Inter-spike-intervals analysis of AER Poisson-like generator hardware

A. Linares-Barranco[a,*], M. Oster[b], D. Cascado[a], G. Jiménez[a], A. Civit[a], B. Linares-Barranco[c]

[a] Arquitectura y Tecnología de Computadores. Universidad de Sevilla, Av. Reina Mercedes s/n, 41012 Sevilla, Spain

[b] Institute of Neuroinformatics. UNI—ETH Zürich, Winterthurerstrasse 190, CH 8057 Zurich, Switzerland

[c] Instituto de Microelectrónica de Sevilla. CSIC, Av. Reina Mercedes s/n. Edificio CICA. 41012 Sevilla, Spain

Abstract

Address–Event–Representation (AER) is a communication protocol for transferring images between chips, originally developed for bio-inspired image-processing systems. Such systems may consist of a complicated hierarchical structure with many chips that transmit images among them in real time, while performing some processing (for example, convolutions). In developing AER-based systems it is very convenient to have available some means of generating AER streams from on-computer stored images. Rank order coding (ROC) and Poisson rate coding are the extremes of spikes coding. In this paper, we present a pseudo-random hardware method for generating AER streams in real time from a sequence of images stored in a computer's memory. The Kolmogorov–Smirnov test has been applied to quantify that this method follows a Poisson distribution of the spikes. A USB–AER board, developed by our RTCAR group, have been used for the measurements. An example scenario of use under the EU CAVIAR project is presented.

Keywords: Bio-inspired; Neuro-inspired; AER; LFSR; Poisson; FPGA; USB

1. Introduction

Primate brains are structured in layers of neurons, in which the neurons in a layer connect to a very large number ($\sim 10^4$) of neurons in the following layer [25]. Many times the connectivity includes paths between non-consecutive layers, and even feedback connections are present. Artificial bio-inspired software models based on such connectivity models have overwhelmed the specialized literature presenting many ways of performing bio-inspired processing systems that outperform more conventionally engineered machines [12,5]. Since these models are software based, they operate at extremely low speeds, because of the massive connectivity they emulate. For real-time solutions direct hardware implementations are required. However, hardware engineers face a very strong barrier when trying to mimic the bio-inspired hierarchically layered structure: the massive connectivity. In present day state-of-the-art very large-scale integrated (VLSI) circuit technologies it is plausible to fabricate on a single chip many thousands (even millions) of artificial neurons or simple processing cells. However, it is not viable to connect physically each of them to even a few hundreds of other neurons. The problem is greater for multi-chip multi-layer hierarchically structured bio-inspired systems. Address–Event–Representation (AER) is an incipient bio-inspired spike-based technique capable of providing a hardware solution to the inter-chip massive connectivity problem.

AER was proposed in 1991 by Sivilotti [26] for transferring the state of an array of analog time-dependent values from one chip to another. It uses mixed analog and digital principles and exploits pulse density modulation for coding information. Fig. 1 explains the principle behind the AER basics. The Emitter chip contains an array of cells (like, for example, a camera or artificial retina chip) where each pixel shows a continuously varying time-dependent state that it changes with a slow time constant (in the order of ms). Each cell or pixel includes a local oscillator (VCO) that generates digital pulses of minimum width (a few nano-secconds). The density of pulses is proportional to the state of the pixel (or pixel intensity). Each time a pixel generates a pulse (which is called "Event"), it communicates to the array periphery and a digital word

*Corresponding author. Tel.: +34 954 556 145; fax: +34 954 552 899.
E-mail address: alinares@atc.us.es (A. Linares-Barranco).

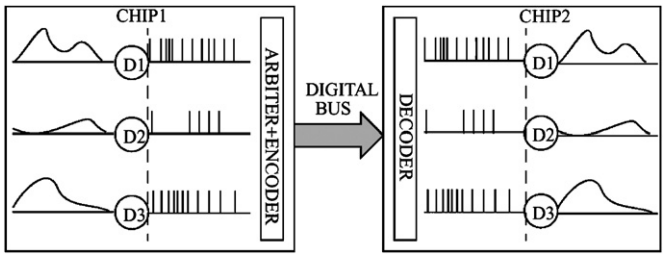

Fig. 1. Illustration of AER inter-chip communication scheme.

representing a code or address for that pixel is placed on the external inter-chip digital bus (the AER bus). Additional handshaking lines (Acknowledge and Request) are also used for completing the asynchronous communication. The inter-chip AER bus operates at the maximum possible speed because of its asynchronous nature. In the receiver chip, the pulses are directed to the pixels whose code or address was on the bus. This way, pixels with the same code or address in the emitter and receiver chips will "see" the same pulse stream. The receiver pixel integrates the pulses and reconstructs the original low-frequency continuous-time waveform. Pixels that are more active access the bus more frequently than those less active.

Transmitting the pixel addresses allows performing extra operations on the images while they travel from one chip to another. For example, inserting properly coded EEPROMs allows shifting and rotation of images. Also, the image transmitted by one chip can be received by many receiver chips in parallel, by properly handling the asynchronous communication protocol. The peculiar nature of the AER protocol also allows for very efficient convolution operations within a receiver chip [28].

There is a growing community of AER protocol users for bio-inspired applications in vision and audition systems, as demonstrated by the success in the last years of the AER group at the Neuromorphic Engineering Workshop series [4]. The goal of this community is to build large multi-chip and multi-layer hierarchically structured systems capable of performing complicated array data processing in real time. The success of such systems will strongly depend on the availability of robust and efficient development and debugging AER tools. One such tool is a computer interface that allows not only reading an AER stream into a computer and displaying it on its screen in real time, but also the opposite: from images available in the computer's memory, generate a synthetic AER stream in a similar manner as would do a dedicated VLSI AER emitter chip [2,17,16].

In the following sections we present and compare three hardware implementations of one of the already existing methods for synthetic AER generation (the random method) [14], by evaluating the nature of the distribution of the events with respect to their inter-spike-intervals (ISIs). For this analysis we have used a hardware USB–AER interface developed by RTCAR group.

2. Synthetic AER generation by hardware

One can think of many software algorithms that would transform a bitmap image into an AER stream of pixel addresses [17]. At the end, the frequency of appearance of the address of a given pixel must be proportional to the intensity of that pixel.

The precise location of the address pulses should not be critical. The pulses can be slightly shifted from their nominal positions because the AER receivers will integrate them to recover the original pixel waveform.

However there are coding schemes where the precise location of the address in time is critical, like Rank-order coding proposed by Thorpe et al. [8,9], but in another spike-timing-based computation it is not, as the Poisson distribution [10], or rate coding [14].

From the software methods already proposed [14], we present the hardware implementation in VHDL of the random method. Some modifications have been made to reduce the resources needed in the FPGA.

This method sends a sequence of addresses to the AER bus without any buffer. Next event to be sent is selected in real time from an image composed by $N \times N$ pixels, with N a power of 2, and each pixel can have up to k gray levels, with $k = 255$ in this implementation. Each event needs a time to be sent. Let us call it time slot. The algorithm would implement a particular way of selecting the next address to be transmitted, through the AER bus, in the current time slot.

At each time slot an event could be placed in the AER bus, or not. This method uses a linear feedback shift register (LFSR) [20,15] for selecting the pixel of the image in charge of sending an event, and also to decide if the event is going to be sent or not. The LFSR has a resolution of $\log(N \times N \times k)$, so the random number obtained for each time slot is divided into:

1. an address for selecting a pixel of the image; and
2. a gray level value.

Thanks to the LFSR, each gray-level value of each pixel is generated only once. If a pixel in the image has a value p, then the method will validate p events along time and will send those p *events*, from the k possible, for this pixel. They will not be perfectly equidistant in time, but in average they will be reasonably well spaced. This method has two fundamental characteristics:

- It is relatively simple to be implemented in hardware.
- The use of the LFSR for generating the pixel and the decision to send or not the event give to this method a random behavior, what implies a Poisson-like distribution of events along time.

Next sections explain in more details the implementation issues for this method with three variations to improve the

Poisson behavior. An example scenario of use is presented and evaluated.

3. Random method

This method is an implementation of LFSR-based random number generators. LFSR random number generators are based on a linear recurrence of the form

$$x_n = (a_1 x_{n-1} + \ldots + a_k x_{n-k}) \bmod 2, \quad (1)$$

where $k>1$ is the order of the recurrence, $a_k = 1$, and $a_j \in \{0,1\}$ for each j. This recurrence is always purely periodic and the period length of its longest cycle is 2^k-1 if and only if its characteristic polynomial

$$P(z) = -\sum_{i=0}^{k} a_i z^{k-i} \quad (2)$$

is a primitive polynomial over the Galois field with two elements [22].

With these premises and limiting the maximum number of address events necessary to transmit an image, we know the number of bits needed for the LFSR and the primitive polynomial. For this implementation, the limit corresponds to a 64×64 image of 256 gray levels, which implies a 20-bit LFSR.

The characteristics polynomial $P(z)$ used for 20 bits is

$$P(z) = z^{20} + z^{17} + 1 \quad (3)$$

which corresponds to the LFSR of Fig. 2 [1].

One possible implementation could have all bits set to '1' after a reset, which is the seed of the random numbers generator. This way, the 20-bit numbers are divided into two parts: the gray level corresponds with the 8 more significant bits, and the address of the pixel in the image is represented by the other 12 bits. The method works as follows:

- For each time slot, the LFSR generates a 20-bit random number.
- The 12 less-significant bits are used to define the address of the pixel.
- Once addressed that pixel, its gray level is compared with the 8 more significant bits of the LFSR.
- If the gray level of the pixel is greater or equal than the 8 MSB of the LFSR, an event is transmitted with the 12 LSB of the LFSR as the address.
- In the other case, no event is produced for this time slot.

The LFSR ensures that each possible event of each pixel is obtained from the LFSR only once per each $(2^{20}-1)$ time slots.

To improve the Poisson distribution we have considered three variations on the LFSR. The comparisons are presented in Section 4.

This 20-bit LFSR will generate $2^{20}-1$ numbers in a random way. This can be called the period of the LFSR. The next $2^{20}-1$ numbers will be generated in the same order than in the previous period. As these numbers covers all the events for one image, if the gray level of the pixel is low (for example with gray level 1), then the time between consecutive events for this pixel will be always the same, and equal to the period of the LFSR. This implies a no Poisson-like distribution, but a uniform distribution. So the behavior of the method with the LFSR presented above seems to follow a uniform distribution for low gray-level pixels and a Poisson distribution for high gray levels. We will probe this with the Kolmogorov–Smirnov (K–S) test in Section 4.

There are two parameters in the LFSR, the seed, and the length. In the next paragraphs we present two variations attending to the seed, and one for the length:

A. LFSR with an incremental seed. In this case, the seed of the LFSR is changed for every $2^{20}-1$ numbers generated, and the seed is changed attending to a counter.
B. LFSR with a bitwise incremental seed. Now the seed is changed attending to a decremental counter. The counter output is taken in the opposite order for the seed: the less significant bit of the counter is the most significant bit of the seed and vice-versa.
C. Twenty-eight-bit LFSR. To increase the period of the LFSR we propose to increase the number of bits. The 8-MSB are used for the gray level comparator, the 12-LSB are used for the pixel selection, and the other 8 bits are not used.

Fig. 3 shows the block diagram of the hardware implementation of the random method for the USB–AER FPGA. The FPGA receives a bitmap from the PC through the USB port of the board. The image is stored into an internal memory implemented into the FPGA. The Control unit then obtain a number from the LFSR, uses 12 bits to address the memory and read a pixel, and it uses another 8 bits from the LFSR to compare with the pixel value and decide if an event is sent or not. Events to be sent are stored

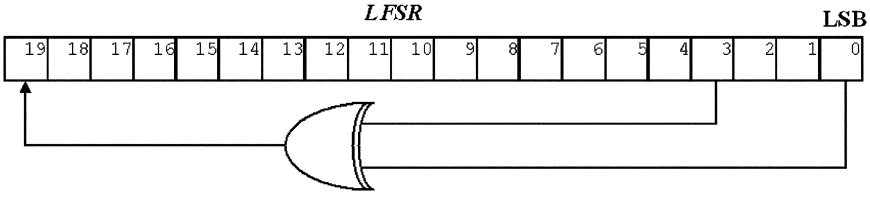

Fig. 2. Linear feedback shift register for random synthetic AER generation.

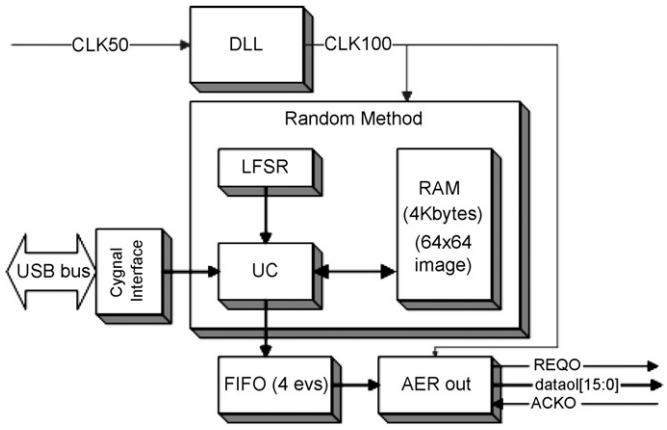

Fig. 3. Hardware implementation of random method.

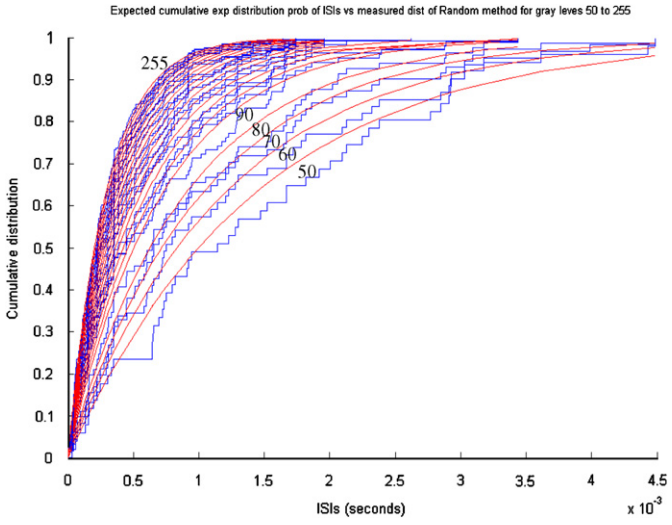

Fig. 4. Expected cumulative exponential ISIs distribution versus measured ISIs distribution generated by the random method for gray levels.

in the FIFO and the AER out state machine send them to the AER bus. All the system works with a 100 MHz clock obtained internally from the 50 MHz input clock using an internal DLL of the FPGA.

4. Inter-spike-intervals distribution analysis

In this section we will compare the ISIs of time of this hardware synthetic AER generation method with the normalized distribution that it should have (a Poisson distribution), using the K–S statistical test.

In neuro-inspired systems, signals can often be modelled by a Poisson distribution [7,21]. A Poisson distribution can be described by the following formula [3]:

$$P_n(T) = \frac{(\lambda T)^n}{n!} e^{-\lambda T}, \tag{4}$$

where P is the probability of having n events time interval T. The distribution of ISIs is the probability that no event

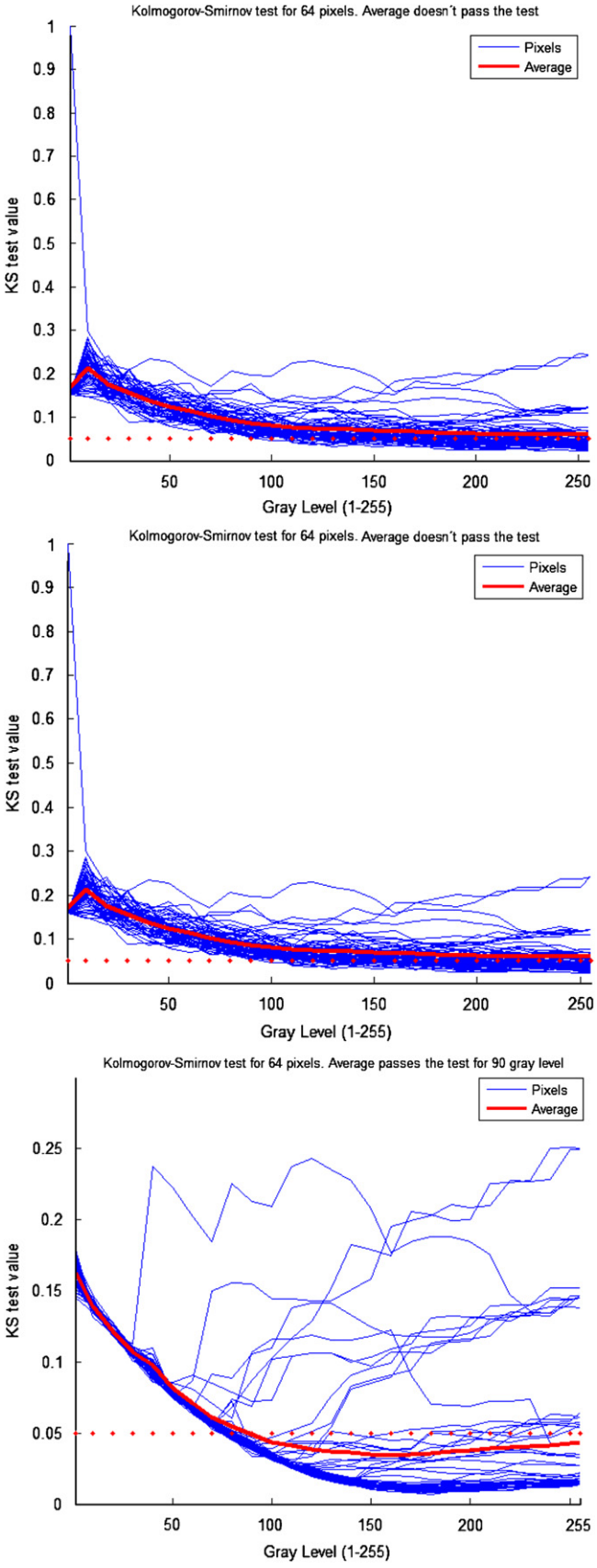

Fig. 5. K–S test results for random method distribution A (first), B (middle) and C (last).

occurs in the interval

$$P_0(T) = e^{-\lambda T} \quad (5)$$

which is the exponential distribution.

A USB–AER board with a Spartan II 200 FPGA [19] has been used for the first random method implementation (without improvements), using images with all pixels to zero, except one, with different gray values. Another USB–AER board, configured as a datalogger [19] that captures events and their timestamp, controlled through MATLAB, has been also used to capture the ISIs. Fig. 4 shows the cumulative probabilistic distribution of ISIs for the events of the pixel (10,10) of one image of 64×64 pixels. It is shown the expected exponential distribution (continuous line) versus the measured distribution generated by the random method (stair line) (with 20-bit LFSR with any seed change), for one pixel and ISIs for gray levels 50–255, 10×10. As can be seen, for high gray levels, the theoretical and observed distributions are close to each other, which imply that the spike train generated with the random method is close to the Poisson distribution.

The K–S test quantifies the difference between two distributions. We have applied this test to compare how good the observed distribution of ISIs follows the theoretical exponential distribution. If the test result is under the 5% one can affirm that the distributions are practically equal. For gray levels greater that 140, the 20-LFSR random method is Poisson like.

As a gray level of 140 is very high, several alternatives have been proposed (described in Section 3) to decrease this frontier.

Fig. 5 shows the result of applying the K–S test to the Random distribution obtained for the three implementations A, B and C presented. In this case the 64 pixels of the diagonal of the 64×64 image have been analyzed. It is shown the K–S test of the 64 pixels, the average (bold line) and the 5% frontier (dotted line).

These 64 pixels do not have the same behavior. This is due to the LFSR. For each pixel address, the sequence of numbers that the LFSR generates is different. For this reason we have calculated also the average. One can see that in average only implementation C passes the K–S-test for gray levels greater than 90.

It is shown that for small gray levels implementations A and B are not much Poisson like. The K–S test is not passed for the average curve, but is passed for some pixels. As the LFSR has a period in which the sequence is repetitive, there are few differences on ISIs for a pixel with low gray level. Furthermore, one can observe that there are soft differences by changing the seed algorithm. This is due to the property of the seed in the LFSR. The seed will change the start point of the $2^{20}-1$ sequence of random numbers, but not the sequence. We address this problem by increasing the number of bits in the LFSR (C), what implies a longer period in the LFSR (256 frames longer in this case). The K–S test has the best result for implementation C, where the average passes the test for a gray level of 90, and there are pixels that pass the test with a 70 gray level.

5. CAVIAR scenario

Complex systems developed by Neuromorphic Engineers require interfaces to interconnect them and to connect them to PCs for debugging and/or other purposes. This milestone was the start point for the development of a set of AER tools under the European Project CAVIAR. We are four different partners working together in the design of a neuromorphic vision system totally based on AER. CAVIAR has connected the biggest AER chain at the moment [24]. This chain is composed by a 64×64 retina that spikes with temporal and contrast changes [13], two convolution chips to detect a ball at different distances from the retina [23], an object chip to filter the convolution activity [18] and a learning stage composed by two chips: delay line and learning [11]. To make all this vision system possible, a set of AER tools for debugging and interconnection [19] purposes are not only useful, but also necessary.

Fig. 6 shows the AER system mounted under CAVIAR. In this chain, we have tested our random method generator (implementation C) in the following ways:

Experiment 1. Using the framegrabber firmware of the USB–AER board, we have connected the AER output of the retina to the USB–AER board. The board is able to integrate events during a period of 50 ms. We have recorded a sequence of frames (a video) of the retina. The stimulus of the retina was a black circle over a white sheet and an irregular black square rotating at a low speed. Then, the retina chip was replaced by a USB–AER configured with the random method AER generator

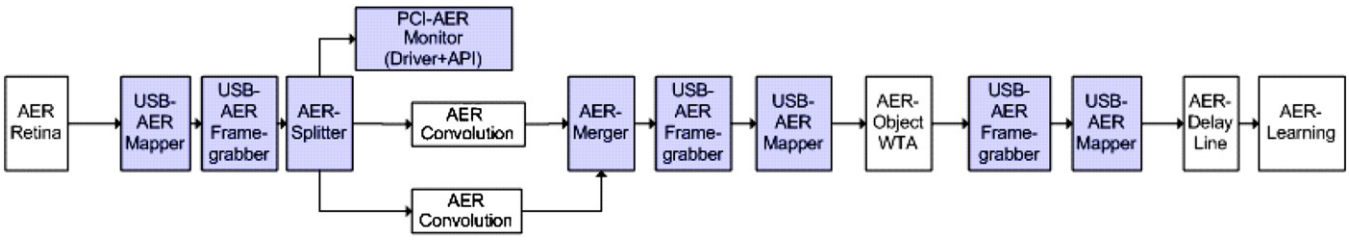

Fig. 6. CAVIAR scenario.

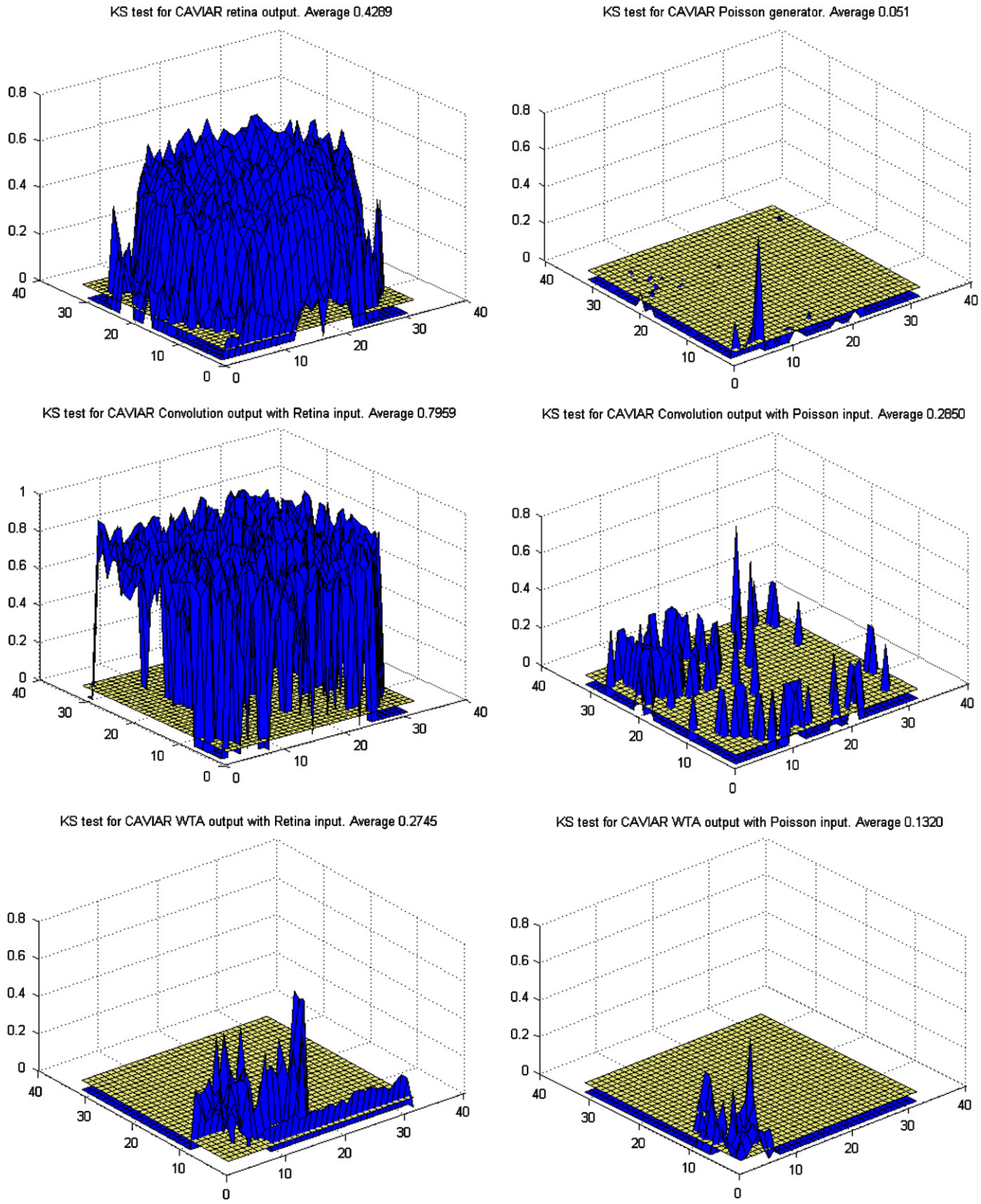

Fig. 7. Kolmogorov–Smirnov test results. Left column: retina events. Right column: Poison generator events. Second row: convolution chip output (experiment 1). Third row: WTA chip output (experiment 2).

Table 1
KS test average results for the CAVIAR convolution and WTA chips stimulated by Tmpdiff retina and random method generator

	K–S retina/ generator output	K–S convolution chip output	K–S WTA chip output
Retina	0.43	0.80	0.27
Random method generator	0.05	0.29	0.13

(implementation C). We logged the sequence of events at the output of the generator and at the output of the convolution chip. Fig. 7 shows the K–S test results for the sequences of events of each pixel of the 64×64 frame (dark figure) and the 5% plane frontier of the test (light curve). The K–S test has been applied to the output of the chips using the retina sequence of events as the input, and using the output of the random method. Table 1 shows the average values of the K–S test. It can be seen that the output of neither the retina, nor the convolution chip are Poisson like. This is due to the design of the chips that are based on the integrate and fire neuron. It is stated that this model of neuron cannot produce a Poisson-like output distribution [27], although the input is Poisson-like.

Experiment 2. We have recorded the sequence of events of the output of the convolution chip with the same input stimulus of the retina using a USB–AER board configured as a framegrabber. Then we have replaced the retina and the convolution chip with the random AER generator 28-LFSR (implementation C). Then we have logged the output of the WTA chip. Fig. 7 shows the K–S test results for each pixel of the 32×32-output images. As can be seen the WTA chip do not produce an output with a Poisson distribution, because the chip is also based on the IF neuron.

Although any output is Poisson like, if we use the random method to stimulate the chips, the output produced is more Poisson like than if we use the retina.

6. Conclusions

We have presented three hardware implementation of the random method for synthetic AER generation. We have demonstrated that the output distribution of events follows a Poisson distribution quite well. Therefore, it could be a very realistic method to be used for the output stage of a neuro-inspired design.

We have demonstrated that for low frequency of events, this method will need more bits in the LFSR implementation to pass the K–S test for the Poisson distribution.

In contrast to already existing AER generators like [6], this USB–AER board with the random AER generator firmware directly generates Poisson spike trains instead of sequencing raw spike trains.

The experiment with the CAVIAR scenario reinforces the Softky and Koch [27] hypothesis that says that integrate-and-fire (IF) neurons require a Random number generator inside the neuron to produce a Poisson distribution. Both, the convolution chip and the WTA object chip, developed under CAVIAR have been designed around the IF VLSI neuron model, with no random number generator inside. Therefore, although one stimulates those chips with a Poisson distribution, they do not reproduce it, as shown in Fig. 7.

Using an implementation of this method at the output stage of an AER chip, the output will warranty a Poisson distribution. This Poisson distribution could be improved by increasing the bits of the LFSR that generates the output random sequence of events.

The USB–AER board running with a syntethic AER generator, the datalogger, and with the MATLAB interface, is a useful tool for testing and debugging AER systems.

Acknowledgements

This work was in part supported by EU Grant IST-2001-34124 (CAVIAR), and Spanish Grant TIC-2003-08164-C03-02 (SAMANTA).

We want to give special thanks to Manuel Rivas for the development of the Datalogger and Switch-AER board and firmware.

References

[1] Alfke, P. Efficient Shift Register, LFSR Counters and Long Pseudo-Random Sequence Generators, Application Note, Xilinx, 1995.
[2] K.A. Boahen, Communicating Neuronal Ensembles between Neuromorphic Chips. Neuromorphic Systems, Kluwer Academic Publishers, Boston, 1998.
[3] J.R. Cogdell, Modeling Random Systems, Pearson Prentice-Hall, 2004.
[4] A. Cohen, R. Douglas, C. Koch, T. Sejnowski, S. Shamma, T. Horiuchi, G. Indiveri, Report to the National Science Foundation, in: Workshop on Neuromorphic Engineering, Telluride, CO, USA, June–July 2001. 〈www.ini.unizh.ch/telluride〉.
[5] T. Crimmins, Geometric filter for speckle reduction, Appl. Opt. 24 (1985) 1438–1443.
[6] V. Dante, P. Del Giudice, A.M. Whatley, PCI–AER hardware and software for interfacing to address-event based neuromorphic systems, Neuromorph. Eng. 2 (1) (2005) 5–6.
[7] P. Dayan, L. Abbot, Theoretical Neuroscience, MIT Press, Cambridge, MA, 2001.
[8] A. Delorme, J. Gautrais, R.R. van, S. Thorpe, SpikeNET: a simulator for modeling large networks of integrate-and-fire neurons, Neurocomputing 26–27 (1999) 989–996.
[9] A. Delorme, L. Perrinet, S.J. Thorpe, Networks of integrate-and-fire neurons using rank order coding B: spike timing dependent plasticity and emergence of orientation selectivity, Neurocomputing 38–40 (2001) 539–545.
[10] D. George, F.T. Sommer, Computing with inter-spike interval codes in networks of integrate and fire neurons, Neurocomputing 65–66 (2005) 415–420.
[11] H. Kolle Riis, P. Haefliger, Spike based learning with weak multi-level static memory, in: ISCAS'04, vol. 5, Vancouver, Canada, May 2004, pp. 393–395.

[12] J. Lee, A simple speckle smoothing algorithm for synthetic aperture radar images, IEEE Trans. Syst. Man Cybernet. SMC-13 (1983) 85–89.
[13] P. Lichtsteiner, T. Delbruck, 64×64 Event-driven logarithmic temporal derivative silicon retina, in: 2005 IEEE Workshop on Charge Coupled Devices and Advanced Image Sensors, Nagano, Japan, June, 2005.
[14] A. Linares-Barranco, G. Jimenez-Moreno, B. Linares-Barranco, A. Civit Balcells. On algorithmic rate-coded AER generation. Trans. Neural Network May (2006).
[15] Linear Feedback Shift Register V2.0. Xilinx Inc. 4 October 2001. ⟨http://www.xilinx.com/ipcenter⟩.
[16] M. Mahowald. VLSI analogs of neuronal visual processing: a synthesis of form and function, Ph.D. Thesis, California Institute of Technology Pasadena, CA, 1992.
[17] A. Mortara, E.A. Vittoz, P. Venier, A communication scheme for analog VLSI perceptive systems, IEEE J. Solid-State Circuits 30 (6) (1995) 660–669.
[18] M. Oster, S.C. Liu, A winner-take-all spiking network with spiking inputs, in: ICECS 2004, Tel Aviv, 2004.
[19] R. Paz, F. Gomez-Rodriguez, M.A. Rodriguez, A. Linares-Barranco, G. Jimenez, A. Civit, Test Infrastructure for Address–Event–Representation Communications, IWANN Lecture Notes on Computer Sciences, vol. 3512, Springer, Berlin, 2005, pp. 518–526.
[20] L'Ecuyer Pierre, F. Panneton, A New class of linear feedback shift register generators, in: Proceedings of the 2000 Winter Simulation Conference.
[21] F. Rieke, D. Worland, R. de Ruyter van Steveninck, W. Bialek, Spikes: Exploring the Neural Code, The MIT Press, Cambridge, 1999.
[22] Bruce Schneier, Applied Cryptography, Wiley, New York, 1996.
[23] R. Serrano, T. Serrano-Gotarredona, A.J. Acosta, B. Linares-Barranco, An arbitrary kernel convolution AER-Transceiver, in: ISCAS'06, Kos, Greece, May 2006.
[24] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, H. Kolle Riis, T. Delbrück, S.C. Liu, P. Häfliger, G. Jimenez-Moreno, A. Civit, T. Serrano-Gotarredona, A. Acosta-Jiménez, B. Linares-Barranco. AER Building blocks for multi-layer multi-chip neuromorphic vision systems, in: NIPS'05, Vancouver, December 2005.
[25] G.M. Shepherd, The Synaptic Organization of the Brain, third ed, Oxford University Press, Oxford, 1990.
[26] M. Sivilotti, Wiring considerations in analog VLSI systems with application to field-programmable networks, Ph.D. Thesis, California Institute of Technology, Pasadena, CA, 1991.
[27] W.R. Softky, C. Koch, The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs, J Neurosci. 13 (1) (1993) 334–350.
[28] T. Serrano-Gotarredona, A.G. Andreou, B. Linares-Barranco, AER image filtering architecture for vision-processing systems, IEEE Trans. Circuits Syst. Fund. Theory Appl. 46(9) 1999.