Synthetic Generation of Address-Events for Neurons Multiplexed Communications Emulation

A. Linares-Barranco, R. Senhadji, I. García-Vargas, F. Gómez, A. Civit, and G. Jiménez.
Arquitectura y Tecnología de Computadores. Universidad de Sevilla.
Av. Reina Mercedes s/n, 41012-Sevilla, SPAIN
{alinares, raouf, ignacio, gomezroz, gaji, civit}@atc.us.es
http://icaro.eii.us.es/

Abstract: Address-Event-Representation (AER) is a communication protocol that emulates the nervous system's neurons communication, and it is typically used for transferring images between chips. It was 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 (e.g. convolutions). In this paper we present several methods for generating AER streams in real time from images stored in a computer's memory. They are analysed, tested by software and compared.

1. Introduction

Address-Event-Representation (AER) was proposed in 1991 by Sivilotti [1] 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 changes with a slow time constant (in the order of magnitude of mili-secconds). 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 representing a code or address for that pixel is placed on the external inter-chip digital bus (the AER bus). Additional handshake lines (Acknowledge and Request) are also used for completing the asynchronous communication. The inter-chip AER bus operates at the maximum possible speed. In the receiver chip the pulses are directed, in the simplest case, 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, like a neuron does in the nervous system. Pixels that are more active access the bus more frequently than those less active.

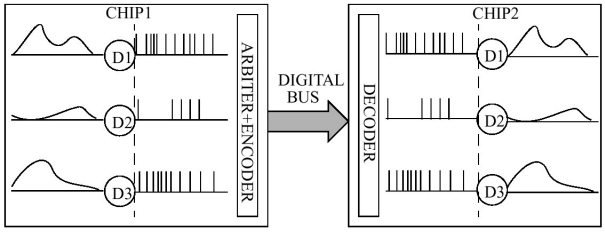


Fig. 1: Illustration of AER inter-chip communication scheme

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 [2].

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 [3]. 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: generate a synthetic AER stream from images available in the computer's memory, in a similar manner as would do a dedicated VLSI AER emitter chip [4-6].

2. Synthetic AER Stream Generation

One can think of many software algorithms that would transform a bitmap image into an AER stream of pixel addresses. At the end, the frequency of appearance of the address of a given pixel must be proportional to the intensity of that pixel. Note that the precise location of the address pulses is not critical. The pulses can be slightly shifted from their nominal positions because the AER receivers will integrate them to recover the original pixel waveform.

Whatever algorithm is used, it will generate a vector of addresses that will be sent to an AER receiver chip via an AER bus. If we have an image of *NxM* pixels and each pixel can have a grey level value from 0 to *K*, one possibility would be to place each pixel address in the address sequence as many times as the value of its intensity, and distributed in equidistant position. In the worst case (all pixels with value *K*), the address sequence would be filled with *NxMxK* addresses. If the images come from a traditional digital media, we have a certain number of frames per second, thus, the time used for transmitting an image needs to be the same for any image intensity charge, leaving blank slots if non-event has to be sent. Each algorithm would

implement a particular way of distributing these address events. Let us propose five different algorithms: the Uniform method, the Scan method, the Random method, the Random-Square method and the Exhuastive method.

2.1 The Scan method

In this method the image is scanned many times. For each scan, every time a non-zero pixel is reached its address is put on the address sequence in the first available slot, and the pixel value is decremented by one. This method is very fast, because it does not need to look for empty slots, although the image needs to be scanned many times (K times in the worst case). However, a non-well behaved event distribution is obtanied due to all pulses of the pixels with low intensity values will appear only at the begining of the sequence.

2.2 The Uniform method

In this method the image is scanned pixel by pixel one time. For each pixel, the generated pulses must be distributed at equal distances. As the sequence is getting filled the algorithm may want to place addresses in slots that are already occupied. In this case, it will put the pulse in the nearest empty slot. Thus, this method will make more mistakes at the end of the process than the beginning. The execution time grows considerably because the collisions have a high-cost in time to resolve them. The best results for this method are obtained for low-intensity images.

2.3 The Random method

This method place the address events in the slot positions obtained by a pseudo-random number generator based on Linear Feedback Shift Registers (LFSR) [7][8]. Due to the properties of the LFSR used, each slot position is generated only once and no collisions appear. If a pixel in the image has the intensity p, then the method will take p values from the pseudo-random number generator and places the pixel address in the corresponding p slots of the address sequence. They will not be equidistant but will appear along the complete address sequence. This method is fast, because the image is swept only once, and because the algorithm does not need to perform searches for empty slots.

Because the LFSR can obtain consecutive, or very close, addresses in a few calls, the method can be enhanced using a b-bit counter for the high part of the address, dividing the address sequence in 2^b slices of the same size. So, for each call to the address generator, 2^b addresses equally distributed are obtained. For any pixel with a grey level value near to 2^b, the problem is solved. However, for higher values, the same problem appears again for the events located at the same slice. When the grey level value decreases, the distribution of the events gets worse, because all events are concentrated at a few neighbouring slices.

Fig. 2 shows the LFSR structure with a 2-bit counter for a 128x128 images with a 256 grey levels.

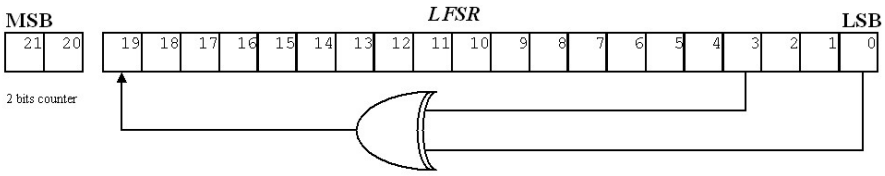


Fig. 2: LFSR with a 2-bit counter for pseudo-random numbers generation.

2.4 The Random-Square method

Using the Random method with a counter from 1 to the maximum grey level, the problem with the high values (mentioned above) is solved, but the event distribution is still poor. Substituting the counter by another LFSR can improve the distribution of pixels with low grey level values.

For a 128x128 image with maximum grey level of 255, one 8-bit LFSR (LFSR-8) is used for selecting 255 slices of 128x128 positions, and another 14-bit LFSR (LFSR-14) selects the position inside the slice. The image is scanned only once. For each pixel a 14-bit number is generated by the LFSR-14, and the LFSR-8 is called as many times as the intensity level of the pixel would indicate. Fig. 3 shows the LFSRs used by the Random-Square method.

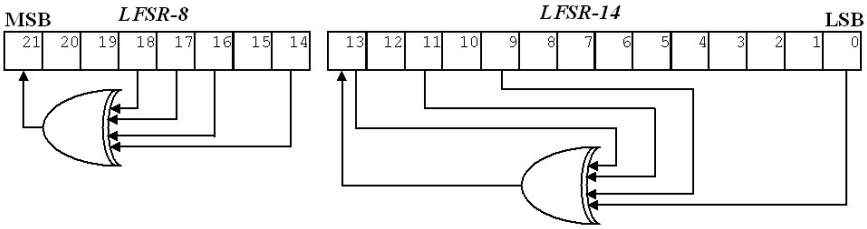


Fig. 3: LFSR-8 and LFSR-14 used by the Random-Square method.

2.5 The Exhaustive method

This algorithm also divides the address event secuence in K slices of NxM positions for an image of NxM pixels with a maximum grey level of K. This method is based on the idea that each possible event (as maximum $NxMxK$) has assigned one fixed position in the address event secuence. In such way, for the slice k, an event of the pixel (i,j) is sent on the time t if the following condition is asserted:

$$(k \cdot P_{i,j}) \bmod K + P_{i,j} \geq K \text{ and } N \cdot M \cdot (k-1) + (i-1) \cdot M + j = t$$

where $P_{i,j}$ is the intensity value of the pixel (i,j) (Fig. 4).

The Exhaustive method tries to distribute the events of each pixel into the K slices at equal distances. For this propose, the algoritm scan the image K times. In the iteration k, if the previous condition is true, then the corresponding event is sent, otherwise the algorithm will wait for the following event (no event is sent at time t).

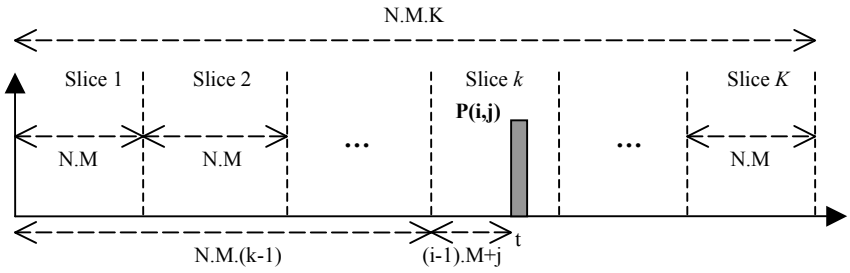


Fig. 4: The event distribution made by the Exhaustive method.

3. Simulation Results

The goal of this section is to compare the behavior of the five methods proposed using as parameters both the execution time and the desviation of the obtained event distribution from the ideal (distribution error). For this task, the methods have been implemented in C++ and an application has been developed which allows the generation of the address events from a grey-scale image and calculate the previously described parameters. Fig. 5 shows a screenshot of this software interface.

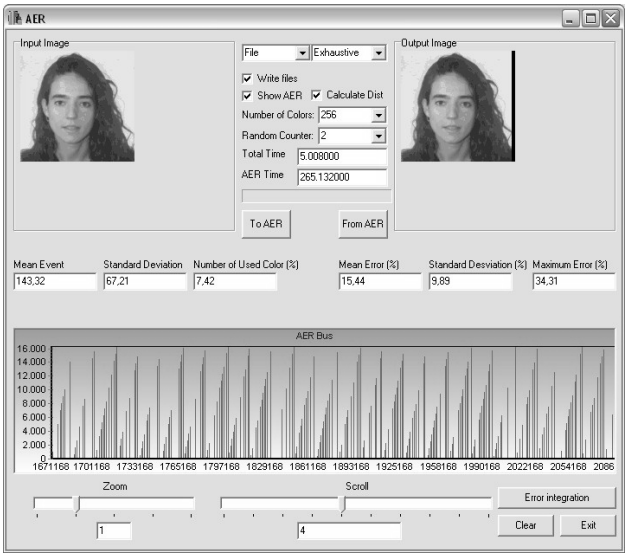


Fig. 5: Software interface.

3.1 The execution time

The simulation has been performed by using a Pentium IV at 1.6Ghz with 256Mb of RAM under Windows XP. The test has been made by using two images: the one shown in Fig. 6 and another with all pixels with the maximum grey value (the white image is the worst case). The results obtained are shown in Table 1.

Fig. 6: Reference image to convert into AER format via software.

	Images	
Method	**Reference**	**White (worst case)**
Scan	0,09	0,26
Uniform	0,38	1,72
Random	0,63	1,09
Random Square	0,84	1,39
Exhaustive	0,23	0,23

Table 1: Execution times in seconds spent by the methods for processing the images

The best results have been obtained by the Scan and the Exhaustive methods. The Exhaustive algorithm has an execution time independent of the total number of events that are generated. However, the Scan algorithm has a better behavior than the Exhaustive for images with a low event number.

The Uniform is the slowest method for the white image because the number of collisions, which have a high-cost in execution time, increases with the event number. However, this time improve very much when the event number of the image is reduced, as can be seen for the reference image case (50% intensity charge).

As the sotware implementation of an LFSR is not very eficient in time, the two pseudo-random number generator based methods are slow. As the Random-Square use two LFSRs, its execution time is greater than that spent by the Random method for all images. Anywhere, these methods can be easily implemented in hardware reducing considerably their execution time.

3.2 The distribution error

In an ideal AER distribution all events for one pixel and image could be equidistant in time. In this section, the event distribution obtanined with each method is evaluated. The proposed distribution error measures how much the event distribution generated by a method deviates from the ideal distribution.

Let suppose D_{ij} is the ideal distance between events of the pixel *(i,j)* of a *NxM* image with *K* grey level values.

$$D_{i,j} = \frac{N \cdot M \cdot K}{P_{i,j}}$$

where $P_{i,j}$ is the intensity value of the pixel *(i,j)*.

Let suppose $d_{i,j}^{k}$ is the distance between the *k*-event ($p_{i,j}^{k}$) and the following event ($p_{i,j}^{k+1}$). The distance of the last event is calculated, by supposing that the next event is the first of a new sequence of the same image (we suppose that the image is continuously re-sent).

$$d_{i,j}^{k} = p_{i,j}^{k+1} - p_{i,j}^{k}$$

Then we can measure the mean error for a pixel as the average of the differences between the ideal and real distance. The error expresion is:

$$e_{i,j} = \frac{\sum_{k=1}^{P_{i,j}} \left| D_{i,j} - d_{i,j}^{k} \right|}{P_{i,j}}$$

It is easy to see that the worst case for this error measure is that in which all the events are together in the address sequence (Fig. 7).

Therefore, in order to compare the error obtained by different methods and images, the error of each pixel must be normalized respect to the maximum error associated to that pixel. The following expression is the maximum error for the pixel *(i,j)*:

$$me_{i,j} = 2 \cdot (D_{i,j} - 1) \cdot (1 - \frac{1}{P_{i,j}})$$

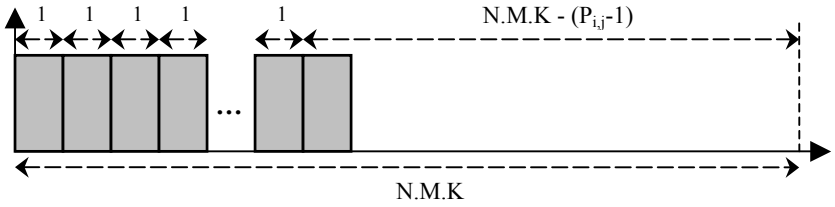


Fig. 7: The worst event distribution.

Finally, we define a matrix with the same dimensions of the image, where the *(i,j)* element represents the normalized error for the pixel *(i,j)*:

$$NE = \begin{pmatrix} e_{1,1}/me_{1,1} & e_{1,2}/me_{1,2} & \cdots & e_{1,M}/me_{1,M} \\ e_{2,1}/me_{2,1} & e_{2,2}/me_{2,2} & \cdots & e_{2,128}/me_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ e_{N,1}/me_{N,1} & e_{N,2}/me_{N,2} & \cdots & e_{N,M}/me_{N,M} \end{pmatrix}$$

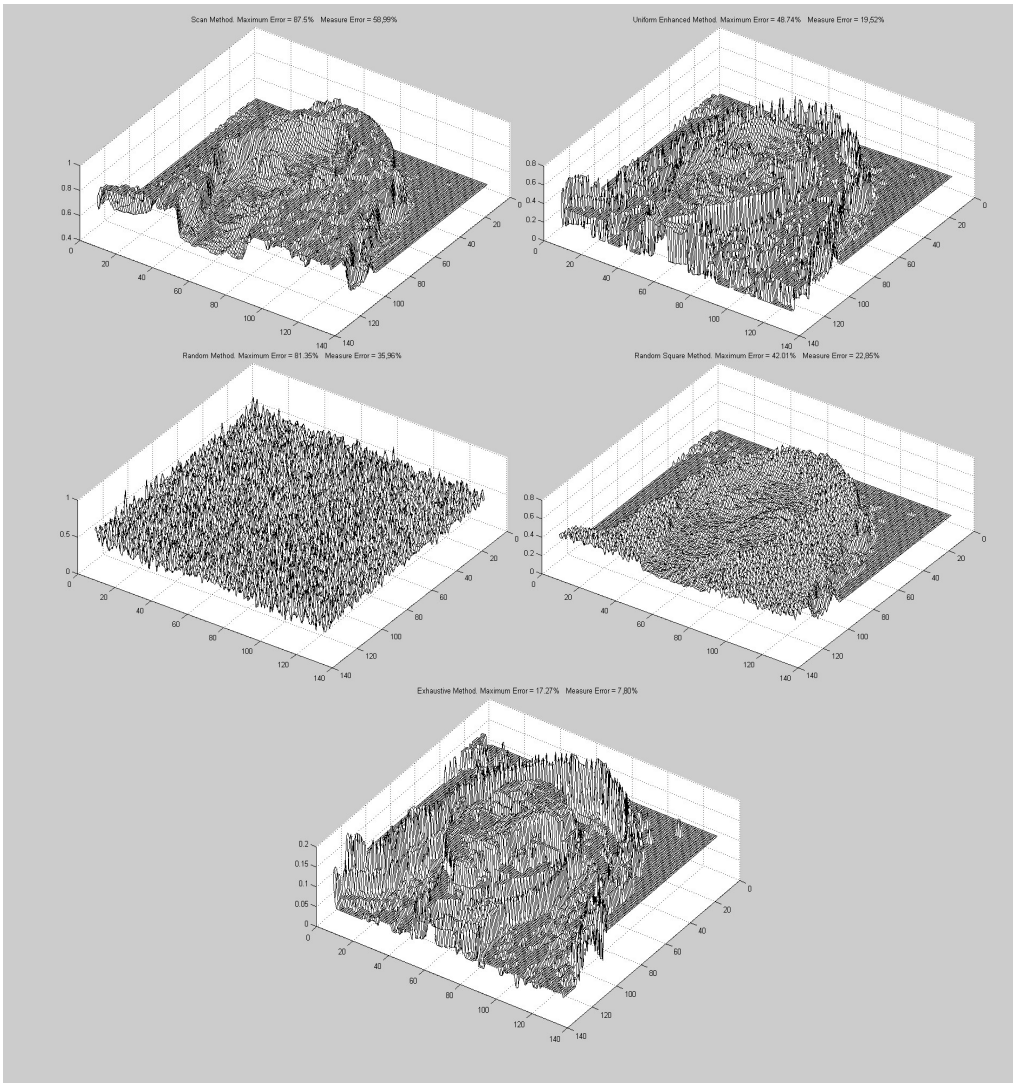
Consider the example image in Fig. 6. For each of the synthetic AER generation methods ("Uniform", "Scan", "Random", "Random-Square" and "Exhaustive"), we have generated by software the corresponding AER distribution and computed the normalized errors. The resulting matrixes are shown in Fig. 8. The Table 2 shows the maximum and mean normalized errors in percent.

As can be seen, the proposed Exhaustive method yields a significant improvement over the other methods. The Random-Square results are better than those obtained by the Random method. As the image is not very charged the results got by the Uniform method are similar to those of the Random-Square. For more charged images the results obtained by the Uniform can be worse.

6. Conclusions and future works

In this paper five methods for generating AER streams in real time from images stored in a computer's memory have been presented. The different methods have been analysed and tested by simulation software. The best results, using the execution time and the distribution error as parameters, have been obtained by the Exhaustive method.

A hardware platform that exploits these techniques is currently under development. For this goal, an analysis of different architectures for real time is being made. A



dedicated hardware to write and read to/from an AER bus is being developed using a standard FPGA-based prototyping board.

Fig. 8: From left to right and from up to down the normalized error of methods: Scan, Uniform, Random, Random-Square and Exhaustive (with different scales)

Method	Max. NE in %	Mean NE in %
Scan	87,5	58,99
Uniform	48,74	19,42
Random (2bits)	81,35	35,96
Random-Square	42,01	22,85
Exhaustive	**17,27**	**7,80**

Table 2: Maximum and mean normalized errors for the five methods proposed.

7. References

[1] M. Sivilotti, *Wiring Considerations in analog VLSI Systems with Application to Field-Programmable Networks*, Ph.D. Thesis, California Institute of Technology, Pasadena CA, 1991.

[2] Teresa Serrano-Gotarredona, Andreas G. Andreou, Bernabé Linares-Barranco. "AER Image Filtering Architecture for Vision-Processing Systems". IEEE Transactions on Circuits and Systems. Fundamental Theory and Applications, Vol. 46, N0. 9, September 1999.

[3] A. Cohen, R. Douglas, C. Koch, T. Sejnowski, S. Shamma, T. Horiuchi, and G. Indiveri, *Report to the National Science Foundation: Workshop on Neuromorphic Engineering*, Telluride, Colorado, USA, June-July 2001. [www.ini.unizh.ch/telluride]

[4] Kwabena A. Boahen. "Communicating Neuronal Ensembles between Neuromorphic Chips". Neuromorphic Systems. Kluwer Academic Publishers, Boston 1998.

[5] Charles M. Higgins and Christof Koch. "Multi-Chip Neuromorphic Motion Processing". January 1999.

[6] VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function. Thesis by Misha Mahowald. California Institute of Technology Pasadena, California 1992.

[7] Pierre L'Ecuyer, François Panneton. "A New Class of Linear Feedback Shift Register Generators". Proceedings of the 2000 Winter Simulation Conference.

[8] Linear Feedback Shift Register V2.0. Xilinx Inc. October 4, 2001. http://www.xilinx.com/ipcenter

***Acknowledgement**: This work has been supported by the research project "CAVIAR: Convolution AER Vision Architecture for Real-Time (IST-2001-34124)" financed by the European Commission.*