Address-Event based Platform for Bio-inspired Spiking Systems

A. Jiménez-Fernández, C.D. Luján, A. Linares-Barranco, F. Gómez-Rodríguez, M. Rivas, G. Jiménez, A. Civit. [1]

Dpto. de Arquitectura y Tecnología de Computadores, Universidad de Sevilla, Av. Reina Mercedes s/n, 41012 Sevilla, Spain. E-mail: alinares@atc.us.es

ABSTRACT

Address Event Representation (AER) is an emergent neuromorphic interchip communication protocol that allows a real-time virtual massive connectivity between huge number neurons, located on different chips. By exploiting high speed digital communication circuits (with nano-seconds timings), synaptic neural connections can be time multiplexed, while neural activity signals (with mili-seconds timings) are sampled at low frequencies. Also, neurons generate 'events' according to their activity levels. More active neurons generate more events per unit time, and access the interchip communication channel more frequently, while neurons with low activity consume less communication bandwidth. When building multi-chip muti-layered AER systems, it is absolutely necessary to have a computer interface that allows (a) reading AER interchip traffic into the computer and visualizing it on the screen, and (b) converting conventional frame-based video stream in the computer into AER and injecting it at some point of the AER structure. This is necessary for test and debugging of complex AER systems. In the other hand, the use of a commercial personal computer implies to depend on software tools and operating systems that can make the system slower and un-robust.

This paper addresses the problem of communicating several AER based chips to compose a powerful processing system. The problem was discussed in the Neuromorphic Engineering Workshop of 2006. The platform is based basically on an embedded computer, a powerful FPGA and serial links, to make the system faster and be stand alone (independent from a PC). A new platform is presented that allow to connect up to eight AER based chips to a Spartan 3 4000 FPGA. The FPGA is responsible of the network communication based in Address-Event and, at the same time, to map and transform the address space of the traffic to implement a pre-processing. A MMU microprocessor (Intel XScale 400MHz Gumstix Connex computer) is also connected to the FPGA to allow the platform to implement event-based algorithms to interact to the AER system, like control algorithms, network connectivity, USB support, etc. The LVDS transceiver allows a bandwidth of up to 1.32 Gbps, around ~66 Mega events per second (Mevps).

1. INTRODUCTION

The Address-Event Representation (AER) was proposed by the Mead lab in 1991 [1] for communicating between neuromorphic chips with spikes (Fig. 1). Each time a cell on a sender device generates a spike, it communicates with the array periphery and a digital word representing a code or address for that cell is placed on the external inter-chip digital bus (the AER bus). Additional handshaking lines (Acknowledge and Request) are used to complete the asynchronous communication. In the receiver chip, the spikes are guided to the cell whose code or address was on the bus. In this way, cells with the same address in the emitter and receiver chips are virtually connected by streams of spikes. These spikes can be used to communicate analog information using a rate code, but this is not a requirement. More active cells access the bus more frequently than those less active. Arbitration circuits usually ensure that cells do not access the bus simultaneously. Usually, these AER circuits are built using self-timed asynchronous logic by e.g. Boahen [2].

Transmitting the cell addresses allows performing extra operations on the events while they travel from one chip to another. For example, the output of a silicon retina can be easily translated, scaled, or rotated by simple mapping operations on the emitted addresses. These mapping can either be lookup-based (using, e.g. an EEPROM) or algorithmic. Furthermore, the events transmitted by one chip can be received by many receiver chips in parallel, by

[1] Authors want to thanks to the NSF for supporting the 2006 Workshop on Neuromorphic Engineering, where this work started. They also want to thanks to Giacomo Indiveri and Daniel Fasnacht for their collaboration in the ideas of the Future Super-AER platform described in Section 3.
National project SAMANTA-II covers the development of the platform.

Bioengineered and Bioinspired Systems III, edited by Paolo Arena, Ángel Rodríguez-Vázquez, Gustavo Liñán-Cembrano, Proc. of SPIE Vol. 6592, 659206, (2007) · 0277-786X/07/$18 · doi: 10.1117/12.724156

properly handling the asynchronous communication protocol. There is a growing community of AER protocol users for bio-inspired applications in vision, audition systems and robot control, 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 massively-parallel data-driven processing in real time [5].

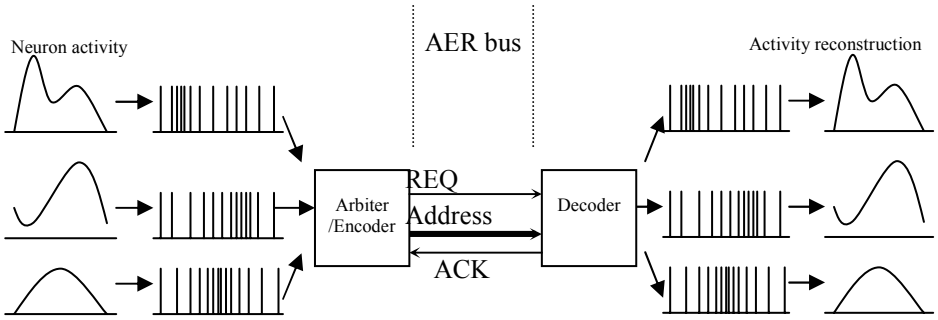

Fig. 1 Rate-coded AER inter-chip communication scheme. REQ is the Request line, ACK is the Acknoledge line and the bus Address hold the neuron code.

It is essential to have a set of instruments which make possible the right communication of these AER chips and can be used for debugging purposes. So this set of instruments has to allow the following tasks: (a) to sequence: to produce synthetic AER event streams [6] that can be used as controlled inputs while testing and adjusting a chip or a set of them, (b) to monitor: to observe the output of any element in the system, and (c) to map: to alter the stream produced by an emitter and send it to a receiver.

There is a set of AER tools based on reconfigurable hardware (FPGA) that can be connected to a computer. They achieve these purposes with a very high AER bandwidth, but with the necessity of a PC for Event processing purposes [7].

There also exists a very powerful set of software tools to be used under a PC for processing and generating AER signals and to simulate AER systems behaviours [6][9].

A new philosophy was born in the last Workshop on Neuromorphic Engineering (Telluride, 2006) to improve this, which is based in the use of an embedded GNU/Linux system over a relatively powerful microprocessor in junction with a FPGA. They are intended to be able to implement the FPGA based circuits and the microprocessor algorithms in a stand alone way.

In Section 2, we review the state of the art of the AER tools[2] actually developed and used by several researchers, and we discuss the fusion of all the functionality. In Section 3, we present a first prototype of this fusion, the Super-AER board. Then, in Section 4, we present some testing results. And finally, Section 5 presents the conclusions and the objectives to cover with the Future Super-AER board.

2. AER TOOLS STATE-OF-ART.

2.1. Rome PCI-AER

This was the first AER-tool developed [3]. It was developed by ISS Laboratory of Physics in Rome by Vittorio Dante and Paolo Del Giudice. The interface consists into a PCI board with 2 FPGAs and a PCI commercial bridge. The interface was able to sequence, monitor and map a list of Address-Event-Representation spikes. For sequencing, the PC was supposed to be in charge of producing the list of events and their corresponding time-stamps. This was used to indicate to the hardware the correct Inter-Spike-Interval for the event transmission. This list of events was transmitted through PCI bus to the hardware by the use of a FIFO. The events in the FIFO were transmitted through a 4-to-4 arbiter. This arbiter managed incomings events from up to 4 AER ports. It was also able to retransmit them to up to 4 output ports. This arbiter could retransmit the events using a mapping table, so they could be translated according to a mapping

[2] http://www.atc.us.es/AERtools

table. Although this is a powerful interface and is able to act as a center of communications of an AER system, it presents two main disadvantages: a PC is needed (it is not for stand alone purposes) and it has a traffic limitation of 1Mevent/s, in the best case.

2.2. CAVIAR PCI-AER

This was the second AER tool developed. It was developed under the CAVIAR EU project, in which the Rome PCI-AER board was used as a start point. The considerations for that project were to have a set of AER tools for testing a debugging purposes, more specialized, with high bandwidth and allowing stand alone configurations. The design of the CAVIAR PCI-AER board was focused in the high bandwidth consideration, so it was developed including the bus mastering and just for sequencing and monitoring events. It could retransmit a sequence of events obtained by software from static sequences of video frames [6].

A Windows driver and an API that implements bus mastering and a Matlab interface are currently available. A Linux version of the driver is also available too.

2.2.1. PCI-AER INTERFACE: Hardware design

The final goal is to transmit an AER sequence to an AER based system (for example a convolution chip) in order to perform video processing. An adequate sequence of events can be generated by software for testing an AER based system. This sequence of events needs to be sent to it. For this purpose, an interface between the computer and the AER bus is needed. The interface's architecture consists of an output FIFO (OFIFO) and an Out-AER state machine to sequence events, and it also consists of an input FIFO (IFIFO) and an In-AER state machine to monitor events. The CAVIAR PCI-AER has two operation modes that can work in parallel:

A. From PCI to AER.

The AER-stream is stored in the computer memory and then it is sent to the AER system through the PCI bus and the OFIFO. This stream is saved in memory using 32 bits for each address event. The sixteen less significant bits represents the address of the pixel that is emitting the event. The sixteen more significant bits represent a time difference from the previous occurrence, in clock cycles. The clock cycle is 30 ns but can be scaled up 16 times. Special words can be used in the OFIFO to make the state machine to wait the maximum time, coded with 16 bits, and then it reads a new word of the same OFIFO without any event transmission. If it is enabled, the OUT-AER state machine keeps continuously reading 32-bit words from it FIFO. For each word, the state machine will wait for the configured number of clock cycles before transmitting the address through the AER output bus. If the acknowledge is delayed, the timer of the OUT-AER state machine will discount this time to the wait counter of the next event. If the result of the discount is negative, no wait will be done for the next event and this value will be used as initial wait for the following event. So the delay between events is not relative to the previous one with this treatment, and a delay in the ACK reception will not cause a distortion in the time distribution of all the events along the time period.

B. From AER to PCI.

The AER sequence arrives to the CAVIAR PCI-AER interface through the input AER port. The AER-IN state machine stores the incoming event (16 LSbits) into the IFIFO with temporal information (16 MSbits). This temporal information is the number of clock cycles since the last event.

The connection to the PCI bus is done by a VHDL bridge [10] that attends to the Plug & Play protocol of the PCI bus, decodes the access to the base address by the operating system and allows the bus mastering and interruptions.

2.3. USB-AER

This was the third AER-tool. It was also developed under the CAVIAR EU project. The USB-AER board allows standalone operation and has several functionalities, like hardware mapping and more.

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, like MMC/SD cards. However, USB input is also provided for development stages.

The USB-AER board includes a relatively large FPGA (Spartan-II 200) that can be loaded from MMC/SD or USB (through the C8051F320 microcontroller), a large SRAM bank (512Kx32 12ns) and two AER ports. Thus the board can be used as a sequencer, a monitor, or an event processor of the AER bus. 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.
The board will act as a different device according to the module that is loaded in the FPGA (through MMC/SD or USB):

2.3.1. Mapper

Mapping functionality consists in a change of the events addresses. Theses changes can be: (a) one to one: this mapping is used in one to one communication, where the addresses on the sender device are different from the ones on the receiver. (b) One to one replicated: This mapping replicates, for each input event, up to 8 times the same output event. (c) One to several: This mapping is used in one to several communications. In this case, one input event is sent to several output addressed. (d) No event with a probabilistic function: this mapping emits or not the mapped event according to a probabilistic function.

2.3.2. Monitor

This functionality allows to record the traffic in an AER bus and to represent it in a VGA or in a computer. Monitor types are: (a) Frame-grabber: the USB-AER board is inserted in the AER bus, and captured the events during a time; then reconstructs an image and sends it to the computer using the USB port. (b) Sniffer: the board is connected to the AER bus without interfering in the traffic. In this case, the sniffer must be faster than the other AER devices connected to the AER bus. The USB-AER captured the events and stores it with a timestamps in the SRAM; then it sends the information to the computer using the USB port. (c) Logger: The USB-AER board is inserted in the AER bus, and captures the events with their timestamps. The difference from the *sniffer* is that the logger interferes with the AER traffic. (d) VGA: the board captures the event and convert it in a B&W image and send it using the VGA connector; or a Gray image, using a VGA-DAC board connected to the out-AER connector of the board.

2.3.3. Sequencer

In this case the USB-AER board is used as an event generator; there are two types of AER traffic generation: (a) Frame-based generation: the FPGA generates events from a digital image. Computer sends a digital image to the board and it converts it in an event stream that can be transmitted through AER-Out bus. Several methods [6] can be used to convert the image to events. (b) Player: the board generates events using timestamp information. This information is sent to the board using the USB port, or this information can be the information captured using the board as a logger o as a sniffer.

The USB-AER board was interfaced in Telluride 04 and Telluride 06 [4] to the current version of the CAVIAR retina, to an AER cochlea developed at INI and to an imager developed at JHU. In the CAVIAR last review, November 06, it was interfaced to all the project chips into a demo for tracking a moving ball.
A Windows XP driver and a MATLAB interface is available. Furthermore, a Linux driver is also available which lets the USB-AER board to be easily integrated with several MATLAB applications developed at INI [9].

2.4. SWITCH-AER

This board allows connecting several AER devices to an AER BUS. The connection possibilities are: 4 to 1 or 1 to 4. This board allows: (a) the connection of more complex AER systems. (b) An easier debugging by inserting PCI-AER or USB-AER board without modifying the structure of the global system under testing.
The board has a CPLD as a communication centre that manages the different modes and controls the protocol lines. It can work in two modes: (a) Merger: 4 input and 1 output, or (b) Splitter: 1 input and 4 output mode. Splitter mode can be unicast (selecting one output) or broadcast. Both functionalities should be configured by jumpers. There are 5 different AER ports; one of them works always as an output, and another as an input. The others three are bidirectional and work as input or output depending on the mode of operation.

2.5. USB2AER

This board is the USB 2.0 high speed version of the PCI-AER board [12]. Basically the platform allows to sequence and monitor event at a maximum rate of 5 Mevps and a sustained rate of 3Mevps. A Windows XP driver and a Java API for MATLAB and a Java application for recording and playing sequences of time stamped events is available.

2.6. SUPER-AER: Necessity of fusion.

All these AER-tools have complementary functionalities. All of them are needed to implement an AER system, like the visual processing AER system based on convolutions developed into CAVIAR EU project [8]. It was used to detect and follow a ball in move. In this system, an AER retina detects the temporal contrast changes and sends the information into AER format to a 4 convolution chips for different ball sizes detection. The 4 convolution AER output are joined into the same AER bus, using an AER merger. The information is sent to an object Winner-Take-All chip that filters the output of the processing and tells just the center of the detected object. This output can be used to control DC motors or Servo motors to follow the recognized object.

In this system, the USB-AER was used for mapping functionalities, adapting and transforming the AER information between the chips, the Switch-AER was used for merging and splitting the AER buses of the chips, and the PCI-AER was used for debugging purposes to sequence and monitor an AER stream generated by the computer. This functionality can be also supported by the USB-AER for small AER streams that can be stored into the SRAM memory of the board (up to 512 events).

Although this kind of systems can be connected, debugged and used for a control application, the system grows considerably because of the huge amount of boards and the size of them. For this reason, it is necessary to develop a new, relatively small and very powerful AER-tool that is able to support the functionalities of the other AER-tools.

This new tool should integrate a FPGA for the VHDL circuits support, an embedded microprocessor for complex event-based algorithms implementation, several AER ports for bidirectional communication to AER chips and other AER boards, network and/or USB connectivity to a PC for configuration purposes. A first prototype with a reduced set of functionalities is described in the next section.

3. PROTOTYPE OF THE SUPER-AER: Reduced functionality

To cover the functionalities described in the previous section, a huge FPGA is needed because the amount of AER ports needed to be connected to the FPGA for switching, mapping and connectivity purposes, apart from the lines needed to connect the embedded microcomputer, the microcontroller used for USB2.0 support and the SRAM memory. So the design of a very ambitious board using a FPGA, with around one thousand available pins, is needed for the Super-AER real board. This implies to use the BGA package, what it is not possible to solder in our laboratory. For these reasons, we decided to implement a first prototype, with a reduced functionality, for testing the different parts before the fabrication of the final version.

This prototype is composed by the fundamental components, but reducing the number of pins to be able to use the Spartan-3 400 FPGA, with 208 pins. Therefore, the prototype will have the Embedded Microprocessor connected to the FPGA through the general purpose input / output pins (60 GPIO) instead that through the Address / Data buses to allow Direct Memory Access (DMA), what is as fast as the system bus of the microprocessor selected. The embedded microprocessor is the Intel XScale PXA255 400MHz. This 32 bit processor offers 32KB of cache memory for data and the same amount for instructions, an MMU, 84 GPIO ports that can be programmed to work as function units to manage serial ports, I2C, PWM, LCD, USB client 1.1, ... The processor is connected to 64MB of RAM and 16MB of Flash Memory as the storage medium for the OS root file system. Another board is attached to the processor's one, providing wireless connectivity to the platform (IEEE 802.11b). This hardware is governed by a multi-task general purpose operating system. It is based on a Linux kernel 2.6, with only architecture dependent patches applied to its sources. The whole system, and obviously the cross-compile tool chain, is compiled using the uClibc[3], a C library for developing embedded Linux systems, which supports shared libraries and threading. This lets the application's binaries to be lighter. No other change has been done to the system referred to a common GNU/Linux one. The user console and the debug one are set to a serial port. Two services are the other provided user interfaces, a remote secure shell server and a HTTP one.

Furthermore, the prototype board supports the USB 2.0 high speed through the CypressFX2 8051 microcontroller. The Super-AER prototype is compatible with the USB-AER board thanks to the Cygnal F320 microcontroller, which support USB2.0 full speed and SD/MMC connectivity.

Two parallel AER busses and 4 LVDS SAER busses, based on SATA connectors, are used. Two of the SAER busses are managed directly from the FPGA and the other two are managed through Ser and Deser commercial chips (Texas Instrument SN65LV1224 and SN65LV1023), which support a full bandwidth of up to 1.32Gbps (full-duplex).
Fig. 2 shows the block diagram and the photograph of the prototype.

[3] [http://uclibc.org]

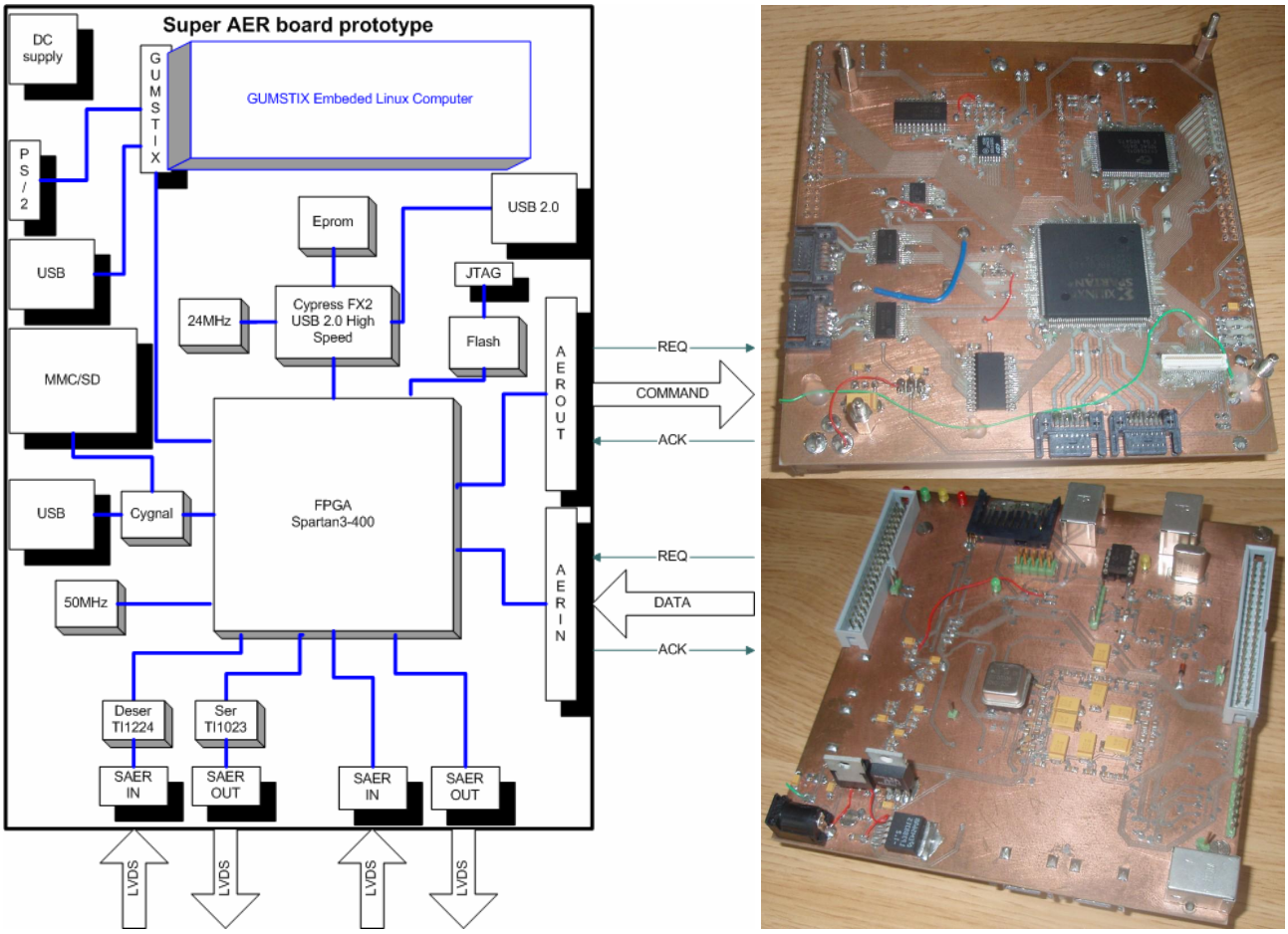

Fig. 2 Super-AER prototype platform block diagram (left) and photograph (right).

4. TESTING THE PLATFORM

In this section we present two experiments for testing the platform: (a) a serial AER based on LVDS transmission over SATA cables and connectors offering the eye diagram; (b) a Microprocessor implementation of a frame-grabber with httpd service over wireless.

4.1. SAER over LVDS Transceivers.

A simple experiment was implemented to analyze the performance of a serial AER bus (SAER), based into LVDS, using commercial ser-deser chips. We demonstrate that SAER is possible and it has a lot of possibilities from the point of view of speed, scalability, and easiness to implement. The experiment consists on programming a simple VHDL code, for the Spartan 3 400 FPGA of the Super-AER first prototype platform, and testing the maximum speed supported by LVDS serializer and deserializer transceivers. The VHDL code describes a simple state machine that generates continuously 10-bit word to transmit them through the serializer chip. It can work with a clock frequency of up to 66 MHz. We have used a 50MHz clock. The serial output, a LVDS signal, is sent using a 100cm SATA cable connected to the SATA receiver connector, which is connected to the deserializer chip. The VHDL code is receiving the 10 bits data continuously and is comparing the transmitted word with the received one.

The two Texas Instrument chips transmit both data and clock into the same differential signal. Fig. 3 shows the oscilloscope LVDS signal and the clock extracted by the oscilloscope. The transmission works at 600 Mbps. It also shows the eye diagram of the differential pair when the experiment is running. It can be seen that for this cable length (100 cm) the speed transmission can be almost twice higher.

Table 1 shows the parameters of both implementations.

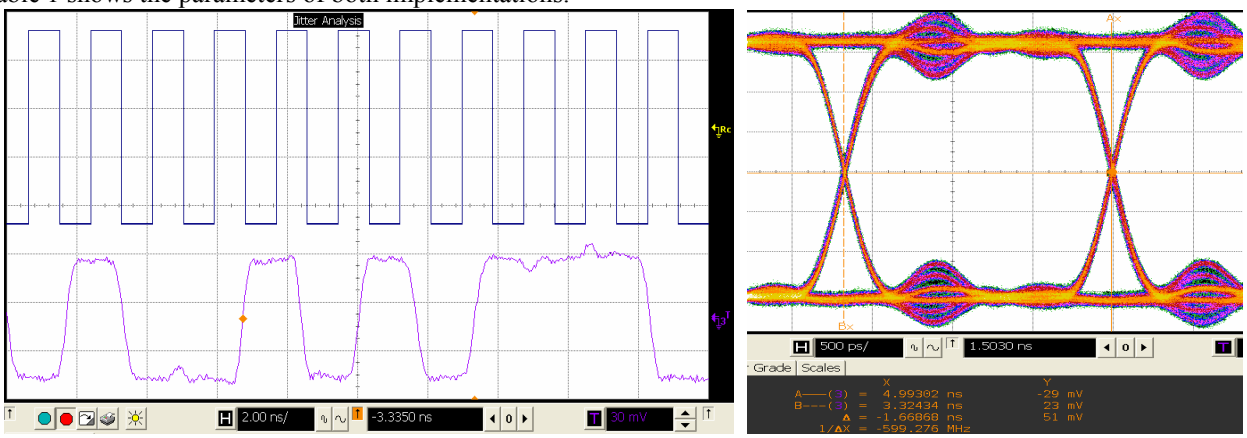

Fig. 3 SAER link oscilloscope with clock extraction (left). SAER link eye diagram (right)

Parameter	Experiment
Total Jitter	150 ps
Symbol duration	1,66 ns
Bit rate	~600 Mbps
Reference clock	50 MHz
Event rate (16-bits)	31,25 Mev/sec

Table 1. SAER link eye diagram parameters.

4.2. Software Implementations in an Embedded System: The Gumstix computer.

This experiment consists on implementing the frame-grabber functionality of the USB-AER tool [7]. The USB-AER frame-grabber was continuously collecting events from the AER bus for a defined period of time, called integration time. The frame was constructed by computing each event when it was received and it was sent to a computer via USB protocol when the integration time had expired. The events that were received during the frame transfer to the computer were lost. This process was restarted again when the transfer was finished.

Using the microprocessor GPIO ports, we have developed several implementations of the frame-grabber and we have compared them to the ideal case: activating the Acknowledge signal when the Request one is active and with no data processing, so in the ideal case we suppose that the processing time of the data doesn't decrease the bandwidth in the AER bus. In this way we will evaluate the performance of this multi-task environment when processing Address-Event-Representation data.

The job of constructing a frame from AER events can be divided into two tasks. One is to acquire the events from the AER bus and the other is to construct the frame from those acquired events. This conceptual scheme can be quickly translate to a "double-buffering" implementation, thinking in the fastest way to do it in a multi-task environment. There are two approaches to this programming technique implementation in this scenario. One is the use of one process for each task and the other is the use of one process for both, implementing these tasks as threads. We will call them "processes implementation" and "threads implementation", respectively.

Both implementations have the same philosophy. Events will be continuously collected and put into a buffer. When this buffer is full, a signal will be sent to the other process or thread and the new received events will be put into the other buffer. So, this is a worst-case linear time algorithm that will let to continuously receive events. The other process or thread will be generating the frame into memory from a buffer or waiting to receive a signal. Therefore, it is also a worst-case linear time algorithm which let to continuously generate the frame or wait until a buffer is ready for its treatment. When a signal is received, the reference to the appropriate buffer will be changed depending on the received signal. This is a worst-case constant time algorithm for the signal handler that let the double buffering buffer-change to be implemented.

We will use IPC Shared Memory method in the first implementation and global variables in the second one, which makes both implementations equivalent from the access to memory point of view. "Polling" will be used to implement

the event acquisition for both. So, they are also equivalent in this other sense. Therefore, the difference between the implementations takes place in how they are affected by the operating system scheduler.
Finally, another process will be used for debugging purposes, independently of the double buffering implementation. This process will be waiting to receive a signal that will be periodically sent by the operating system. Then, it will wake up and put the frame in memory into a BMP file. This last could be viewed by connecting to the HTTP server on the platform. Also, they will be used to test the implementations under situations with other processes running.

There are two main parameters which define the operating system scheduler influence, the scheduling policy and the frequency of the timer interrupts.
The scheduling policy determines how the processes will be executed in a multi-task operating system. The Linux kernel 2.6 version presents several ones. These can be chosen without recompiling the sources. The kernel offers system calls to let the processes to choose the scheduling policy that will rule their execution. A dynamic priority based on execution time scheduling policy, a real-time fixed priority FIFO one and a real-time fixed priority round robin one are offered by the kernel. The first one is the common policy on UNIX systems. The other two scheduling policies differ from each other in how processes with the same priority are reorganized to take the microprocessor again, using a FIFO criterion or a round robin one, respectively. A process whose execution is managed by one of these two policies is, obviously, not influenced by the first of all. Even more, preference will be given, of course, to a process in these scheduling situations than the managed by the first policy ones.
The real-time scheduling policies try to ensure a short response time for a ruled by them running process. Also, no lower-priority processes should block its execution but this situation actually happens. The kernel code is not always assumed to be pre-emptive[4]. So a system call from a lower-priority process may block the execution of higher-priority one until it has finished. Therefore, the support for real-time applications is weak although the processes response time is improved referred to the common scheduling policy. Every process in a Linux system is normally ruled by the first one. Therefore, a process running continuously cannot be set to be ruled by one of the offered real-time policies without making the whole rest of the system unresponsive. We will use the common scheduling policy for our implementations in this study in order to evaluate the performance of them in a general purpose multi-task environment.
The frequency value of the timer interrupts is the other parameter that mainly influences on a multi-task operating system performance. The period of time assigned to a process for its execution in the microprocessor is generally called quantum, whose value is defined by the frequency of the timer interrupts one and is decremented each timer interrupt. Therefore, a more fine-grained resolution system can be achieved by raising it. On the other hand, an extra instruction overhead has to be paid due to a higher number of timer interrupts. This implies context switches from process to interrupt handler and from this last to the first, the handler execution, and possible cache and TLB[5] pollution, which may result in an impoverishment of the system performance. This value is set before the Linux kernel compilation process. The default one is 100Hz for the ARM architecture. We have study the performance of both implementations under this default value and a 1000Hz one. The results will be presented in the next section.

Fig. 4 shows the Event Reception Rate (EvRR) over the time for the two implementations referred before (using processes or threads). It is mainly stable at its highest value, which is briefly decreased due to the processor assignment to other processes. This reduction evolves sometimes to a harsh value when the frequency of the timer interrupts is set to the default, 100Hz. This undesirable value is 200keps for the processes implementation and 259keps for the threads one. The processes based implementation presents an EvRR oscillating from 530keps to 450keps for a frequency of timer interrupts of 100Hz and from 500keps to 430keps for 1000Hz, being the first values the stable ones. The intervals for the threads implementation are 770keps to 620keps and 770keps to 660keps, respectively, being again the stable ones. These stable values are the unique ones when no other user process is running, and so we called them the stable values.
We have also implemented an application which only performs the response to the handshake protocol. No event direction is calculated from the microprocessor GPIO ports or event storage is done. The EvRR is 1'3Meps in this case, which implies a time between events of 760ns. We have also measured the time between events when there is either the event direction calculation and its storage into a buffer during the handshake protocol. The result is 1'16µs, which should be the ideal case. We have also set the implementations to be ruled by the round robin priority fixed real-time scheduling policy, achieving an EvRR of 840keps. Therefore, the time between events is 1'19µs. This value is near the ideal one but as we explained before, and so expected, the system was unresponsive for other tasks. The threads

[4] It has to be compiled with this option and it is only supported in 2.6 versions.
[5] Translate Lookaside Buffer, a cache used to improve the speed of virtual address translation containing parts of the operating system's page table.

implementation presents 770keps, which implies that it performs the event acquisition and the event treatment with a time between events of 1'3µs, approximately. Therefore, it offers a multi-task environment useful for other simultaneous tasks with an 11% deviation from the ideal.

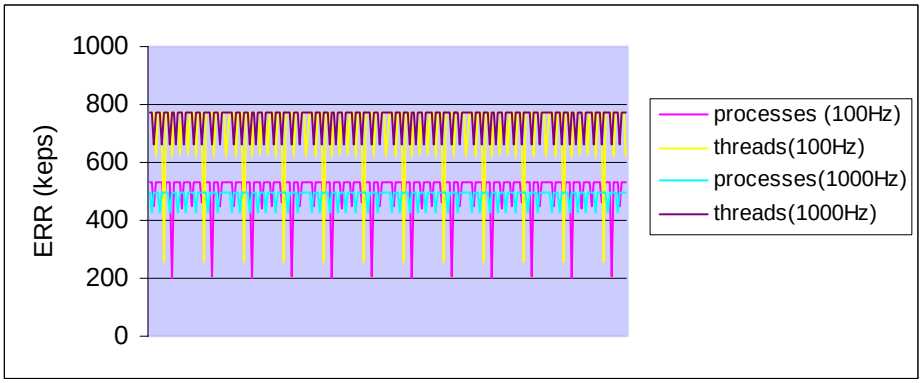

Fig. 4 Event Reception Rate (EvRR) in Kilo-events per second (keps) over the time for all the cases of study.

Although the processor offers a mechanism to detect any level change at any of its GPIO ports, generating hardware interrupt when it occurs, the minimum pulse width duration to guarantee this detection is 1µs [11]. Therefore, the time between events is, at least, 2'4µs, because the time due to interrupts handlers overhead, context changes … are not considered and so this option was ruled out. To obtain higher bandwidth the Direct Memory Access (DMA) solution has to be implemented that implies a future work for the next version of this platform.

5. CONCLUSIONS

A very powerful prototype of a new philosophy of AER-tools is presented in this paper. This new platform has the principal aim to integrate the functionalities and advantages of the previous ones. In this line, this prototype has been fabricated in order to test the viability of the platform. This is demonstrated with the two most critical experiments offered in this paper (SAER over LVDS and Microprocessor functionality).

The next version of the platform will have approximately the same components, but will include the SRAM memory accessible from the FPGA, another kind of connector for supporting other manufacturer of embedded computers (Toradex), and the most important: up to eight SAER based on commercial Ser-Deser for LVDS, what will allow to implement dynamic switching of the AER communications in Real-Time.

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

[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] A. Cohen et al., Report to the National Science Foundation: Workshop on Neuromorphic Engineering, Telluride, Colorado, USA, June-July 2006. [www.ine-web.org]

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

[6] A. Linares-Barranco, G. Jimenez-Moreno, A. Civit-Ballcels, and B. Linares-Barranco. "On Algorithmic Rate-Coded AER Generation". *IEEE Transaction on Neural Networks*. May-2006.

[7] R. Paz, F. Gomez-Rodriguez, M. A. Rodriguez, A. Linares-Barranco, G. Jimenez, A. Civit. Test Infrastructure for Address-Event-Representation Communications. IWANN 2005. LNCS 3512. pp 518-526. Springer Verlag.

[8] 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, NIPS'05, December-2005.

[9] E. Chicca, A. M. Whatley, P. Lichtsteiner, V. Dante, T. Delbruck, P. Del Giudice, R. J. Douglas, and G. Indiveri. “A multi-chip pulse-based neuromorphic infrastructure and its application to a model of orientation selectivity”. IEEE Transactions on Circuits and Systems I. Accepted for publication on 2007. [http://ieeexplorer.ieee.org]

[10] R. Paz. “Análisis del bus PCI. Desarrollo de puentes basados en FPGA para placas PCI”. Trabajo de investigación para obtención de suficiencia investigadora. Sevilla, Junio 2003.

[11] Intel Press, Intel PXA255 Processor Developer’s Manual, 2004.

[12] R. Berner, et al., "A 5 Meps $100 USB2.0 Address-Event Monitor-Sequencer Interface," in ISCAS 2007, 2007, pp. (in press).