A Binaural Neuromorphic Auditory Sensor for FPGA: A Spike Signal Processing Approach

Angel Jiménez-Fernández, Elena Cerezuela-Escudero, Lourdes Miró-Amarante,
Manuel Jesus Domínguez-Morales, Francisco de Asís Gómez-Rodríguez,
Alejandro Linares-Barranco, and Gabriel Jiménez-Moreno

Abstract—**This paper presents a new architecture, design flow, and field-programmable gate array (FPGA) implementation analysis of a neuromorphic binaural auditory sensor, designed completely in the spike domain. Unlike digital cochleae that decompose audio signals using classical digital signal processing techniques, the model presented in this paper processes information directly encoded as spikes using pulse frequency modulation and provides a set of frequency-decomposed audio information using an address-event representation interface. In this case, a systematic approach to design led to a generic process for building, tuning, and implementing audio frequency decomposers with different features, facilitating synthesis with custom features. This allows researchers to implement their own parameterized neuromorphic auditory systems in a low-cost FPGA in order to study the audio processing and learning activity that takes place in the brain. In this paper, we present a 64-channel binaural neuromorphic auditory system implemented in a Virtex-5 FPGA using a commercial development board. The system was excited with a diverse set of audio signals in order to analyze its response and characterize its features. The neuromorphic auditory system response times and frequencies are reported. The experimental results of the proposed system implementation with 64-channel stereo are: a frequency range between 9.6 Hz and 14.6 kHz (adjustable), a maximum output event rate of 2.19 Mevents/s, a power consumption of 29.7 mW, the slices requirements of 11 141, and a system clock frequency of 27 MHz.**

Index Terms—**Address event, artificial cochlea, FPGA, neuromorphic engineering, pulse frequency modulation (PFM), real-time audition.**

I. Introduction

NEUROMORPHIC engineers study, design, and develop neuroinspired systems, such as analog very large-scale integration (aVLSI) chips for sensors [1], [2], neuroinspired processing [3], filtering or learning systems [4], [5], neuroinspired control pattern generators [6], neuroinspired robotics [7], [8], software frameworks [9], and basic spiking building blocks for creating larger scale systems [10], [11]. Spiking systems are neural models that mimic the neuron layers of the brain for processing purposes. Signals in these systems are composed of short pulses in time, called spikes. The term spike (in this context) is derived from action potentials in neurons. Communication between spike-based layers, with hundreds or thousands of neurons per layer, can become very complex, because one-to-one (or one-to-many) connections are needed between neurons that need to be distributed easily over several chips or FPGAs. This problem was solved with the introduction of address-event representation (AER), Mead Laboratory in 1991 [13].

Traditional digital signal processing (DSP) techniques commonly apply multiply–accumulate (MAC) operations over a collection of discrete samples codified as fixed or floating point representation. MAC operations often require dedicated and complex resources, i.e., float-point multipliers, which are available in FPGAs as dedicated expensive resources in relatively small quantities. Therefore, applying a sequence of MAC operations over a data set with these units requires to multiplex them in time. So they are reused with different input data and output results, which are stored in a global memory. It often requires high-frequency clock signals to achieve a competitive data throughput. Furthermore, large memory depths to store intermediate data and results are needed. These facts are reflected in the power consumption and circuitry complexity.

On the other hand, spike signal processing (SSP) has the aim to implement the basic operations that commonly are performed in DSP, but over spike rate coded signals [10]. Thus, operations are performed directly over spike streams, being equivalent to simply adding or removing spikes at the right moment (although is not evident which). The currently available circuits that implement SSP operations [10] use general purpose FPGA resources, as counters, comparators, and logic gates. This allows to build large-scale dedicated systems in hardware, which process spike coded signals in real time using low frequency clocks in a fully parallel way for (low cost) FPGAs.

In this paper, we present a novel way to process the sound wave using SSP. Our proposed neuromorphic auditory sensor (NAS) for FPGA transforms the information in the acoustic wave into an equivalent spike rated representation, and then uses a set of Cascade spike-based low-pass filters (SLPFs)

This work was supported in part by the European Regional Development Fund within the Spanish Government through the BIOSENSE Project under Grant TEC2012-37868-C04-02 and in part by the European Regional Development Fund within the Andalusia Coun-cil through the MINERVA Project under Grant P12-TIC-1300. *(Corresponding author: Elena Cerezuela-Escudero.)*

The authors are with the Robotics and Computer Technology Laboratory, E.T.S.I. Informatica, University of Seville, Sevilla 41012, Spain (e-mail: ajimenez@atc.us.es; ecerezuela@atc.us.es; lmiro@atc.us.es; mdominguez@atc.us.es; gomezroz@atc.us.es; alinares@atc.us.es; gaji@atc.us.es).

TABLE I
SUMMARY OF CHARACTERISTIC OF SOME ANALOG COCHLEAE

Ref.	Number of Channels	Frequency range	Dynamic Range	Event Rate	Power cons.
[2]	64x2	50Hz-50kHz (adjustable)	36dB	10 MEvents /Sec.	18-26 mW
[16]	360	200Hz-20kHz	52dB	33 kSpikes /Sec.	51.8 mW
[17]	64x2	200Hz-6.6kHz	46dB	Not provided	56.32 mW

TABLE II
SUMMARY OF CHARACTERISTIC OF SOME DIGITAL COCHLEAE

Ref.	Number of Channels	Frequency range	Slices / Utilization	Max. clock (MHz)
[22]	88	1,006-7,630 Hz	6,800 slices	44.15
[25]	16	150-3,400 Hz	11,048 slices	Not provided
[28]	1,224	20 -20,657 Hz	113,760 slices	142

bank inspired on Lyon's model of the biological cochlea [13]. This auditory system processes information directly encoded as spikes using pulse frequency modulation (PFM), decomposes PFM audio into a set of frequency bands, and propagates that information by means of an AER interface.

The biological cochlea is a part of the inner ear that plays a central role in hearing. It moves in response to vibrations caused by sound signals entering the ear and vibrating the basilar membrane. Thousands of hair cells on the membrane sense the vibration and excite the spiral ganglion cells, which generate action potentials, or spikes, that travel along nerve fibers to higher order auditory brain areas. Because of the physical properties of the basilar membrane, high-frequency inputs activate the basilar membrane area closest to the base of the cochlea, while low-frequency waves travel further down the membrane [14]. The first silicon cochlea was proposed by Lyon and Mead [15]. In their design, the membrane basilar was modeled by a Cascade of 480 second-order filter sections. There are several VLSI implementations of the cochlea based on Lyon's design (for example, [16]–[19]). One of them is even commercially available [2]. Some of these implementations, [2], [18], [19], use AER to represent the identification address of active channels. Table I shows the characteristics of the latest analog implementations. On the other hand, digital models of the cochlea have also been documented in the literature [20]–[28], summarizing some of them in Table II.

In biological cochleae, the acoustic wave is filtered mechanically and its frequency components are represented by neural pulses in the auditory nerve. An analog cochlea transforms the sound wave into an analog electronic signal that is processed by aVLSI filters. In digital cochleae, the sound wave is transformed into digital data that is processed by VLSI discrete digital filters. The main difference between analog and digital filters is the precision achieved when working with the information. Analog filters never suffer from truncating resolution errors, while digital filters never suffer from transistor mismatch at the fabrication level. When the audio signal is represented in an ideal PFM and the filters are applied in that domain, this NAS would offer the benefits of both types (no resolution errors and no mismatch errors). However, the implementation presented in this paper did not have an ideal PFM representation because the way audio is converted to spikes, which experiences some truncating resolution errors.

This paper is structured as follows. Section II presents the global architecture of the NAS, and shows how it was developed with spike-based building blocks [10]. These blocks work mainly with the spike frequencies in a different way to the integrate-and-fire (IF) neuron [3]. Section III presents the design flow for synthesizing a parameterized NAS, with a new set of features. One of the most difficult tasks is to properly tune the NAS, so in Section IV, we propose using a genetic algorithm (GA). The NAS has a high degree of scalability, and may, therefore, consume a huge number of FPGA resources. Section V presents a resource study based on the synthesis results for different NAS sizes. Sections VI and VII describe a test scenario, using a Virtex-5 development board from Xilinx, and present the experimental results obtained. The NAS responses to diverse stimulus are also analyzed and characterized. Finally, some conclusions are presented in Section VIII.

II. BINAURAL NAS ARCHITECTURE

The architecture of the NAS is shown in Fig. 1(a). It comprises two different Cascade banks of time domain SLPFs. These are described in detail in [10] and [29]. The aim is to decompose two digitalized audio streams (the left and right ear's audio signals) into a set of bands, having previously converted them into trains of spikes [30]. The streams are decomposed in time domain spike-based filter banks, the information of each channel being encoded and passed on as spikes. The output spikes from the filters are collected by an AER monitor [31] that transmits the spike information using a four-stage asynchronous AER protocol.

Each complete NAS was modeled by a time domain spike-based filter bank, or Cascade filter bank (CFB). Each CFB comprising several stages connected in Cascade for that particular NAS; the architecture of a single stage is shown in Fig. 1(c). Each stage has a time domain SLPF and an element capable of subtracting two spike coded signals (both elements will be explained later in detail in this paper). As in previous implementations of AER cochleae [2], [15], [20], [22], a series of several Cascade-connected stages subtracted the information from consecutive SLPF output spikes in order to reject out-of-band frequencies, obtaining a response equivalent to that of a bandpass filter. The SLPF architecture for implementing the NAS banks was presented earlier in [10], but for a better understanding, we will briefly introduce the elements that are needed to design an SLPF. All the elements were written in VHDL and designed as

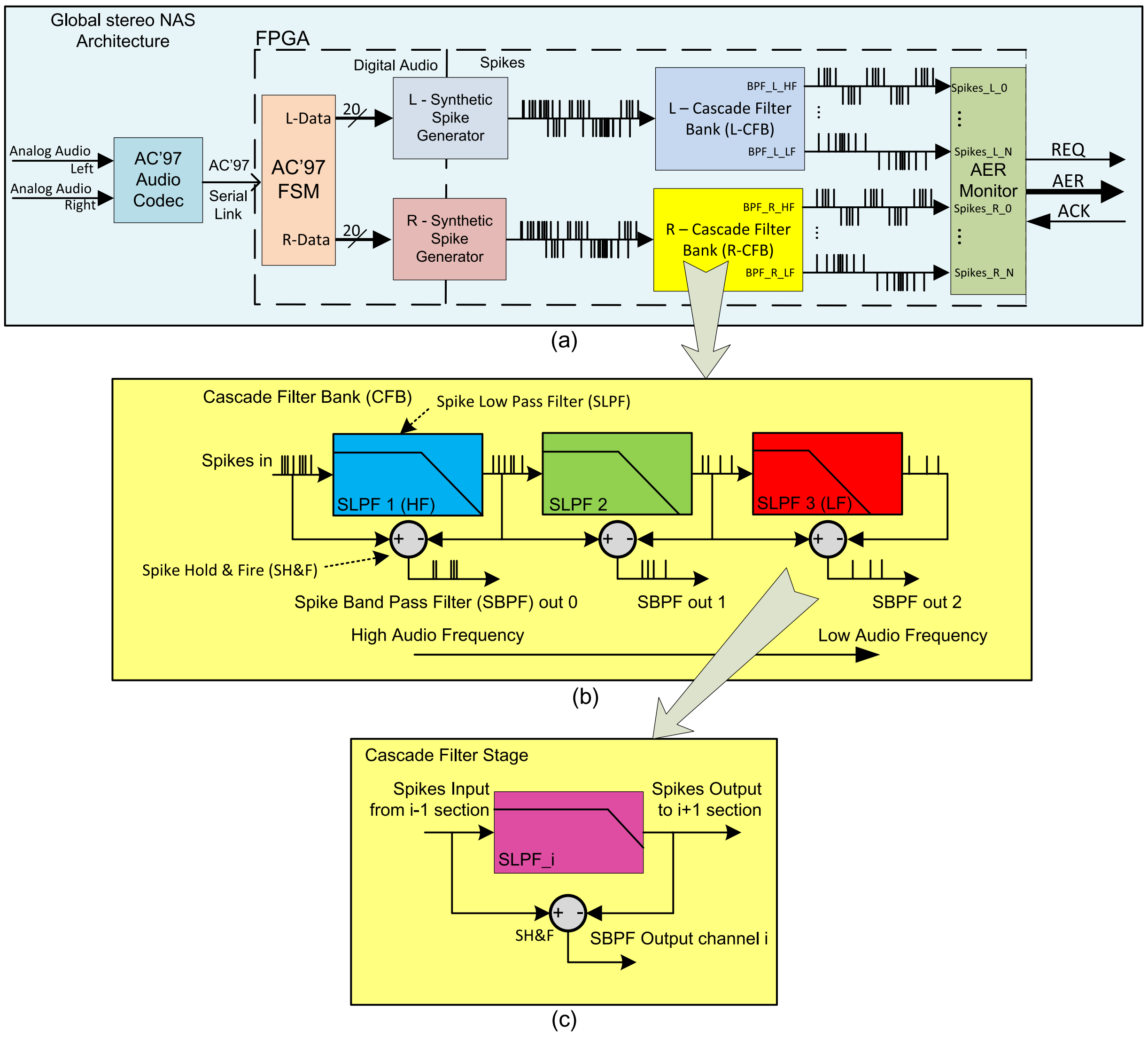


Fig. 1. (a) Global NAS architecture. (b) Filter banks with Cascade topology, CFB. (c) Single CFB stage containing an SLPF and an SH&F.

small general purpose building blocks. Each of these blocks performs a specific primitive arithmetic operation on the spike streams, and can be combined with others to build large spike processing systems of the type already used in closed-loop spike-based PID controllers [32], and trajectory generators for object tracking [33].

A. Reverse Bitwise Synthetic Spike Generator

Digital sound samples received from a commercial audio codec were immediately converted into a stream of spikes by a digital synthetic spike generator (SSG) capable of converting a discrete number (SSG input) into a fixed spike frequency rate (SSG output). These output spikes represented the audio information that would excite the CFB. This SSG is also the formed part of other elements in the CFB capable of processing spike-coded signals. This will be explained later.

Although there are several ways to design a digital SSG [30], [34], the implementation in this paper used the reverse bitwise method for synthetic AER event generation [reverse bitwise SSG (RBSSG)] described in [30] and [35].

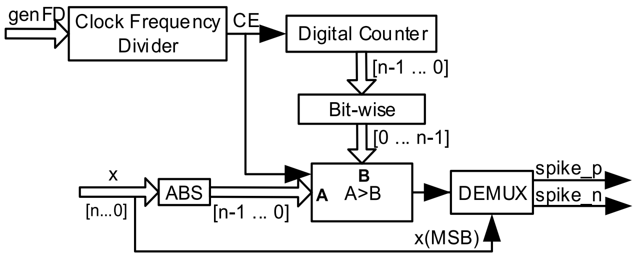


Fig. 2. RBSSG [10].

This architecture was selected mainly for its low resource needs (a digital counter and a comparator) and closer to uniform temporal spike distribution.

An SSG would generally be capable of generating a synthetic spike stream with a frequency proportional to a constant ($k_{\text{BWSpikesGen}}$) and an input value (x), as in

$$\text{RBSSG}(x)_{\text{SpikesRate}} = k_{\text{BWSpikeGen}} * x. \quad (1)$$

Fig. 2 shows the RBSSG circuit. It uses a continuous digital counter [Fig. 2 (top)], the output of which is reversed

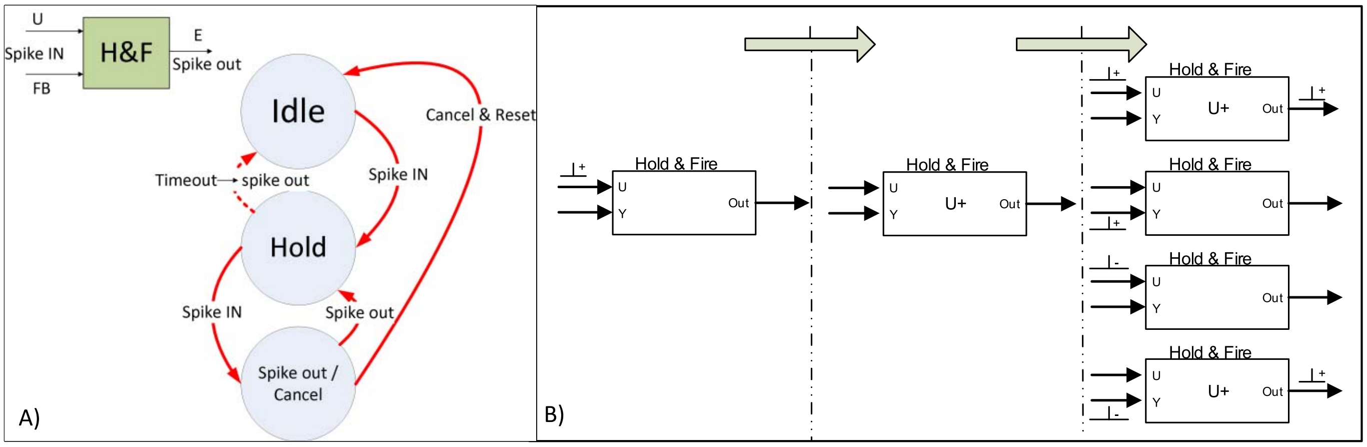


Fig. 3. (a) SH&F state machine. (b) Evolution example from a positive U spike (U+).

bitwise and compared with the input absolute value [ABS(x)]. When the input absolute value is greater than the reversed counter value, a new spike is fired [Fig. 2 (bottom)]. The RBSSG ensures uniform spike distribution over time, thanks to the bitwise reversal output of the counter. Since the excitation signal may be signed, it is necessary to generate positive and negative spikes. We, therefore, used a demultiplexor (DEMUX) to select the right output (positive or negative) for the spikes generated. The DEMUX selection signal is the x input sign, or x(MSB). Finally, a clock frequency divider [Fig. 2 (top-left)] was included to accurately adjust the RBSSG gain. The clock frequency divider activates a clock enable signal, which divides the spike generator clock frequency according to a frequency divider signal (genFD). The RBSSG gain ($k_{\text{BWSpikeGen}}$) can thus be calculated as in

$$k_{\text{BWSpikeGen}} = \frac{F_{\text{CLK}}}{2^{N-1}(\text{genFD}+1)}. \tag{2}$$

F_{CLK} represents the system clock frequency, N is the RBSSG bit length, and genFD is the clock frequency divider value. These parameters can be modified to set up an RBSSG gain that meets design requirements.

B. Spike Hold & Fire

The spike hold & fire (SH&F) block performs the subtraction between two spike trains. Subtracting one spike input signal (f_U) from other (f_Y) means creating a new spike signal with a spike rate ($f_{\text{SH\&F}}$) that is the difference between the two input spike rates

$$f_{\text{SH\&F}} = f_U - f_Y. \tag{3}$$

The function of the SH&F block is to hold the incoming spikes for a fixed period of time while input evolution is monitored to decide whether output spikes must be produced. Fig. 3(a) shows the SH&F behavior as a state machine, and Fig. 3(b) shows how SH&F can evolve from a positive input spike. In Fig. 3(a), state machine can be seen that the transition between hold state and idle state is governed by a timeout that represent the fixed period of time mentioned earlier. In [32], that time was set to 10 μs with a configurable range between 1 μs and infinite. In this paper, we use an

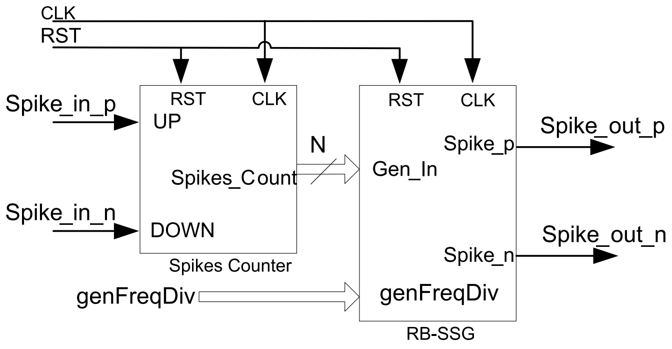


Fig. 4. SI&G block diagram [10].

infinite period of time, so there is no real transition between hold and idle states. SH&F has two inputs: U (positive input) and Y (negative input, commonly used as feedback). Let us suppose that a positive spike, $U+$, is received as U input [Fig. 3(b) (left)]. The $U+$ state is held internally [Fig. 3(b) (center)]. SH&F will do nothing if no more spikes are received. When a new spike arrives, it behaves according to the corresponding spike input port and sign [Fig. 3(b) (right)]. As shown in Fig. 3(b) (top-right), if SH&F receives a positive spike ($U+$), the previously held spike is fired as a positive spike ($U+$), and the new one is held internally ($U+$ state). If a negative input spike is received in port U ($U-$), or a positive spike is received in port Y ($Y+$), the previously held spike is cancelled and no output spike is produced. Finally, if a negative spike ($Y-$) is received in Y [Fig. 3(b) (bottom)], the previously held spike is fired and the last one received is held with a positive sign ($U+$ state). This SH&F behavior can be extended to deal with any kind of input spikes ($U-$, $Y+$, or Y$-$ states) using the same logic: hold, cancel, and fire spikes according to the port and sign of the input spike. Further information about its behavior and temporal response can be found in [10] and [32].

C. Spike Integrate & Generate

The building block called spike integrate & generate (SI&G) [10] integrates spikes. SI&G comprises a spike counter and an RBSSG, as shown in Fig. 4.

The spike counter is a digital counter, the value of which increases when a positive spike is received and decreases when a negative spike is received. The counter output represents the spike's integration value. To convert the integrated count to spikes again, the spike counter output is connected to an RBSSG input, which will generate a new stream of spikes. These spikes will have a frequency proportional to the spike count, or the spike integration value. There is no delay between the integrator and the generator part of this model. The frequency response is, therefore, instantaneous with regard to the integrator part. Furthermore, this element can be seen as a neuron model, where the membrane potential is the output of the integrator (as in the IF neuron), but with a frequency response (not a single action potential spike as in IF neuron models). The output is, therefore, a stream of action potentials with an interspike-interval (ISI) time that is a function of the integrator.

Considering this SI&G architecture, the SI&G spike output frequency can be calculated as expressed in (4). SI&G gain, $k_{\mathrm{SI\&G}}$, is set using the RBSSG parameters included. Actually, it is equivalent to $k_{\mathrm{BWSpikeGen}}$, and can be tuned with the same parameters we previously introduced in (2) to set up the RBSSG gain (N, F_{CLK}, and genFD)

$$\begin{aligned} f_{\mathrm{SI\&G}} &= k_{\mathrm{SI\&G}} * \int_0^t f_{\mathrm{inputSpikes}} dt \\ &= \frac{F_{\mathrm{CLK}}}{2^{N-1}(\mathrm{genFD}+1)} \int_0^t f_{\mathrm{inputSpikes}} dt. \end{aligned} \tag{4}$$

As in continuous systems, the equivalent SI&G transfer function in the time domain can be calculated using a Laplace transform analogy [10]. Other authors also compare spiking systems with dynamical systems, such as [12]. The SI&G transfer function is presented in (5). It is equivalent to an ideal integrator with a gain of $k_{\mathrm{SI\&G}}$, but in a spike-based context

$$\mathrm{SI\&G}(s) = \frac{k_{\mathrm{SI\&G}}}{s} = \frac{F_{\mathrm{CLK}}}{2^{N-1}(\mathrm{genFD}+1) * s}. \tag{5}$$

D. Spike Frequency Divider

The spike frequency divider (SFD) divides the spike rate of an input spike train by a constant number. An SFD can be implemented in many ways: for example, by using simple counters, by firing one spike when several spikes have been received, or using probabilistic techniques that decide whether or not to propagate an incoming spike using a random number generator, implemented with linear feedback shift registers. These techniques have one common problem: the output spike rate is correct on average, but the spikes are not distributed uniformly over time, introducing high-frequency noise in terms of ISI. To ensure uniform spike distribution, we designed this RBSSG-inspired SFD block.

Fig. 5 shows the block's internal components. The digital comparator output drives a buffer that allows or prevents input spikes from passing through the SFD. In general, input spikes will increase the digital counter, the reverse value of which is then compared with the *spikesDiv* signal. If that reversed counter output is lower than *spikesDiv*, the output buffers will be enabled, allowing the next spike to travel through this

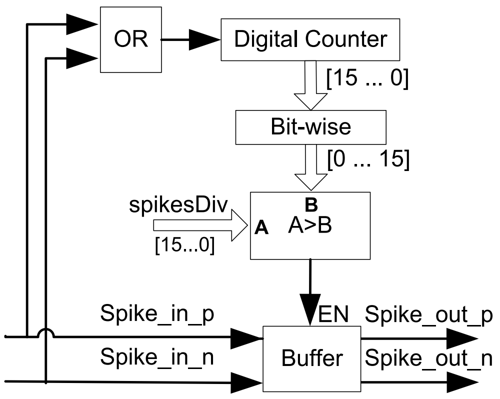


Fig. 5. SFD [10].

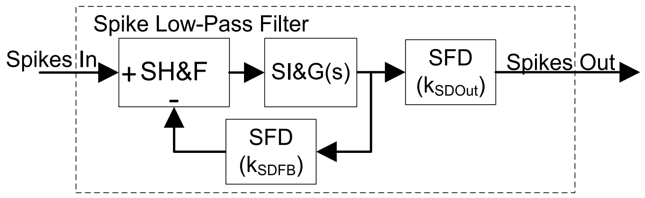


Fig. 6. SLPF Architecture [10].

component. The output buffers only enable output spikes when *spikesDiv* is uniform over time.

The SFD transfer function is shown in (6), where N represents the SFD number of bits, and *spikesDiv* the previously presented signal. The SFD transfer function is equivalent to a gain block with a value in the range of [0, 1], with 2^N possible values

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

E. Time-Domain Spike-Based Low-Pass Filters

The time-domain SLPF block filters high-frequency changes in the input spike rate. To build this element, we used feedback: the SI&G output is its inputs after processing by an SFD and by SH&F, as shown in Fig. 6. The idea was to integrate input spikes with an SI&G, subtracting SI&G output spikes with input spikes using SH&F, and thus performing a basic filter operation, with no great accuracy and with a fixed gain of 1. To achieve better accuracy and improve SLPF gain, two SFDs were included. The most important one is the SFD placed in the feedback loop, which has to divide the feedback spike frequency. This implies less input spikes in the SH&F and, therefore, a higher number of spikes in the SI&G input. It also makes the filter gain higher than 1, because SI&G integrates more spikes, and makes it possible to choose the cutoff filter frequency more accurately, as will be discussed later. A second SFD was placed at the SLPF output, making it possible to decrease SI&G output bandwidth. SLPF gain is, therefore, fully adjustable. According to SI&G feedback topology, and considering both SFD gains, it is possible to calculate the SLPF equivalent transfer function using basic systems theory. Equation (7) shows the ideal SLPF transfer function. The SI&G(s) can be obtained from (4).

TABLE III
BUILDING BLOCKS UTILIZATION SUMMARY

RBSSG	SH&F	SI&G	SFD	SLPF
37slices	17slices	15slices	33slices	77 slices

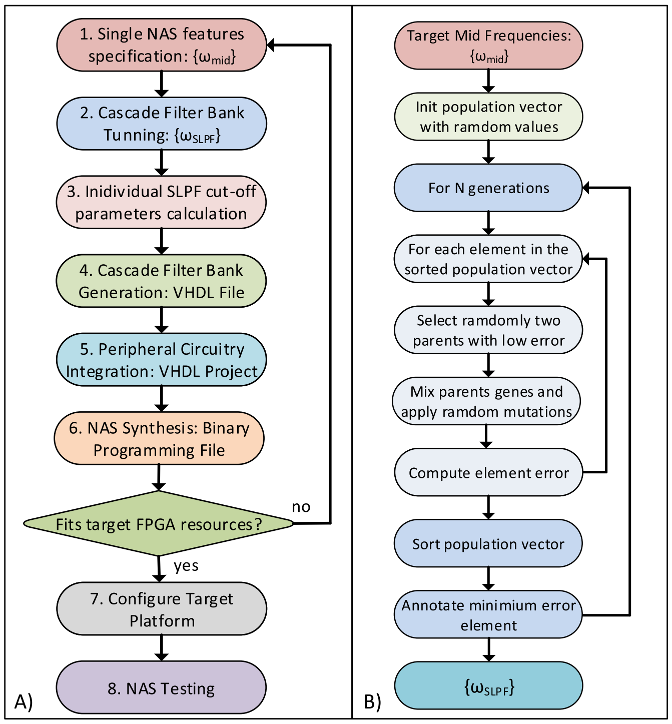

1. Single NAS features specification: {ω_{mid}}
2. Cascade Filter Bank Tunning: {ω_{SLPF}}
3. Inidividual SLPF cut-off parameters calculation
4. Cascade Filter Bank Generation: VHDL File
5. Peripheral Circuitry Integration: VHDL Project
6. NAS Synthesis: Binary Programming File
Fits target FPGA resources?
no
yes
7. Configure Target Platform
8. NAS Testing
A)
Target Mid Frequencies: {ω_{mid}}
Init population vector with ramdom values
For N generations
For each element in the sorted population vector
Select ramdomly two parents with low error
Mix parents genes and apply ramdom mutations
Compute element error
Sort population vector
Annotate minimium error element
{ω_{SLPF}}
B)

Fig. 7. (a) Spike-based NAS design flow. (b) GA flow for CFB tuning.

k_{SFDOut} represents output SFD gain and k_{SFDFB} represents the gain of the SFD placed in the feedback loop. Both are detailed in (6)

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

$$= \frac{k_{\text{SFDOut}} * k_{\text{SI\&G}}}{s + k_{\text{SFDFB}} * k_{\text{SI\&G}}} \tag{7}$$

$$\omega_{\text{SLPF}} = k_{\text{SFDFB}} * k_{\text{SI\&G}} \tag{8}$$

$$k_{\text{SLPF}} = F_{\text{SLPF}}(s) = \frac{k_{\text{SFDOut}} * k_{\text{SI\&G}}}{k_{\text{SFDFB}} * k_{\text{SI\&G}}} = \frac{k_{\text{SFDOut}}}{k_{\text{SFDFB}}}. \tag{9}$$

The SLPF transfer function is equivalent to a first-order low-pass filter with a pole. The theoretical filter cutoff frequency, ω_{SLPF}, in rad/s, can be determined by the product between $k_{\text{SI\&G}}$ and k_{SFDFB} in (8). Equivalent SLPF gain can be set with the relation between the values of k_{SFDFB} and k_{SFDOut}, as expressed in (9).

Table III lists the FPGA resources (slices) for the building blocks mentioned earlier. They have been synthesized and implemented separately in the FPGA, what implies an incomplete use of slices. It can be noticed that when the SLPF is synthesized, resources are optimized because slices utilization is more compact.

III. NAS DESIGN FLOW

This section presents the design flow needed to achieve a real VLSI implementation of the NAS architecture described in Section II (Fig. 1). Since that the NAS architecture was conceived for implementation in a programmable digital device, such as an FPGA, this design flow can be seen as a generic process for building a full custom auditory system, with a particular set of parameterized features and for a specific application. Our NAS design flow has eight stages. They are shown in Fig. 7(a). The NAS design process was almost automatic, as user interaction was only needed to link the different software applications required in the different stages. These software tools were: 1) MATLAB, for NAS parameter tuning, automatic VHDL design file generation, and testing and 2) Xilinx ISE Design Suite, for synthesis and FPGA bit stream downloading.

The first step was to define the features of a single NAS, i.e., to specify the number of channels, N_{ch}, and the midfrequencies of each channel. This led to a set of midfrequencies $\{\omega_{\text{mid}}\}$ that defined the frequency features of the NAS response. Considering that the NAS acts like a spectrum analyzer, this task had to be done considering a number of questions, including application specification requirements (source signal frequency features, the number of channels, frequencies of interest, and so on) and the capabilities of the target FPGA (general and specific resource requirements) where the NAS was to be loaded.

The second step was to tune a CFB to fit, or at least provide the best approximation to, the desired midfrequencies set, $\{\omega_{\text{mid}}\}$. This involved finding a set of SLPF cutoff frequencies, $\{\omega_{\text{SLPF}}\}$, for the $N_{\text{ch}+1}$ SLPFs that would make up the CFB. Due to the Cascade architecture of the filter bank, the setting of these cutoff frequencies was critical because the choice of a particular frequency for an intermediate SLPF would affect all consecutive channels. Because of the complexity of an analytical search for $\{\omega_{\text{SLPF}}\}$, we addressed this task using a GA that is explained in Section IV.

Once a set $\{\omega_{\text{SLPF}}\}$ that satisfied the NAS specification requirements was found, the third step was to convert the cutoff frequencies of the SLPF to the corresponding parameters ($k_{\text{SI\&G}}$, k_{SDFB}, and k_{SDOut}), according to (8) and (9). Since we needed an SLPF with a 0-dB gain in the bandpass, $k_{\text{SDFB}} = k_{\text{SDOut}}$.

The fourth step was to generate a VHDL file that would implement the CFB with the right cutoff frequencies for each SLPF. A MATLAB script took the number of SLPFs and their parameters and automatically generated the VHDL file to implement the required CFB.

At this point, the core of our NAS was ready to process audio spike information. In the fifth step, the CFB was integrated with the peripheral circuitry. This circuitry had two components: 1) a finite state machine (FSM) for decoding the incoming audio from the AC'97 audio codec [Fig. 1(a) (left)] and feeding the digitalized audio samples to the RBSSG and 2) an AER monitor for monitoring the spike output activity of each SLPF in the CFB and converting it into a stream of AER events in the output of the proposed system [Fig. 1(a) (right)]. Depending on requirements and on the available resources of the target FPGA, the NAS could be built as a monaural or binaural sensor, so in this step, only one CFB needed to be added for a monaural NAS, or two CFBs for a binaural NAS. All the results presented in this paper are for a binaural NAS with two identically parameterized CFBs.

After integrating and connecting all the NAS components [AC'97 FSM, RBSSG, CFB(s), and AER monitor], the sixth step was to synthesize the binaural NAS with selected FPGA manufacturer tools (in our case, Xilinx). If the number of logical resources required for NAS synthesis exceeded the target FPGA capabilities, we had two options: 1) reduce the number of NAS channels or 2) go to a monaural CFB per FPGA, and then go back to the first step of the procedure. Section V presents an analysis of hardware requirements according to the number of NAS channels, and may, therefore, be of use to designers wishing to determine an FPGA's capability to implement a specific NAS. The seventh step was to load the binary file obtained from the Xilinx Design Suite into the FPGA and the eighth step was to perform operational tests.

IV. SLPF Bank Tuning With a Genetic Algorithm

This section shows the mechanism for adjusting all the parameters of the CFB. It was necessary to have a previously defined set of midfrequencies for each channel, $\{\omega_{\text{mid}}\}$. Equation (10) shows the theoretical transfer function of the ith NAS channel. Due to the Cascade architecture, which implies strong coupling between NAS midfrequencies, and the higher order of the transfer function, it was very difficult to find an analytical solution for all the SLPF cutoff frequencies, $\{\omega_{\text{SLPF}}\}$, in the CFB. There were several possible ways to solve this problem, for example, an iterative approximation algorithm, backpropagation, and so on. In this paper, we opted to use a GA derived from the classical design flow proposed in [36] for filter bank tuning. This choice was a bioinspired alternative, and allowed us to measure and calculates whether a certain set of SLPF cutoff frequencies $\{\omega_{\text{SLPF}}\}$ was accurate or not. It provided us with a set of midfrequencies that matched, or were very close to, the desired frequencies $\{\omega_{\text{mid}}\}$ by evolving a population vector for several generations, considering the SLPF imperfections and the effects of Cascade coupling

$$\text{BPF}_i(S) = \prod_{k=1}^{i}\left(\frac{\omega_{\text{SLPF}\_k}}{(s+\omega_{\text{SLPF}\_k})}\right) - \prod_{j=1}^{i+1}\frac{\omega_{SLPF\_j+1}}{(s+\omega_{\text{SLPF}\_j+1})}. \tag{10}$$

A. GA Data Structure and Target Function

Fig. 8 shows the data structure of the GA for CFB tuning. We built a population vector in which every element had a set of genes representing the cutoff frequencies set for the SLPFs of a Cascade bank, $\{\omega_{\text{SLPF}}\}$, sorted from higher to lower frequencies. Average error was stored, to later shorten the population vector.

The target function of the GA was to minimize deviation between the ideal midfrequencies $\{\omega_{\text{mid}}\}$ and the equivalent bandpass filter midfrequencies of an element in the population vector. To determine $\{\omega_{\text{mid}}\}$ of a population vector element, we simulated the full NAS model using the Control Toolbox included in the MATLAB. This has a high computational load and requires a considerable amount of computation time. The exact length of time depends directly on the number of CFB channels for every element in the population vector. Equation (11) was used to compute the deviation of each individual filter, and (12) to calculate the average deviation for all the filters

$$\text{Error}_{\text{Norm}\_i} = \frac{\left|\sqrt{\omega_{\text{SLPF}\_i} * \omega_{\text{SLPF}\_i+1}} - \omega_{\text{mid}\_i}\right|}{\omega_{mid\_i}} \tag{11}$$

$$\text{Error}_{\text{Norm}} = \frac{\sum_{i=1}^{N}\text{Error}_{\text{Norm\_i}}}{N}. \tag{12}$$

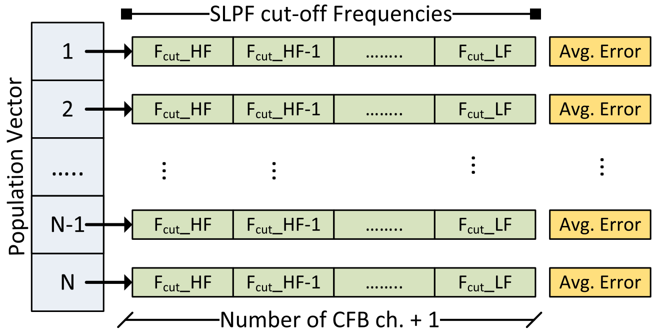


Fig. 8. Population vector and gene codification.

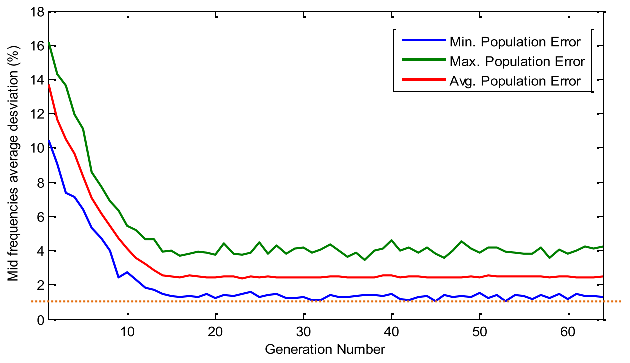


Fig. 9. GA error evolution through 64 generations for a 32-channel CFB.

B. GA Flow

Fig. 7(b) shows the GA iteration flow, starting from a specified $\{\omega_{\text{mid}}\}$ (initial NAS specifications). The population vector was initialized with a random set of SLPF cutoff frequencies, $\{\omega_{\text{SLPF}}\}$. After all the initializations, the error of each element was computed, and the population vector sorted considering this error.

After this, the main loop of the GA starts, allowing the population vector to evolve for a defined number of generations. Each generation replaces elements of the population vector with crossovers of the best elements (lowest error) of the old populations. To do this, in each generation, the population vector has to be filled with new elements, so for each element in the population vector, we need to find a random pair of parents among the lower error elements in the previous generation, and generate a new element in which $\{\omega_{\text{SLPF}}\}$ is

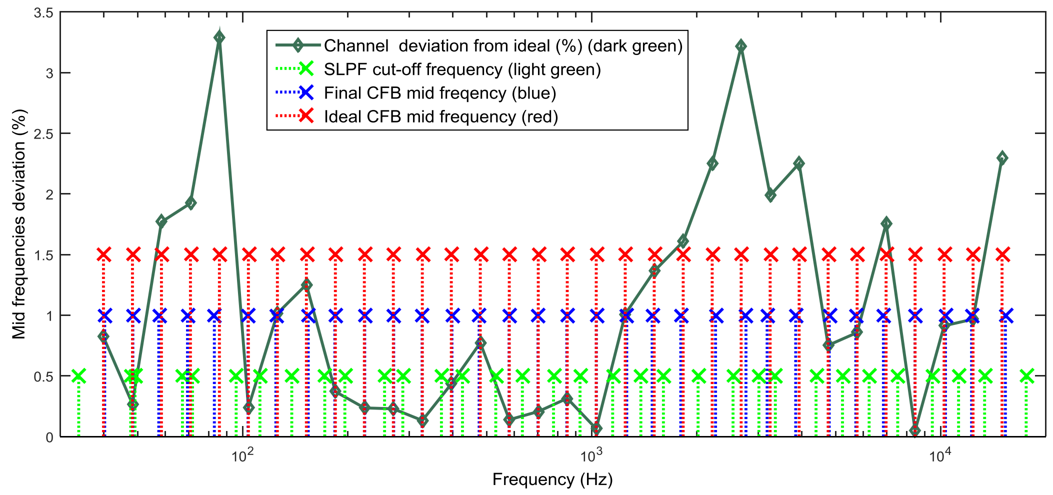


Fig. 10. GA results: ideal and final CFB midfrequencies, SLPF cutoff frequencies, and channel relative error.

a mixture of the parents' genes and a small, added random number (mutation). Finally, the NAS is simulated and the relative error is computed. Once the population vector is filled with a new generation, it is sorted, taking the best elements (those with the least error) in the first positions of the population vector for a new generation.

C. GA Execution Results

After executing the GA, we obtained the element with the lowest error from among the generations. This element provided the set $\{\omega_{\text{SLPF}}\}$ with the best fit $\{\omega_{\text{mid}}\}$ in our entire GA. Fig. 9 shows the evolution of the population vector error after executing the GA for a 32-channel CFB through 64 generations. Using an Intel Core i5 4670 at 3.4 GHz and applying parallel optimizations in the MATLAB, this execution took 3329.45 s.

Fig. 9 contains the minimum, maximum, and average errors after GA execution with (11) and (12). In the first generations, the error was very high, nearly 14%; however, with the genetic evolution of the population vector, it quickly decreased. After 20 generations, the error started to reach minimum values below 2%. In the next generations, the error did not present significant changes; however, gene mutation can facilitate a better CFB fine-tuning, so we expanded the study to 64 iterations. In iteration 45, one of the population members reached the minimum error.

Fig. 10 shows the solution provided by the element with the lowest error found by the GA. Fig. 10 contains the ideal CFB midfrequencies, $\{\omega_{\text{mid\_ideal}}\}$ (red crosses), the midfrequencies that were found, $\{\omega_{\text{mid}}\}$ (blue crosses), the SLPF cutoff frequencies, $\{\omega_{\text{SLPF}}\}$ (green crosses), and the relative midfrequency deviation of each channel (dark green line), using (11). In accordance with (12), this element had an average error of 1.573%. This is comparable to cochlear implants, where users are able to discriminate sentences with a deviation below 10% [37].

V. NAS Synthesis

Once $\{\omega_{\text{SLPF}}\}$ was obtained for every SLPF contained in the CFB, the last step was to generate the VHDL files to implement the already parameterized NAS and perform the tests. As commented earlier, this was done using automatic scripts: first calculating the SLPF's own set of parameters ($k_{\text{SI\&G}}$, k_{SDFB}, and k_{SDOut}), and then automatically generating the VHDL file of the CFB using previously calculated parameters. Finally, one or two instances of the CFB were integrated with the peripheral circuitry (AC'97 FSM and AER Monitor). It is possible to add more than two CFBs for some noncommon applications, e.g., a quadrasonic audition system. The nature of this architecture makes it scalable not only in the number of channels, but also in the number of CFBs.

To measure hardware requirements for the purpose of choosing the right FPGA for a particular design, several binaural NASs were synthesized with different features (number of channels). This was done for the Virtex-5 FPGA (XC5VFX70T) included in the ML507 Xilinx development board [38]. It should be noted that this NAS architecture, thanks to its implementation of spike-based building blocks, does not require specialized FPGA resources (i.e., multipliers, phase-locked loop, embedded processors, DSP, and so on). It requires only common digital logic (counters, comparators, adders, and registers) with a low number of bits and a low connectivity: only two wires are needed for internal communication of the stages of the CFB, and a few I/O pins for transmitting the NAS spikes as AER events.

Table IV presents the synthesis and implementation study results, including the number of CFB channels, the hardware requirements as FPGA slices, the maximum operative system clock frequency, power demand, and the number of external I/O signals required. The bigger the CFB implemented, the more FPGA resources are required. Around 40% of the XC5VFX70T was needed for a 24-channel stereo NAS (two × CFBs with 12 channels each), and almost 100% was needed for a 128-channel stereo NAS. Maximum operating clock frequency decreases with the number of channels, from 180 MHz for a 24-channel stereo NAS to 87 MHz for a 128-channel stereo NAS. This is because the greater the need for logical resources the more levels of asynchronous logic and longer paths are introduced into the internal FPGA routing. Using the Xilinx XPower tool to estimate power demand,

TABLE IV
STEREO NAS HARDWARE REQUIREMENTS

CFB channels	Slices / Utilization	Max. clock (MHz)	Power (mW)	I/O Signals
2x12	4,286/ 38.26%	179.95	6.6	15
2x16	4,415/ 38.41%	171.73	7.2	16
2x24	6,301/ 56.25%	113.74	8.6	17
2x32	7,606/ 67.91%	99.84	14.3	17
2x48	10,241/ 91.43%	91.86	18.1	18
2x64	11,141/ 99.47%	87.31	29.7	18

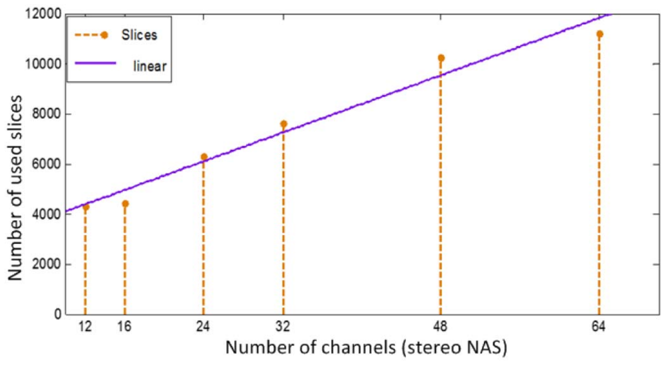


Fig. 11. Logical resource requirements versus the size of CFB.

we simulated the power consumption for different NASs. This is shown in the fourth column of Table IV. Power also increases with NAS channels as the required slices increase. The highest power required for a 128-channel NAS is <30 mW.

The number of required slices was measured against the number of CFB channels. It is interesting to note that the number of required slices decreases per channel as the number of channels increases. This is due to the peripheral circuitry, which must always be added regardless of the number of CFB channels.

In Fig. 11, the y-axis shows the quantity of required slices and the x-axis shows the number of CFB channels, indicating how slice requirements are relatively linear with respect to the number of CFB channels. A linear regression of these values is also shown, making it possible to estimate how many slices will be needed for a specific design using (13). This equation indicates that there is a fixed cost (in terms of slices) of 2663 slices and an additional cost of 71.6 slices for every CFB channel added to the NAS

$$\text{Total slices} \approx 71.6 * \text{channelNumber} + 2663. \tag{13}$$

The last column of Table IV contains the number of I/O signals required in different synthesized NASs. The number of I/O signals may change depending on the number of CFB channels. This happens because the AER address space

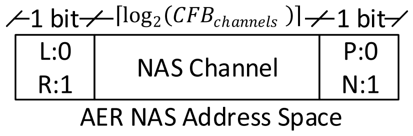


Fig. 12. Structure of the AER address event for NAS spike communication.

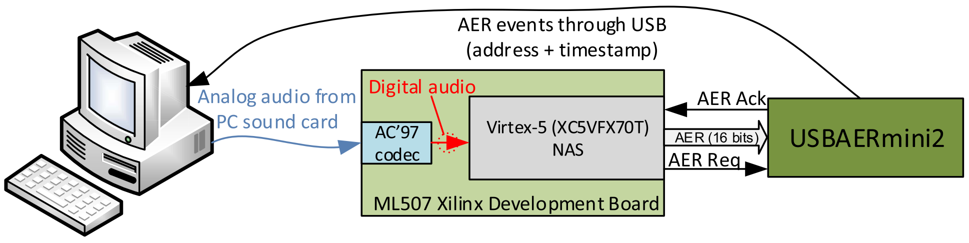
needs to have enough AER addresses to provide a unique address to the output spikes of each of the CFB channels. This implies that the width of the AER bus is related to the size of the CFB. Fig. 12 shows the general structure of the AER events implemented by the included AER monitor. The output AER events have three fields: the first field has a fixed width of one bit and represents the spike's polarity (positive or negative), and the second field contains the identification number (or address) of the CFB channel that fired the spike and its width should be enough to encode the channel number (6 b for 64 channels), while the third field again has a fixed width of 1 b and indicates which CFB fired the spike (left or right). With this AER event structure, we are able to give a unique address to every spike firing inside the CFB. The width of the AER events, in bits, is, therefore, a factor that directly affects the number of I/O signals in the NAS.

The number of NAS I/O signals can be calculated as an addition of the number of fixed signals (clock, reset, AC'97 link, and AER protocol lines) and the width of the AER bus (which varies with the number of CFB channels). This calculation can generally be performed using (14). The following signals are required. One signal for the NAS clocking, six additional signals to manage the AC'97 audio link, two signals that are used for the asynchronous communication of the AER bus (request and acknowledge), and, finally, a number of I/O signals equivalent to the AER bus width

$$\begin{aligned} N_{I/O} &= 2(\text{Clk, Rst}) + 6(\text{AC Link}) \\ &\quad + 2(\text{AER Control}) + \text{AER Bus Width}. \end{aligned} \tag{14}$$

VI. NAS Test Scenario

The NAS test scenario was designed and built to implement a synthesized NAS inside a real platform that would allow us to characterize and analyze the NAS behavior. For experimental testing, a Xilinx development board (ML507) [38] was used, which, among other components, included a Virtex-5 FPGA (XC5VFX70T) and an AC'97 audio codec. The idea was to implement the NAS in the FPGA, receive the analog audio using the AC'97 codec through a standard digital audio AC link, and transmit the AER events through the general-purpose input/output (GPIO) pins available on the board. The use of the AC'97 commercial audio chip imposed a maximum sample frequency for the analog audio signal of 48 kHz, limiting the temporal capabilities of our NAS. For ecolocalization purposes, for example, a higher sampling frequency is needed. In a future study, we intend to explore how the performance and specifications of the proposed NAS can be improved by using faster ADCs or new circuits for directly converting analog signals to PFM.



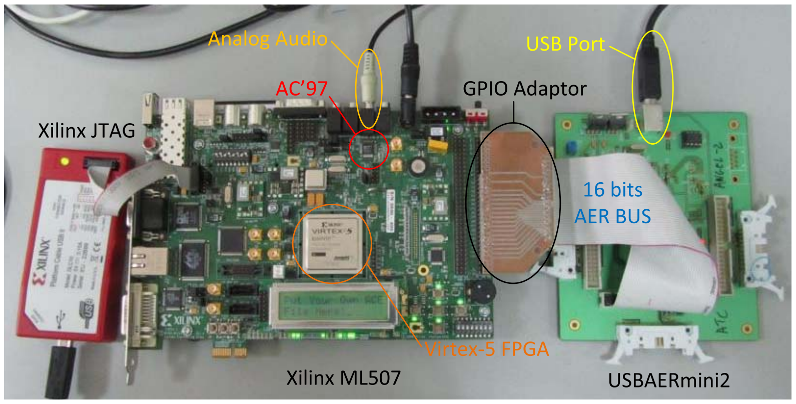


Fig. 13. NAS test scenario block diagram (top) and photograph (bottom).

Fig. 13 (top) shows the block diagram for the test scenario. The computer on the left is the master device; the audio line-out of its sound card is connected to the line-in of the AC'97 audio codec included in the ML507 [Fig. 13 (center)]. The computer sound card is used to stimulate the NAS with different sounds. This AC'97 audio codec is able to digitize a stereo audio signal with a resolution of 20 b with a sample rate of 48 kSamples/s.

The audio stimulus is then processed by an NAS loaded inside the FPGA, generating a stream of AER events, which are sent to the computer via USB, using the USBAERmini2 platform detailed in [39] [Fig. 13 (right)]. The USBAERmini2 represents a bridge between AER buses and the USB bus, so the NAS AER activity can be monitored, sending the AER events directly to the computer with a minimum interevent temporal resolution of 0.2 μs, using jAER open source software jAER [39].

Fig. 13 (bottom) shows a photograph of the real test scenario comprising the ML507 development board (center), an attached Xilinx JTAG programmer (left), and the USBAERmini2 (right). The Virtex5 FPGA and the AC'97 audio codec inside the ML507 are marked. Connected to a ML507 GPIO port, there is a small adapter that adapts the GPIO signals to the AER bus, in accordance with the CAVIAR standard [40]. Finally, the AER bus is connected to the USBAERmini2, which sends AER events to the computer through the USB port.

VII. Experimental Results

Using the test scenario described earlier, several experiments were conducted to analyze the CFB response and extract the NAS features. The features study focused on a 2×64 channel binaural NAS with a 27-MHz clock, because this is the biggest NAS (in terms of CFB channels) that can be loaded into the Virtex5 FPGA. This NAS needs an AER bus width of 8 b, providing an AER space of 256 addresses (0 to 127 for the left CFB, and 128 to 255 for the right CFB, including the polarity bit).

A. Temporal Response

The first experimental result obtained was the temporal response of the NAS. Fig. 14 (top) shows the cochleagram created in the presence of a woman saying "*En un lugar de la mancha*", the first sentence in the famous Spanish novel *Don Quixote*. In Fig. 14 (top), the x-axis represents real time and the y-axis represents the AER address, adding a blue dot every time an AER address appears in the AER bus. Fig. 14 (bottom) (addresses from 0 to 127) shows the left NAS AER activity and Fig. 14 (top) (addresses from 128 to 255) shows the right NAS AER activity. In general, both NASs have delayed responses due to the CFB Cascade architecture, where there is an additional phase delay in the SLPFs implemented by the CFB. Both the left and right CFBs have a similar temporal response, but Fig. 14 shows that the responses are not identical.

Looking at these spikes from a higher level point of view, we can reconstruct the spike rate of the different CFB channels over time, obtaining a representation equivalent to the sonogram in Fig. 14 (center) [2], [42]. Fig. 14 (center) shows the left CFB instantaneous spike rate as a color map, where the x-axis is time, the y-axis is the NAS channel, and the color represents the spike rate. The six words used for the NAS stimulation are clearly visible in Fig. 14 (center), with a specific spike rate through the NAS channels. Fig. 14 (bottom) shows the original audio spectrogram for this sentence.

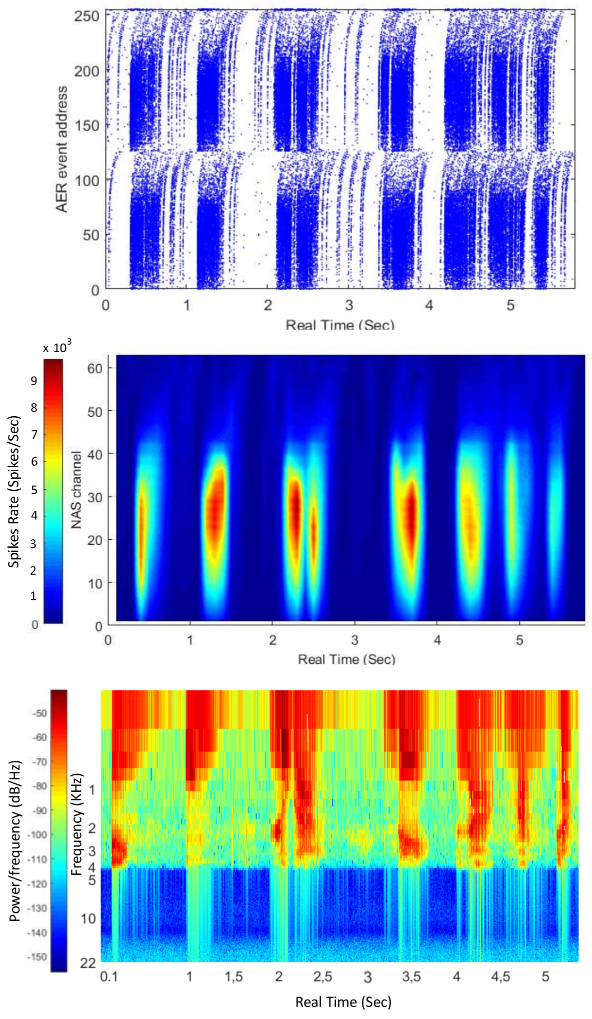


Fig. 14. Experimental cochleagram of a 128-channel stereo NAS (top), reconstructed sonogram of a 64-channel mono-NAS (center), and original audio spectrogram (bottom) corresponding to a woman saying "*En un lugar de la mancha*".

For this example, the maximum frequency that the NAS is detecting is 4 kHz, which corresponds to channel 10 according to Fig. 15 (bottom). And this is the lowest active channel in Fig. 14 (center) where this female voice is in channels 10–35 (300 Hz–4 kHz).

B. Frequency Features

This test studied frequency behavior. An audio frequency sweep was carried out, stimulating the NAS with a set of pure audio tones, and simultaneously monitoring AER activity in the NAS output.

The audio tones had an amplitude of 1 V_{RMS} and varied in a frequency from 10 Hz to 22 kHz. Fig. 15 (top) shows the left NAS bode diagram, that is to say, the spike rate (y-axis) versus the audio tone frequency (x-axis) for each of the CFB channels (different colors). Fig. 15 (top) shows how, in general, the NAS channels act like a set of bandpass filters, rejecting out of band audio tones. Fig. 15 (top) also shows that the spike rate of the bandpass of the CFB channels

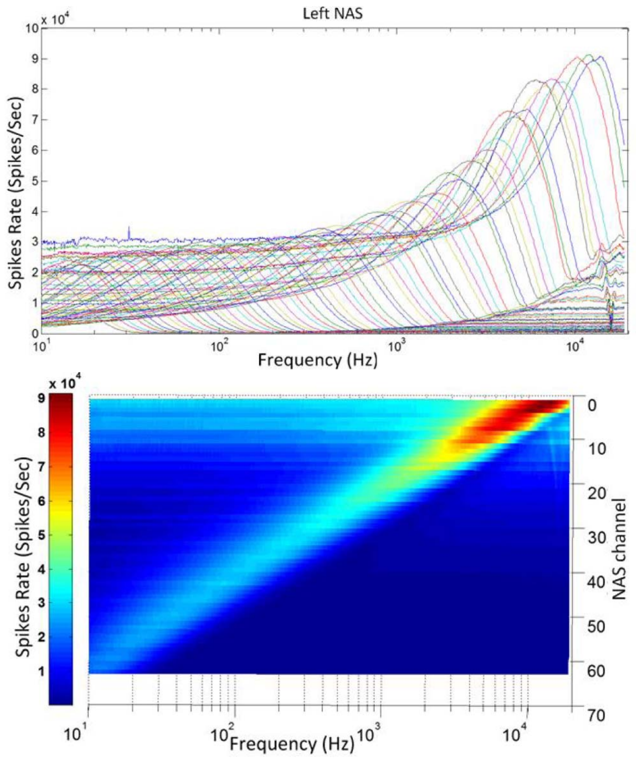


Fig. 15. Frequency response of a 64-channel NAS. Top: 1 V_{RSM} audio tone. Bottom: surface representation of the same frequency response.

decreases with lower frequencies, while the higher frequency channels show the addition of an offset. The same information is represented as a color mesh in Fig. 15 (bottom), where the x-axis represents the tone frequency, the y-axis the NAS channel, and the color intensity the channel's output spike rate. Fig. 15 (bottom) shows how every channel is more active in a specific audio frequency band, the light blue band, and presents no activity outside that band, in the dark blue area. It also indicates that high-frequency channels have progressively higher activity in their bands, with a diagonal gradient moving from light blue at low frequency to red at high frequencies. This effect is due to overlapping with SLPF rejection bands in the Cascade. A higher frequency filter will attenuate softly the band near to cutoff frequency. This effect is increased along SLPF Cascade, showing an accumulative effect in low frequency channels, what implies a stronger attenuation.

This information can be used to measure some parameters related to common bandpass filters [midfrequency, quality factor (Q factor), and bandwidth], and thus to characterize features of the CFB channels in greater detail. Fig. 16 (top) shows the midfrequency, and the spike rate at this frequency, of every CFB channel. In Fig. 16 (top), the x-axis represents the audio frequency tone and the y-axis represents the spike rate. A line is added at the midfrequency of the CFB channels. The distribution of the midfrequencies is relatively uniform. However, as mentioned in Section III-C, it is not perfect, because the GA used to tune the CFB channels was designed to approximate a set of values, and it is very difficult to find a perfect solution. The highest frequency channel has a

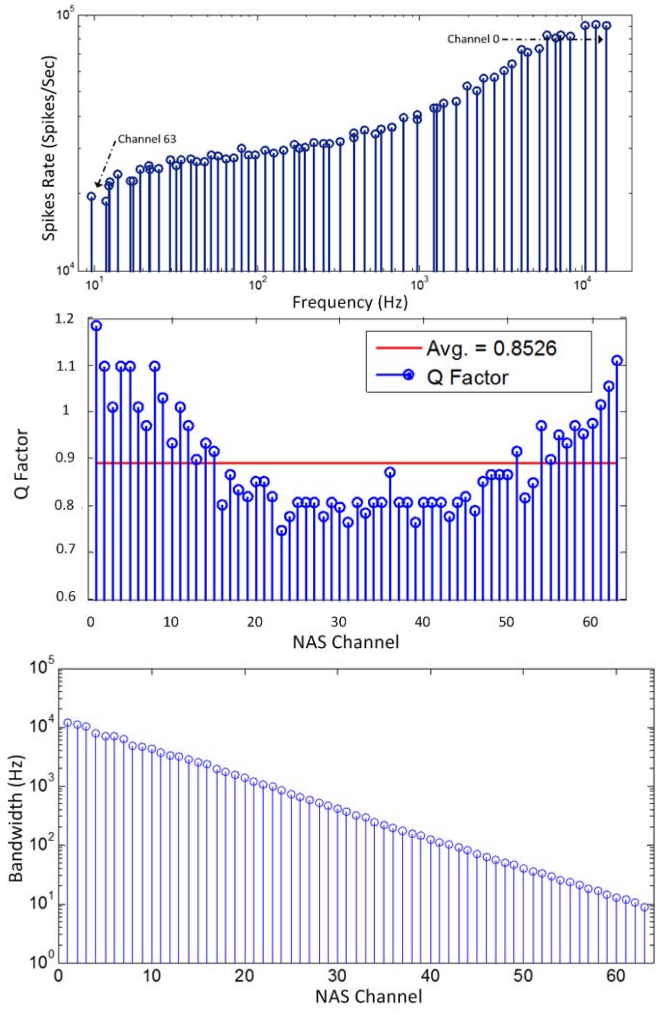


Fig. 16. Top: midfrequencies and maximum spike rate of a 64-channel NAS (1 V_{RMS} audio tone). Center: Q factor of a 64-channel NAS. Bottom: bandwidth of a 64-channel NAS.

midfrequency of 14.06 kHz, and the lowest frequency channel a midfrequency of 9.6 Hz, producing a global equivalent NAS bandwidth greater than 14 kHz. The spike rate of the CFB channels in their midfrequency decreases with the tone frequencies. The highest frequency CFB channel has a maximum spike rate of more than 90 kSpikes/s and the lowest frequency CFB channel has a maximum spike rate of around 20 kSpikes/s. This effect of decreasing CFB spike rates also impacts the Cascade architecture, where spike-based low-pass (SLPF) filters reject bands overlap and SLPF gain through the CFB sections progressively decreases.

The next parameter we studied was the Q factor of the CFB channel response. Fig. 16 (center) shows the Q factor of every CFB channel, where the x-axis represents the CFB channel number and the y-axis the Q factor. CFB channel Q factors vary from 1.2 to 0.75 with an average value of 0.85. Again, the Q factor changes from one channel to another due to the imperfection of the CFB tuning, a factor directly dependent on the distance between the consecutive SLPFs that make up the CFB. Finally, Fig. 16 (bottom) shows the bandwidth of every CFB channel, where the x-axis represents the CFB channel number and the y-axis the absolute bandwidth frequency. Fig. 16 (bottom) shows how bandwidth is distributed logarithmically from 10 kHz to 10 Hz. Bandwidths have been calculated from data represented in Fig. 15. For each channel, let us suppose the peak is 0 dB. If we draw a line at −3 dB for each bode diagram, the bandwidth is calculated by subtracting the high and low frequencies of the intersection.

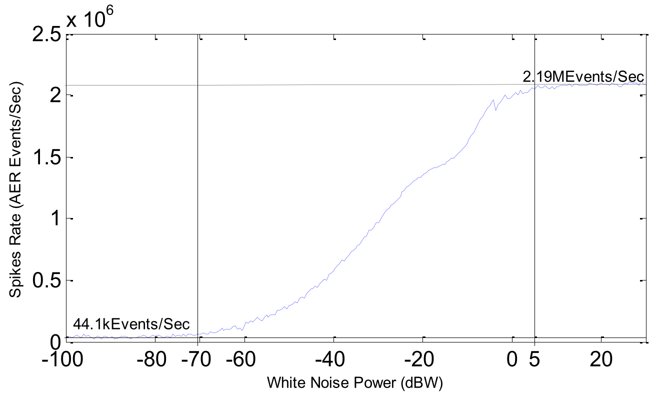


Fig. 17. 64-channel stereo NAS event rate for different levels of audio white noise.

C. Dynamic Range

The dynamic range (DR) of an NAS represents the range of audio power against the NAS sensitivity. The NAS was excited with white audio noise at different levels, and the AER event rate was then analyzed. Fig. 17 shows the experimental results. White audio noise power is shown in the x-axis, in correlation with the NAS's absolute AER event rate (y-axis). The NAS studied had an AER background activity (the absence of audio or low audio power) of around 44.1 kEvents/s. At an audio level of −70 dBW, AER activity gradually rose to +5 dBW, reaching an AER event rate saturation level of 2.19 MEvents/s. The DR, in terms of audio level, was +75 dBW for the couple AC'97 and NAS, while the AC'97 used claims +80 dBW.

VIII. Conclusion

Emergent SSP techniques offer the opportunity to design new neuromorphic systems that represent real alternatives to common digital systems. Despite the debate over which type of system—classic digital systems or neuromorphic systems—is better, neuromorphic systems are actually different from digital systems. They are based on different ideas, closer to biology than to traditional computing. In the audio context, traditional digital systems have to process several samples in a buffer, because sound makes sense along time, where fast Fourier transform calculation prior to specific processing. However, NAS provides audio directly and continuously decomposed into its frequency components as an AER events stream. This allows real-time event-by-event audio processing (without the need for buffering), using neuromorphic processing layers.

In this paper, we have presented the architecture and design flow for an NAS, which implements in the frequency domain a set of auditory filters whose outputs are inspired in the

TABLE V
SUMMARY OF NAS CHARACTERISTIC

Number of Channels	64x2 (adjustable)
Frequency range	9.6Hz- 14.06kHz (adjustable)
Dyn. range (AC'97+NAS)	75dB
Event Rate	2.19Mevents/sec
Power Consumption (NAS)	29.7mW
Slices requirements	11,141
System clock frequency	27MHz

functionality of the last stage of a biological cochlea, the inner hair cells. This response has been also obtained in other analog and digital cochlea emulated implementations [2], [18], [19]. The real time response is transferred from this auditory sensor using an AER interface. This NAS represents a parallel computational system, in that spikes flow between dedicated spike processing hardware units without sharing or multiplexing any computational elements. It is thus able to operate with low clock frequencies, 27 MHz in this case, and has low power consumption (below 30 mW) in the FPGA side (total platform power consumption has to include AC'97 one). Spike-based building blocks do not require dedicated resources, such as floating or fixed point multipliers. Their most complex operations are to increase/decrease one single register and to perform arithmetic comparisons. The cochleae presented in [22], [25], and [28] require 77, 92, and 690 slices/channels, respectively. The presented NAS requires 87 slices/channels. It should be noted that [25] and [28] include other functionality, such as automatic gain control of the outer hair cells.

Table V shows a summary of the NAS characteristics for comparison with previous implementations. As is shown in Table I, the DR of the NAS is at least 20 dB higher than that of earlier analog cochleae. This is possible thanks to use of a specific audio codec that provides a DR of +80 dB; however, in combination with NAS, this decreases until +75 dB. The NAS can be implemented for a wide variety of FPGAs and development boards at a low cost, making this technology ideal for aiding/improving research into neuromorphic audio processing systems. These systems have potential applications in real-time audio processing, and offer a better understanding of how biological auditory systems work. They could even constitute a new model for cochlear implants.

The presented NAS is able to represent the input layer of a spiking neuronal network, and can be used for a number of purposes, such as ecolocalization, noise robust speech recognition, person identification, audio classification, quality control, and so on. Neuromorphic audio processing offers new opportunities for exploring and implementing new algorithms for real time binaural ecolocalization, including interaural time differences and interaural-level differences. However, the main limitation of this architecture resides in the need for a digital audio codec capable of sampling the sound at a fixed rate. This is a critical issue with regard to the temporal accuracy of auditory information.

The tuning and synthesis of the NAS are automatic, making it possible to produce a full NAS from a set of specifications, while its flexibility and scalability allow us to build custom auditory systems with sets of features appropriate for specific FPGAs, and thus satisfy the requirements of specific applications. For testing and behavioral analysis of the NAS, a test scenario was built where a 2×64 channel stereo NAS was loaded and analyzed. The NAS analyzed displayed the expected behavior of a bank of spike-based bandpass filters, providing streams of spikes representing audio frequency components, and manifesting a DR of 75 dBW. The results showed that the gain in the NAS channels decreased from high to low frequency. To improve this aspect, we are now working to develop active cochleae models capable of dynamically adjusting filter features to model the behavior of outer hair cells [17].

Current research is focused on the development and implementation of applications for use in fields, such as speech recognition, auditory feature classification, person identification, quality control, and so on. As an example of a current development, [43] shows a convolutional spiking neural network for audio tone classification using a 64-channel stereo NAS as input stage, with a success percentage of 97.5%. These applications also address sensory fusion, where neuromorphic retina information is combined with NAS events to develop new real-time neuromorphic processing systems. For example, in [44], there is combined DVS128 retina [1] with an NAS to measure the speed of dc motor, with an accuracy of 94.33%.

REFERENCES

[1] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 $\times$ 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.

[2] S.-C. Liu, A. van Schaik, B. A. Mincti, and T. Delbruck, "Event-based 64-channel binaural silicon cochlea with Q enhancement mechanisms," in *Proc. IEEE Int. Symp. Circuits Syst.*, May/Jun. 2010, pp. 2027–2030.

[3] G. Indiveri, E. Chicca, and R. Douglas, "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity," *IEEE Trans. Neural Netw.*, vol. 17, no. 1, pp. 211–221, Jan. 2006.

[4] R. Serrano-Gotarredona *et al.*, "On real-time AER 2-D convolutions hardware for neuromorphic spike-based cortical processing," *IEEE Trans. Neural Netw.*, vol. 19, no. 7, pp. 1196–1219, Jul. 2008.

[5] P. Hafliger, "Adaptive WTA with an analog VLSI neuromorphic learning chip," *IEEE Trans. Neural Netw.*, vol. 18, no. 2, pp. 551–572, Mar. 2007.

[6] C. Liu, Q. Chen, and D. Wang, "CPG-inspired workspace trajectory generation and adaptive locomotion control for quadruped robots," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 41, no. 3, pp. 867–880, Jun. 2011.

[7] A. Linares-Barranco, F. Gomez-Rodriguez, A. Jimenez-Fernandez, T. Delbruck, and P. Lichtensteiner, "Using FPGA for visuo-motor control with a silicon retina and a humanoid robot," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 1192–1195.

[8] A. Jimenez-Fernandez, A. Linares-Barranco, R. Paz-Vicente, G. Jimenez-Moreno, and R. Berner, "Spike-based control monitoring and analysis with address event representation," in *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl.*, May 2009, pp. 900–906.

[9] T. Bekolay *et al.*, "Nengo: A python tool for building large-scale functional brain models," *Frontiers Neuroinformat.*, vol. 7, pp. 1–48, Jan. 2014.

[10] A. Jimenez-Fernandez, A. Linares-Barranco, R. Paz-Vicente, G. Jiménez, and A. Civit, "Building blocks for spikes signals processing," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2010, pp. 1–8.

[11] A. S. Cassidy *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *Proc. Int. Joint Conf. Neural Netw.*, Aug. 2013, pp. 1–10.

[12] M. Mahowald, "VLSI analogs of neuronal visual processing: A synthesis of form and function," Ph.D. dissertation, Dept. Comput. Neural Syst., California Inst. Technol., Pasadena, CA, USA, 1992.

[13] R. Lyon, "A computational model of filtering, detection, and compression in the cochlea," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, May 1982, pp. 1282–1285.

[14] A. F. Jahn and J. Santos-Sacchi, *Physiology of the Ear*, 2nd ed. San Diego, CA, USA: Singular, 2001.

[15] R. F. Lyon and C. Mead, "An analog electronic cochlea," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 36, no. 7, pp. 1119–1134, Jul. 1988.

[16] B. Wen and K. Boahen, "A silicon cochlea with active coupling," *IEEE Trans. Biomed. Circuits Syst.*, vol. 3, no. 6, pp. 444–455, Dec. 2009.

[17] T. J. Hamilton, C. Jin, A. van Schaik, and J. Tapson, "An active 2-D silicon cochlea," *IEEE Trans. Biomed. Circuits Syst.*, vol. 2, no. 1, pp. 30–43, Mar. 2008.

[18] J. Lazzaro and J. Wawrzynek, "A multi-sender asynchronous extension to the AER protocol," in *Proc. 16th Conf. Adv. Res. VLSI*, Mar. 1995, pp. 158–169.

[19] N. Kumar, W. Himmelbauer, G. Cauwenberghs, and A. G. Andreou, "An analog VLSI chip with asynchronous interface for auditory feature extraction," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 45, no. 5, pp. 600–606, May 1998.

[20] C. D. Summerfield and R. F. Lyon, "ASIC implementation of the Lyon cochlea model," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 5. Mar. 1992, pp. 673–676.

[21] A. Mishra and A. E. Hubbard, "A cochlear filter implemented with a field-programmable gate array," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 49, no. 1, pp. 54–60, Jan. 2002.

[22] M. P. Leong, C. T. Jin, and P. H. W. Leong, "An FPGA-based electronic cochlea," *EURASIP J. Appl. Signal Process.*, vol. 7, pp. 629–638 Jan. 2003.

[23] C. K. Wong and P. H. W. Leong, "An FPGA-based electronic cochlea with dual fixed-point arithmetic," in *Proc. Int. Conf. Field Program. Logic Appl.*, Aug. 2006, pp. 1–6.

[24] F. Gomez-Rodriguez *et al.*, "AER auditory filtering and CPG for robot control," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 1201–1204.

[25] R. V. Dundur, M. V. Latte, S. Y. Kulkarni, and M. K. Venkatesha, "Digital filter for cochlear implant implemented on a field-programmable gate array," *Proc. World Acad. Sci., Eng. Technol.*, vol. 33, pp. 468–472, Sep. 2008.

[26] I. Gambin, I. Grech, O. Casha, E. Gatt, and J. Micallef, "Digital cochlea model implementation using Xilinx XC3S500E spartan-3E FPGA," in *Proc. 17th IEEE Int. Conf. Electron., Circuits, Syst.*, Dec. 2010, pp. 946–949.

[27] C. Mugliette, I. Grech, O. Casha, E. Gatt, and J. Micallef, "FPGA active digital cochlea model," in *Proc. 18th IEEE Int. Conf. Electron., Circuits Syst.*, Dec. 2011, pp. 699–702.

[28] C. S. Thakur, T. J. Hamilton, J. Tapson, A. van Schaik, and R. F. Lyon, "FPGA implementation of the CAR model of the cochlea," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 2014, pp. 1853–1856.

[29] M. Domínguez-Morales, A. Jimenez-Fernandez, E. Cerezuela-Escudero, R. Paz-Vicente, A. Linares-Barranco, and G. Jimenez, "On the designing of spikes band-pass filters for FPGA," in *Proc. 21st Int. Conf. Artif. Neural Netw.*, 2011, pp. 389–396.

[30] F. Gomez-Rodriguez, R. Paz, L. Miro, A. Linares-Barranco, G. Jimenez, and A. Civit, "Two hardware implementations of the exhaustive synthetic AER generation method," in *Proc. 8th Int. Work-Conf. Artif. Neural Netw.*, 2005, pp. 534–540.

[31] E. Cerezuela-Escudero, M. J. Dominguez-Morales, A. Jiménez-Fernández, R. Paz-Vicente, A. Linares-Barranco, and G. Jiménez-Moreno, "Spikes monitors for FPGAs, an experimental comparative study," in *Proc. 12th Int. Work-Conf. Artif. Neural Netw.*, 2013, pp. 179–188.

[32] A. Jimenez-Fernandez, G. Jimenez-Moreno, A. Linares-Barranco, M. J. Dominguez-Morales, R. Paz-Vicente, and A. Civit-Balcells, "A neuro-inspired spike-based PID motor controller for multi-motor robots with low cost FPGAs," *Sensors*, vol. 12, no. 4, pp. 3831–3856, Mar. 2012.

[33] F. Perez-Peña *et al.*, "Neuro-inspired spike-based motion: From dynamic vision sensor to robot motor open-loop control through spike-VITE," *Sensors*, vol. 13, no. 11, pp. 15805–15832, Nov. 2013.

[34] A. Linares-Barranco, G. Jimenez-Moreno, B. Linares-Barranco, and A. Civit-Balcells, "On algorithmic rate-coded AER generation," *IEEE Trans. Neural Netw.*, vol. 17, no. 3, pp. 771–788, May 2006.

[35] R. Paz-Vicente, A. Linares-Barranco, A. Jimenez-Fernandez, G. Jimenez-Moreno, and A. Civit-Balcells, "Synthetic retina for AER systems development," in *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl.*, May 2009, pp. 907–912.

[36] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley, 1989.

[37] D. Başkent and B. Edwards, "Simulating listener errors in using genetic algorithms for perceptual optimization," *J. Acoust. Soc. Amer.*, vol. 121, no. 6, pp. EL238–EL243, May 2007.

[38] *Xilinx Virtex-5 FXT FPGA ML507 Evaluation Platform*, accessed on Jul. 18, 2016. [Online]. Available: http://www.xilinx.com/products/boards/ml507/docs.htm

[39] R. Berner, T. Delbruck, A. Civit-Balcells, and A. Linares-Barranco, "A 5 Meps $100 USB2.0 address-event monitor-sequencer interface," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 2451–2454.

[40] *jAER Open-Source Software Project*, accessed on Jul. 18, 2016. [Online]. Available: https://sourceforge.net/projects/jaer/

[41] R. Serrano-Gotarredona *et al.*, "CAVIAR: A 45k neuron, 5M synapse, 12G connects/s AER hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking," *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1417–1438, Sep. 2009.

[42] R. Sarpeshkar, M. W. Baker, C. D. Salthouse, J.-J. Sit, L. Turicchia, and S. M. Zhak, "An analog bionic ear processor with zero-crossing detection," in *IEEE Int. Dig. Tech. Papers. Solid-State Circuits Conf.*, vol. 1. Feb. 2005, pp. 78–79.

[43] E. Cerezuela-Escudero, A. Jimenez-Fernandez, R. Paz-Vicente, M. Dominguez-Morales, A. Linares-Barranco, and G. Jimenez-Moreno, "Musical notes classification with neuromorphic auditory system using FPGA and a convolutional spiking network," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2015, pp. 1–7.

[44] A. Rios-Navarro, E. Cerezuela-Escudero, M. Dominguez-Morales, A. Jimenez-Fernandez, G. Jimenez-Moreno, and A. Linares-Barranco, "Real-time motor rotation frequency detection with event-based visual and spike-based auditory AER sensory integration for FPGA," in *Proc. Int. Conf. Event-Based Control, Commun., Signal Process.*, Jun. 2015, pp. 1–6.