Hardware and software for interfacing to address-event based neuromorphic systems

Address event representation (AER)[1] has long been considered a convenient transmission protocol for neuromorphic devices. This is because messages are transmitted by an asynchronous sequence of all-or-none, 'digital' spikes, carrying information in the (analog) pattern of inter-spike time intervals. One evolution of the originally proposed point-to-point AER protocol is a many-to-many, fully-arbitrated protocol that supports the design of fairly general-purpose prototyping systems (as in the Silicon Cortex project[2]).

One missing and long-needed feature of AER-based systems is the ability to acquire data from complex neuromorphic systems, to stimulate them using suitable data, and to support the design and the operation of multi-chip AER-based systems. Such needs have been only partially met so far through commercial or home-made hardware and software built for a particular purpose and idiosyncratically dependent on the specific system. We have implemented a general-purpose solution to this problem in the form of a PCI (peripheral component interconnect) board,[3] developed at the Italian Institute of Health, supported by software developed at the Insitute for Neuroinformatics. The article on page 4 of this issue illustrates the use of the this board with VLSI networks of spiking neurons

The PCI-AER board can handle up to four sender and four receiver chips. Figure 1 illustrates the three main functions that the board can perform independently in a typical environment: monitoring, sequencing, and mapping.

The monitoring function involves reading and time-stamping events from the AER bus, and making them available to the host PC for further processing. The *monitor* taps any AER transaction and, as the transaction is detected, stores a time label and the address in the FIFO (first in, first out). This decouples the management of AER events from the PCI read operation. Extra bits are used to identify time labels and address words so that the software can recover if words are missing due to FIFO overruns. Figure 2, left, illustrates the monitor data path.

Another function that is performed is sequencing: generating spike traffic on the AER bus on the basis of pre-computed information (e.g., to emulate hardware components). The ability to inject pre-computed spike sequences into the AER bus also allows software simulations and VLSI hardware to be seamlessly interchanged. The *sequencer* is decoupled from the PCI bus using a FIFO. The sequencer checks the FIFO state and, if it is not empty, reads its content. Depending on the two most significant bits, it can either generate a transaction on the AER bus (a spike), or it can generate a relative or an absolute time delay. The transactions generated by the sequencer can be transmitted via any one of the four output channels. Figure 2, middle, illustrates the sequencer data path.

Finally, the *mapper* maps incoming AER-*in* to outgoing AER-*out* addresses, and can operate in three modes: *pass-through, one-to-one,* and *one-to-many*. In the pass-through mode, the AER-in address is simply replicated at AER out, whereas in one-to-one mode the AER-in address is used as a pointer to a look-up table. The retrieved content will be the target address at AER out. In the case of the one-to-many mode, the AER-in address generates multiple events to be dispatched to different targets. This is achieved by using the AER-in address as a pointer into the same look-up table as in one-to-one mode, but in this case the retrieved content is used as a further pointer to a list of AER-out addresses.

The mapper itself is composed of three main parts. First, the *mapper-in* forwards AER input, as events, to the mapper FIFO. It can also complete the AER transaction in the absence of a receiver chip. Second, the FIFO is needed to decouple the management of the AER-in and AER-out fluxes. Lastly, the *mapper-out* manages the look-up table scanning and the corresponding AER-out transactions. Figure 2, right, illustrates the mapper data path.

The main interface can adapt to different AER standards, and is connected via a cable to a small board that is directly connected to the AER bus. To enable the functionality of the board to be accessed robustly and conveniently from user programs, we have provided a device driver for Linux and, layered on top of this, a C library.

The open-source, GNU-General-Public-Licensed (GPL)[4] driver provides full integration of the PCI-AER board into Linux systems following the Unix *everything-is-a-file* model. This allows AE data streams to be read and written using standard Unix read and write calls and/or supports the use of standard shell redirection and command-line tools. The driver supports separate logical devices for each of the major functional blocks of the board—mapper, monitor, and sequencer—and can support multiple boards. It also ensures that the AE data streams being read or written remain coherent. For instance, it prevents multiple programs from attempting to read the monitored AE data at the same time, thus splitting the data stream unpredictably. It also forces word-multiple-sized reads

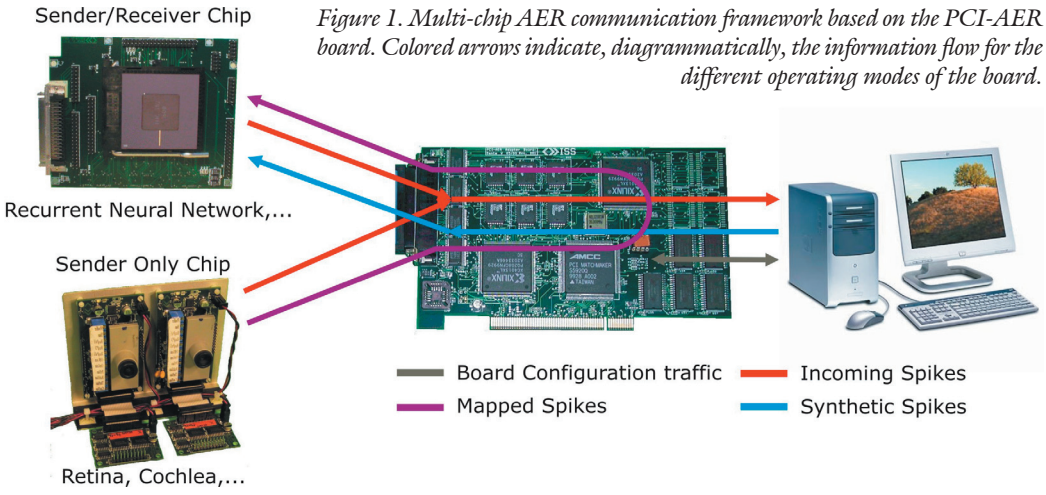

Figure 1. Multi-chip AER communication framework based on the PCI-AER board. Colored arrows indicate, diagrammatically, the information flow for the different operating modes of the board.

***Dante et al.,** continued on p. 6*

Dante, continued from p. 5

and writes to prevent corruption of the data streams due to misalignment problems. IOCTL (input/output control) calls are provided to set and get all possible sensible configuration states, and user programs are prevented from putting the board into an inconsistent state. The driver also performs memory management for the mapper SRAM (static random-access memory) so that the user does not need to perform necessary but onerous and error-prone indexing arithmetic. This helps the user to avoid accidentally corrupting the mapping table. At the time of writing, the driver is available for Linux kernel versions 2.4 and 2.6.

The library consists mainly of thin, fast wrappers for the driver functions (open, close, read, write, flush) and IOCTL calls. However, functions are also provided to convert from the PCI-AER hardware-specific format to a generic inter-spike interval/address-event format for reading, and vice versa for writing. The conversion function for reading also implements a state machine to attempt recovery when data are received out of order in the event of monitor FIFO overruns or other hardware errors.

Both driver and library are fully documented.

Some users have written small applications in C or C++ for spike-train generation and data logging directly using the API (application programming interface) provided by the library. Dylan Muir of the Institute of Neuro Informatics (INI) has developed a Matlab toolbox for the offline generation and manipulation of spike trains to be sent to, or read from, the PCI-AER board via library and driver. Matthias Oster (see bottom of page 9) has developed a client-server architecture on top of the library to enable the use of the board on-line from within Matlab, including real-time data display. Future developments should include a refinement of this client-server architecture to enable multiple data-sinks (including simple graphical display tools and other software packages) to read the monitored AE stream concurrently in a coordinated way. Other possible future developments include a command-line-driven configuration tool, Java support, and a stimulation tool for the on-line generation of AE patterns to drive the sequencer.

The design of the PCI-AER interface is now being developed to provide better integration of the AER-related functions with other needs, such as setting and control. We are aiming for better performance. In the meantime, the current board and associated software are being used by several groups (including two European-Union-funded projects), and are made available to interested research groups through INI thanks to a formal agreement between it and the ISS.

Vittorio Dante, Paolo Del Giudice, and Adrian M. Whatley*

Complex Systems Unit
Department of Technologies and Health (ISS)
Rome, Italy
E-mail: {dante, paolo.delgiudice}@iss.infn.it
http://neural.iss.infn.it

*Institute of Neuroinformatics (INI)
Uni/ETH Zurich, Switzerland
E-mail: amw@ini.phys.ethz.ch
http://www.ini.unizh.ch/~amw

References

1. J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead, *Winner-take-all networks of O(n) complexity,* **Advances in Neural Information Processing Systems,** pp. 703-711, Morgan Kaufmann, San Mateo, CA, 1989.
2. S. R. Deiss, R. J. Douglas and A. M. Whatley, *A pulse code communication infrastructure for neuromorphic systems,* **Pulsed Neural Networks**, pp. 157-178, MIT Press, Cambridge, MA, 1999
3. V. Dante and P. Del Giudice,*The PCI-AER interface board,* **2001 Telluride Workshop on Neuromorphic Engineering Report,** pp. 99-103, 2001: http://www.ini.unizh.ch/telluride/previous/report01.pdf
4. http://www.gnu.org/copyleft/gpl.html

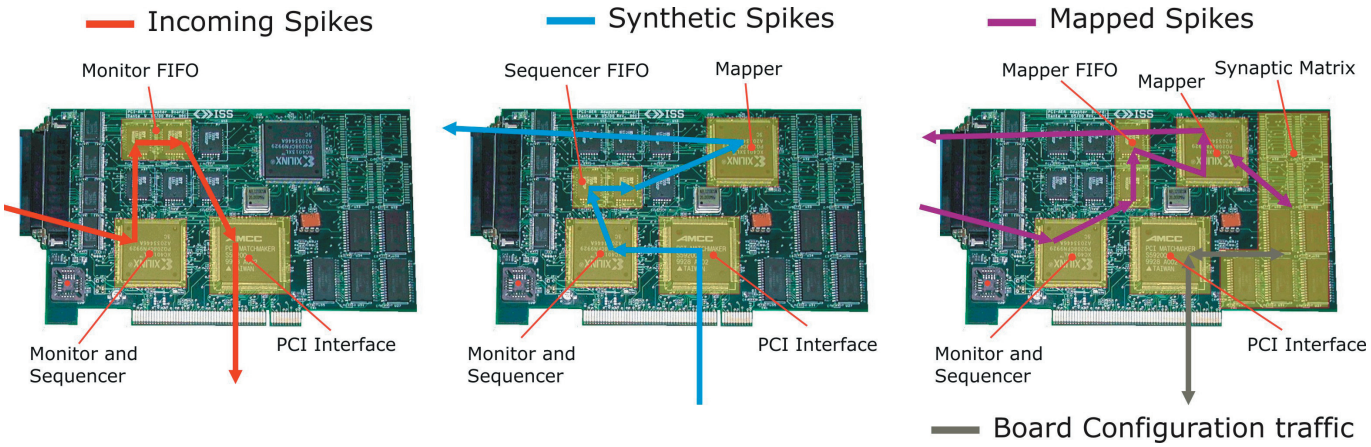

*Figure 2. The three panels show, from left to right, information flow for the **monitor**, **sequencer,** and **mapper** functionalities.*