RASHPA: A DATA ACQUISITION FRAMEWORK FOR 2D X-RAY DETECTORS

F. Le Mentec, P. Fajardo, T. Le Caër, C. Hervé, A. Homs
ESRF 6 Rue Jules Horowitz, 38000 Grenoble, France

Abstract

As last generation X-ray detectors are capable of producing very high data rates in the range of 1 to 100 GBytes/second, the implementation of high throughput data acquisition systems is essential for the efficient use of those data streams at high brilliance synchrotron radiation facilities. This paper introduces RASHPA, a data acquisition framework optimised for 2D X-ray detectors sufficiently generic and scalable to be used in a large diversity of new high performance detector developments. the paper describes the basics of the conceptual design as well as the scheme chosen by the ESRF for the first demonstrators that combines a highly configurable multi link PCI Express over cable based data transmission engine with a carefully designed LINUX software stack.

INTRODUCTION

RASHPA is a data acquisition framework for X-ray 2D detectors currently under development at the European Synchrotron Radiation Facility that aims at standardizing the data transmission pipeline from the detector up to the software application for further processing, visualization or storage. From the perspective of this work, a 2D detector is considered to be a device segmented in an arbitrary number of detector modules that operate and transfer data in parallel. Each module includes a data acquisition controller, referred as RASHPA controller, that is in charge of pushing the data into the address space of the backend computing infrastructure.

As a framework, RASHPA consists of a specification defining the functional concepts as well as the hardware and software interfaces, and a middleware running on the backend computers that is fully generic and independent of the detector. For its practical implementation, the ESRF is also developing a set of hardware blocks that allows to build and integrate the RASHPA controllers in the detector hardware in conformity with the specification.

In this article, we first introduce some basic considerations about technology choices, then we present the framework design and finally, we give some details of the prototype implementation currently under development. For the sake of simplicity and unless it is explicitly mentioned, the rest of this paper refers to a basic configuration in which the detector is made of a single module, and therefore it includes only one RASHPA controller. In the same way, the backend infrastructure is also simplified and consists of a single computer referred as ”workstation”.

TECHNOLOGICAL CONSIDERATIONS

Zero-copy Memory Transfer

A fundamental goal of the project is to achieve zero-copy memory transfers: data are pushed by the RASHPA controller from the detector memory directly into the destination applicative buffers without intervention of the backend workstation CPUs. Furthermore, the middleware makes it possible for the application to exploit the workstation platform capabilities, such as NUMA factors and CPU affinities. By eliminating intermediate memory copies and enabling smart data placement policies, RASHPA allows for an efficient use of the workstation memory subsystem. Such considerations are especially important in the context of high performance computing systems [1].

Choice of the Data Transport Layer

While it requires a direct memory access capable transport layer, RASHPA is not tied to a specific one. However for the first implementations an important design choice has been the selection of PCI Express over cable [2]. From a strictly functional point of view this is an ideal option due to:

- *reliability*: PCIe has built in support for both control flow, data integrity and packet ordering. This removes the needs for additional protocol layers,
- *scalability*: a single point to point link is built upon one or more dynamically negotiated lanes. Different bandwidth requirements can thus be addressed by adjusting the lane count, which makes RASHPA automatically spans a wide range of detector bandwidth requirements,
- *low latency*: while RASHPA primary targets high throughput data transfers, PCIe supports low latency word based read and write operations. It includes interrupts (so called MSIs), which is particularly interesting as the middleware is largely event based,
- *native integration*: over the years, PCIe has become the default peripheral interconnect of x86 based platforms. As a part of this interconnect, there is no added overhead for a RASHPA based detector to communicate with the workstation CPU, memory or other PCIe endpoints.

Despite these benefits, the limited availability of PCIe over cable products and the lack of standardization of optical cabling form factor is still an issue. The current specification [3] only addresses copper cabling which limits the

ISBN 978-3-95450-139-7

Figure 1: Architecture of a basic RASHPA system.

link length to a few meters and becomes cumbersome as the lane count increases. Third party solutions based on optical fiber exist today [4], but the convenience of relying RASHPA implementations on PCIe in the future will very much depend on the practical availability of off-the-shelf PCIe over cable products. If proven wrong, and given that RASHPA does not fix a specific transport layer, it would be possible to replace it in our current prototypes without a major redesign of the architecture. For instance, we are investigating 10GbE as a replacement and have already a basic working implementation.

Foreseen Technological Integration

To reach peak performances, a general trend in both computing and storage technology areas is to decrease device access related overheads. For instance, NVIDIA is running the GPUDirect initiative, that allows a third party device to address directly a GPU memory. A similar approach exists for disk storage: backed by influential actors such as Intel and Oracle, NVM Express allows accessing solid state drives through an optimized PCIe interface. The emergence of these promising technologies can be seen as an additional motivation to build RASHPA PCIe based controllers, as coming generations of systems would be potentially able of transferring data directly from the detector to disk or to GPU coprocessors for on-the-fly data analysis.

FRAMEWORK DESIGN

Overview

Figure 1 is a conceptual diagram of how the RASHPA framework fits the data transmission pipeline of a basic detector system consisting of a unique detector module and a single backend workstation. From a hardware point of view, the RASHPA controller consists of a specific logic interfacing the detector readout electronics as well as a set of hardware blocks handling data transmission. The RASHPA specification defines the functionality of these blocks that can be seen as a default implementation. The main components are:

- a scheduler in charge of generating and dispatching memory transfer requests,
- engines to access workstation memory,
- an embedded CPU in charge of handling configuration and control requests from the workstation.

RASHPA also defines and implement a middleware on the workstation that initialises and manages the data transfer process and provides a standard programming interface to client applications.

Detector Readout Logic

Since the framework aims at being detector agnostic, few assumptions are made on interfacing the RASHPA controller with the detector specific electronics:

- RASHPA expects images resulting from the detector front-end to be built into intermediate working memories before data can be sent to the workstation. These memories can be either randomly (BRAM, DDR) or FIFO addressable,
- an implementation for driving the detector readout exists and must share the same FPGA as the RASHPA memory transfer scheduler. As acquisition progresses, this logic informs RASHPA about data availability (memory identifier, offset and size), through a protocol defined in the RASHPA specification.

Memory Transfer Scheduler

The memory transfer scheduler generates contiguous memory block transfer requests as data become available from the detector. A single request contains the following information:

- an address identifying the block working memory and the starting offset,

Figure 2: Structure of the RASHPA middleware.

- the workstation destination address,
- the block size,
- data transport layer specific informations.

The request is passed to the memory access engine. If the memory access engine is implemented on multiple FPGAs, the scheduler selects the appropriate one according to the destination address and simple load balancing policies.

Memory Access Engine

The memory access engine is in charge of transmitting contiguous memory blocks from the detector to the workstation memory. All the required information come from the memory block scheduler requests.

Conceptually, there is only one memory access engine. In simple designs, the memory access engine can eventually share the same FPGA as the transfer scheduler. However, achieving high bandwidth requires to aggregate multiple data transport links and thus, implement the memory access engine on several FPGAs.

Configuration and Control CPU

For increased flexibility, the RASHPA hardware embeds a CPU running a daemon on top of a full featured LINUX operating system. The daemon listens for incoming configuration and control requests from the workstation middleware. The requests are translated to low level registers accesses over an internal dedicated PCIe link that connects the CPU to the RASHPA hardware. The communication between the embedded CPU and the workstation can either use a TCP based connection or the PCIe downstream data link.

Workstation Middleware

The workstation middleware is in charge of providing client application with data transmission services through a well defined application programming interface (API). It targets LINUX based platforms. A typical client is the LIMA data acquisition and detector control library widely used at ESRF and other synchrotron radiation facilities [5]. The middleware layers are depicted in Figure 2:

- *configuration*: this layer is in charge of building a description so that the hardware knows how, when and where to transmit data. For instance, it includes the workstation memory buffer addresses. This description is implemented as a file whose structure follows an XML dialect defined by the RASHPA specification. This file is sent to the detector embedded CPU over the configuration link. The CPU interprets this information to configure and initialize the hardware,
- *device abstraction*: the RASHPA hardware may consist of one or more PCIe endpoints. Moreover, we mentioned that PCIe may eventually get replaced by another technology with different hardware capabilities (interrupts ...). Thus, the device abstraction layer hides low level details and provides the software with a generic interface to access the underlying devices. While most of the layer is implemented in user space, special operations such as interrupt handling require a thin driver,
- *memory management*: zero copy memory transfers require to work with the workstation physical address space. However, typical client application buffers are allocated in the process virtual address space. The middleware thus provides a virtual memory allocator on top of an internally managed physical memory allocator. If applicable, this allocator is NUMA aware and allows for efficient memory placement,
- *unified event model*: there are multiple sources of events that would make sequential programming difficult. For instance, data transmission completion is

ISBN 978-3-95450-139-7

sent by the detector using an interrupt along with auxiliary data; LINUX informs about PCIe device adding or removal using system notifications; Timers and callbacks may be registered by the client application itself. For these reasons, the middleware programming model is largely event based and provides with software abstractions that unify the different event sources.

FIRST PROTOTYPE

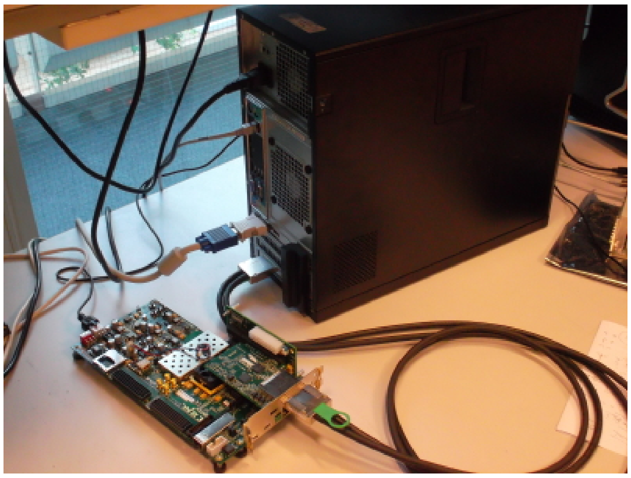

Figure 3: Prototyping platform.

A prototype is currently under development to implement and validate various aspects of the RASHPA framework. The hardware setup is shown in Figure 3 and consists of a detector module emulator connected to a workstation by a single data link. The data link uses a PCIe cabling expansion kit from One Stop Systems [6] connected to the workstation root complex that allows for a Gen2 x4 PCIe setup, giving an effective throughput of 2 GB/s. The same manufacturer provides today kits for PCIe Gen3 x16.

The detector module emulator is built around a KC705 Xilinx development board [7], which features a Kintex 7 FPGA, 1GB DDR3 memory and a PCIe x8 connector. The FPGA design relies on EBONE [8], a general purpose interconnect developed at the ESRF.

The workstation is an Intel 64 bits quad core i7 based platform installed with LINUX and the RASHPA software.

Benchmarking this prototype pointed an effective data transfer rate of 1GB/s, half the theoretical effective bandwidth. These results are encouraging, as they confirm that a performant implementation of the RASHPA flexible design can be made. Optimization areas have already been spot in the implementation to increase the bandwidth, which do not impact the RASHPA design.

ACKNOWLEDGEMENTS

A relevant part of this work is carried withing the CRISP EU project of the 7th Framework Programme (http://www.crisp-fp7.eu).

CONCLUSION

This paper presented the design of the RASHPA acquisition framework. Directions enabling both performance and flexibility are now established, and development of the first prototype is currently in progress. ESRF research programs and the accelerator upgrade second phase will soon allow to refine and validate the framework against real applications.

REFERENCES

[1] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, M. Roth, "A Holistic Approach to Memory Placement on NUMA Systems".

[2] PCI Express Special Interest Group, `http://www.pcisig.com/specifications/pciexpress`

[3] PCI Express External Cabling 1.0 Specification, `http://www.pcisig.com/specifications/pciexpress/pcie_cabling1.0`

[4] PLX Technology and Avago Technologies, "A Demonstration of PCI Express Generation 3 over Fiber Optical Link".

[5] A. Homs, L. Claustre, A. Kirov, E. Papillon, S. Petitdemange, "LIMA: A Generic Library for High Throughput Image Acquisition".

[6] One Stop Systems PCIe expansion kits, `http://www.onestopsystems.com`

[7] Xilinx KC705 development kit, `http://www.xilinx.com`

[8] ESRF Electronic Unit, EBONE OHR repository, `http://www.ohwr.org/projects/e-bone`