, like for example using simple counters, firing one spike when several spikes have been received, or with probabilistic techniques by deciding to propagate, or not, an incoming spike using a random number generator. These techniques present a problem, output spikes rate is correct in average, but these spikes are not distributed homogenously in time. To ensure spikes distribution we have implemented this block inspired in the way that RB-SSG works. Fig. 5 shows SFD internal components, it's very similar to RB-SSG with three differences: first of all, input spikes increases the digital counter, not every clock cycle like RB-SSG. Next difference turns around the *spikesDiv* signal, in the RB-SSG this signal works like generator input, being its output spikes rate proportional to this value, in SFD *spikesDiv* behaves like the constant to divide, being its value compared with the digital counter output reversed bit wise. Digital comparator output drives a buffer that allows, or not, input spikes to pass through SFD. In general, input spikes will increase the digital counter, which reverse value will be compared with *spikesDiv* signal and in the case that reversed counter output is lower than *spikesDiv*, output buffers will be enabled, allowing next spike to travel across this component. So, output buffers only enable output spikes according *spikesDiv* homogenously in time. SFD transfer function can be calculated using (6), where *N* represents the SFD number of bits, and *spikesDiv* the signal presented before.

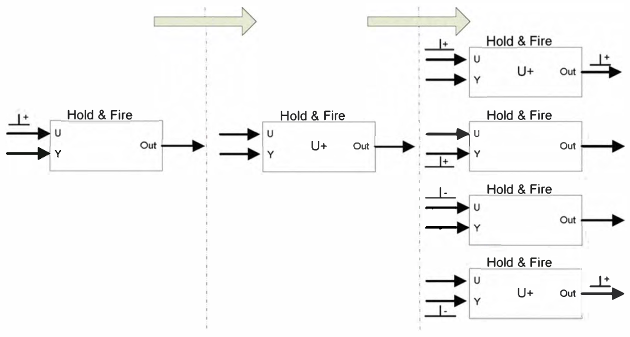

Fig. 4. Spike Hold&Fire evolution form a positive U spike.

$$F_{SFD} = \frac{F_{outSpikes}}{F_{inputSpikes}} = \frac{spikesDiv}{2^N} \tag{6}$$

SFD transfer function is equivalent to a gain block with a value in the range of [0,1], with 2^N possible steps. SFD accuracy can also be adjusted with N (N=16 bits in Fig.5) getting an accuracy of 2^{-N}.

III. Feeding back the Spikes Integrate & Generate

Using these basic building blocks as primitive operations, is possible to combine them for designing new building blocks for more complex SSP, like for example spikes frequency filters.

If we generate a spikes stream from an analog signal, spikes frequency will be proportional to analog signal amplitude, as exposed before in (1). In consequence, analog signal amplitude changes will be represented by spikes frequency changes. As analog frequency filters modifies signal frequency components from amplitude changes, spikes filters will work on spikes rate changes frequency components. For example, a spikes low pass filter will attenuate spikes rate high frequency components, so if constant spikes rate signal is applied to spikes filter input, as an analog input step, spikes output frequency will start to increase exponentially, as expected from an analog filter output signal amplitude, until spikes rate reach a steady state frequency. Next section presents in detail two spikes filters, the first one will be equivalent to a low pass filter, and the next one to a high pass filter, but in the context of spikes coded signals.

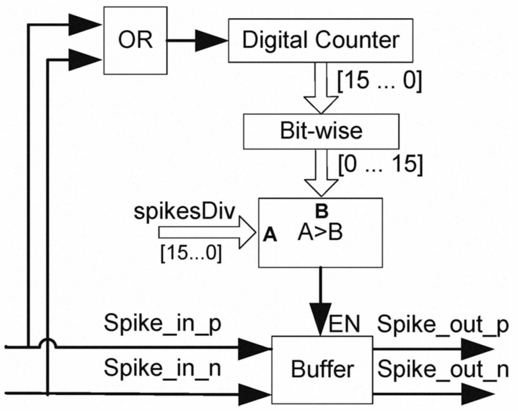

Fig. 5. Spikes Frequency Divider internal components.

A. Spikes Low Pass Filters

The Spikes Low Pass Filter (SLPF) block will filter high frequency changes on input spike rate. To build this new block we have feedback a SI&G using a SH&F and two SFD, as it is shown in Fig. 6. The idea is to ingrate input spikes with a SI&G, subtracting SI&G output spikes with input spikes using a SH&F, performing in this way a basic filter, without a great accuracy and a fixed gain to 1. To avoid this problem and improve SLPF features, two SFD have been included. The most important one is the SFD placed in the feed-back loop, its work is to divide feedback spikes frequency, subtracting less input spikes in the SH&G, providing a higher number of spikes at the SI&G input. This fact affects to filter gain, being higher than one, because SI&G integrates more spikes, and cut-off filter frequency can be selected with an improved accuracy, as will be discussed later. The inclusion of SFD in the feedback increases filter gain, so another SFD is placed before SLPF output, allowing to decrease SI&G output, being SLPF gain fully adjustable.

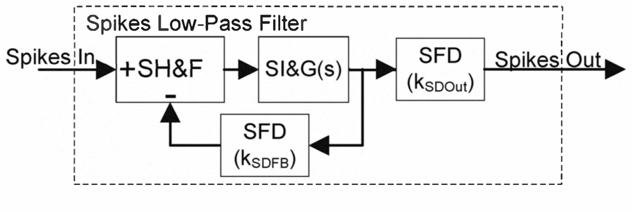

Fig. 6. Spikes Low Pass Filter Architecture

According to SI&G feedback topology and taking into account both SFD gain, we can calculate SLPF equivalent transfer function using basic systems theory. Next equation shows SLPF ideal transfer function, where SI&G(s) can be obtained from (4), k_{SDout} represents output SFD gain and

k_{SDFB} the gain of the SFD placed in the feedback loop, both detailed in (6).

$$F_{SLPF}(s) = \frac{F_{outSpikes}\ (s)}{F_{inputSpikes}\ (s)} = \frac{k_{SFDOut}\ * SI\&G(s)}{1 + k_{SFDFB}\ * SI\&G(s)} = \frac{k_{SFDOut}\ * k_{I\&G}}{s + k_{SFDFB}\ * k_{I\&G}} \tag{7}$$

SLPF transfer function is equivalent to a first order low pass filter with a pole. Theoretical filter cut-off frequency, $\omega_{cut\text{-}off}$ in rad/sec, can be determined by the product between $k_{I\&G}$, and k_{SDFB}. Thanks to the fact that k_{SDFB} can be set with a 2^N bits value, the accuracy of $\omega_{cut\text{-}off}$ can be very sharp.

$$\omega_{cut-off} = k_{SFDFB} * k_{I\&G} \tag{8}$$

Equivalent SLFP gain can be set with the relation between the values of k_{SDFB} and k_{SDOut}.

$$k_{SLPF} = \lim_{s\to 0} F_{SLPF}(s) = \frac{k_{SFDOut}\ * k_{I\&G}}{k_{SFDFB}\ * k_{I\&G}} = \frac{k_{SFDOut}}{k_{SFDFB}} \tag{9}$$

B. *Spikes High Pass Filters*

Using an SLPF and SH&F we have built a spike-based high pass filter (SHPF). This block will filter low frequency components in a spike-based signal, allowing passing only high frequency components. The idea is to perform a low pass filtering of input spikes, using a SLPF, and subtract SLPF output with original spikes signal using a SH&F, as Fig. 7 shows.

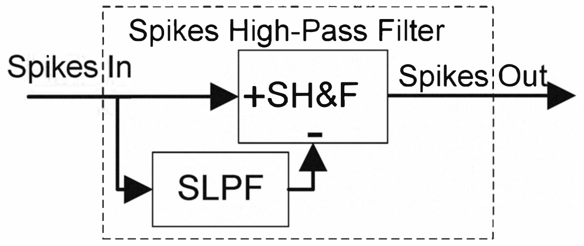

Fig. 7. Spikes High-Pass filter Architecture

Using SLPF transfer function, in (7), we can calculate SHPF equivalent transfer function as follows:

$$F_{SHPF}(s) = 1 - F_{SLPF}(s) = 1 - \frac{k_{SFDOut}\ * k_{I\&G}}{s + k_{SFDFB}\ * k_{I\&G}} = \frac{s + (k_{SFDFB}\ * k_{I\&G} - k_{SFDOut}\ * k_{I\&G})}{s + k_{SFDFB}\ * k_{I\&G}} \tag{10}$$

Resulting SHPF transfer function contains a pole and a zero. Because we are looking for high-pass filter, the zero should be placed in the origin. To meet this constrain, according to (10), k_{SFDFB} must be equal to k_{SFDOut}, or in other words, SLPF gain has to be 1, as exposed in (9). Taking into account this constrain, SHPF transfer function is reduced to:

$$F_{SHPF}(s) = \frac{s}{s + k_{SFDFB}\ * k_{I\&G}} \tag{11}$$

And SHPF cut-off frequency can be calculated as the product of k_{SFDFB} and $k_{I\&G}$:

$$\omega_{cut-off} = k_{SFDFB}\ * k_{I\&G} \tag{12}$$

However, gain of band pass filter is fixed to one. DC signal component is rejected, as transfer function limits evidence.

$$\begin{aligned} k_{SHPF}(s \to 0) &= \lim_{s\to 0} \frac{s}{s + k_{SFDFB}\ * k_{I\&G}} = 0 \\ k_{SHPF}(s \to \infty) &= \lim_{s\to \infty} \frac{s}{s + k_{SFDFB}\ * k_{I\&G}} = 1 \end{aligned} \tag{13}$$

IV. Simulation Scenario and Results

This section presents experimental results from MATLAB simulations of previously exposed SLPF and SHFP. For this purpose we have designed a simulation scenario using Simulink with the addition of Xilinx System Generator. Thanks to these tools we can simulate, among other things, Simulink elements together with VHDL entities, providing a powerful interface to stimulate and analyze the behavior of written VHDL files. Fig. 8 shows the scenario used for simulations, it contents two presented building blocks, at left a RB-SSG, and at right, the spikes filter that we want to test (SLPF and SHPF, with all their possible parameters as constants), actually any other building block can be placed in test filter slot. A Simulink signal generator provides a stimulus signal for the RG-SSG input, generating at RB-SGG output the stimulus spikes for testing the spikes filter, being spikes frequency proportional to input signal amplitude, as presented before in (1). Synthetic generated spikes will be the spike test filter input. Spikes filter under test will process incoming spikes, providing a new spikes stream that is the result of the simulation. Adding at the end, figure right, two more Simulink blocks, one for sending output spikes directly to MATLAB work space, and a scope to watch spikes.

Our aim is to present diverse filters responses, so different features spikes filters have been simulated, as for SLPF and as for SHPF. Fixing different parameters sets, getting equivalent filters with various cut-off frequencies and gain (in the case of SLPF). Simulation parameters are presented in Table I, it contains from left to right: number of filter under test, and the kind of filter used; next four rows show filter parameters, bits number, frequency divider, and both SFD values; finally last three row provides equivalent ideal filter parameters, cut-off frequency, gain in absolute value, and rise time. Simulations results are presented in Fig 9, this figure is composed by two different kinds of simulations, in a) there are spikes filters temporal response, and in b) the frequency

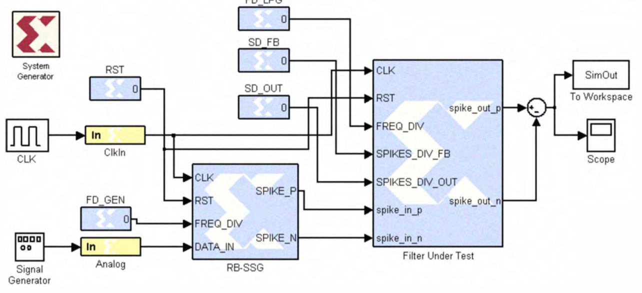

Fig. 8. Simulation scenario designed for spikes filters testing

TABLE I
SPIKES FILTERS SIMULATION PARAMETERS

Test Case	Spikes Filter	Bits Number	Frequency Div.	Spikes Div. Out	Spikes Div. Feedback	$\omega_{cut\text{-}off}$	Filter Gain (abs)	Rise Time
1	SLPF	13	0	0.5147	0.5147	1kHz	1	0.6366mSec
2	SLPF	11	0	0.634	0.634	5kHz	1	0.1273mSec
3	SLPF	9	0	0.634	0.634	20kHz	1	0.0318mSec
4	SLPF	13	0	0.6691	0.5147	1kHz	0.5	0.6366mSec
5	SLPF	13	0	0.6691	0.5147	1kHz	1.3	0.6366mSec
6	HLPF	13	0	0.5147	0.5147	1kHz	1	0.6366mSec
7	HLPF	11	0	0.634	0.634	5kHz	1	0.1273mSec
8	HLPF	9	0	0.634	0.634	20kHz	1	0.0318mSec

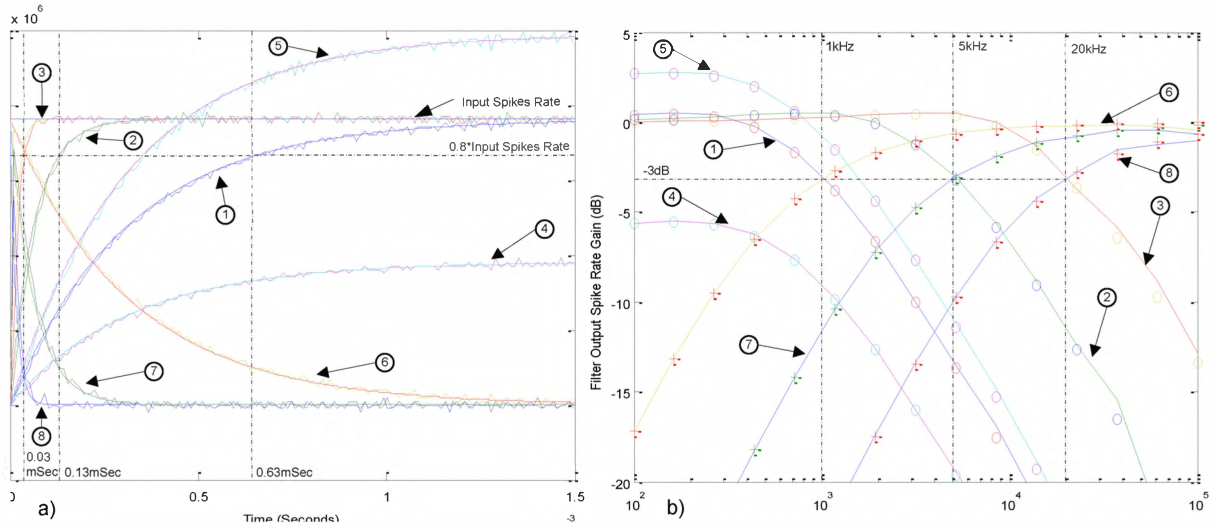

Fig. 9. Simulation Results: a) Spikes Filters time response b) Spikes Filters Bode diagram

response. As first simulations we present a set of temporal simulations, Fig. 9 a), simulating all spikes filters of Table I for a fixed time (1.5mSec). This figure shows spikes filters temporal output, being x axis simulation time, and y axis instantaneously spikes rate. For temporal simulation these spikes filters have been excited by a constant spike rate input (7.6MSpikes/Sec), as an analog voltage step, in blue. Filter output spikes have been analyzed and their instantaneous frequency along time (in diverse colors) have been reconstructed. Theoretical responses of analog ideal equivalent filters have been also added to the figure, to compare simulation response respect to ideal response. In the case of SLPF (1-5), output spikes frequency start to increase exponentially, like expected from an ideal low pass-filter, with a rise time near to 4 times the inverse of $\omega_{cut\text{-}off}$. SLPF output spikes rate reach a steady state value proportional to SLPF gain, reaching input spikes frequency when is 1 (1-3), higher output spikes rates in the case that SLPF gains is higher than 1 (5), and opposed effect happens when SLPF gains is lower than 1 (4). When SHPF is simulated, its output spike rate behaves like an impulse (6-8), with high spike rate at start, and decreasing output spikes frequency exponentially until no spike is fired, denoting that DC component is completely rejected. This response represents input temporal changes, with the addition of some dynamics, adjustable with SHPF cut-off frequency; however SHPF filter gain cannot be adjusted and it is fixed to 1. No timing issues are detected with SHPF as in digital high-pass filter using same topology, because spikes represent continuous signals, and they are subtracted on the fly, while spikes are arriving.

Both SLPF (1-5) and SHPF (6-8) have a similar behavior from expected theoretical temporal responses, confirming that filters work equivalently to previously calculated Laplace transfer functions.

After simulating the temporal behavior of designed spikes filters, we have studied their frequency responses. If our aim were to characterize an analog filter in frequency domain, one simple way to perform this task could be to excite the analog filter with pure frequency tones (fixed frequency sinusoidal voltage signal), and annotate analog filter output power for each voltage tone. Translating this experiment to spikes domain, we are going to stimulate RB-SSG with an input sinusoidal signal, getting a spikes output which frequency changes according the sinusoidal signal, codifying a pure tone as spikes rate changing. Then generated spikes will stimulate the spikes filters, whose output will be a spikes stream with same input frequency tone, but with its amplitude and phase modified by the spikes filters. So we have exited every spikes filter with a set of spikes coded tones, making a frequency sweep, and recording all the spikes filter output power for every that tone, obtaining finally the spikes filter Bode diagram. Results of these simulations are presented in Fig. 9 b), x axis represents the frequency of input tones in Hz, and y axis contains the spikes filter gain in dB from 100Hz to 100kHz. Figure contains SLPF simulated and theoretical responses marked with a cross, and also SHPF responses, where theoretical responses are marked in these cases with circle. SLPF with a gain of 1 (1-3) have predicted gain of 0dB in the pass band, and then gain starts to decrease when frequency is near to cut-off frequencies, cutting them with a gain of 3dB, providing an attenuation of 20dB by decade as expected of analog filters. SLPF with different gains (4-5), having a gain in the band pass of -6dB and +2.6dB respectively, and attenuating frequency components from the cut-off frequency as 0dB SLPF. SHPF (6-8) are opposed to 0dB SLPF, attenuating low frequency components, with a gain of -3dB on them cut-off frequencies, and a 0dB gain in their pass band.

Both simulations, temporal and frequencial, have evidenced that spikes filters work as we predicted theoretically in previous sections. All spikes filters show a great accuracy with ideal values and a closer behavior to analog filters, although spikes filters are pure digital, working with binary signals values.

V. SSP BUILDING BLOCKS SYNTHESIS AND PERFORMANCE

Spikes filters have been synthesized in a commercial FPGA, to determine hardware requirements and analyze spikes filters performance. Each spikes filter has been synthesized for a Spartan 3 FPGA family manufactured by Xilinx, the XC3S400. Table II contains synthesis results, showing the spikes filter synthesized; spikes filter number of bits, number of slices consumed, maximum operation frequency and number of equivalents operations performed. Spikes filters have been synthesized for different number of bits, because this is a parameter that is determinated in synthesis time and it is needed to allocate inside the FPGA several counters and comparators of different length when final circuit is being synthesized. The SLPFs need between 115 and 139 slices depending of the number of bits selected, with different working frequencies, from 121MHz to 111MHz. As higher is the number of bits, higher is the hardware consumption and lower is the working frequency. Results of SHPFs are very similar, however SHPFs include an additional SH&F, what will slightly increase the number of slices, and in consequence, it will decrease working frequency.

Finally we have analyzed spikes filters performance, although, in general, analyzing spikes processing systems performance uses to be difficult and debatable, because this kind of system are very different to systems based on sequential traditional processors. We have analyzed filters performance inspired in the way that AER-based convolutions have been measured in [19] and [20], analyzing spikes performance according to the operations that AER convolutions perform compared with the equivalent digital processing systems. Our idea is to determinate how many operations are needed to implement digital filters with the same features than spikes filters, estimating how many equivalent operations can be performed by a spikes filter.

Finite Impulse Response (FIR) filters uses to be a common way to filter discrete signals in a DSP, we are going to use this kind of filter as reference because are simpler than Infinite Impulse Response (IIR) filters. The idea of a FIR filter is to multiply a history of input signal samples by a fixed set of coefficients, adding the results of all of these operations to get the output sample. In (14) we show the general equation of a FIR filter of order 1. This filter needs last two samples of the input samples (x), and two fixed coefficient, b_{0-1}, that contains filter behavior. Output sample (y) is the result of the operation of multiplicate-and-accumulate (MAC) between these values.

$$y_{LPF}(n) = b_0 * x(n) + b_1 * x(n-1) \qquad (14)$$

To perform this operation, the DSP should execute many instructions, not only arithmetic operations. DSPs need to access to memory in a sequential way to obtain input samples and coefficients, for a later execution of arithmetic operations. Then DSP need to shift input samples to be ready for next sample. Overhead instructions decrease dramatically the DSP performance in effective operations per seconds. FIR filter needs 1 addition and 2 multiplications, if we assign a classical set of weights to both operations, typically 1 and 4 respectively, we can calculate the number of ideal operations that a FIR filter needs is 9. This number of operations is ideal, because it doesn't take care of memory access or arithmetic operations related again to memory access. However we are going to use this number of operations as reference, because spikes filters don't access to memory at all, and also can execute this operation in a massive parallel way. Table II last row shows the number of operations that can be performed by the different spikes filters related to the same digital operation, obtained from the product between spikes filter maximum operating frequency and equivalent number of operations, 9. Spikes filters can reach a number of operations equivalents higher than 1.1Gops / Sec, and no lower than 0.99Gops / Sec, denoting a good performance. In addition, spikes filters do not need any complex hardware like floating point ALUs, so spikes filter can be replicated many times, performing high number of operations. Compared with commercial DSP like [20], that reaches ideally 2.5Gops, if we allocate in a medium FPGA, like is selected Spartan 3, 3 or more spikes filters, FPGA will be performing more operations than this DSP (in ideal conditions) using only a 1% of the FPGA.

TABLE II
SSP BUILDING BLOCKS SYNTHESIS RESULTS AND PERFORMANCE

Building Block	Number of bits	Occupied slices. Max 3584	Max. operating frequency (MHz)	Eq. operations by second (Gops)
SLPF	8	115	121.4	1.0927
SLPF	12	130	119.7	1.0773
SLPF	16	139	111.6	1.0044
HLPF	8	132	120.8	1.1196
HLPF	12	144	119.3	1.0737
HLPF	16	153	110.6	0.9954

VI. SSP BUILDING BLOCKS PRACTICAL APPLICATIONS

Although SSP building blocks are in a early state, some filed of applications can be found, however new SSP building blocks should be proposed and designed to perform more complex operations. Currently authors are starting to work on two possible applications of SSP building blocks. First field of application is in spikes-base robotics controls. Nowadays these kind of controls implements proportional (P) closed-loop controls based only in the use of spikes [17] and have been used to control mobile robots [18]. Thanks to SI&G and SHPF spikes-based controls can be improved, designed Proportional-Integral-Derivative (PID) controls using only spikes, providing better and new spikes-based controls. Far beyond, SHPF transfer function is equivalent to a lead network when SLPF inside has a gain different to 1, that is a very common control structure in industry.

Other possible application is the design of a neuro-inspired synthetic cochlea. An example of an analog cochlea can be found in [2], biological cochlea decompose sounds in its frequency components, this fact is mimicked by this analog cochlea as a set of analog band-pass filters, whose analog output value is codified to spikes by a determinate circuit (the inner-hair-cell circuit, IHC). Analog cochlea use to show some practical problems related to AVLSI circuits, like the introduction of noise due to use of analog filters, and problems with circuit mismatch, that affects to analog filters cut-off frequencies. In the sense of this cochlea, it processes analog sound information to convert it later to spikes. Using spikes filters, why don't convert incoming sound in spikes and then process spikes directly? So a very useful application could design spikes band pass filters (SBPF) as similar as analog band pass filters is possible, to try to avoid the problems that's this kind AVLSI systems presents.

VII. CONCLUSIONS

This paper presents building blocks inspired in the way that biology codifies, transmit, and process the information, the spike rate coded representation. Although there is no evidence that nature performs same operations than presented blocks, we continue working in this way because it is interesting in the sense that SSP building blocks adapts already known digital operations to spikes based operations, but with the advantages of representing information with spikes. These advantages are well known, like massively parallel operations, the absence of a sample period, or the simplicity of the hardware needed to process this information. SSP building blocks performs operations equivalent to basic analog systems, but using pure digital circuits. Those new spike-based components evidence that nature have reach efficient solutions, processing signals without the need of complex information representation (e.g. floating point numbers), avoiding complicated arithmetic hardware (e.g. multipliers). Being the most complex operation performed by our blocks the addition of one to a counter. We have simulated and synthesized these new elements, showing a good behavior, low hardware consumption and higher performance from the computational point of view.

ACKNOWLEDGMENT

This work has been supported in part by the Andalucía Council with the BrainSystem project (P06-TIC-01417), and by the Spanish projects: SAMANTA II (TEC2006-11730-C03-02) and VULCANO (TEC2009-10639-C04-02)

REFERENCES

[1] P. Lichtsteiner, et al. "A 128×128 120dB 15 us Asynchronous Temporal Contrast Vision Sensor". IEEE Journal on Solid-State Circuits, vol. 43, No 2, Feb-2008.

[2] Chan, V. et al,, "AER EAR: A Matched Silicon Cochlea Pair With Address Event Representation Interface". IEEE Transactions on Circuits and Systems I: Regular Papers, Volume 54, Issue 1, Jan. 2007

[3] R. Serrano-Gotarredona, et al.. "On Real-Time AER 2-D Convolutions Hardware for Neuromorphic Spike-Based Cortical Processing. IEEE Transactions on Neural Networks, Vol. 19, No 7. July-2008.

[4] Oster, M et al "Quantifying Input and Output Spike Statistics of a Winner-Take-All Network in a Vision System" IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007.

[5] P. Hafliger. "Adaptive WTA with an Analog VLSI Neuromorphic Learning Chip". IEEE Transactions on Neural Networks, vol. 18, No 2,. March-2007.

[6] G. Indiveri, et al. "A VLSI Array of Low-Power Spiking Neurons and Bistables Synapses with Spike-Timig Dependant Plasticity". IEEE Transactions on Neural Networks, vol. 17, No 1. Jan-2006.

[7] F. Gomez-Rodríguez, et al "AER Auditory Filtering and CPG for Robot Control". . IEEE International Symposium on Circuits and Systems ,ISCAS 2007.

[8] A. Linares-Barranco, et al. "Using FPGA for visuo-motor control with a silicon retina and a humanoid robot". . IEEE International Symposium on Circuits and Systems, ISCAS 2007.

[9] Telluride Cognitive Neuromorphic workshop: https://neuromorphs.net/

[10] Capo Caccia Cognitive Neuromorphic workshop: http://capocaccia.ethz.ch

[11] G. Shepherd, "The Synaptic Organization of the Brain". Oxford University Press, 3rd Edition, 1990.

[12] E. Chicca, et al. "An event based VLSI network of integrate-and-fire neurons ". IEEE International Symposium on Circuits and Systems, ISCAS 2004 Misha Mahowald. VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function. PhD. Thesis, California Institute of Technology Pasadena, California, 1992.

[13] R. Serrano-Gotarredona, et al., "CAVIAR: A 45k-neuron, 5M-synapse AER Hardware Sensory-Processing-Learning-Actuating System for High-Speed Visual Object Recognition and Tracking," IEEE Trans. on Neural Networks, Volume 20, Issue 9, Sept. 2009 Pags.: 1417 - 1438

[14] F. Gomez-Rodriguez, et al. "Two Hardware Implementation of the Exhaustive Synthetic Aer Generation Method". LNCS. Vol. 3512. 2005. Pag. 534-540

[15] R. Paz-Vicente, et al."Synthetic retina for AER systems development". International Conference on Computer Systems and Applications, AICCSA 2009.

[16] A. Jiménez-Fernández, et al. "AER-based robotic closed-loop control system". IEEE International Symposium on Circuits and Systems, ISCAS 2008.

[17] A. Jiménez-Fernández, et al. "AER and dynamic systems co-simulation over Simulink with Xilinx System Generator". IEEE International Conference on Electronics, Circuits and Systems, ICECS 2008.

[18] A. Linares-Barranco, et al. "FPGA Implementations comparison of Neuro-Cortical Inspired Convolution Processors for Spiking Systems". International Work-Conference on Artificial Neural Networks, IWANN 2009.

[19] A. Linares-Barranco, et al. "On the AER Convolution Processors for FPGA". IEEE International Symposium on Circuits and Systems, ISCAS 2010.Texas Intruments TMS320C6720 datasheet: http://focus.ti.com/lit/ds/symlink/tms320c6727b.pdf