A PCI based high-fanout AER mapper with 2 GiB RAM look-up table, 0.8 $\mu$s latency and 66 MHz output event-rate

Daniel Bernhard Fasnacht, Giacomo Indiveri
Institute of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland

Abstract—Neuromorphic systems have been increasing in size and complexity in recent years, thanks also to the adoption of the Address-Event Representation (AER) as a standard for transmitting signals among chips, and building multi-chip event-based systems. AER mapper devices that route Address-Events from multiple sources to different multiple destinations are crucial components of these systems, as they allow users to flexibly compose multi-chip setups, re-configure them, and program different architecture or network topologies.

In this paper we present a PCI based high-fanout AER mapper with 2 GiB RAM look-up table, 0.8 $\mu$s latency and 66 MHz output event-rate. Integrated with a PC it forms a very flexible and affordable AER experimental platform which is suitable for prototyping and research projects. Indeed, multiple instances of this system are already being used to perform various types of AER experiments. In addition, the system’s ability to implement probabilistic address-event mappings further extends the range of experiments that can be performed using this platform.

We describe the hardware system implementation details, compare our approach to previously proposed ones, and present experimental results which demonstrate how the system provides optimal performance for experiments with high average fanout and how for low fanout mappings the limiting factor is given by the 0.8 $\mu$s latency induced by the PC on each random memory access.

I. INTRODUCTION

A. Neuromorphic Multi-Chip AER systems

Event-based neuromorphic systems typically comprise multiple custom hybrid analog/digital VLSI chips that communicate among each other using the Address-Event Representation (AER). In this representation input and output spikes (Address-Events) are transmitted using asynchronous digital pulses that encode the address of the sending node. Analog information is carried in the temporal structure of the inter-pulse intervals and in their mean frequency, very much as it is believed to be conveyed in spike trains of biological neural systems.

Figure 1 illustrates an example of AER communication: addresses of sending nodes on a source chip are encoded and written on a digital bus as soon as an event is produced; on the receiver side, the incoming Address-Events are decoded and consumed by the corresponding receiving node. This example represents a simple direct connection from single source nodes to corresponding single destination nodes (where the number of source nodes, destination nodes and connections are all equal).

To create more complex network topologies between neuromorphic chips, with arbitrary connectivity patterns, with fan-in and/or fanout greater than one, it is necessary to implement a way to map events from source addresses to one or more desired destination addresses. A variety of mapping approaches are possible, from algorithmic mappings to memory-based look-up tables.

Such mappings are usually realized with custom (digital) hardware devices (mappers). While single neuromorphic chips typically offer little flexibility in the type of networks or architectures that they can implement (due to the analog nature of the circuits used in these chips), neuromorphic Address-Event systems that use mappers to manage the communication of signals among its computing elements offer the advantage of re-configurability: by changing the mapping algorithm, or the content of the mapper’s look-up tables, arbitrary network topologies can be implemented, learned, or evolved.

In theory multi-chip AE systems could have reconfigurable...
on-chip connectivity or even on-chip mapping functionality, without requiring the use of an explicit off-chip mapper. But reconfigurable mappings require memory. As memory is a very costly resource to implement on-chip, the mapping functionality implemented on-chip would result (and in practice is) very limited.

To overcome this problem and extend the limited mapping capabilities of the neuromorphic chips in the system it is necessary to use dedicated mapping devices. Such AER mappers commonly use SRAM chips to store the mapping information and FPGAs to handle the I/O interfaces and to control the mapping process.

Typical mappers have up to 4 MiB\(^1\) of SRAM to store the mapping tables, and use parallel AER interfaces. In this paper we present a new AER mapper that can store mapping tables up to 2 GiB size, and uses high-speed serial AER interfaces, far superior to the parallel AER interface typically used for transmitting AER signals between boards or devices [2].

II. STATE OF THE ART

A wide range of multi-chip AER systems has been presented in the past, which approached the mapping of AEs (Address-Events) problem in different ways.

1) **Silicon Cortex:** In the Silicon Cortex project [1] Digital Signal Processors (DSPs) were used to interconnect pairs of neuromorphic chips and do AE (Address-Event) mapping between them. The mapping table was stored in up to 64 KiB of RAM. A simple single-indirection table format (look-up table) was used.

2) **Rome PCI-AER:** In [3] the authors used a custom PCI board for building an AER multi-chip system around a PC as the central platform. This PCI-AER board was developed by V. Dante and colleagues at ISS, Rome. It contains multiple FPGAs, FIFO and SRAM chips, and a dedicated PCI-I/O chip implementing the PCI protocol.

The Rome PCI-AER board supports monitoring\(^2\) and sequencing\(^3\) of AER streams, but also mapping AEs directly on the PCI-AER board, without any interaction with the actual PC. For mapping it has 4 MiB of discrete SRAM available. It supports either storing one-to-one mappings, where an incoming AE can be mapped to only one output AE (or discarded), or one-to-many mappings, where the mapping table is stored in a double-indirection format (an incoming AE is first “mapped” to a pointer in the SRAM, which points to the actual output values of the mapping).

With the “double-indirection” mapping method, the 4 MiB of SRAM available can be used efficiently. However mapping tables of this type can not be easily modified or extended, and have to be recreated as a whole in case of re-configuration.

This mapper has been used in a multitude of projects in various labs [3], [4].

3) **CAVIAR USB-AER:** As part of the CAVIAR project [5], a “USB-AER” board with mapping capability was presented. The board consists of parallel AER input and output connectors, a central FPGA and 2 MiB of discrete SRAM.

Originally the board was programmed to support “double-indirection” mapping methods. In [6] the USB-AER board was modified to implement probabilistic mappings, i.e. AEs were mapped or discarded with a probability that could be set for each source value. The same board was also modified to implement configurable delays in AEs [7]. In this variant, a delay is read from a table linked to the source address, and the AE is then put in a queue until the delay cycle is completed.

4) **Monitor & Sequencer based Mappers:** All the AER hardware that supports monitoring and sequencing of AEs to/from a computer can in principle be used also to perform mapping on the same computer. The reason why this approach is not commonly used is because it has a serious drawback compared to all devices mentioned above: getting data into the computer (monitoring), processing (mapping) it and sending it back out again (sequencing) takes considerable amounts of time. To reduce latencies, these operations have to be performed on large blocks of AEs, and not on single AEs. Even in this case, input-to-output latencies are typically of tens to hundreds of milliseconds. Even worse, these latencies can vary drastically due to the nature of time-sharing operating systems.

Such latency properties render most approaches to map AEs involving computers as described almost unusable.

III. MAPPER SYSTEM OVERVIEW

The AER mapper we propose is addressing these latency issues, while following design choices dominated by:

1) the requirement to interface the mapper to pre-existing neuromorphic hardware that makes use of the Serial AER interface [2].

2) the requirement to affordably store and quickly access LUT-type mapping tables hundreds of megabytes in size.

Affordable FPGA systems do not have large availability of RAM memory, characterized by size, bandwidth and access latency. On the other hand, in PC hardware gigabytes of very fast DRAM is readily available at prices which are orders of magnitude lower to memory with equivalent specifications on FPGA development platforms. For this reason we chose to combine PC hardware and FGPAx using the PCI interface.

The task of the look-up table based mapper presented here is simple: it has to receive AEs on it’s input, look up the corresponding output AEs in it’s mapping table, and send them to the output interface.

Input and output signals, in this mapper, are transmitted using a custom high speed Serial AER interface. The mapper look-up table\(^4\) is stored in the main memory of a PC motherboard.

The actual mapper device is a PCI card with an FPGA and the Serial AER input and output interfaces. It is connected to a PC motherboard with 4 GiB of RAM, out of which 2 GiB are used to store the mapping table.

---

\(^1\)IEC binary prefixes are used throughout the text, i.e. Gi denotes 1024\(^3\)

\(^2\)timestamps and recording of AER streams on a computer

\(^3\)replaying timestamped AER streams from a computer

\(^4\)Look-up table (LUT) and mapping table are used interchangeably as the mapper presented uses a simple LUT type mapping table.
A. Mapper Components

As displayed in Fig. 3 the mapper hardware thus consists of: (1) a PC motherboard with RAM to store the mapping table, (2) Serial AER I/O interfaces to receive input AEs and transmit mapped output AEs (3) a PCI card with an FPGA that performs the look-ups in the PC RAM.

1) PC Motherboard & RAM: The LUT is stored and managed using a PC motherboard, with 4 GiB of RAM installed. All other components of a regular PC are kept to the bare minimum or even removed. A USB memory stick is used as a hard-drive replacement containing a minimal Linux setup that manages the download of mapping tables over the network.

2) 3 GHz Serial AER Interface: The PCI FPGA board extends with a daughter-board carrying the components necessary for the mapper AER input and output interfaces. These AER interface used is a Serial AER interface based on a dedicated Serializer-Deserializer chip from Texas Instruments and SATA connectors and cables as presented in [2].

This interface allows to transfer AEs at very high speeds between boards in a multi-chip, multi-board setup. Neuromorphic sensors and processing chips are connected to the mapper through this interface and the AEX board also presented in [2]. The Serial AER interface used achieves event-rates of 156 MHz with 16 bit AEs or 78 MHz with 32 bit AEs.

3) PCI FPGA Board: The main piece of the mapper hardware is a PCI board containing a FPGA at its core. The FPGA, a Xilinx Spartan-3, connects both to the PCI bus connector and to I/O headers for connecting the daughter-boards.

The FPGA contains logic to handle the Serial AER input and output, the mapping process, and the PCI interface enumeration and I/O in order to access the mapping table in the PC main memory.

IV. IMPLEMENTATION DETAILS

A. PC Hardware

The PC hardware consists of a motherboard with the Intel P35/ICH9R chip-set, 4 GiB DDR2 RAM and a regular CPU.⁵

Unlike in the most recent Intel & AMD CPUs, in this architecture the memory controller is not in the CPU, but in the north-bridge. If a PCI device needs to access the main memory the data-path is PCI ⇐ south-bridge ⇐ north-bridge ⇐ DRAM. This means that the CPU is not involved in the actual mapping operation. It is just used to run a minimal Linux environment to provide access to the system over the network. Graphics card, monitor or keyboard are not part of a typical setup as all user-interaction is done through the network.

B. Mapping Table Setup & Management

1) Physical Memory Map: In order to reserve most of the main memory for the mapping table, the operating system is told to use only 1 GiB of RAM. This is achieved by providing the mem=1G statement⁶ as a Linux boot parameter via the GRUB⁷ boot-loader. The mapping table is stored in the next 2 GiB of RAM. The top-most gigabyte is left untouched. BIOS and PCI devices reserve address regions just below 4 GB which must not be overwritten by the mapping table.

The resulting physical memory map is thus:

<table>
<thead>
<tr>
<th>memory range</th>
<th>used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 GiB – 1 GiB</td>
<td>Operating System</td>
</tr>
<tr>
<td>1 GiB – 3 GiB</td>
<td>Mapping Table</td>
</tr>
<tr>
<td>3 GiB – 4 GiB</td>
<td>Reserved for BIOS &amp; PCI</td>
</tr>
</tbody>
</table>

2) Setting Mapping Tables: The user can transfer a mapping table, or a script to generate a mapping table via the network. The memory region reserved for the mapping table can then be written to through an mmap() on /dev/mem.

3) Mapping Table Format: A very simple mapping table format is used. The input AEs are 16 bits wide.

The 2 GB of mapping table space are split uniformly into 64 Ki pieces giving 32 KiB of RAM per possible input value. With 2 Bytes per AE this allows us to store up to 16 Ki of output AEs for a given input AE. A reserved sentinel address value (0xFFFF) is used to mark the end of a sequence of output AEs.

This means that one input AE can generate up to 16 Ki – 1 output AEs, i.e. the maximum fanout of the mapper is ~16 Ki.

One advantage of this very simple mapping table format is that it requires only one contiguous access to the mapping table RAM in order to get all the values for one input AE. This is beneficial, as random access to main memory is much more costly than sequential access.

Also the calculation to find the right memory address for a given input AE is simple. The numeric value of the input AE is multiplied with 32 KiB, and the mapping table offset (1 GiB) is added. For the FPGA this is an almost free operation as we multiply with a power of two, which is simply a bit shift. The resulting bit-shift and addition can be done in nanoseconds resulting in a single cycle operation in the FPGA.

The mapping table format is of course not very memory efficient. If a certain source address does not map to anything, the respective memory cannot be used for storing additional mapping information for another source address.

⁵Motherboard: Asus P5K WS (Intel P35/ICH9R), RAM: Mushkin 2x2 GiB, DDR2 800MHz (PC2-6400), CPU: Intel Core 2 Duo E6600 2.4 GHz
⁷GRUB boot-loader: http://www.gnu.org/software/grub/
C. Mapper PCI Board

The basis for the custom mapper hardware is a modified PCI FPGA development board containing a Xilinx Spartan-3 FPGA.\(^8\) This FPGA connects to both the PCI bus and a custom daughter-board that implements the Serial AER interface (depicted in Fig. 2).

D. FPGA Programming

As illustrated in the FPGA block-diagram (Fig. 4) the FPGA contains logic to handle the Serial AER input and output, the mapping process, and the PCI interface enumeration and I/O in order to access the mapping table in the PC main memory.

1) Custom PCI Implementation: The mapper uses a custom PCI implementation that was developed from scratch without using any IP-cores. This was motivated to get as much control over the PCI communication as possible and the ability to analyze and optimize any aspect of it.

Other PCI VHDL implementations and IP-cores were considered, but various readily available PCI implementations were analyzed and found to be not appropriate because they were either incomplete or did implement burst-read/-write transactions in a way that was not fast enough for our needs. PCI implementations without published HDL were ruled out because of the limited low-level customizability. In our PCI VHDL implementation we chose to ignore Arbitration Latency and the master Latency Timer\(^9\). This means that other PCI bus-masters can be kept from getting access to the PC bus longer than usual. In the case of the PCI mapper this is a desired behavior though, as ongoing mapping activity should not be suspended to grant bus access to other PCI devices.

E. Probabilistic Mapping Mode

The mapper can also be configured to use an alternative mapping scheme that supports probabilistic mappings. In this mode, the mapping table data is interpreted as follows: instead of the default pair of 16 bit target addresses, the 32 bits are used

```
    31  30..................24  23........................................0
        7 bit probability value
        24 bit target address
        last-in-fanout marker
```

Fig. 6. Probability Mapping Word Format

Our custom PCI implementation has undergone practical testing in combination with a multitude of different motherboards and has shown to be working reliably.

<table>
<thead>
<tr>
<th>PCI Implementation Spec Sheet</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCI 3.0, 32 bit, 33 MHz</td>
</tr>
<tr>
<td>PCI-Target support</td>
</tr>
<tr>
<td>PCI-Initiator support (bus-mastering)</td>
</tr>
<tr>
<td>Customizable PCI Configuration Space defaulting to one 64 MiB Base Address Register</td>
</tr>
<tr>
<td>Supports all types of PCI read/write transactions implemented including multi-read/write (burst-transfers) without wait states</td>
</tr>
<tr>
<td>Supports full bus performance of 132 MB/s for both read &amp; write transfers</td>
</tr>
</tbody>
</table>

2) Example PCI Configuration-Read Transaction: Figure 5 shows a scope-trace\(^10\) of a basic PCI Configuration transaction. The signals are the PCI bus signals Clock (33 MHz), FRAME, IRDY (Initiator Ready), TRDY (Target Ready). Clock and signals are source-synchronous. The traces match the PCI spec waveforms\(^11\) as desired.

3) Mapping and Serial AER FPGA Parts: While the PCI core takes most of the VHDL and FPGA logic resources used, the FPGA also implements the Serial AER interface and the mapper control logic. Both the mapper control and Serial AER interface contain FIFOs in the data-path (in blue/dashed in Fig. 4).

As the mapper interfaces to neuromorphic chips through the AEX board, the Serial AER interface is identical to the same interface on that board. Both the AEX board and the 3 GHz Serial AER interface are explained in detail in [2].

The mapper control logic is straightforward because the mapping table format is very simple. For every input AE we calculate the right spot in the mapping table to read the output AEs. We then initiate a PCI burst-read from the calculated address and read until we find the sentinel value, chosen to be 0xFFFF.

\(^8\)Development board: Raggedstone1, http://enterpoint.co.uk/ FPGA: Xilinx Spartan-3 1500 kGates, XC3S1500-4FGG456C
\(^9\)PCI Specification 3.0, Section 3.5.4 & 6.2.4, PCI-SIG
\(^10\)Oscilloscope used: Tektronix TDS 3054B
\(^11\)PCI Specification 3.0, Figure 3-4, PCI-SIG
to store 24 bit target addresses, a 7 bit probability value and one bit to mark the last word in a fan-out sequence (see Fig. 6).

The \(2^7\) probability values are interpreted as uniformly distributed in the range \([0, 1.0]\) inclusive. This means that the probability can be selected in steps of \(1/127\).

The trivial cases of probability zero (0000000) and one (1111111) are implemented directly by discarding or unconditionally forwarding that address.

For the other values we draw from a pseudo random number generator (PRNG) that provides values in the range of \([0, 126]\) inclusive. If the probability value for the current address is strictly greater than the random number, the address is forwarded and discarded otherwise.

The random number generator is based on an open source PRNG code\(^{12}\). From this PRNG 7 bit values are drawn. If the value is out of our range (i.e. we draw the value 127) the step is repeated. A small number of random values that are in our desired range are stored in a FIFO, to reduce the probability that the process stalls because the drawing step has to be repeated.

The main PRNG is constantly kept running and not only active when we draw a random number from it. As the time from the start of the FPGA to the start of the mapping process is essentially random, there is no need to seed the PRNG registers explicitly.

Throughout the rest of the text the original mapping mode is discussed and not the alternate probability mapping mode. The performance numbers given are not affected by the probability mapping mode, except that the output bandwidth is reduced by a factor of two as we can transfer only one destination address per PCI bus cycle, i.e. the output event rate in the probability mapping mode is 33 MHz.

V. RESULTS

A. Typical Usage in Multi-Chip Setups

A practical multi-chip setup such as [8] typically consists of a number of neuromorphic chips, each of which is directly connected to one AEX board, and an AER mapper. These components are then connected using the Serial AER interface in a loop topology. Each AEX board has a certain address-space assigned. If incoming AE fall into that space, they are sent to the local chip. Otherwise they are forwarded (kept in the loop). This allows the mapper to send AEs to all chips, and allows all chips to send AEs to the mapper. The full network connectivity is controlled at one single point, the mapper described here. Several examples of multi-chip systems have been built, with this mapper, using up to five multi-neuron AER chips. In all experiments carried out up to now, the mapper has not been considered the limiting factor.

B. Performance Measurements

In order to perform measurements of the latency and output-bandwidth of the mapper, a number of digital signals normally only available within the FPGA were routed to a debug connector and then used record the scope traces in this sections.

1) Latency: Figure 7 shows a scope trace taken to measure the latency of the mapper. The traces are signal denote: MappingInProgress (top), ReadInputEvent (middle), WriteOutputEvents (bottom). The mapping table is set up to map each event to itself only, thus we have a fanout of one.

The trace starts with input data becomes available. ReadInputEvent is first active for one cycle to read the incoming data. Then MappingInProgress is asserted which means that the mapper issues a memory-read on the PCI bus. Now the mapper waits for the memory to deliver the data requested. When WriteOutputEvents becomes active for one cycle, this means that the mapper received the first word from the memory, in this case being 0xFFFFXXXX where 0xFFFF is the output event, and 0xXXXX the sentinel value. With the sentinel being detected the memory-read stops, and little after, the next input event is being read and processed.

The measurement figures on the right of the traces show the high-time of MappingInProgress signal, which is about one PCI bus cycle (30 ns), and the duration from the rising-edge on MappingInProgress to the rising-edge on WriteOutputEvents, i.e., the time from the input event being available until the first mapped output event is there (ca. 0.8 \(\mu\)s).

2) Bandwidth Limits: Figures 8 and 9 show a plot of the same signals as in Fig. 7, but with a mapping table fanout of 64 and of 1024. Note that the timescale is different for each plot.

The observed latency of 774 ns at a fanout of 64 is the same as in the previous latency measurement for a fanout of one. The other measurement in the figures (CH3+Width) tells

---

\(^{12}\)OpenCores project systemc-rng, http://opencores.org/
us that the mapper takes 983.7 ns to map and send out 64 events and 15.27 μs for 1024 events. We can calculate that the mapper actually maps two AEs per PCI bus-cycle (30 ns). This is as expected given that our AEs are 16 bit and the PCI bus transfers 32 bits per cycle.

We can conclude that the bandwidth bottleneck is the maximum physical limit of the PCI bus.

VI. LIMITATIONS AND FUTURE WORK

A. Limitations

1) LUT mapping only: Although probabilistic mappings can be efficiently implemented on the current mapper, the mapper only supports look-up table mappings. Delay or algorithmic mappings are not very well suited for this system. These types of mappings require multiple non-sequential accesses to the main memory, each of which takes 0.8 μs as elaborated.

2) Non-Simultaneous Fanout: One issue with high-fanout mappings of the type presented here is that the order in which AEs are transmitted is always the same. For high-fanout cases this can cause artefacts as an AE at the end of a fanout can be considerably later than one at the beginning of the fanout, even though that these AEs are assumed to be at the same time, i.e. the fanout is instantaneous and not serial.

3) System Scalability: One single central mapper as presented here does not scale well from a certain size of the system on. For a system with hundreds of chips, for example, a central mapper is not feasible. Such systems need to be approached by distributed and/or hierarchical mapping schemes.

B. Future Work

The non-simultaneous fanout is a problem that will be investigated and approached by some degree by permuting parts of the fanout AEs within the FPGA. The limit in the current hardware for this is given by the amount of block-SRAM available within the FPGA.

To approach the scalability issue, a next generation mapper will consist of multiple distributed mapping units, each of which combining FPGA, SRAM and DRAM components. Another important design aspect will be to create a mapping hierarchy, where the fanout-generation is kept as close to it’s consumer as possible.

VII. Conclusion

We presented an AER mapper based on a hybrid system comprising both FPGA and PC hardware. A key integrating component of the system proposed is the high-performance PCI interface implementation developed exclusively for building this mapper. The mapper can store up to 2 GiB of mapping data and is thus suitable for high-fanout mapping applications.

Together with dedicated AEX boards hosting neuromorphic chips the mapper provides a highly flexible system for building experimental multi-chip AER setups.

We presented measurements of the key characteristics latency and bandwidth figures, demonstrating that the input-to-output latency is 0.8 μs, and that the limiting factor is given by the time required to access the main memory from PCI. We demonstrated that the mapper can support a sustained output event-rate of up to 66 MHz, and this is the maximum supported by the PCI bus.

Acknowledgment

This work was supported by the EU ICT FP7 Grant ICT-231467-eMorph.

References