

# Test Infrastructure for Address-Event-Representation Communications

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

Departamento de Arquitectura y Tecnología de Computadores,  
Universidad de Sevilla,  
Av. Reina Mercedes s/n, 41012-Sevilla, Spain  
rpaz@atc.us.es  
<http://www.atc.us.es>

**Abstract.** Address-Event-Representation (AER) is a communication protocol for transferring spikes between bio-inspired chips. Such systems may consist of a hierarchical structure with several chips that transmit spikes among them in real time, while performing some processing. To develop and test AER based systems it is convenient to have a set of instruments that would allow to: generate AER streams, monitor the output produced by neural chips and modify the spike stream produced by an emitting chip to adapt it to the requirements of the receiving elements. In this paper we present a set of tools that implement these functions developed in the CAVIAR EU project.

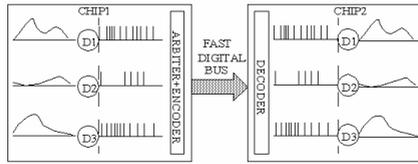
## 1 Introduction

Address-Event-Representation (AER) was proposed in 1991 by Sivilotti [1] for transferring the state of an array of neurons from one chip to another. It uses mixed analog and digital principles and exploits pulse density modulation for coding information. The state of the neurons is a continuous time varying analog signal.

Fig. 1 explains the principle behind the AER. The emitter chip contains an array of cells (like, e.g., an imager or artificial retina chip) where each pixel shows a state that changes with a slow time constant (in the order of milliseconds). Each pixel includes an oscillator that generates pulses of minimum width (a few nanoseconds). Each time a pixel generates a pulse (called "event"), it communicates with the periphery and its address is placed on the external digital bus (the AER bus). Handshaking lines (Acknowledge and Request) are used for completing the communication.

In the receiver chip the pulses are directed to the pixels or cells whose address was on the bus. This way, pixels with the same address in the emitter and receiver chips will "see" the same pulse stream. The receiver cell integrates the pulses and reconstructs the original low frequency continuous-time waveform.

Transmitting the pixel addresses allows performing extra operations on the images while they travel from one chip to another. For example, inserting memories (e.g. EEPROM) allows transformations of images.

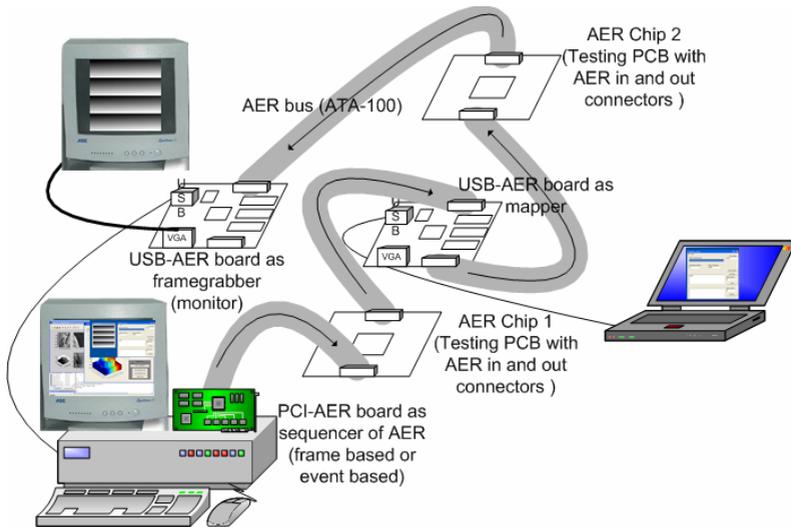


**Fig. 1.** AER inter-chip communication scheme

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 [2]. The goal of this community is to build large multi-chip hierarchically structured systems capable of performing complicated array data processing in real time. To make these objectives possible it is essential to have a set of instruments that would allow to:

- Sequence: Produce synthetic AER event streams that can be used as controlled inputs while testing and adjusting a chip or set of chips.
- Monitor: Observe the output of any element in the system.
- Map: Alter the stream produced by an emitter and send the modified stream to a receiver

For these purposes we have designed and implemented two different instruments: a PCI board capable of sequencing and monitoring events at a rate of over 15Mevents/s and a versatile board that can be used for sequencing, monitoring and mapping. This last board can be used either in a stand alone mode or connected to an external



**Fig. 2.** AER tools usage scenario

computer through a USB bus. A possible scenario for these tools is shown in fig. 2 where a computer with a PCI-AER board produces output for AER chip1. The output from this chip is remapped by a USB-AER board and fetched to AER chip 2. The stream produced by chip 2 is monitored by another USB-AER board which can send its output directly to a VGA monitor or to a computer through USB bus.

## 2 Sequencing and Monitoring AER Events

To be useful for debugging an AER tool should be able to receive and also send a long sequence of events interfering as little as possible with the system under test. Let's start explaining the meaning of interfacing in the context.

As neurons have the information coded in the frequency (or timing) of their spikes, the pixels that transmit their address through an AER bus also have their information coded in the frequency of appearance of those addresses in the bus. Therefore, interspike-intervals (ISIs) are critical for this communication mechanism. Thus, a well designed tool shouldn't modify the ISIs of the AER.

The ISIs may be difficult to preserve depending on the nature of the emitter and/or receiver chips. Let's suppose the case of having an AER emitter chip connected to an AER receiver chip, and we want to debug their communication. In principle, there are two possibilities: connecting to the bus an AER sniffer element, or to introducing a new AER element in between the emitter and the receiver.

- The sniffer element will consist on an AER receptor that captures the address and stores it with a timestamp in memory for each request that appears on the AER bus. The problem in this case is that the speed of the emitter and receiver protocol lines could be faster than the maximum speed supported by the sniffer (15 ns per event in some existing chips), causing events to be lost. Another typical problem could be that the throughput of the AER bus (unknown in principle) would be so high that the interface memory cannot be downloaded to the computer's memory on time. This also implies that events are lost.
- The other possibility is to introduce a new AER element between the two chips. In this case the emitter sends the event to the AER element and the AER element sends the same event to the receiver chip. The problem now is that the new AER element will always introduce a delay in the protocol lines, and may also block the emitter if it is not able to keep up with its throughput. Therefore, ISIs are not conserved. But the behaviour will be the same than if we connect the emitter to a slower receiver.

The throughput problem requires using very fast PC interfaces and the problem of very fast emitter or receiver protocols can be reduced by using a very high frequency clock for the stages that interface with the AER protocols.

## 3 PCI-AER Interface

Before the development of our tools the only available PCI-AER interface board was developed by Dante at ISS-Rome (See [2]). This board is very interesting as it embeds

all the requirements mentioned above: AER generation, remapping and monitoring. Anyhow its performance is limited to 1Mevent/s approximately. In realistic experiments software overheads reduce this value even further. In many cases these values are acceptable but, currently many address event chips can produce (or accept) much higher spike rates.

As the Computer interfacing elements are mainly a monitoring and testing feature in many address event systems, the instruments used for these proposes should not delay the neuromorphic chips in the system. Thus, speed requirements are at least 10 times higher than those of the original PCI-AER board. Several alternatives are possible to meet these goals:

- Extended PCI buses.
- Bus mastering.
- Hardware based Frame to AER and AER to Frame conversion.

When the development of the CAVIAR PCI-AER board was started, using 64bit/66MHz PCI seemed an interesting alternative as computers with this type of buses were popular in the server market. When we had to make implementation decisions the situation had altered significantly. Machines with extended PCI buses had almost disappearing and, on the other hand, serial LVDS based PCI express [3] was emerging clearly as the future standard but almost no commercial implementations were in the market. Therefore, the most feasible solution was to stay with the common PCI implementation (32 bit bus at 33MHz) and consider PCI express for future implementations. Speed improvements, thus, should come from the alternative possibilities.

The previously available PCI-AER board uses polled I/O to transfer data to and from the board. This is possibly the main limiting factor on its performance. To increase PCI bus mastering is the only alternative. The hardware and driver architecture of a bus mastering capable board is significantly different, and more complex, than a polling or interrupt based implementation.

Hardware based frame to AER conversion doesn't increase PCI throughput but, instead, it reduces PCI traffic. First some important facts have to be explained. It is well known that some AER chips, especially grey level imagers where pulse density is proportional to the received light intensity, require a very large bandwidth. This is also the case of many other chips when they are not correctly tuned. For example let's consider a Grey level 128\*128 imager with 256 grey levels. In a digital frame based uncompressed 25fps format, it would require a bandwidth of  $128*128*25=0.39\text{MBytes/s}$ . The maximum requirements for an "equivalent" system that would output AER supposing the number of events in a frame period is equal to the gray level and considering the worst case where all pixels spike with maximum rate is:

$$2\text{bytes/event}*256\text{events/pixel}*number\ of\ pixels/\ frame\ period= 200\text{MBytes/s}$$

The meaning of this figure should be carefully considered. A well designed AER system, which produces events only when meaningful information is available, can be very efficient but, an AER monitoring system should be prepared to support the bandwidth levels that can be found in some real systems. These include systems that have not been designed carefully or that are under adjustment. Currently the available

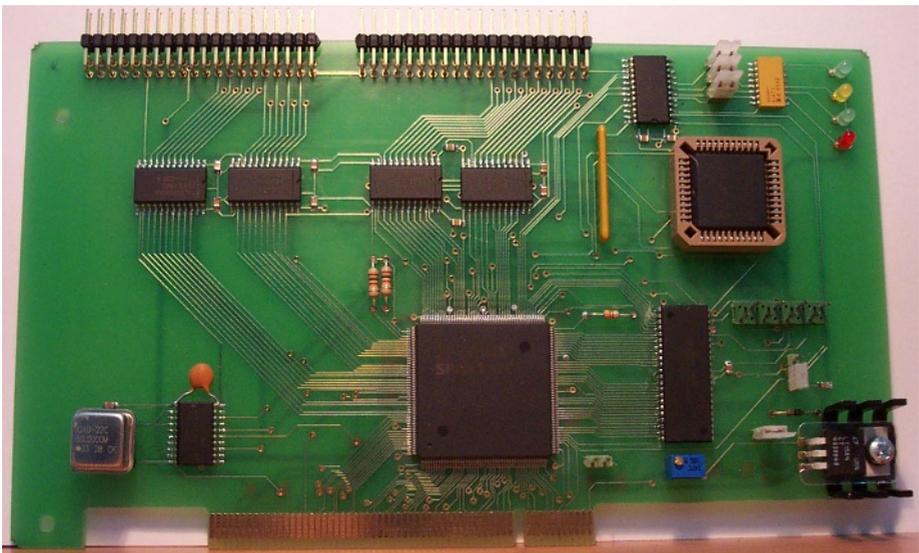
spike rates, even in these cases, are far from the value shown above but, some current AER chips may exceed the 40Mevent/s in extreme conditions.

The theoretical maximum PCI32/33 bandwidth is around 133Mbytes/s. This would allow for approximately 44Mevent/s considering 2 bytes per address and two bytes for timing information. Realistic figures in practice are closer to 20Mbyte/s. Thus, in those cases where the required throughput is higher a possible solution is to transmit the received information by hardware based conversion to/from a frame based representation. Although this solution is adequate in many cases, there are circumstances where the developers want to know precisely the timing of each event, thus both alternatives should be preserved.

Implementing AER to Frame conversion is a relatively simple task as it basically requires counting the events over the frame period. Producing AER from a frame representation is not trivial and several conversion methods have been proposed [4][5].

The theoretical event distribution would be that where the number of events for a specific pixel is equal to its associated grey level and those events are equally distributed in time. The normalized mean distance from the theoretical pixel position in time to the resulting pixel timing with the different methods is an important comparison criterion. In [6] it is shown that, in most circumstances, the behavior of the methods is similar and, thus, hardware implementation complexity is an important selection criterion. From the hardware implementation viewpoint random, exhaustive and uniform methods are especially attractive.

As a result of these considerations the design and implementation of the CAVIAR PCI-AER board was subdivided into a set of intermediate steps in which initially no mastering was implemented. Later Bus mastering was included and hardware based frame to AER conversion was included as a last step.



**Fig. 3.** CAVIAR PCI-AER board

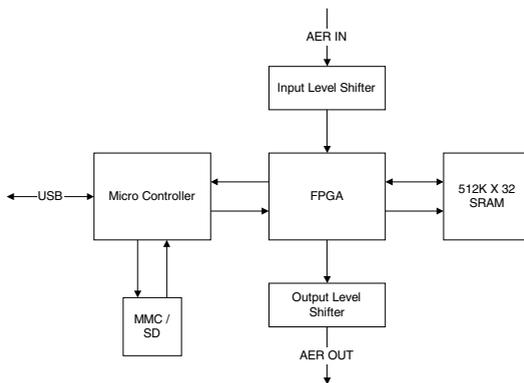
The physical implementation of all the steps is equal. They differ in the VHDL FPGA code and in the operating system dependant driver. The first design was a VIRTEX based board which was completely redesigned after the first tests. It was established that most of the functionality, demanded by the users, could be supported by the larger devices in the less expensive SPARTAN-II family. The Spartan Version of the board is shown in fig. 3.

Currently a Windows driver that implements bus mastering is being tested. The Linux version with bus mastering is still under development. An API that is compatible, as much as permitted by the different functionality, with that used in the current PCI-AER board has been implemented. MEX files to control the board from MATLAB have also been developed.

## 4 USB-AER

The CAVIAR PCI-AER board can perform Address Event sequencing and monitoring functions but has no hardware mapping capabilities. Although software based mapping is feasible a specific device for this purpose is needed if we want to build AER systems that can operate without requiring any standard computer. This standalone operating mode requires to be able to load the FPGA and the mapping RAM from some type of non volatile storage that can be easily modified by the users. MMC/SD cards used in digital cameras are a very attractive possibility. However in the development stage the users prefer to load the board directly from a computer and, for this purpose USB seems the most suitable solution.

Many AER researchers would like to demonstrate their systems using instruments that could be easily interfaced to a laptop computer. This requirement can also be supported with the USB-AER board as it includes a relatively large FPGA that can be loaded from MMC/SD or USB, a large SRAM bank and two AER ports. Thus the board can be used also as a sequencer or a monitor. Due to the bandwidth limitations of full speed USB (12Mbit/s) Hardware based event to frame conversion is essential in this board for high, or even moderate, event rates.



**Fig. 4.** USB-AER Block Diagram

The USB-AER board is based around a Spartan-II 200 Xilinx FPGA, with a 512K\*32 12ns SRAM memory bank. The board uses a Silicon Laboratories C8051F320 microcontroller to implement the USB and the MMC/SD interface. A simple VGA monitor interface is also provided to allow the board to act as a monitor (frame grabber). A block diagram of the board is shown in fig. 4.

The board will act as a different device according to the module that is loaded in the FPGA either through a MMC/SD card or from the USB bus. Currently the following Modes are implemented:

- Mapper: 1 event to 1 event and 1 event to several events.
- Monitor (frame-grabber): using either USB or VGA as output.
- Sequencer: based on hardware frame to AER conversion using the Random or Exhaustive method.

The power consumption is around 220mA (1.63W) working as random sequencer. And its size is 100x130 mm.

Modules are under development to capture sequences of up to 512K events with timestamps and to play these sequences back. These modules are very interesting when a researcher wants to use the output stream produced by a chip from another researcher as input to his or her chip.

This new board was interfaced in Telluride 04 [7] to the current version of the CAVIAR retina and to an imager developed at JHU. Later in the CAVIAR meeting in September 04 it was interfaced to the remaining project chips. The USB-AER board is shown in fig. 5.

A simple interface to control this board is available under windows. It allows loading modules into the FPGA, uploading or downloading data to the FPGA, and showing the received images when the board acts as a monitor.



**Fig. 5.** USB-AER Board

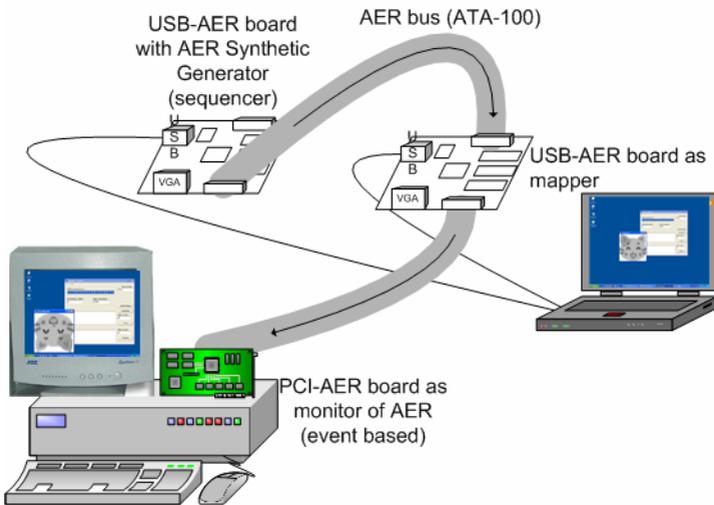
A Linux driver for the USB-AER is currently under test. With this driver the USB-AER board can be easily integrated with several MATLAB applications developed at INI [8].

## 5 Conclusions

A set of tools has been developed that allow efficient testing and demonstration of address event based systems. A demonstration scenario is shown in fig. 6. In this case a USB-AER board is generating a stream of events that correspond to the image shown in Fig. 6 in the laptop screen, which is rotated by a second USB-AER board used as a mapper to produce an AER event stream, which is captured by a PCI-AER board used as a monitor to produce the output shown in the desktop PC screen.

In this scenario only the presented tools are shown. In real world cases the tools are used to evaluate or tune neural chips. In the CAVIAR project the chips have been interfaced to two different retinas, a convolution chip, a winner take-all chip and a learning chip.

Although the AER protocol is asynchronous, the VHDL codes for both boards have been developed synchronously, due that both the PCI and the USB buses are synchronous. To avoid the metastability two cascade flip-flops have been used for the input protocol lines.



**Fig. 6.** Demonstration Scenario

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

## 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](http://www.ini.unizh.ch/telluride)]
3. A. Wilen, J. Schade, R. Thornburg. "Introduction to PCI Express", Intel Press, 2003.
4. 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
5. 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.
6. Linares-Barranco, A.; Jimenez-Moreno, G.; Civit-Ballcells, A.; Linares-Barranco, B.; On synthetic AER generation. ISCAS '04. Proceedings of the IEEE 2004 May 2004 Pages:V-784 - V-787 Vol.5
7. Avis Cohen, et.al., Report on the 2004 Workshop On Neuromorphic Engineering, Telluride, CO. June - July , 2004 [[www.ini.unizh.ch/telluride/previous/report04.pdf](http://www.ini.unizh.ch/telluride/previous/report04.pdf)]
8. M. Oster, Serverbased Software Architecture for AER systems [<http://www.ini.unizh.ch/~mao/AerSoftware/SoftwareOverview.pdf>]