Spike Processing on an Embedded Multi-task Computer: Image Reconstruction

Carlos Luján-Martínez, Alejandro Linares-Barranco, Manuel Rivas-Pérez, Ángel Jiménez-Fernández, Gabriel Jiménez-Moreno and Antón Civit-Balcells*

[1]Computer Architecture and Technology Department,
University of Seville, Seville, Spain
{cdlujan,alinares,mrivas,ajimenez,gaji,civit}@atc.us.es

Abstract — *There is an emerging philosophy, called Neuro-informatics, contained in the Artificial Intelligence field, that aims to emulate how living beings do tasks such as taking a decision based on the interpretation of an image by emulating spiking neurons into VLSI designs and, therefore, trying to re-create the human brain at its highest level. Address-Event-Representation (AER) is a communication protocol that has embedded part of the processing. It is intended to transfer spikes between bioinspired chips. An AER based system may consist of a hierarchical structure with several chips that transmit spikes among them in real-time, while performing some processing. There are several AER tools to help to develop and test AER based systems. These tools require the use of a computer to allow the higher level processing of the event information, reaching very high bandwidth at the AER communication level. We propose the use of an embedded platform based on a multi-task operating system to allow both, the AER communication and processing without the requirement of either a laptop or a computer. In this paper, we present and study the performance of a new philosophy of a frame-grabber AER tool based on a multi-task environment. This embedded platform is based on the Intel XScale processor which is governed by an embedded GNU/Linux system. We have connected and programmed it for processing Address-Event information from a spiking generator.*

1 Introduction

The Address-Event-Representation, AER, was proposed by the Mead lab in 1991 for communicating between neuromorphic chips with spikes [1], as was mentioned before. Figure 1 shows the principle behind the AER. Each time a cell on a sender device generates a spike, it communicates with the array periphery. A digital word representing a code or address for that cell is placed then on the external inter-chip digital bus, the AER bus. This word is called event. Additional handshaking lines, Acknowledge and Request, are

*This work was supported by Spanish grant TEC2006-11730-C03-02 (SAMANTA 2).

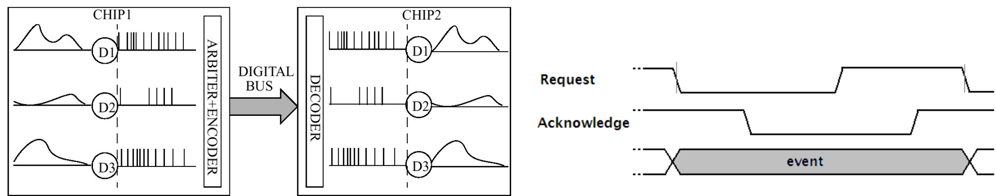

Figure 1: Rate-coded AER inter-chip communication scheme.

used for completing the asynchronous communication. In the receiver chip, the spikes or events are guided to the cells whose code or address appeared 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, by relating the analog information to the time between two spikes that correspond to the same neuron, although this is not a requirement. More active cells access the bus more frequently than those that are less active. The use of arbitration circuits usually ensure that cells do not access the bus simultaneously. These AER circuits are generally built using self-timed asynchronous logic by e.g. Boahen [2].

In addition, transmitting the cell addresses allows performing extra operations on the events while they travel from one chip to another, making AER not only a communication channel. 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 properly handling the asynchronous communication protocol.

In artificial vision systems based in AER, it is widely used the *rate-coded* AER e.g., [3], [4], [5], [6], [7], [8] and [9]. In this scheme, each cell corresponds to a pixel and its activity is transformed into pixel event frequency. This scheme may be inefficient for conventional image transmission: Monochrome VGA resolution[1] yields a peak rate of (480$\times$640 pixels/frame) $\times$ (256 spikes/pixel) $\times$ (25 frames/s) $\times$ (19 bit/spike) = 37 Gbit/s. On the other hand, the lost of some events does not mean a degradation in the application when using *rate-coded* AER. Let suppose a pixel which intensity is 255, considering 256 gray levels. Its corresponding event should appear 255 times during the time frame, being this time the one at which the events in the bus corresponds to the same frame. If some events from that pixel are lost, the receptor will also interpret that pixel as one of higher intensity. Therefore, not only most information, even more, enough information has been received although some has been lost. So *rate-coded* AER scheme, increases the tolerance of the whole system. Also, preprocessed images are usually transmitted instead of raw images, such as edges or contrast [3], in which 20 gray levels are satisfactory and only a small percentage of all pixels, between 1-10 %, will present appreciable contrast. Therefore, previous full VGA peak rate is reduced in two or three orders of magnitude. In addtion, present day AER hardware uses image resolutions between 64$\times$64 and 128$\times$128 pixels at the most, thus adding another one or two order reduction in the peak rate. So,

[1] 480$\times$640 pixel frames, at 25 frames per second, with 8 bits per pixel.

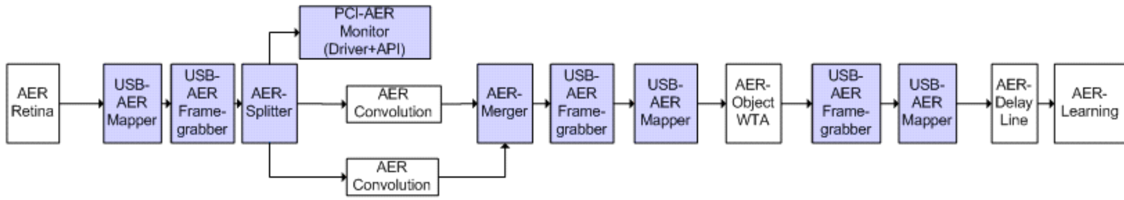

Figure 2: CAVIAR Scenario.

most of the AER hardware systems supports *rate-coded* AER.

In general, AER is useful for multistage processing systems, in which as events are generated at the front end. They travel and are processed down the whole chain (without waiting to finish processing each frame). Also, in multistage systems, information is reduced after each stage, thus reducing the event traffic. A design of a neuromorphic vision system totally based on AER has taken place under the European IST project CAVIAR, *"Convolution Address-Event-Representation (AER) Vision Architecture for Real-Time"* (IST-2001-34124) [10]. Figure 2 shows the AER system mounted under CAVIAR. This chain is composed by a 64×64 retina that spikes with temporal and contrast changes [11], two convolution chips to detect a ball at different distances from the retina [9], an object chip to filter the convolution activity [12] and a learning stage composed by two chips: delay line and learning [13]. The maximum throughput rate takes place at the output of the silicon retina. Although it is able to emit 4 Mevents/s, real applications, such as someone walking along a corridor or even the beat of an insect wing, vary from 8 to 150 Kevents/s [11], respectively.

There is a growing community of AER protocol users for bioinspired 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 [14]. 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 [15]. These complex systems require interfaces to interconnect them and to connect them to PCs for debugging and/or high level processing. There is a set of AER tools mostly based on reconfigurable hardware that achieve these purposes with a very high AER bandwidth, as shown in "Table 1", but with the need of a PC for the higher level processing. Generally, buffers of event streams are prepared on the PC e.g., [16], and sent via these AER-tools to the AER bus or an obtained event stream is sent to a PC and a high level processing is done then, such as learning algorithms for the VLSI neuronal network, development of connectivity, models of orientation selectivity, which are not always easily portable to pure hardware solutions e.g., [17] and [18].

A new philosophy was born at the last Workshop on Neuromorphic Engineering (Telluride, 2006) to improve this, which is based in the use of an embedded GNU/Linux system running over a relatively powerful microprocessor with network connectivity. This will let neuromorphic engineers to use AER standalone platforms for high level event processing when developing or building AER systems.

We present in this paper a totally microprocessor based solution, where the AER bus

AER-tool Name	Event Rate
Rome PCI-AER	1 Mevents/s
CAVIAR PCI-AER	8 Mevents/s
USB-AER	25 Mevents/s
mini USB-AER	300 Kevents/s
USB2AER	5 Mevents/s

Table 1: Event Rate for some previous AER-tools in chronological order (first to last). The communication to or from the PC is done by the PCI bus or the USB protocol. Rome PCI-AER [14], USB2AER [19], CAVIAR PCI-AER and USB-AER [20] are based on reconfigurable hardware, FPGA, and an C8051F320 MCU is used for mini USB-AER [21].

is connected directly to it by using its general purpose I/O ports, as a first approach and in order to study the advisability of its use within AER based systems. We will solve the image reconstruction from event streams problem for this purpose, which requires a high AER bandwidth when no preprocessing is done and will let evaluate the performance of the embedded system. Also, we have compared them with other hardware solutions. Therefore, there are either no reconfigurable and specific hardware to manage the AER traffic or to process the event information.

2 Spike Processing over Multi-task

2.1 The Platform

The platform is composed by a powerful embedded processor and a multi-task general purpose operating system. The first one is the Intel XScale PXA255 400 MHz. This 32 bit processor offers 32 KB of cache memory for data and the same amount for instructions, an MMU, 84 GPIO[2] ports that can be programmed to work as function units to manage serial ports, I2C, PWM, LCD, USB client 1.1,... This processor is connected to 64 MB of RAM and 16 MB 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.

[2] General Purpose Input Output.

[3] [http://uclibc.org] (2007)

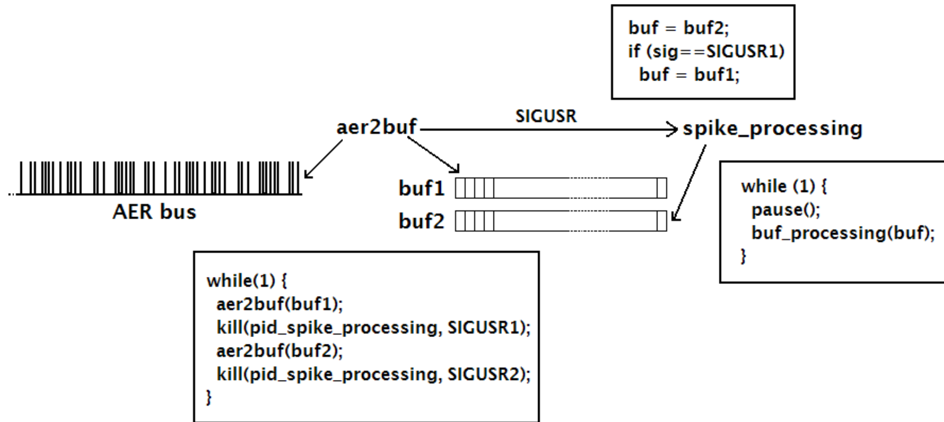

Figure 3: Software architecture for high level spiking processing over a multi-task system when receiving events from the AER bus.

2.2 Software Architecture

Although AER allows performing extra operations on the events while they travel from one chip to another, as mentioned before, high level spike processing is not applied individually to one event but a set of them. So, we propose a double-buffering scheme for the AER communication and this high level event processing on this system, splitting up both into two concurrent tasks, trying to make the most of the time between events arrivals for spike processing. Also, this separation makes the development of this kind of applications easier. Only special spike processing has to be developed due to the AER communication is obviously always the same.

AER was developed for multiplexing in time the spike response of a set of neuro-inspired VLSI cells. Neuro-inspired cells are not synchronized. They send a spike or event when they need to send it and the AER periphery is responsible to send it into AER format with the minimum possible delay, and therefore, the AER scheme is asynchronous. As the event arrival is asynchronous, the event buffer filling is also asynchronous. We propose the use of signals, which are asynchronous too, for notifying the double-buffering buffer exchange.

When a process receives a signal, it processes the signal immediately, without finishing the current function or even the current line of code. The operating system stops its execution and assigns the processor to the signal handler that has been registered for that signal. A signal handler should perform the minimum work necessary to respond the signal and return control to the main program then. So, we suggest a buffer references exchange to the appropriate buffer depending on the received signal as the signal handler.

High level spike processing will be applied to a set of events, so it will be done when a buffer has been filled. Figure 4 shows the three possibilities based on the signal communication and the task latencies. If filling the buffer, either at AER communication level or computing them, lasts more than consuming it, every event will be treated. If not, the

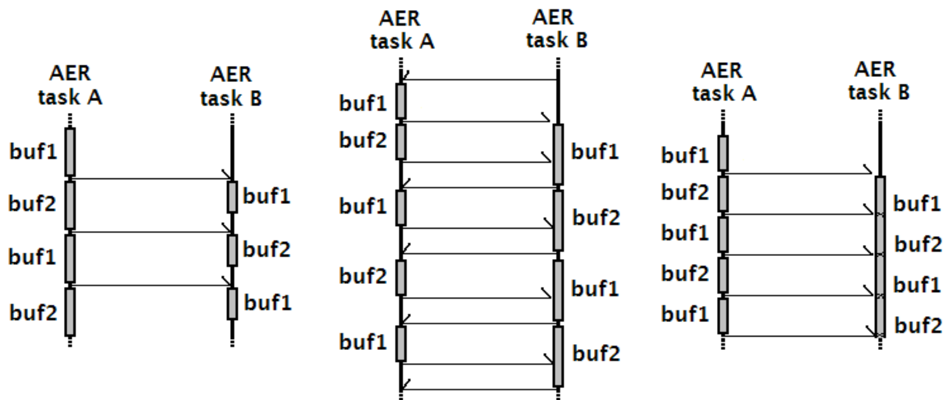

Figure 4: Double-buffering driven execution flowchart. AER task A, filling the buffer, drive the execution. In the left, filling the buffer lasts more than consuming it, AER task B. In the centre, an acknowledge signal is used when AER task B has finished using the buffer. In the right, AER task B aborts and starts using an updated buffer.

consuming task will work with the last updated events and, together with the *rate-coded* AER's feature of "losing some events does not necessary mean losing information", it does not implies to be always an undesirable situation. Finally, a returned signal from the task that consumes the buffer could be added to the scheme, as a "ready signal", ensuring the processing or the reception-emission of every event, independently of the latencies of both tasks.

The processor offers a mechanism to detect any level change at any of its GPIO ports, generating hardware interrupt when it occurs with a minimum pulse width duration to guarantee this detection is 1 μs [22]. It is necessary to detect the two Request signal levels to implement the AER hand-shake protocol. In addition, over 0.17 μs are needed to set a bit on a GPIO in this processor. Two sets have to be done for generating the AER Acknowledge signal. Therefore, the minimum time between events would be, at least, 2.34 μs. It should be greater considering the time penalty due to the interrupts handlers execution, context changes... which implies a event rate fewer than 427 Kevents/s only for the AER communication task.

AER communication is asynchronous, so either the number of consecutive events or the time between two of them can not be supposed. Free spikes or bursts of them can appear in the bus. Although hardware interrupts release the processor for computation tasks until data is ready at I/O, if spikes are presented as bursts of events the event rate will be reduced. Also, if there is no event traffic at the AER bus for a enough long period of time, there is no high level spike processing to do and so, there is no need to release the processor. Therefore this option may be ruled out, and polled I/O may be used.

From a computational point of view, both, filling a buffer from the AER bus or sending to it, makes the AER communication to be a worst-case linear time algorithm. The proposed double buffering buffer exchange is a worst-case constant time algorithm. Therefore, this software architecture presents worst-case linear time complexity, whose wors-

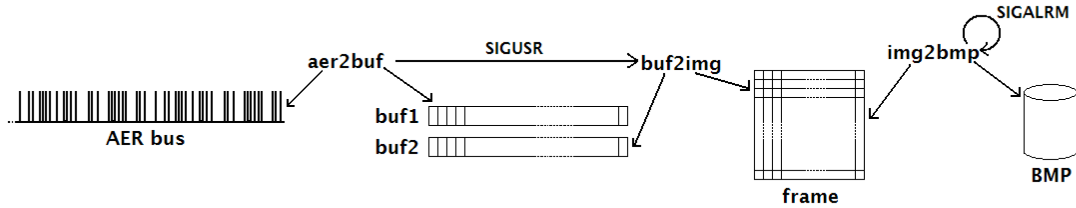

Figure 5: Software architecture for reconstructing the frame from an AER event stream.

ening may only take place at the high level spike processing task.

3 Image Reconstruction from Event Stream

Going from asynchronous AER to synchronous frame based representation video is more or less straightforward. If T_{frame} is the duration of a single frame, a 2-D video frame memory is reset at every time $t = n \times T_{frame}(n \in [0, \infty))$, called the integration time. Then, for each event address, the memory position for this address (x, y), is incremented by 1. Finally, the content of the 2-D memory is transferred to the computer screen and reset again at $t = (n+1) \times T_{frame}$. This is more or less how state-of-the-art AER hardware engineers visualize their AER systems outputs on computers [23].

There is a previous hardware implementation of a frame-grabber used as a monitor [20], based on this idea, that sends the frame via the USB protocol when the integration time has expired. The events that will be received during the frame transfer to the computer are lost. This process will be restarted again when the transfer will have finished. Each event is computed into the frame when it is received.

We propose an AER-communication driven execution policy with no "ready" like signal, which will decide the execution rate (see Figure 4). So, events will be continuously collected and put into a buffer, "aer2buf". When this buffer is full, a signal will be sent to the high level processing task and new received events will be put into the other buffer. The spike processing task, "buf2img", will be generating the frame into memory from a buffer or waiting to receive a signal, so it will only consume processor execution when it is needed. 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.

4 Results

We have developed a processes and a threads implementation for reconstructing a frame from an event stream. We use IPC Shared Memory method in the first one and global variables in the second one for the shared data, which makes both implementations equivalent from the access to memory point of view.

An USB-AER board will play the role of the AER emitter. It will be responsible to transform a binary representation of a frame into the corresponding events and to send them [24]. These will be sent to the platform via the AER bus, whose pins will be directly connected to the processor's GPIO ports. The frame is downloaded to the USB-AER

Test	Timer	No other process		Other processes	
		ER	WER	ER	WER
Processes	100 Hz	540	450	530	200
Threads	100 Hz	770	620	770	259
Processes	1000 Hz	500	430	500	430
Threads	1000 Hz	775	660	770	660

Table 2: Image reconstruction Event Rate, ER, and Worst Event Rate, WER, in Kevents/s, for: 1) processes and threads implementation; 2) under a system with a value of the frequency of timer interrupts of 100 Hz and 1000 Hz; and 3) when there is no and there are other processes running concurrently.

from the PC, no preprocessing is done, such as referred in Section 1, and it will continuously be sending the same frame translated into event streams. This board is able to achieve an event rate up to 25 Mevents/s. Having this event rate will let us to evaluate the performance of the embedded computer, which should be the bottle-neck.

An oscilloscope probe will be clipped to the Request signal pin and it will be used to measure the event rate, due to each cycle at this signal implies an event communication. The usual mechanisms to compute the execution time of a task and its duration, either provided by the hardware or the operating system, would interfere on the obtained value by incrementing it. So the need of including this kind of instructions is avoided by using the oscilloscope. The event rate will be the frequency of the Request signal, which will be calculated by it. The time that the process is ready to run and waiting to take the processor for its execution is also considered in this value, which makes it a real measure of the mean even rate for AER communication and spike processing.

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 can be viewed by connecting to the HTTP server on the platform. Also, these processes will be used to test the implementations under situations with other ones running.

4.1 Processes vs Threads Implementation

We have executed both implementations and studied their evolution over the time. The threads implementation achieves an event rate, ER, of 770 Kevents/s, while the processes one reaches 540 Kevents/s. These values are reached even if there are other processes running on the system and are mainly maintained over the time.

Both implementations present a momentary reduction of the ER. When no other process is running, these worst event rates, WOR, are 620 Kevents/s for the threads implementation and 450 Kevents/s for the other one. These oscillating values define event rate intervals that are relatively small but WOR evolves sometimes to a harsh value of 259 Kevents/s and 200 Kevents/s for each implementation, respectively, when there are other processes running on the system. Although these last WOR values appear momentarily, they suppose a main degradation of the spike processing performance.

4.2 Frequency Value of the Timer Interrupts

A more fine-grained resolution system can be achieved by raising frequency value of the timer interrupts, which not only implies a shortest process response time but a quicker turnover of scheduler's processes queue. 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[4] 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 100 Hz for the ARM architecture.

We have also study the performance of both implementations under a value of 1000 Hz. At this one, the event rate is not affected by the fact of other processes running on the system. The ER is 775 Kevents/s and WOR is 660 Kevents/s for the threads implementation and 500 Kevents/s and 430 Kevents/s, respectively, for the other one. So, it has been achieved that the influence of other processes on the event rate is transparent for a frequency value of the timer interrupts of 1000 Hz.

4.3 Scheduling Policies

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. Basically, a base priority is initially assigned to the process based on the frequency value of the timer interrupts. Its new priority is calculated by the scheduler when this last is executed using the execution time associated to the process. This priority will determine when the process will be executed again. 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, which is desirable when development an AER device. 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[5]. 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

[4]Translate Lookaside Buffer, a cache used to improve the speed of virtual address translation containing parts of the operating system's page table.

[5]It has to be compiled with this option and it is only supported in 2.6 versions.

cannot be set to be ruled by one of the offered real-time policies without making the whole rest of the system unresponsive.

We have set our threads implementation to be ruled by the real-time fixed priority round robin scheduling policy, achieving an event rate of 840 KEvents/s continuously maintained over the time. Therefore, the time between two consecutive events is 1.19 μs. This value is near the one, but as we have explained before, and so expected, the system was unresponsive for other tasks. If other processes e.g., network,... are needed, a combination of the scheduling policies at runtime based on the application state, receiving events or waiting for them, could increase the performance of the system with no degradation on the multi-task environment response.

4.4 AER Communication vs Spike Processing Tasks

We have also measured the exact time between events for the system using the oscilloscope, which is 1.16 μs. Therefore, the system presents an event rate of 862 Kevents/s without either the spike processing task or other processes running on the system. The threads implementation presents 770 KEvents/s, which implies that it performs the event acquisition and the event treatment with a mean time between events of 1.29 μs, approximately. Therefore, it offers a multi-task environment useful for other simultaneous tasks with an 11% deviation from the maximum that can be achieved with the system. Under the real-time round robin scheduling policy, the mean time between events is 1.19 μs, so spike processing implies an 2.5% from the maximum.

In Section 1, a neuromorphic vision system totally based on AER has been presented. The maximum throughput rate takes place at the output of the silicon retina and vary from 8 to 150 Kevents/s for real applications [11]. The higher demanding value, 150 Kevents/s, implies a mean time between events of 6.66 μs. The time of the reception of an event of our system is 1.16 μs. So, there is a mean time of 5.5 μs for any kind of high level spike processing, which means up to 2200 instructions on a 32-bit processor at 400 MHz.

5 Conclusion

We have presented a new philosophy of implementing a frame-grabber using a standalone multi-task environment directly connected to the AER bus and achieving up to 840 KEvents/s while constructing a frame. Although this rate is not as fast as those gotten by the hardware implementations of AER tools, it lets the execution of more than 2200 32-bit processor instructions between two spikes, as a mean value for high spike bandwidth on real applications, and so, allowing high level spike processing while performing AER communication.

Acknowledgements

We want to special thank our colleague at Computer Architecture and Technology Department from the University of Seville, Francisco Gómez-Rodríguez, for his great interest and his useful comments when carrying out this work. We would also like to thank the NSF sponsored Telluride Neuromorphic Engineering Workshop, where this idea was born in a discussion group participated by Daniel Fasnacht, Giacomo Indiveri, Alejandro Linares-Barranco and Francisco Gómez-Rodríguez.

References

[1] M.A. Sivilotti. *Wiring considerations in analog VLSI systems, with application to field-programmable networks*. PhD thesis, California Institute of Technology Pasadena, CA, USA, 1992.

[2] K. Boahen. Communicating Neuronal Ensembles Between Neuromorphic Chips, Neuromorphic Systems Engineering, 1998.

[3] K.A. Boahen and A.G. Andreou. A contrast sensitive silicon retina with reciprocal synapses. *Advances in Neural Information Processing Systems*, 4:764–772, 1992.

[4] M. Mahowald. *An Analog VLSI System for Stereoscopic Vision*. Kluwer Academic Publishers, 1994.

[5] A. Mortara and EA Vittoz. A communication architecture tailored for analog VLSI artificialneural networks: intrinsic performance and limitations. *Neural Networks, IEEE Transactions on*, 5(3):459–466, 1994.

[6] A. Mortara, EA Vittoz, and P. Venier. A communication scheme for analog VLSI perceptive systems. *Solid-State Circuits, IEEE Journal of*, 30(6):660–669, 1995.

[7] P. Venier, A. Mortara, X. Arreguit, and EA Vittoz. An integrated cortical layer for orientation enhancement. *Solid-State Circuits, IEEE Journal of*, 32(2):177–186, 1997.

[8] C.M. Higgins and C. Koch. A Modular Multi-Chip Neuromorphic Architecture for Real-Time Visual Motion Processing. *Analog Integrated Circuits and Signal Processing*, 24(3):195–211, 2000.

[9] R. Serrano-Gotarredona, T. Serrano-Gotarredona, AJ Acosta-Jimenez, and B. Linares-Barranco. An arbitrary kernel convolution AER-transceiver chip for real-time image filtering. *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, page 4, 2006.

[10] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, H.K. Riis, T. Delbrück, SC Liu, S. Zahnd, et al. AER Building Blocks for Multi-Layer Multi-Chip Neuromorphic Vision Systems. *2005 Neural Information Processing Systems Conference (NIPS 2005), Vancouver*, 2005.

[11] P. Lichtsteiner and T. Delbruck. 64×64 Event-Driven Logarithmic Temporal Derivative Silicon Retina. *Proceedings of IEEE Workshop on Charge-Coupled Devices and Advanced Image Sensors*, pages 157–160, 2005.

[12] M. Oster and S.C. Liu. A winner-take-all spiking network with spiking inputs. *Electronics, Circuits and Systems, 2004. ICECS 2004. Proceedings of the 2004 11th IEEE International Conference on*, pages 203–206, 2004.

[13] HK Riis and P. Hafliger. Spike based learning with weak multi-level static memory. *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on*, 5, 2004.

[14] 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, 2001.

[15] M. Mahowald. *VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function*. PhD thesis, California Institute of Technology, 1992.

[16] A. Linares-Barranco, G. Jimenez-Moreno, B. Linares-Barranco, and A. Civit-Balcells. On algorithmic rate-coded AER generation. *Neural Networks, IEEE Transactions on*, 17(3):771–788, 2006.

[17] M. Oster, A.M. Whatley, S.C. Liu, and R.J. Douglas. A hardware/software framework for real-time spiking systems. *ICANN2005, Int. Conf. on Artificial Neural Networks*, 3696:161–166, 2005.

[18] E. Chicca, A. M. Whatley, P. Lichtsteiner, V. Dante, T. Delbruck, P. Del Giudice, R. J. Douglas, and I. Indiveri. A Multichip Pulse-Based Neuromorphic Infrastructure and Its Application to a Model of Orientation Selectivity. *Circuits and Systems I: Regular Papers, IEEE Transactions on [Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on]*, 54(5):981–993, 2007.

[19] R. Berner and et al. A 5 Meps $100 USB 2.0 Address-Event Monitor-Sequencer Interface. *Circuits and Systems, 2007. ISCAS 2007. Proceedings. 2007 IEEE International Symposium on*, page (in press), 2007.

[20] R. Paz, F. Gomez-Rodriguez, MA Rodriguez, A. Linares-Barranco, G. Jimenez, and A. Civit. Test Infrastructure for Address-Event-Representation Communications. *International Work-Conference on Artificial Neural Networks (IWANN'2005). Lecture notes in computer science*, pages 518–526, 2005.

[21] F. Gomez-Rodriguez, R. Paz, A. Linares-Barranco, M. Rivas, L. Miro, S. Vicente, G. Jimenez, and A. Civit. AER tools for communications and debugging. *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, page 4, 2006.

[22] Intel Press. *Intel PXA255 Processor Developer's Manual.* Intel Press, 2004.

[23] E. Culurciello, R. Etienne-Cummings, and KA Boahen. A biomorphic digital image sensor. *Solid-State Circuits, IEEE Journal of*, 38(2):281–294, 2003.

[24] F. Gomez-Rodriguez, R. Paz, L. Miro, A. Linares-Barranco, G. Jimenez, and A. Civit. Two hardware implementations of the exhaustive synthetic AER generation method. *Computational Intelligence and Bioinspired Systems. Lecture Notes in Computer Science*, 3512:534–540, 2005.