The ReadoutCard Userspace Driver for the New Alice O^2 Computing System

Konstantinos Alexopoulos and Filippo Costa
On behalf of the ALICE Collaboration

***Abstract*— A large ion collider experiment (ALICE) experiment focuses on the study of the quark-gluon plasma as a product of heavy-ion collisions at the CERN large hadron collider (LHC). During the long shutdown 2 of the LHC in 2019–2020, a major upgrade is underway in order to cope with a hundredfold input data rate increase with peaks of up to 3.5 TB/s. This upgrade includes the new online–offline computing system called O^2. The O^2 readout chain runs on commodity Linux servers equipped with custom peripheral component interconnect express (PCIe) field-programmable gate array (FPGA)-based readout cards: the PCIe Gen 3 × 16, Intel Arria 10-based common readout unit (CRU), and the PCIe Gen 2 × 8, Xilinx Vertex 6-based Common ReadOut Receiver Card (CRORC). Access to the cards is provided through the O^2 ReadoutCard userspace driver, which handles synchronization and communication for direct memory access (DMA) transfers, provides base address registers (BAR) access, and facilitates card configuration and monitoring. The ReadoutCard driver is the lowest level interface to the readout cards within O^2 and is in use by all central systems and detector teams of the ALICE experiment. This communication presents the architecture of the driver and the suite of tools used for card configuration and monitoring. It also discusses its interaction with the tangent subsystems within the O^2 framework.**

***Index Terms*— Base address register (BAR), data acquisition systems, direct memory access (DMA), drivers, field-programmable gate arrays.**

I. Introduction

A LARGE ion collider experiment (ALICE) is an experiment at the CERN large hadron collider (LHC) focusing on the study of the quark-gluon plasma—a state of matter which existed shortly after the Big Bang—as a product of heavy-ion collisions. Currently, the second long shutdown (LS2) of the LHC is underway, allowing for preparations for Run3, which will run at significantly higher luminosity. The main physics topics addressed by the upgrade require measurements characterized by a very low signal-over-background ratio, making traditional triggering strategies inefficient. Hence, the time projection chamber (TPC) necessitates the implementation of continuous readout, capable to keep up with an interaction rate of 50 kHz [1]. During LS2, ALICE is upgrading its detector and software systems to fulfill the above requirements and achieve a higher resolution.

A. O^2 Computing System

The upgrade includes the new online–offline computing system that is called O^2 [2], extending across two major computing slices, the first level processor (FLP) and the event processing node (EPN). The O^2/FLP subsystem is comprised of 200 FLPs responsible for detector readout. They are equipped with specialized data acquisition cards that interface with the front-end electronics (FEE) of the detectors. FLPs are designed to handle triggered and continuous readout operations, without discarding any events. A first-level grouping of the read-out events takes place within the FLPs before the data flow to the next subsystem. O^2/FLP also includes the quality control system and services for control, configuration, monitoring, logging, and bookkeeping. The O^2/EPN subsystem is comprised of 250 EPNs that are responsible for synchronous calibration and reconstruction before the data reaches storage.

For what concerns O^2 , the data originate from the readout cards of the experiment, namely the Common ReadOut Receiver Card (CRORC) [3] and the common readout unit (CRU) [4], which are connected to the front-end cards of the detector electronics. In this document, we focus on the ReadoutCard userspace driver, which controls and provides a communication interface to the cards of the experiment, serving as the lowest level interface to them within the O^2 framework, as shown in Fig. 1.

The ReadoutCard package was initially published in [5]. Since then, the userspace driver has been heavily developed to improve implementation details and extend core functionality. The existing library interfaces have been improved and fragmented to focus their functionality to their expected uses, reducing unnecessary complexity. Moreover, several interfaces have been introduced to facilitate access to new software components. The available tools have been extended to provide a complete suite to configure and monitor the status of the cards. Finally, ReadoutCard has been integrated with the O^2 monitoring [6] and O^2 configuration [7] components. This article presents the state of ReadoutCard following these developments.

II. Hardware

The readout chain of the experiment is designed around two different readout cards: the CRORC and the CRU.

Manuscript received October 30, 2020; revised March 11, 2021; accepted May 23, 2021. Date of publication July 20, 2021; date of current version August 16, 2021.

The authors are with the European Organization for Nuclear Research, CERN, 1221 Geneva, Switzerland (e-mail: konstantinos.alexopoulos@cern.ch; filippo.costa@cern.ch).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNS.2021.3098185.
Digital Object Identifier 10.1109/TNS.2021.3098185

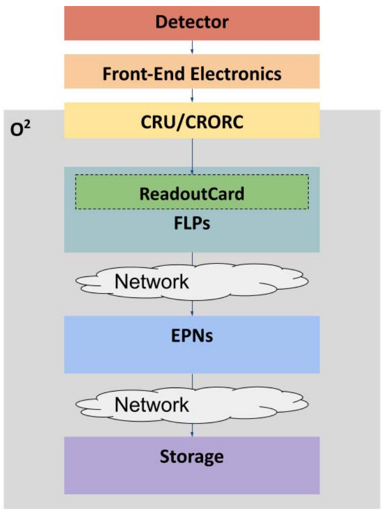

Fig. 1. O^2 facility.

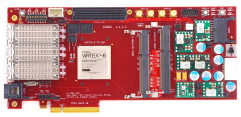

Fig. 2. CRORC.

Fig. 3. CRU.

A. CRORC

The CRORC in Fig. 2 was used to collect data from the majority of detectors of the ALICE experiment during Run2. It is a peripheral component interconnect express (PCIe) Gen 2 × 8 card based on the Xilinx Vertex 6 field-programmable gate array (FPGA), equipped with 12 optical links able to run up to 6 Gb/s each.

The CRORC uses the detector data link (DDL) protocol [8]. A big part of its old firmware has been reused and extended so that it can address the new needs of the detectors and the O^2 facility during Run3. This enables the reuse of DDL-enabled hardware from Run2 for a few detectors that would not benefit from higher throughput capabilities.

B. CRU

The CRU in Fig. 3 is the main readout card for the ALICE experiment during Run3. It is based on the PCIe40 [9] hardware designed for LHCb, a PCIe Gen 3 × 16 card equipped with the Intel Arria 10 FPGA. Connection to the FEE of the detectors happens via up to 24 optical fibers, which can be used for readout, trigger, timing, and/or slow control, depending on individual needs.

The CRU uses the gigabit transceiver (GBT) protocol [10] for its readout links and its firmware is under active development.

C. FLPs

Following an extensive software and hardware assessment process [11] and a competitive tender, the DELL PowerEdge R740 servers were selected to run the FLP portion of the O^2 computing facility. The FLPs come in two flavors, silver and gold, depending on computing needs for data processing. Both flavors run CERN CentOS 7 and are equipped with 96 GB of 2666 MT/s DDR4 memory and a 480 GB solid state drive (SSD) at 6 Gbps. The Silver version uses 2 Intel Xeon Silver Cascade-Lake 4210s, each with 10 cores at 2.2 GHz, whereas the Gold version uses 2 Intel Xeon Gold Cascade-Lake 6230s, each with 20 cores at 2.1 GHz. A single server may be equipped with up to four CRORCs or up to three CRUs, depending on detector and FEE topology and configuration.

III. Kernel Driver

The first layer over the PCIe interface to the cards is the portable driver architecture (PDA) Userspace IO (UIO) kernel module [12], developed by the Frankfurt Institute for Advanced Studies (FIAS). PDA also provides a userspace library in C language, which supports device enumeration and provides a handle to PCIe devices.

Through the PDA handle, the following functionalities may be exploited.

1) Registering direct memory access (DMA) memory targets with the input–output memory mapping unit (IOMMU), which maps the PCIe physical address space to the virtual address space of the CPU. This allows the mapping of separate memory regions to a contiguous memory space while also preventing invalid memory accesses.
2) If the IOMMU component is not physically present in the system or is inactivated, PDA generates DMA scatter-gather lists in order to ensure access to the nonconsecutive physical memory spaces of the DMA engine of the device.
3) Memory mapping the base address registers (BAR) of the device to a virtual address space. This allows the execution of BAR operations on the process level to be carried out by means of simple memory reads and writes.

IV. Userspace Driver

ReadoutCard is an open-source [13] C++ userspace driver and library, which wraps around PDA functionality to access and control the cards. Both cards of the experiment, the CRORC and the CRU, are fully supported by the Readout-Card package, and access is published through a BAR interface as well as a high-level DMA channel interface.

A. Addressing

ReadoutCard accesses the cards on the level of an *endpoint*, which coincides with a PCIe endpoint.

TABLE I
CRU/CRORC ACCESS INTERFACES

Card	CRU		CRORC					
Endpoint	0	1	0					
Link #	0-11	12-23	0	1	2	3	4	5
BAR #	0/2	0/2	0	1	2	3	4	5
DMA Channel	0	1	0	1	2	3	4	5

The CRORC has a total of six optical connections, all within a single endpoint, and for each of them, a BAR handle can be acquired and a DMA channel opened. In other words, the access granularity level of the CRORC is a link for all interfaces.

The CRU boasts 24 optical connections, with each physical card separated into two logical *endpoints*. Each endpoint owns 12 of the 24 links. Contrary to the CRORC, the CRU follows a different scheme to access its links. It publishes two BARs per endpoint: BAR 0 publishes DMA-related registers and facilitates DMA orchestration on the endpoint level, whereas BAR 2 publishes the rest of the registers, i.e., configuration, monitoring, and so on. For what concerns the DMA functionality of the CRUs links, each endpoint orchestrates the DMA transfer through a single DMA channel.

Table I summarizes the addressing details that are specific to each card.

ReadoutCard offers multiple ways of addressing an endpoint.

1) Using the **PCI address** comprised of bus, device, and function number assigned to the PCI device (e.g., `3b:00.0`).
2) Using the **Serial ID** of the device, a pair of the serial number of the card, which is unique to every physical card, and its endpoint (0 or 1) (e.g., `0233:1`).
3) Using the **Sequence ID** of the endpoint, a sequential number assigned by ReadoutCard aiming to simplify development and testing (e.g., #`2`).

B. Enumeration

For what concerns the PDA driver, PCIe devices are enumerated on kernel module insert. When ReadoutCard requests a handle, it receives a handle to a PCI device, which can, by means of the vendor and device ID, be classified as a CRORC or a CRU. Upon receiving the device handle by PDA, ReadoutCard builds upon it to create a `RocPciDevice` object. This object holds a reference to the PDA device handle, initialized handles to the BAR interface(s) of the device, as well as a `CardDescriptor` struct. The latter serves as a standalone identifier for a given endpoint, holding addressing information.

The handle acquisition process takes place every time a process requests a device scan or an interface to a specified device through the ReadoutCard library. Information gathered during the initial stages remains available throughout the lifetime of the process, allowing for an efficient transition between devices and interfaces.

C. Compatibility

The driver is accessing the card at the lowest level through its published registers in the BAR. Between firmware versions, the functionality of a given set of registers may change, ranging from a simple shift of control bits to a completely different submodule. As a result, a firmware incompatibility between what is expected by the driver and what is used by the device can lead to an unexpected state within the device or simply faux operations.

To address this, ReadoutCard checks the firmware of the device in question against a list of supported versions. This check is implemented on the level of the DMA interface as well as for the various command line tools that need to be protected. In all cases, an option to bypass the check is available, which may be necessary during development and testing in between releases.

D. Synchronization

For its synchronization needs, ReadoutCard employs an internal mutex, which uses an abstract UNIX socket as its underlying locking mechanism, called `SocketLock`. The `SocketLock` attempts to bind to the UNIX socket with the specified name, and if it is successful, it keeps the connection open, thus guaranteeing atomic access to the underlying resource, in this case PDA access.

As an example, the PDA library is not thread-safe. It is necessary to ensure that exclusive access is granted whenever a PDA operation is taking place; otherwise, the kernel module or the device might end up in an unsafe state, with undefined side effects.

Even though the `SocketLock` is not the most performant among modern interprocess synchronization mechanisms (see boost::interprocess [14]), securing access for the needs of ReadoutCard takes place during channel initialization, leaving throughput-dependent sections unaffected. Moreover, robustness is much more important, as it is dictated by the large number of users and processes that are active on an FLP at any time. The `SocketLock` guarantees that in case of a pathological scenario (e.g., a process crash), resources will be automatically released, eliminating the need for any manual intervention.

E. BAR Interface

ReadoutCard utilizes PDA to request a map of the BAR of the device in its virtual memory space. Consequently, ReadoutCard implements operations for BAR reads and writes by indexing its BAR-mapped virtual address space. By performing basic BAR limits checks, it ensures that only legal BAR accesses are carried out.

BAR accesses are used for all communication with a device, being that data-taking orchestration, configuration, or monitoring. BAR communication needs to be exclusive during data-taking and, depending on the subcomponent, during configuration. Atomicity during data-taking is anyway guaranteed as will be discussed in Section IV-F, whereas atomicity during configuration is assured through tangent packages on a case-by-case basis.

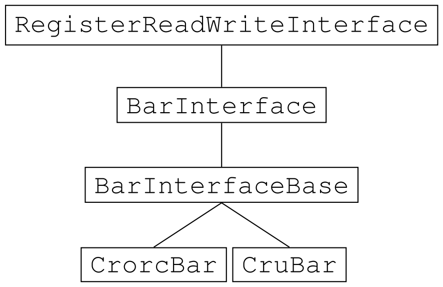

Fig. 4. BAR class hierarchy.

As a result, the BAR interface does not itself enforce mutual exclusion. This is vital to ensure low-latency and service continuity, as multiple tools and users need to constantly communicate with the cards in a responsive way. The number of concurrent processes accessing the BAR interface of a card is only limited by the resources of the system.

Implementation: BAR accesses rely on two interfaces. The `RegisterReadWriteInterface` defines functions for reading, writing, and modifying a register. The `BarInterface` defines functions that utilize the BAR and are common between the CRU and the CRORC. The `BarInterfaceBase` initializes the device if needed, maps the BAR address space to virtual memory, and implements the `RegisterReadWriteInterface`. The `BarInterfaceBase` is extended for every card, so as to implement all card-specific operations, resulting in the `CrorcBar` and `CruBar` classes, as shown in Fig. 4.

F. DMA

DMA is used for data acquisition, which presents the heaviest requirements in terms of throughput. A DMA transfer is facilitated through the use of a DMA channel, which is uniquely opened on the level of an endpoint.

Initializing a DMA channel involves a handshake procedure between the card and the driver, which makes sure that the card is properly reset and relevant structures on both sides, such as buffers, FIFOs, and counters, are initiated in order to support data-taking. Moreover, a userspace shared memory buffer is registered with PDA as the DMA buffer of the transfer, which is in turn validated by ReadoutCard.

Starting with this process, mutual exclusion needs to be guaranteed, as concurrent actions on the same channel will certainly have unintended consequences, invalidating the conditions of the ongoing transfer and overwriting contended buffer memory, leading to a data-taking halt or corrupted data.

To mitigate this, ReadoutCard once again exploits `SocketLock` functionality, acquiring the mutex before any hand-shaking takes place and holding it until the data-taking is finished and relevant buffers have been purged. As a consequence, any attempt to open a DMA channel, while an established transfer on this channel is in progress, will fail gracefully with no side effects.

1) Memory Layout: DMA transfers are performed using DMA channel buffers, made up of shared memory, and are comprised of Superpages. A Superpage (usually 1 MiB) is the granularity level on which the driver and the cards communicate. Each Superpage contains DMA pages, which have a variable size of up to 8 KiB, even within the same Superpage (see Fig. 5). Every DMA page includes a header followed by the payload. With regard to its DMA functionality, the driver is agnostic to the content and structure of a Superpage.

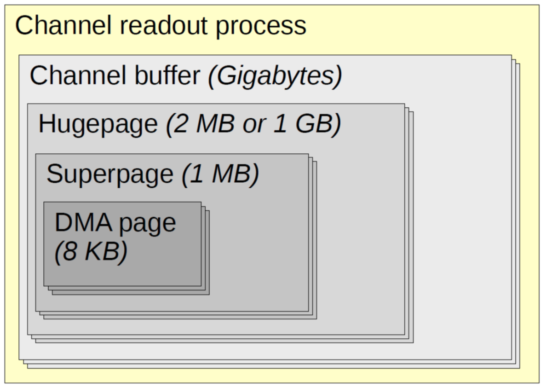

Fig. 5. Memory layout (typical sizes).

2) Communication: As discussed before, DMA channel functionality is always wrapped in a `SocketLock` acquisition associated with the endpoint in question, which guarantees the mutual exclusion property. It is also necessary to clean up potentially leftover PDA buffers as a result of past process crashes. This action also happens under the protection of a `SocketLock`, albeit a different one, the one that is tied to PDA communication.

In order to orchestrate the DMA transfer, the DMA channel holds open interfaces to the BAR, or BARs, of the card, and also publishes convenience functions to monitor health metrics of the transaction, such as the number of dropped packets or the FPGA temperature of the card.

On the successful outcome of the above, DMA buffer registration needs to take place. ReadoutCard provides the PDA driver with a pointer to a userspace buffer to be registered for DMA. For performance reasons, this buffer should be as physically contiguous as possible. To address this, ReadoutCard supports hugepage-backed buffers. Hugepage support is a Linux kernel feature that enables OS support of memory pages larger than the default (usually 4K). ReadoutCard uses 1 GiB hugepages (or 2 MiB as a fallback), greatly limiting the number of DMA-related memory pages. As a consequence, memory page swaps to disk and page table entry lookup times are greatly reduced. Hugepages have been an asset that was heavily used during development and testing. However, ReadoutCard will accept any shared memory buffer as long as it is contiguous. In any other case, the IOMMU needs to be enabled; otherwise, an exception will be thrown.

After PDA initializations, DMA buffer registration, and some exchange of information between ReadoutCard and the underlying card have concluded, data-taking may commence. For both CRORC and CRU, the underlying mechanism enabling DMA transfers is largely the same and both cards are "Superpage aware." The ReadoutCard functionality with regard to DMA transfers boils down to Superpage address and

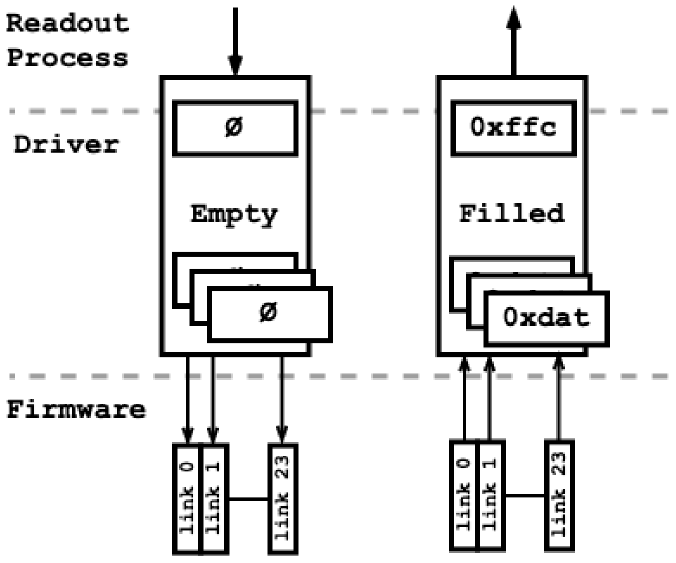

Fig. 6. Superpage data flow.

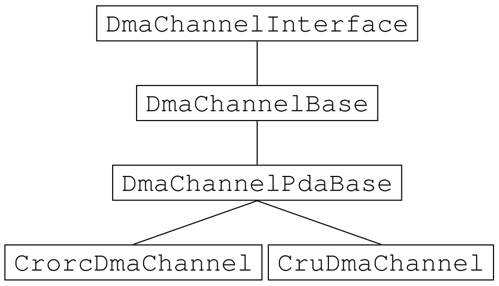

Fig. 7. DMA class hierarchy.

size information exchange, which is facilitated through the use of two Superpage queues, as shown in Fig. 6.

The first one is called `TransferQueue` and manages the "free" Superpages, i.e., the Superpages that are empty and may be populated by the card. When such a Superpage becomes available in the queue, ReadoutCard checks the FIFOs of the enabled links of the card for free slots and proceeds to push the Superpage, or Superpages, in a round-robin fashion between the links. A Superpage push consists of translating the address of the free Superpage, from the virtual address space of the process to the bus address space, before placing it and its size in the relevant FIFOs and notifying the firmware through the BAR.

When a Superpage is filled or "ready," the firmware notifies ReadoutCard by means of an incrementing per-link counter, which will in turn transfer this Superpage to the second queue, the `ReadyQueue`. Superpages in this queue are already filled with data and are ready to be read out by the relevant process.

3) Implementation: The functionality of a DMA channel is dictated by the `DmaChannelInterface`, as outlined in Fig. 7, which includes interfaces for controlling the DMA state and interacting with the two Superpage queues and the device.

The `DmaChannelBase` implements the methods that are common between the card-specific classes, takes care of DMA channel locking, and includes the logging facilities.

DMA buffer-related functions are implemented in `DmaChannelPdaBase`, which also incorporates the high-level DMA start/stop/reset functionality by means of a simple state machine.

Finally, the card-specific DMA Channel classes, i.e., `CrorcDmaChannel` and `CruDmaChannel`, inherit the above and implement device-specific communication, namely the interface to exchange Superpages with the cards.

V. Software

Apart from its userspace driver functionality, the ReadoutCard package also publishes a library that may be primarily used for data-taking through its DMA interface, and control, configuration, and/or monitoring through its BAR interface. Building upon these interfaces, ReadoutCard also provides a set of utilities, the RoC tools, which provide complete solutions for operations that need to be performed on the cards in both development and production contexts.

A. Library

In order to provide adequate support for the needs of the experiment, ReadoutCard provides a number of interfaces with different scopes.

1) DMA: A high-level C++ interface to the DMA channel of the cards is used by processes to perform readout. This is supplied as a well-defined application programming interface (API) that provides functions to control the DMA state by performing start and stop operations. It also provides functions interfacing the two Superpage queues of the driver facilitating the transfer. These functions include operations to push, pop, fill Superpages, and check the current status of the queues. Moreover, the DMA channel offers some convenience functions to get information regarding the underlying card, such as its type, PCI address, firmware info, and corresponding Nonuniform Memory Access (NUMA) node, as well as to monitor health metrics of the transaction, such as the number of dropped packets and the FPGA temperature of the card.

The DMA interface is exploited by O^2 Readout [15], the high-level readout process of the experiment, to access the card, initialize, and perform data-taking.

2) BAR: A high-level C++ interface to the BARs of the cards is heavily used for the internal needs of the driver and by external tools. External tools normally use an interface that supports only simple BAR operations: read, write, and modify the `RegisterReadWriteInterface`. Internally, however, ReadoutCard uses a derived interface, the `BarInterface`, which also defines functions reporting on the status of the cards and facilitating configuration. This offers a complete view of the underlying card for more complex use cases.

Apart from the custom detector solutions utilizing the BAR interface, it also serves as the card communication channel for O^2 Alice low-level front end (ALF) [16], the process that publishes interfaces to detector control system (DCS) [17], so that the latter can access the card and trigger slow control operations.

3) Python BAR: A python wrapper to the BAR interface is used in situations where rapid development cycles are paramount, mainly firmware development and detector teams.

4) CardConfigurator: For a C++ interface to a class orchestrating card configuration, the `CardConfigurator` class is initialized for a specific card with configuration parameters that may be passed programmatically or parsed from a configuration uniform resource identifier (URI), as facilitated by the O^2 Configuration library. It consequently configures all the subcomponents of the card in a modular way.

5) Miscellaneous: ReadoutCard also offers a number of interfaces with a more limited scope to publish software component functionality that may be needed by tangent packages. Examples include interfaces to use the firmware checker, as described earlier, control the pattern player, which controls the pattern player module in the CRU, or header files containing, e.g., register addresses.

B. Tools

The command line interface (CLI) tools provide configuration, monitoring, and testing functionality for the readout cards. The ones providing status output have been integrated with the O^2 Monitoring facility, which takes care of propagating the state to the centrally managed subsystems. Indicatively, the following conditions hold.

1) *`roc-list-cards`:* It lists the available readout cards in the system and provides addressing options and device information (e.g., firmware version and NUMA node).
2) *`roc-config`:* It configures the components and links of the cards in a modular fashion. It takes care not to unnecessarily reconfigure modules unless explicitly forced to.
3) *`roc-status`:* It reports card configuration status, i.e., reports the resulting state of a `roc-config` execution. It also reports the status of the links, namely if they are UP or DOWN coupled with information regarding the optical connections.
4) *`roc-metrics`, `roc-pkt-monitor`:* It reports card metrics and packet statistics. These tools differ from `roc-status` because they report transient information, specific to the current run state, like the current packet rate.
5) *`roc-bench-dma`:* It performs readout and extensive error checking. Heavily used to facilitate development and testing cycles, especially in conjunction with the firmware teams.
6) *`roc-reg-read`, `roc-reg-write`, `roc-reg-modify`:* It performs the most basic, lowest level, register read, write, and modify operations.

Fig. 8 summarizes the integration of ReadoutCard with the tangent software components in the context of the FLP.

VI. Performance

Benchmarks were executed on an FLP, as described in Section II, equipped with the Silver version of the CPU and two CRUs.

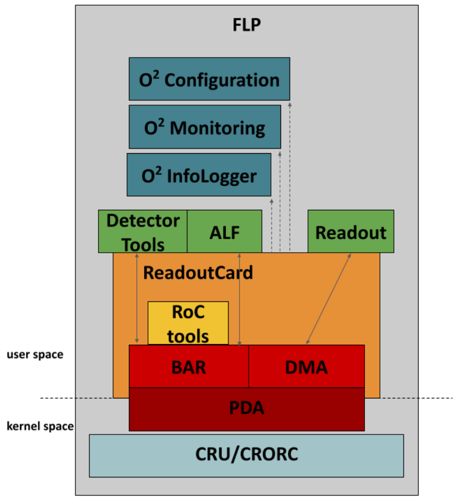

Fig. 8. ReadoutCard integration. Solid lines represent users and dotted lines represent dependencies.

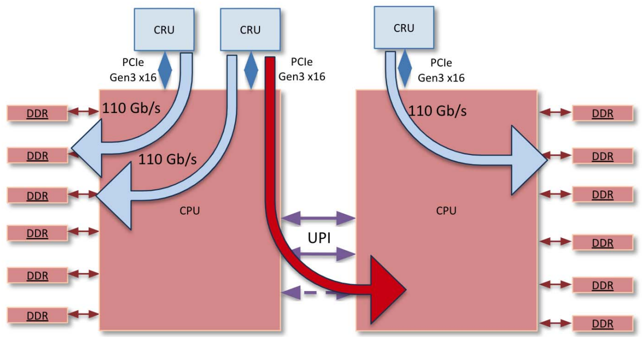

Fig. 9. NUMA node locality.

The FLPs are equipped with dual-socket CPUs, forming two areas of locality, where CPUs are directly attached to their own local RAM. In the NUMA architecture, these are called NUMA nodes. For communication between these two areas, Intel uses the Intel ultrapath interconnect (UPI) [18], a low-latency coherent interconnect for scalable multiple-processor systems in a single shared address space.

The readout cards installed on the FLP are connected to one of the two CPU sockets and thus are part of a single node. In case the process accessing the PCIe endpoint is accessing an NUMA node that is not local to the specific readout card, data will have to flow over the UPI link Fig. 9, which for high-throughput DMA transfers, will be a performance bottleneck. It is thus imperative that processes are pinned to the correct NUMA node when running DMA, a requirement that has been addressed by utilizing the `numactl` utility.

A. DMA Performance

To measure DMA performance, each CRU was configured to use the DMA data generator (DDG), an internal module generating packets. In order to maximize throughput, eight optical links were enabled on the endpoint level. The DMA throughput measurements are presented cumulatively with

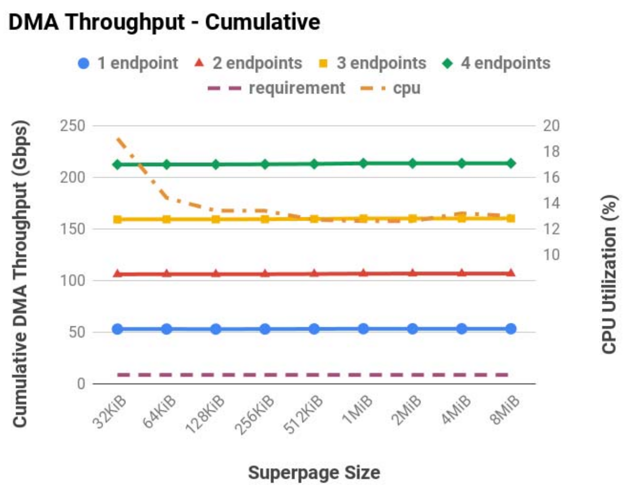

Fig. 10. DMA throughput–Superpage size–CPU utilization.

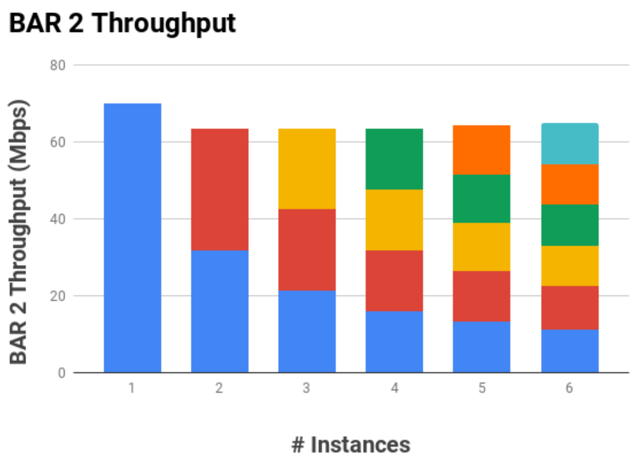

Fig. 11. BAR 2 throughput—# Instances. Different colors indicate bandwidth distribution between instances.

regard to endpoints used and as a function of the Superpage size. Moreover, CPU utilization is also measured for the different scenarios. These tests were facilitated by the `roc-bench-dma` benchmarking tool provided by ReadoutCard. The results can be seen in Fig. 10.

For a single endpoint, the DMA throughput reaches a stable 53 Gbps, which greatly surpasses the requirement of 8.75 Gbps. In addition, the Superpage size does not in any way affect the DMA throughput, which remains stable across the 32 KiB–8 MiB range.

For all four endpoints, i.e., the two CRUs, concurrently running DMA at full speed, the throughput scales well without any bandwidth losses to give a cumulative 212 Gbps.

Throughput remains unaffected in relation to the Superpage size changes for all endpoint layers. The same is not true, however, for the CPU utilization, for which we see a sharp drop as the Superpage size increases. A minimum of 13% of utilization is reached for sizes of 512 KiB–2 MiB, dictating the optimum size range to be used. In addition, the low utilization allows for other processes within the FLP to exploit adequate CPU resources for their needs.

B. BAR Performance

BAR performance was measured with the `roc-bar-stress` tool, which runs a process accessing a single BAR interface and stressing it for a number of operation cycles comprising of sequential read and write operations, at full speed. For the purposes of these benchmarks, a process was run for ten million cycles. The output of this process is equivalent to the total usable bandwidth of a single interface. For BAR 0, the maximum throughput measured reaches 80 Mbps or around 2.5 million 32-bit operations per seconds. For BAR 2, the maximum throughput is slightly lower at 70 Mbps, which roughly translates to around two million 32-bit operations per second. This inconsistency between the two BAR interfaces is due to the fact that they are connected to firmware components that operate on different clocks and thus operate at different speeds. BAR performance is consistently exceeding the operational requirements of the system.

BAR 0 is used for DMA, and under normal circumstances, it should only be accessed by a single process. For BAR 2 on the other hand, multiple concurrent processes are expected to run at the FLP at any time, including monitoring daemons as well as user-initiated operations. Consequently, it is important that the bandwidth of the BAR is distributed fairly among all processes and no starvation is observed. The results of this investigation are shown in Fig. 11. The throughput for multiple instances accessing BAR 2 presents a performance loss when compared to a single instance. As there are no foreseen scenarios for which BAR saturation is expected, this small ($\approx$8%) discrepancy is of no current concern. Nevertheless, when increasing the number of concurrent instances, the cumulative throughput of the interface remains stable and the available bandwidth is demonstrably distributed fairly among all actors.

VII. Conclusion

ReadoutCard is a userspace driver controlling the two cards of the ALICE O^2 computing system. It provides a library with abstract interfaces to access the cards as well as a suite of command-line tools that publish functionality and facilitate development. The ReadoutCard package is the very first layer to access the readout cards within the O^2 facility and has been in heavy use by all subdetectors and subsystems of the experiment for the last three years.

Moving toward operations, the ReadoutCard package will continue to adapt to the evolving requirements. In addition, the suite of tools will be extended to provide more options for debugging both during the detector commissioning phase and production. On the software side, improvements are due with regard to integration with other O^2 software components, specifically the ones facilitating logging [19] and control [20]. Finally, an effort to improve the continuous integration for the package is foreseen, which will also cover scenarios for the various detector use cases.

References

[1] B. Abelev, "Upgrade of the ALICE experiment: Letter of intent," *J. Phys. G, Nucl. Part. Phys.*, vol. 41, no. 8, Aug. 2014, Art. no. 087001, doi: 10.1088/0954-3899/41/8/087001.

[2] P. Moreira, M. Krzewicki, and P. V. Vyvre, "Technical design report for the upgrade of the online-offline computing system," in *Proc. ALICE*, vol. 19, 2015, pp. 33–48.

[3] H. Engel, D. Eschweiler, and D. Francis, "The C-RORC PCIe card and its application in the ALICE and ATLAS experiments," in *Proc. TWEPP*, Aix En Provence, France, Sep. 2014, pp. 1–10.

[4] O. Bourrion *et al.*, "Versatile firmware for the common readout unite (CRU) of the LHC ALICE experiment," in *Proc. TWEPP*, Madrid, Spain, Sep. 2019, pp. 2–6.

[5] P. Boeschoten and F. Costa, "The ALICE O^2 common driver for the C-RORC and CRU read-out cards," in *Proc. ACAT*, Seattle, WA, USA, 2017, pp. 1–6, Art. no. 032001, doi: 10.1088/1742-6596/1085/3/032001.

[6] A. Wegrzynek, V. C. Barroso, and G. Vino, "Monitoring the new ALICE online-offline computing system," in *Proc. ICALEPCS* Barcelona, Spain, Oct. 2017, pp. 195–200, doi: 10.18429/JACoW-ICALEPCS2017-TUBPA02.

[7] *Configuration*. Accessed: Oct. 16, 2020. [Online]. Available: https://github.com/AliceO2Group/Configuration

[8] F. Carena *et al.*, "DDL, the ALICE data transmission protocol and its evolution from 2 to 6 Gb/s," in *Proc. TWEPP*, Paris, France, Sep. 2014, pp. 1–8, Art. no. C04008.

[9] J. P. Cachemiche *et al.*, "The PCIe-based readout system for the LHCb experiment," *J. Instrum.*, vol. 11, Feb. 2016, Art. no. P02013, doi: 10.1088/1748-0221/11/02/P02013.

[10] P. Moreira *et al.*, "The GBT project," in *Proc. TWEPP Conf.*, Paris, France, Sep. 2009, pp. 342–346.

[11] F. Costa *et al.*, "Assessment of the ALICE O^2 readout servers," in *Proc. CHEP*, Adelaide, Australia, Nov. 2019, pp. 1–6.

[12] D. Eschweiler, "Efficient device drivers for supercomputers," Ph.D. dissertation, Dept. Comput. Sci. Math., Goethe Univ. Frankfurt, Frankfurt, Germany, 2016.

[13] *ReadoutCard*. Accessed: Oct. 16, 2020. [Online]. Available: https://github.com/AliceO2Group/ReadoutCard

[14] *Boost Interprocess Library*. Accessed: Sep. 25, 2020. [Online]. Available: https://www.boost.org/doc/libs/1_74_0/doc/html/interprocess.html

[15] S. Chapeland and F. Costa, "Readout software for the ALICE integrated Online-Offline (O2) system," in *Proc. CHEP*, 2018, p. 1041, doi: 10.1051/epjconf/201921401041.

[16] *ALF*. Accessed: Oct. 16, 2020. [Online]. Available: https://github.com/AliceO2Group/ALF

[17] P. Chochula *et al.*, "Challenges of the ALICE detector control system for the LHC RUN3," in *Proc. ICALEPCS* Barcelona, Spain, Oct. 2017, pp. 1–9, doi: 10.18429/JACoW-ICALEPCS2017-TUMPL09.

[18] *Intel Phantomx Ultra Path Interconnect Intel Phantom xUPI*. Accessed: Oct. 16, 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/intel-xeon-processor-scalable-family-technical-overview.html

[19] S. Chapeland *et al.*, "The ALICE DAQ infoLogger," in *Proc. CHEP*, Amsterdam, The Netherlands, Oct. 2013, Art. no. 012005, doi: 10.1088/1742-6596/513/1/012005.

[20] T. Mrnjavac and V. Chibante, "Barroso: Towards the alice online-offline (O2) control system," in *Proc. CHEP*, 2018, p. 1033, doi: 10.1051/epjconf/201921401033.