Two Hardware Implementations of the Exhaustive Synthetic AER Generation Method

F. Gomez-Rodriguez, R. Paz , L. Miro, A. Linares-Barranco,
G. Jimenez, and A. Civit

Arquitectura y Tecnología de Computadores, Universidad de Sevilla,
Av. Reina Mercedes s/n, 41012-Sevilla, Spain
gomezroz@atc.us.es

Abstract. Address-Event-Representation (AER) is a communications protocol for transferring images between chips, originally developed for bio-inspired image processing systems. In [6], [5] various software methods for synthetic AER generation were presented. But in neuro-inspired research field, hardware methods are needed to generate AER from laptop computers. In this paper two real time implementations of the exhaustive method, proposed in [6], [5], are presented. These implementations can transmit, through AER bus, images stored in a computer using USB-AER board developed by our RTCAR group for the CAVIAR EU project.

1 Introduction

Sivilotti in 1991 proposed a new communications protocol [1] to interconnect neuro-inspired system, Address-Event-Representation (AER). The idea of AER is to provide an interconnection mechanism to communicate neuron-based chips. One neuron sends information to the next neurons layer sending pulses. The frequency of these pulses is proportional to the stimulus received by the neurons.

In biological systems, neurons are connected by point to point connections. In silicon based systems when we want to connect neurons located in different chips, point to point connections are not feasible (each chip can implement thousands of neurons communicate to a similar number of neurons in other chips). As it is not possible to interconnect chips using the required number connections lines, AER tries to solve this using a time multiplexed high speed digital bus.

An emitter chip, like a retina, is composed by cells that send pulse streams. The frequency of these pulses is proportional to the stimulus these cells are receiving. An arbiter combines all the streams and sends the address of the selected cell through the AER bus. At the receiver side, a decoder has to de-multiplex incoming events and send each event to the corresponding cell. This cell can integrate received pulses in order to reconstruct correct value.

To properly synchronise devices attached to this bus, a simple asynchronous handshake protocol based in two signals is used: a request signal (REQ) driven by the emitter device, and an acknowledge signal (ACK) driven by the receiver.

By connecting several AER devices, we can get a bio-inspired system able to make complex image or audio processing in real time. Many papers about AER have been presented like [2], [3], and [4].

In [5] and [6] some software methods to convert an image stored in a computer memory into an AER stream were proposed. AER system researchers need computer interfaces for testing and debugging purposes. On PCI based interfaces the bus bandwidth allows software to be used to produce individual events. If frame based videos have to be converted this can be done by software methods. This is not the case when using lower bandwidth USB connections, as in this case, it is necessary to transmit frames through the USB bus and perform frame to AER conversion in hardware. In this paper we present two hardware implementations of the Exhaustive method [5] [6] and we evaluate theirs behaviour.

2 Exhaustive Method for Synthetic AER Generation

The exhaustive method for synthetic AER generation has been presented in [5] and [6]. A. Linares-Barranco et al. present a software implementation of this method, this paper present two hardware implementations of the exhaustive method. Before presenting each implementation, let us see how the software exhaustive method is used to transmit an image through AER bus.

The time needed to transmit an image is called frame period. This algorithm divides the period 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 (with maximum of *NxMxK* per frame period) has assigned one fixed position in the address event sequence. In thus way, for the slice *k*, an event of the pixel *(i,j)* is sent on the time *t* if the following condition is fulfilled:

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

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

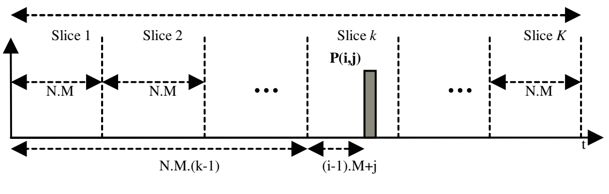

Fig. 1. The event distribution made by the Exhaustive method

The Exhaustive method tries to distribute the events of each pixel into the *K* slices at equal distances.

3 "Modulus" Exhaustive Method Implementation

The "modulus" implementation tries to emit each event as near as possible to the correct place. The idea of this implementation consists in dividing the period in slices; in each slice all the image is scanned. To decide if a pixel emits an event the following expression is used:

$$slice * p_i \bmod 256 < p_i \tag{2}$$

where:

slice: is a counter which provides the slice number
P_i*:* is the pixel value.

This formulation is very similar to formulation presented in [7]. The implementation tries to distribute the events in the best way possible. But the division of the period is sufficient to put all the events in theirs correct place. The exhaustive method shares the events, minimizing the errors.

Let us see a simple example of the exhaustive method approach. Consider an image with 8 grey levels, Table 1 shows how the method decides when a pixel has to emit an event. Columns represent grey levels (0 to 7) and rows represent slice; each cell is the value of $window * p_i \bmod 8$, in dark is marked the slices where each event has to emit depending on its value.

Fig. 2 shows a diagram of the system,

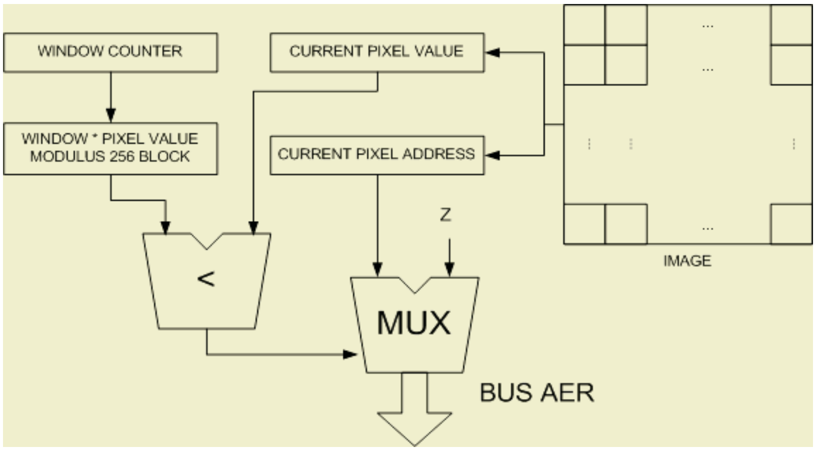

Fig. 2. "Modulus" system diagram

The size of the slice counter and pixel value are given by the number of grey levels, so $size = \log_2 K$ where K is the number of grey levels, usually K = 256. The size of the pixel address depends on the size of the image, thus $size\_address = \log_2 size\_image$. For this implementation the size of the image is 1024 pixels.

Table 1. Events distribution using "modulus" exhaustive methods implementation with 8 grey levels

	Gray levels							
Slices	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

4 "Bitwise" Exhaustive Method Implementation

Bitwise implementation tries to get a very simple hardware approach to exhaustive method to minimize processing time and increase event rate. The idea of exhaustive method is to divide a frame time needed to send a whole image into slices. Each slice has a fixed time slot for each pixel. For a specified pixel, each slice has a slot and the number of slices that send an event is proportional to the bright level of the pixel. The distance between two events of a pixel should be constant. If a pixel has a value P, it should emit P times in K-1 slices (K is the number of values allowed, from 0 to K-1, so for 0 level no events are emitted and for K-1, events are sent it all slices).

Bitwise implementation is based on a binary counter driven by a periodic clock signal. This binary counter is divided into two parts. The least significant part is used to address the pixel, and the most significant part is used as "slice counter" to identify a slice in a period. A full run of the whole counter last a period: (K-1)*N*M clock cycles).

In order to determine if a pixel has to send an event, the slice number and pixel value have to be compared. If pixel level is greater than the bit-reversed slice counter, then the event has to be sent. This bit reversal is needed to homogenise event distribution. Bit reversal is very popular because it is used in FFT algorithm, and also in many ciphering algorithms to get maximum bit dispersion [8].

Table 2. Events distribution using "bitwise" exhaustive methods implementation with 8 grey levels

	Gray levels							
Slices	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	0	0	0	0	0	0	0
1	4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2	2
3	6	6	6	6	6	6	6	6
4	1	1	1	1	1	1	1	1
5	5	5	5	5	5	5	5	5
6	3	3	3	3	3	3	3	3
7	7	7	7	7	7	7	7	7

As it is show in Table 2 for K=8, we have K-1 slices from 0 to K-2. For maximum grey level (value=7) all slices are used. For grey level 1 one by each 7 is used. A pure binary counter will count one more slice, adding an additional empty slice because no pixel is going to be emitted during slice number. This additional slice decreases performance, increasing period by T/(K-1). This is not very important for high K values.

The whole system design is shown in Figure 3:

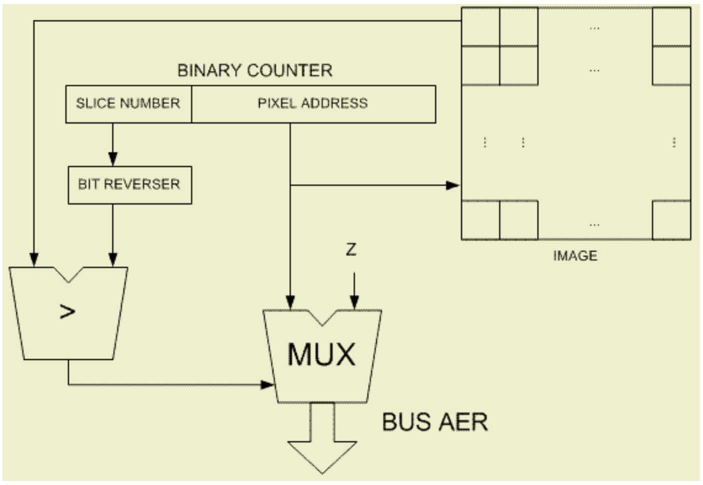

Fig. 3. "Bitwise" system diagram

5 Implementations Comparison

Both implementations have been programmed into board developed by RTCAR group at the University of Seville, called USB-AER board. USB-AER board has a FPGA that can be programmed for multiple applications. The implementations presented have been programmed in an USB-AER board in order to test their reliability.

In order to evaluate how both methods distribute the events in [6], an error measurement method is presented. We use this test to evaluate both implementations of the exhaustive method. The mathematical expression of the error measurement is:

$$Normalized\_error = {}^{e}\!/_{m} \tag{3}$$

where:

$$e = \frac{\sum_{k=1}^{ns} |D - d_k|}{ns} \tag{4}$$

$$me = 2 \cdot (D-1) \cdot (1 - \frac{1}{P}) \text{ with } P \neq 1$$

ns is the number of samplers.

D is the theoretical event distance

d is the real event distance

p is the pixel value

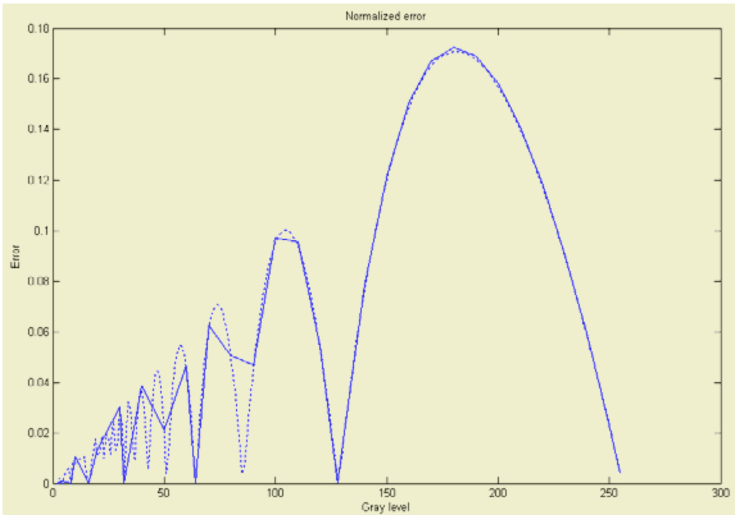

Fig. 4. Normalized error for “modulus” Exhaustive method implementation

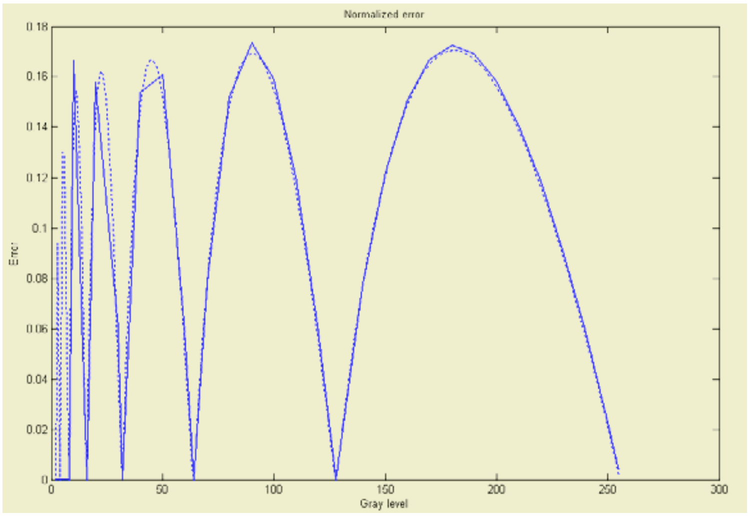

Fig. 5. Normalized error for "bitwise" Exhaustive method implementation

The test consists in transmitting an image with all pixels set to zero except one. This pixel changes its value, in order to test how implementation follows the theoretical event distribution for each grey level. The images have 256 grey levels. Figures 4 and 5 present the error distribution for each implementation.

Figure 4 and Figure 5 show, in solid line, the normalized error for “modulus” and “bitwise” hardware implementation respectively (taken 10 by 10 and power of 2 grey values); dotted line represents the theoretical implementation normalized error.

Both implementations present zero error when grey level is a power of 2, because in this case the period is divisible by these values. This is expected for all exhaustive method implementations.

The main differences between both implementations are for low pixel values. While "modulus" implementation presents smaller errors than "bitwise" implementation, this last implementation is simpler, because it does not need to compute any multiplication.

6 Conclusions

Two hardware implementations of the exhaustive method are presented. These implementations make possible the interconnection between computers and neuro-inspired systems, through an AER-BUS. Thus a computer can be used as an AER stream generator to develop other systems even with a relatively low bandwidth connection to the generator interface. This is typically the case of laptop computers with no PCI slots.

Acknowledgements

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

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. 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]
3. Kwabena A. Boahen. "Communicating Neuronal Ensembles between Neuromorphic Chips". Neuromorphic Systems. Kluwer Academic Publishers, Boston 1998.
4. Misha Mahowald. VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function. PhD. Thesis, California Institute of Technology Pasadena, California, 1992.
5. A. Linares-Barranco. Estudio y evaluación de interfaces para la conexión de sistemas neuromórficos mediante Address- Event-Representation. Ph.D. Thesis, University of Seville, Spain, 2003
6. A. Linares-Barranco, R. Senhadji-Navarro, I. García-Vargas, F. Gómez-Rodríguez, G. Jimenez and A. Civit. Synthetic Generation of Address-Event for Real-Time Image Processing. ETFA 2003, Lisbon, September. Proceedings, Vol. 2, pp. 462-467.
7. Alejandro Linares-Barranco, Gabriel Jimenez-Moreno, Antón Civit-Ballcels, and Bernabé Linares-Barranco. On Synthetic AER Generation. ISCAS 2004. Vancouver, Canada, May, 2004.
8. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. Numerical recipes in c: the art of scientific computing ISBN 0-521-43108-5.1992. Cambridge University Press.