Development of a Read-Out Receiver Card for Fast Processing of Detector Data

ALICE HLT Run 2 Readout Upgrade and Evaluation of Dataflow Hardware Description for High Energy Physics Readout Applications

Dissertation

for attaining the Ph.D. degree

of Natural Sciences

submitted to the Faculty for Computer Science and Mathematics

of the Johann Wolfgang Goethe University

in Frankfurt am Main

by

Heiko Engel

born in Sinsheim, Germany

Frankfurt 2019

(D 30)

accepted by the Faculty for Comuter Science and Mathematics of the

Johann Wolfgang Goethe University as a dissertation.

Dean:	Prof. Dr. Andreas Bernig
Expert assesor:	Prof. Dr. Udo Kebschull Prof. Dr. Lars Hedrich
Date of the disputation:	10.07.2019

Abstract

Development of a Read-Out Receiver Card for Fast Processing of Detector Data

ALICE HLT Run 2 Readout Upgrade and Evaluation of Dataflow Hardware Description for High Energy Physics Readout Applications

Programmable hardware in the form of FPGAs found its place in various high energy physics experiments over the past few decades. These devices provide highly parallel and fully configurable data transport, data formatting, and data processing capabilities with custom interfaces, even in rigid or constrained environments. Additionally, FPGA functionalities and the number of their logic resources have grown exponentially in the last few years, making FPGAs more and more suitable for complex data processing tasks. ALICE is one of the four main experiments at the LHC and specialized in the study of heavy-ion collisions. The readout chain of the ALICE detectors makes use of FPGAs at various places. The Read-Out Receiver Cards (RORCs) are one example of FPGA-based readout hardware, building the interface between the custom detector electronics and the commercial server nodes in the data processing clusters of the Data Acquisition (DAQ) system as well as the High Level Trigger (HLT). These boards are implemented as server plug-in cards with serial optical links towards the detectors. Experimental data is received via more than 500 optical links, already partly pre-processed in the FPGAs, and pushed towards the host machines. Computer clusters consisting of a few hundred nodes collect, aggregate, compress, reconstruct, and prepare the experimental data for permanent storage and later analysis. With the end of the first LHC run period in 2012 and the start of Run 2 in 2015, the DAQ and HLT systems were renewed and several detector components were upgraded for higher data rates and event rates. Increased detector link rates and obsolete host interfaces rendered it impossible to reuse the previous RORCs in Run 2.

This thesis describes the development, integration, and maintenance of the next generation of RORCs for ALICE in Run 2. A custom hardware platform, initially developed as a joint effort between the ALICE DAQ and HLT groups in the course of this work, found its place in the Run 2 readout systems of the ALICE and ATLAS experiments. The hardware fulfills all experiment requirements, matches its target performance, and has been running stable in the production systems since the start of Run 2. Firmware and software developments for the hardware evaluation, the design of the board, the mass production hardware tests, as well as the operation of the final board in the HLT, were carried out as part of this work. 74 boards were integrated into the HLT hardware and software infrastructure, with various firmware and software developments, to provide the main experimental data input and output interface of the HLT for Run 2. The hardware cluster finder, an FPGA-based data preprocessing core from the previous generation of RORCs, was ported to the new hardware. It has been improved and extended to meet the experimental requirements throughout Run 2. The throughput of this firmware component could be doubled and the algorithm extended, providing an improved noise rejection and an increased overall mean data compression ratio compared to its previous implementation. The hardware cluster finder forms a crucial component in the HLT data reconstruction and compression scheme with a processing performance of one board equivalent to around ten server nodes for comparable processing steps in software.

The work on the firmware development, especially on the hardware cluster finder, once more demonstrated that developing and maintaining data processing algorithms with the common low-level hardware description methods is tedious and time-consuming. Therefore, a high-level synthesis (HLS) hardware description method applying dataflow computing at an algorithmic level to FPGAs was evaluated in this context. The hardware cluster finder served as an example of a typical data processing algorithm in a high energy physics readout application. The existing and highly optimized low-level implementation provided a reference for comparisons in terms of throughput and resource usage. The cluster finder algorithm could be implemented in the dataflow description with comparably little effort, providing fast development cycles, compact code and at, the same time, simplified extension and maintenance options. The performance results in terms of throughput and resource usage are comparable to the manual implementation. The dataflow environment proved to be highly valuable for design space explorations. An integration of the dataflow description into the HLT firmware and software infrastructure could be demonstrated as a proof of concept. A high-level hardware description could ease both the design space exploration, the initial development, the maintenance, and the extension of hardware algorithms for high energy physics readout applications.

Table of Contents

Chapter 1

Introduction

1.1 High Energy Physics Detectors and Goals

Basic research aims for a deeper understanding of our world with numerous past, present, and future experiments on different physical aspects all over the world. High energy physics particle collision experiments, in particular, push the understanding of the fundamental structure of matter. These collisions provide conclusions about small-scale interactions of the original constituents. Particle densities, as they occur in current collider experiments, reveal signatures of how the state of the universe was close to its origin at the big bang. Basic research, in combination with applied science, is a driving force for future applications and technologies and brings innovations in fields far beyond physics.

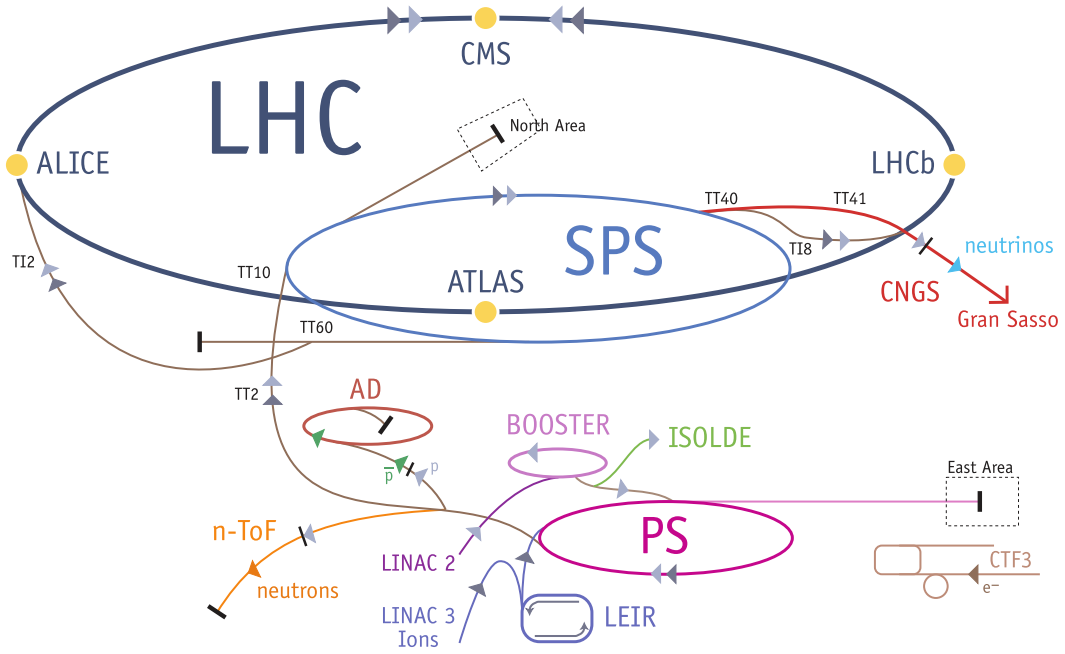

Figure 1.1: CERN accelerator complex, source: Haffner/CERN [Haf 13].

The Large Hadron Collider (LHC) [Eva+ 08] at the European Organization for Nuclear Research (CERN) near Geneva, Switzerland, is currently the most powerful particle collider in the world. It contains two parallel circular beamlines with a circumference of 27 km in an underground tunnel. A number of accelerators and storage rings to feed the LHC and several connected experiments make up a huge complex of systems as shown in Figure 1.1. Both protons and ions can be accelerated close to the speed of light and made colliding in the LHC at energies up to 6.5 TeV

per proton. Four big experiments are located at four crossing points of the beamlines: ALICE, ATLAS, LHCb, and CMS.

ALICE (A Large Ion Collider Experiment) [Aam+ 08] focuses on heavy-ion physics and the study of the quark-gluon plasma (QGP). ATLAS (A Toroidal LHC Apparatus) [Aad+ 08] is specialized in the verification of the Standard Model and searches for physics beyond that model. Together with CMS, ATLAS confirmed the discovery of the Higgs particle in 2012/2013 [Ros 12]. CMS (Compact Muon Solenoid) [Ado+ 08] is, in addition to the standard model and Higgs physics, investigating supersymmetry and heavy-ion physics. LHCb (LHC-beauty) [Alv+ 08] is dedicated to heavy-flavor physics for the verification of the charge-parity-violation.

The LHC started its first operational period with particle collision end of 2009. The first heavy-ion runs with Pb–Pb collisions took place end of 2010. Until early 2013, the LHC delivered colliding beams to the experiments at energies up to 4 TeV per proton. This period from 2009 to 2013 is called Run 1. Starting early 2013, the LHC was shut down for a two-year maintenance period in order to roughly double the beam energies. This period is called "Long Shutdown 1" (LS1). After LS1, the LHC restarted for Run 2 in mid-2015. This second run period is planned to last until end of 2018. Another shutdown period is scheduled after 2018 (LS2) to further increase the luminosity and to prepare the experiments for Run 3 planned for 2020 to 2023 [Heu 15].

All four big LHC experiments are highly complex systems, each with numerous individual detectors to investigate signatures of particle collisions from different aspects and by exploiting different physical properties of the collision products. Each detector translates particle signatures at some point into electrical signals that can be read out and processed to reconstruct the collision vertex, the original constituents, and their energy. A common challenge for all experiments is the orchestration of these massively-parallel electrical signals, the digitization, the correlation, and synchronization of data from different sources, as well as the transport to processing facilities before a reconstruction can even start. A typical detector readout chain starts with the front-end electronics (FEE) located close to the detector. The FEEs mostly contain specialized and custom-built electronics, often with Application Specific Integrated Circuits (ASICs), to achieve high processing rates in electrically and spatially constrained environments in combination with radiation tolerance. Depending on the detector type follows a data aggregation stage that merges detector data from multiple FEE instances onto a smaller number of higher-bandwidth links towards the data acquisition system. This link is often implemented as a serial optical link to achieve electrical isolation and a spatial separation of the detectors in the radiation environment from the data acquisition system in a radiation-free zone. The interface between the serial optical links from the detectors and the computer clusters of the data acquisition system is realized with readout boards, either integrated into the server PCs, as crate systems or as standalone instances connected to the data acquisition systems network infrastructure. A common approach across almost all recent physics detector readout solutions is the use of Field Programmable Gate Arrays (FPGAs). These are not only used as readout boards, but partly also at the data aggregation stage near the detectors or in the trigger electronics.

1.2 Field Programmable Gate Arrays in Detetor Readout

Field Programmable Gate Arrays (FPGAs) were derived from programmable read-only memory (PROM) and programmable logic devices (PLDs) in the mid-1980s. They contain arrays of logic elements with the logic operation of each element and the interconnects between them configurable and re-configurable by the user. A photo of an FPGA is shown in Figure 1.2.

FPGAs are nowadays used in various applications. They have their place between ASICs, Digital Signal Processors (DSP), Central Processing Units (CPU), and Graphics Processing Units (GPU). While ASICs are generally more efficient in terms of performance or radiation tolerance, they are costly for any low- to mid-volume products. Their functionality is static once produced. FPGAs, on the other hand, are reconfigurable. This feature also provides a lower time-to-market for FPGA-based products because hardware production can start once the interfaces to

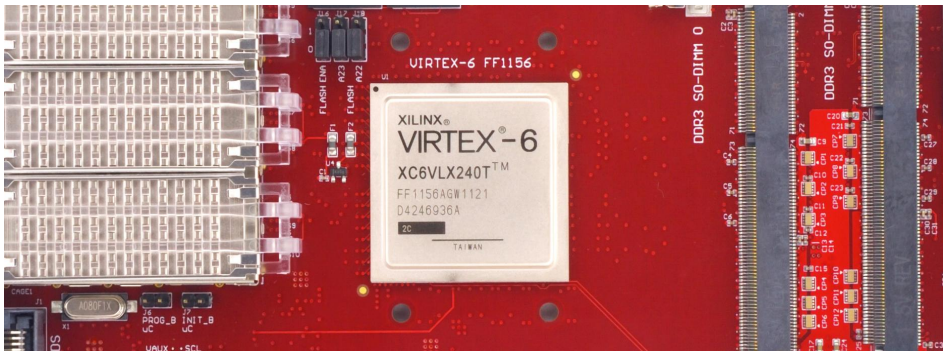

Figure 1.2: Photo of a Field Programmable Gate Arry (FPGA) chip. Photo: Cerntech.

the FPGA are defined, but long before the internal functionality needs to be defined or even implemented. When hardware flexibility is required, FPGAs can still beat DSPs and GPUs despite the lower clock frequencies due to their configurability, the massive parallelism, and the integrated processing blocks. These blocks typically provide the functionality of several DSPs at once and the logic elements can create more flexible combinations of processing steps compared to GPUs. While a CPU is clocked significantly faster than an FPGA, the CPU always requires instructions to complete a full cycle of reading from a register or memor, perform the requested logic operation, and write the result back. Pipelining, vector operations, branch prediction, caches, and other concepts can reduce this latency for independent calculations. Additionally, CPUs operate on fixed data bus widths. FPGAs can easily compete with CPUs for applications where operations can be directly chained or implemented in a massively parallel manner.

The biggest advantage for FPGAs in detector readout chains is their flexibility and parallelism. Detectors are highly complex systems consisting of large numbers of specifically designed custom components handling enormous data rates. Having FPGAs in the chain for operations that are known to be doable with such a device opens the option to build the hardware before or in parallel to the development of the corresponding firmware. On top of this, the ability to reconfigure the FPGA makes it possible to gradually add new features, adjust processing steps to the behavior and characteristics of the actual system, or simply fix bugs that show up only after the production or installation.

1.2.1 FPGA Architecture

The first FPGAs were mainly only what the name suggests: arrays of logic elements that are programmable by the user. While the programmable array of logic elements is still there and growing with each device generation, a number of other features were added over time. External interfaces are highly configurable to operate on various input/output standards and protocols. On-chip memory is available to be used as random access (RAM) or read-only memory (ROM), or wrapped into a first-in-first-out (FIFO) data queue. Digital Signal Processors (DSPs) provide fast hardware blocks for integer and single-precision floating-point multiply-accumulate operations. FPGAs use look-up tables (LUTs) to implement logic operations. Current FPGAs typically have LUTs with six inputs and one or two outputs. They have dedicated connections to implement fast carry chains using several LUTs and have flip-flops (FFs) tightly connected for sequential logic. Figure 1.3 shows a simplified schematic of an FPGA logic block.

On top of this, a number of specialized hardware blocks are available. Internal transceiver blocks provide highly configurable parallel interfaces to fast serial links. Closely connected are hardware blocks for interfaces to a host PC or to a network infrastructure. A number of FPGAs additionally contain a CPU in the same package. The functionality of all FPGA building blocks is defined with a configuration file provided by the user. Depending on the FPGA type, this configuration is either volatile and has to be programmed on each power-on, or it is non-volatile and has to be programmed only once.

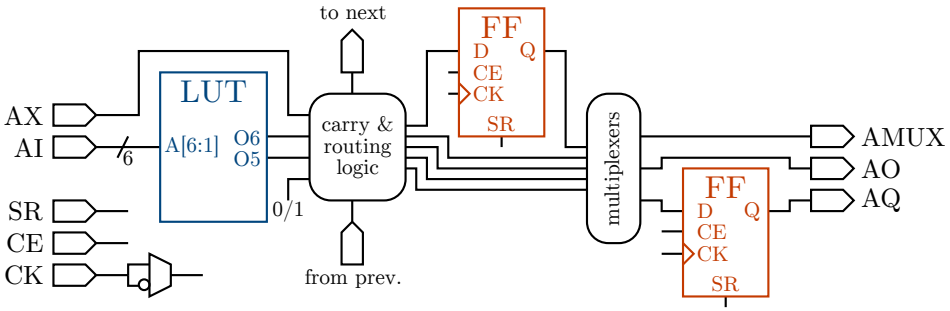

Figure 1.3: Illustration of an FPGA logic block with a 6-input look-up table (LUT) and two flip-flops (FF), similar to the internals of Xilinx Virtex-6 Configurable Logic Blocks [Xil 12].

The main FPGA vendors at the time of this writing are Xilinx and Altera/Intel with the highest-performing SRAM-based models, Microsemi (formerly Actel) with a focus on radiation-hardened flash-based or one-time-programmable devices, and Lattice for simple low-cost devices. Each vendor provides its own set of tools to transform a user-provided hardware description of the device functionality into a configuration file that can be loaded into the FPGA.

1.2.2 Evolution of FPGA Capabilities Over the Last Years

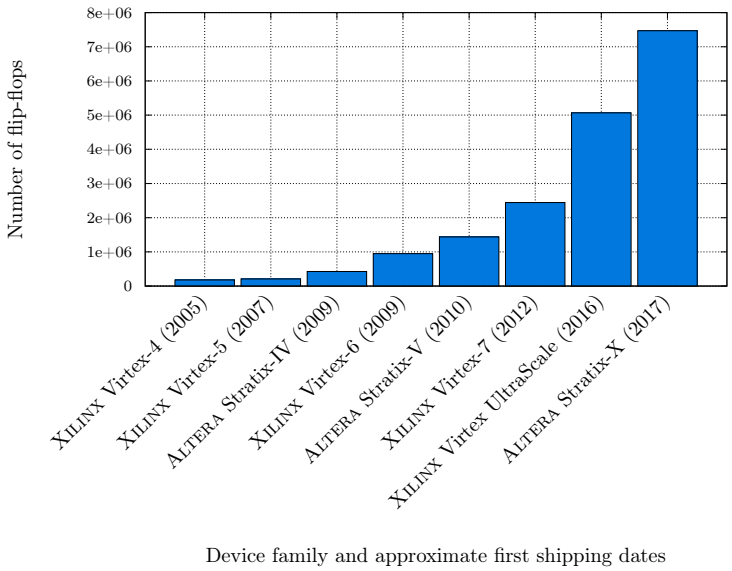

Figure 1.4: Number of flip-flops available in the biggest devices of FPGA families over the last years. The numbers were collected from datasheets and press releases from https://xilinx.com and https://altera.com.

Looking back to the previous generations of FPGAs clearly shows that the number of resources available in these devices increased almost exponentially. Announcements for future FPGA generations seem to keep this trend. Figure 1.4 shows the number of flip-flops available in several current and previous FPGA generations, looking at the biggest available device from each series. This trend is also visible in the number of logic elements, block RAMs, DSPs, and others. However, these are more vendor-specific and are, therefore, hard to compare across vendors. On top of the plain number of resources, the internal components in the FPGAs get faster with each device generation. This provides the ability to run the same functionality at higher clock frequencies or with wider data buses. The strong increase in the number of DSP blocks also makes FPGAs attractive for applications towards High-Performance Computing (HPC), a field currently dominated by CPU and GPU architectures. With Intel acquiring Altera in 2015 [Int 15] the CPU

and FPGA computing worlds moved closer with CPU/FPGA-hybrid devices. The overall development still follows "Makimoto's Wave" [Mak 13] of cyclical alternation between standardization and customization in the semiconductor industry. The conclusions drawn from the extension of this wave fit, at least up to now, with the tight integration of CPUs with application specific hardware blocks and FPGAs in System-on-Chip (SoC) or System-in-Package (SiP) architectures as announced for the upcoming device generations. While these developments open a number of new application fields and may change existing applications, they further encourage the use of these devices in the highly parallel field of detector readout and detector data processing.

1.3 The ALICE Detectors and Online Systems

ALICE consists of 19 individual detectors that analyze different aspects of the signatures from particle collisions. A schematic drawing of the overall system is shown in Figure 1.5 with labels for each detector. The experiment consists of a central part within the red solenoid magnet and a forward muon spectrometer system plus a couple of smaller detectors. A subset of the detectors from the central barrel is briefly described here. A full description of the experiment with all detectors is available in [Aam+ 08].

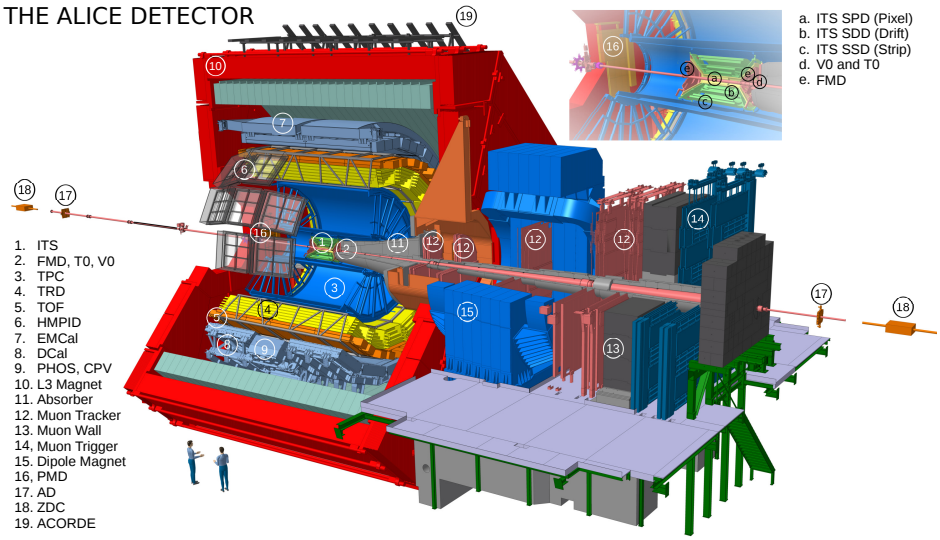

Figure 1.5: ALICE schematics. Source: A. Tauro/CERN [Tau 17].

The crossing point of the two particle beams is in the innermost center of the cylindrical volume that is also shown in the upper right corner of the figure. Collision products leave the interaction point in any direction, creating signals across different detectors. The innermost detector system is the Inner Tracking System (ITS). It consists of six layers of silicon detectors of three different kinds (Silicon Pixel Detector, SPD; Silicon Drift Detector, SDD; Silicon Strip Detector, SSD). They provide a very fast trigger signal, very accurate position of the collision vertex, and information for the particle identification (PID) as well as particle tracking.

Around the ITS sits the Time Projection Chamber (TPC), the biggest detector in ALICE both in volume and data rate. The TPC is a cylindrical gas volume with strong electric fields from the central electrode towards the end caps. Figure 1.6 shows a schematic drawing of the TPC geometry and an illustration of its working principle. The TPC is roughly 5 m in length and around 2.5 m in diameter. The collision products ionize the gas while passing the TPC volume. This leaves traces of electron-ion pairs along the particle trajectory. The strong electric field from the central electrode separates these pairs and makes the electrons drift towards the end caps with a static velocity. The electron drift time from the innermost regions of the TPC near the central electrode to the end caps is around 88 µs. A system of wire layers in the readout chambers on the end caps amplifies the signal and induces charge into arrays of readout pads. This charge

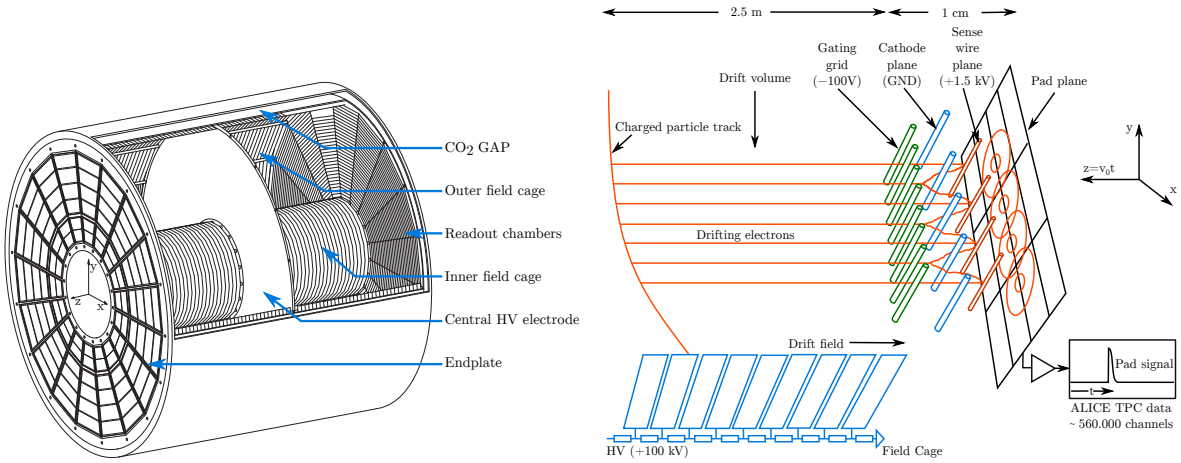

Figure 1.6: Schematic overview of the Time Projection Chamber (left) and its working principle (right). Images: [ALI 13] (modified) / [And 06] (modified).

is proportional to the original space charge in the TPC volume. The charge distribution across the pads is digitized and read out. The TPC is the main tracking device of ALICE.

The Transition Radiation Detector (TRD) is located around the TPC in the form of a hollow cylinder. It consists of six layers. Each layer contains a radiator material that causes leptons to create transition radiation signatures depending on their particle type. The TRD provides fast triggers together with particle tracks. The Time-of-Flight (TOF) detector with precise time measurements of the particles from the collision point to the TOF detector and the High-Multiplicity Particle Identification Detector (HMPID) using Ring-Imaging Cherenkov (RICH) technology complement the particle identification for different transverse particle momentum ranges.

By combining data from the different detectors, the individual particle collisions can be reconstructed. An example of a reconstructed lead–lead collision from 2015 is shown in Figure 1.7. The particle tracks are drawn across the volume of the ITS detectors and the TPC.

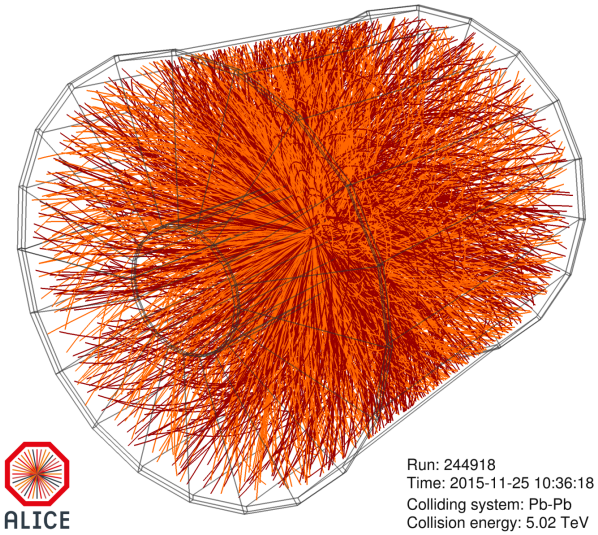

Figure 1.7: One of the first lead–lead collision at 5.02A TeV recorded in Nov. 2015. Image: Weber, Andronic / CERN [Web$^+$ 16].

1.3.1 ALICE Online Systems

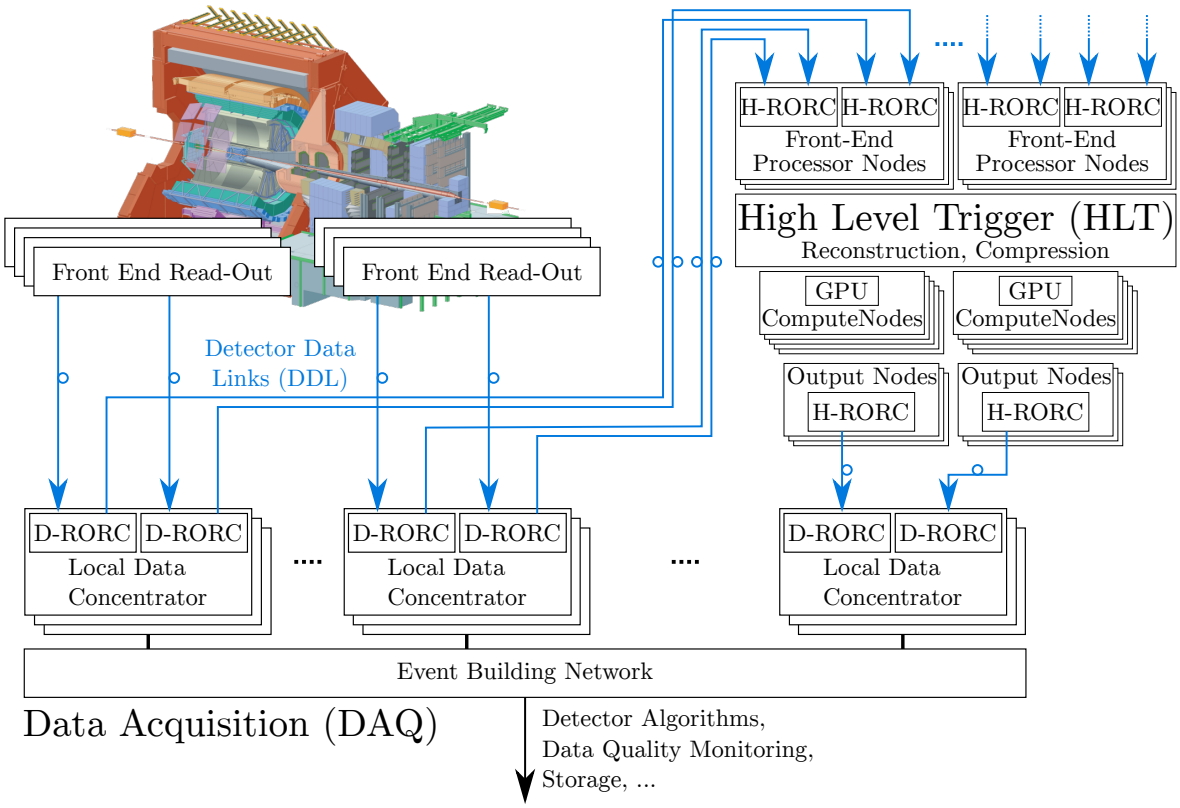

Figure 1.8: ALICE online architecture during Run 1 from 2009 to 2012.

The ALICE online systems manage the operation of the detectors and control the transfer of experimental data from the individual detector systems to a permanent data storage. The online systems mainly consist of the trigger system, the Data Acquisition system (DAQ), the High Level Trigger (HLT), as well as the Detector Control System (DCS) and the Experiment Control System (ECS). These and are described in detail in [Fab+ 04].

A number of detector systems can provide very fast logic trigger signals depending on specific detector measurements, such as a high multiplicity or a high transverse particle momentum. These signals are handled by the trigger system in multiple trigger levels and finally combined into different physics triggers that start the readout of a given set of detectors. Groups of detectors can be triggered independently. Fast detectors can be read out much more frequently than slower ones with this scheme. In return, this means that the readout process after a trigger signal does not necessarily contain corresponding data from every detector. All detectors and the trigger system are synchronized with a central clock provided by the LHC machine. Experimental data from the detectors is marked with a time stamp derived from this clock already in the front-end electronics of each detector. The connection between detector electronics and the data acquisition system is handled with the ALICE Detector Data Link (DDL), a bi-directional serial optical link protocol. The DDL is described in more detail in Section 1.3.2. The optical connection provides electrical, spatial, and logical separation between the detector systems in the experiment cavern and the online systems on the surface. The number of optical links for each detector depends on the required bandwidth and the level of data aggregation that is done already on the detector. During the first and second LHC run period, ALICE is using a bit more than 500 DDLs between the DAQ system and the detectors. 216 of these links are used by the TPC as the biggest detector of ALICE.

The DAQ system handles the dataflow between the detectors and the permanent data storage in the CERN computing center. An overview of the dataflow from the detectors to the DAQ system

is shown in Figure 1.8. The DAQ system is a computer cluster consisting of commercial server, network, and storage hardware. At the interface to the detectors, the DDLs are used both for experimental data transfer from the detectors and for command and configuration upload to the detectors. The interface between the DDLs and the servers is realized with FPGA-based Read-Out Receiver Cards (RORCs). The DAQ Read-Out Receiver Cards (D-RORCs) used during Run 1 are described in more detail in Section 1.3.3.1. The D-RORCs are installed in the first level of server computers, called Local Data Concentrators (LDCs). Each LDC gathers the data fragments sent by the detectors via the DDLs connected to that machine. A second level of machines, the Global Data Collectors (GDCs), collects associated data from all LDCs to build full events. A high-bandwidth storage network provides local data storage for several hours of data-taking. Detector data is then transferred to the CERN computing center for permanent storage. At the end of the first LHC run period in 2013, the DAQ system consisted of 185 LDCs with 440 D-RORCs and 83 GDCs with a theoretical data recording capacity of 4.5 GB/s [Car$^+$ 14].

A copy of the incoming detector data is sent to the High Level Trigger (HLT) using the DDL protocol. The data stream is mirrored inside the D-RORC FPGAs. One copy is read out into the DAQ system and the other copy is transmitted to the HLT via secondary serial optical links on the D-RORCs. The HLT is implemented as a computer cluster using commercial hardware. It provides online data compression and reconstruction. The HLT is described in more detail in Section 1.4. The compressed HLT results are sent back to the DAQ system using the DDL. The HLT decisions, as part of the compressed experimental data, instruct the DAQ system to discard the raw detector data that was received in the detector LDCs in favor of the compressed data from the HLT. The TPC data size was reduced with this method by around a factor of four in Run 1 [Kol 12]. The HLT is an indispensable data compression step especially for heavy-ion physics recording where the raw data rate from the detectors exceeds the maximum storage bandwidth of the DAQ system. For proton-proton collisions, the compression enables to store more physics data with the same permanent storage quota. The interface to the DDLs as the main experimental data input and output of the HLT is realized also with custom FPGA-based readout boards, the HLT Read-Out Receiver Cards (H-RORCs). The H-RORC was developed independently from its DAQ counterpart for Run 1 because it came with additional hardware requirements that are explained in Section 1.3.3.2.

1.3.2 The Detector Data Link (DDL)

The Detector Data Link (DDL) [Rub$^+$ 99] is a serial optical transmission protocol used in ALICE for the data transport between the detector front-end electronics, the DAQ system, and the HLT. The DDL was developed by the DAQ group and the Wigner Research Center for Physics in Budapest for the ALICE Run 1 readout system and is in use in several data acquisition systems around the world. The DDL is designed as a bi-directional point-to-point link to support both the data transport from the detectors and the flow control or configuration upload to the detectors. In order to make it usable with different detectors, it is implemented in a modular way with well-defined interfaces. A schematic overview of a typical DDL connection is shown in Figure 1.9.

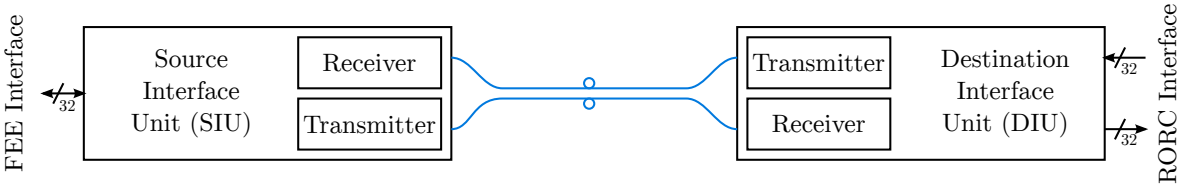

Figure 1.9: Schematic overview of the DDL building blocks.

The Source Interface Unit (SIU) is usually attached to the detector front-end electronics. It provides a bi-directional 32 bit parallel data interface in combination with a couple of control signals towards the front-end electronics. It is optimized to receive detector data on its parallel electrical interface and send them via the optical link. The bi-directional interface reduces the

number of electrical connections to the data source to a minimum. The active data direction of the parallel bi-directional interface to the SIU is controlled from its remote link partner.

On the receiving side is the Destination Interface Unit (DIU) which is typically connected to or integrated into the RORC. The DIU has two independent uni-directional 32 bit interfaces for data reception and command transmission. The high-level data stream across the DDL consists of 32 bit words. All components in the chain can be controlled by sending special command words to the DIU. These command words can be addressed to the DIU, the remote SIU or the remote front-end electronics connected to the SIU in order to send configuration data or to request status information. The DDL implements a Cyclic Redundancy Check (CRC) algorithm in both the DIU and the SIU to ensure the integrity of the transferred data across the optical link.

The DDL implementation in ALICE during Run 1 used the optical link at 2.125 Gbps, resulting in a data bandwidth of around 210 MB/s per DDL. Optical patch panels are installed in several places between the detectors in the experiment cavern and the DAQ system in the counting rooms on the surface. The lengths of the optical cables vary between around 80 m up to around 130 m depending on the SIU location in the detector and the cable routing. More than 500 DDL connections are used between the detectors and the DAQ system to read out the full experiment.

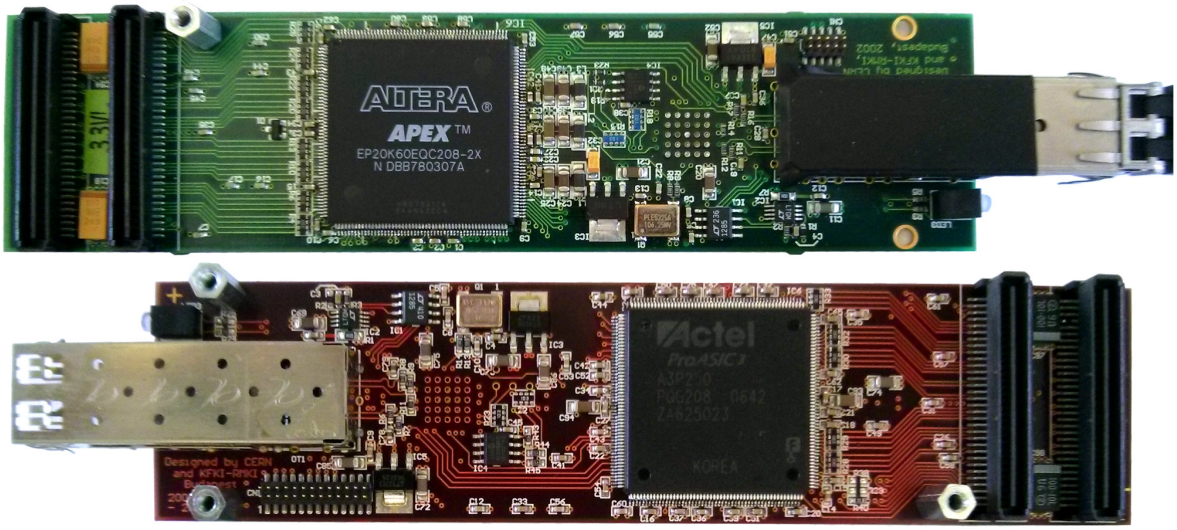

Figure 1.10: Source/Destination Interface Unit (SIU/DIU) modules for the DDL readout as used in ALICE. The bottom module provides radiation tolerance with a flash-based FPGA.

The DIU and the SIU are available as pluggable electronic modules as shown in Figure 1.10. Regular Simple Formfactor Pluggable (SFP) transceiver modules are used for the signal translation from electrical to optical and vice versa. Dedicated serializer/deserializer chips translate the serial electrical data stream into a low-level 16 bit parallel data bus or vice versa. A small FPGA on the module provides the DDL protocol handling functionality. The 32 bit DDL interface is available via the two Compact Mezzanine Connectors (CMC) at the rear end of the modules. The module with the green Printed Circuit Board (PCB) is an example of a DDL module using an Altera APEX FPGA. It can be used as DIU or SIU depending on the FPGA configuration. The red module is an example of a DDL module using an Actel ProASIC3 flash-based FPGA. This module is primarily designed for operation as an SIU module close to the detectors because its FPGA provides a certain tolerance against radiation-induced errors.

The DDL can be operated either by connecting an SIU with a DIU or by interconnecting two DIUs. The DIU–DIU setup provides a full duplex data link across both modules. The SIU–DIU setup has to be configured for one data direction due to the single but bi-directional interface of the SIU. In the ALICE case, both configurations are used as shown in Figure 1.11. All optical links are bi-directional, the arrows are shown to indicate the primary flow of detector data. The detector electronics use the SIU as their interface to the DDL. The data direction of the parallel

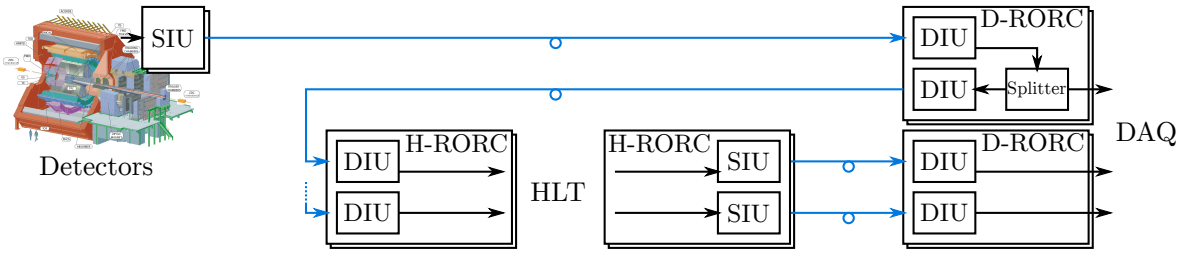

Figure 1.11: DDL interconnection architecture in ALICE.

interface to the SIU is configured from the DAQ system. On DAQ side, the D-RORCs implement the corresponding DIU interface to receive the detector data from DDL. The connection for the detector data copy to the HLT is realized with a DIU–DIU setup. The data path for the compressed HLT data back to DAQ uses again a SIU–DIU configuration. With this setup, the general dataflow is fully controlled from the DAQ system. The DAQ system triggers a bus turnaround in the detector SIUs and the HLT SIUs when starting the readout and closes the link again after the end of a run.

The DDL with its constituent parts DIU, SIU, and optical fiber forms a transparent data channel for 32 bit words between the detectors and the DAQ and HLT systems. All experimental data from the detectors is packed into multiples of 32 bit words. The data block being transferred over a single DDL upon a trigger is called a "sub-event". The collection of all corresponding sub-events from all participating detectors for a given trigger forms an "event". In order to unify data handling procedures, all sub-events follow a common data format. Each sub-event starts with the ALICE Common Data Header (CDH) [Div+ 07]. This header contains a unique identification number (event ID) that can be found as well in all corresponding sub-events transferred via the other DDLs. It holds information about the type of the trigger and the subset of detectors that were read out. The encoding of sub-event data after the CDH is detector-specific. Special control flags in the SIU and the DIU interface mark the end of a sub-event.

1.3.3 The Read-Out Receiver Cards

The Read-Out Receiver Cards (RORCs) in ALICE provide the interface between the optical DDLs and the computer clusters of the DAQ and the HLT systems. They are realized as server plug-in boards with optical interfaces for the DDLs and an electrical computer bus interface to the host machine. The RORCs of the DAQ and HLT systems were developed as independent projects for Run 1 due to different requirements for the two use cases.

1.3.3.1 D-RORC

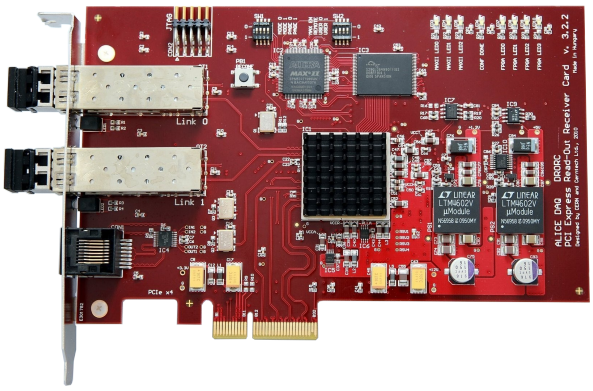

Figure 1.12: PCI-Express variant of the DAQ Read-Out Receiver Card. Photo: [Car+ 14].

The DAQ Read-Out Receiver Card (D-RORC) exists in two hardware variants with different interfaces to the host machine. The earlier variant provides a PCI-X (Peripheral Component Interconnect eXtended) interface to the host machine. The newer variant uses the PCI-Express interface. A photo of the PCI-Express D-RORC is shown in Figure 1.12. The board hosts two optical transceiver modules to connect up to two DDLs. The pluggable DIU modules shown in Figure 1.10 are not used here but the DDL protocol is directly implemented in the FPGA firmware. The D-RORCs use Altera Stratix-II FPGAs and provide a raw bandwidth of roughly 800 MB/s (PCI-X) or 1000 MB/s (PCI-Express) to the host machine. A special requirement of the D-RORCs are Low Voltage Differential Signaling (LVDS) connections to the TPC Busy Box [Muk 07]. The event IDs from all TPC sub-events received in the D-RORCs are sent to the Busy Box via this dedicated connection. The Busy Box ensures that all D-RORCs have received all sub-events or pauses the trigger. The connection to the Busy Box uses the same connectors and cables as used for Ethernet. The connector is visible below the transceiver modules in the figure above.

1.3.3.2 H-RORC

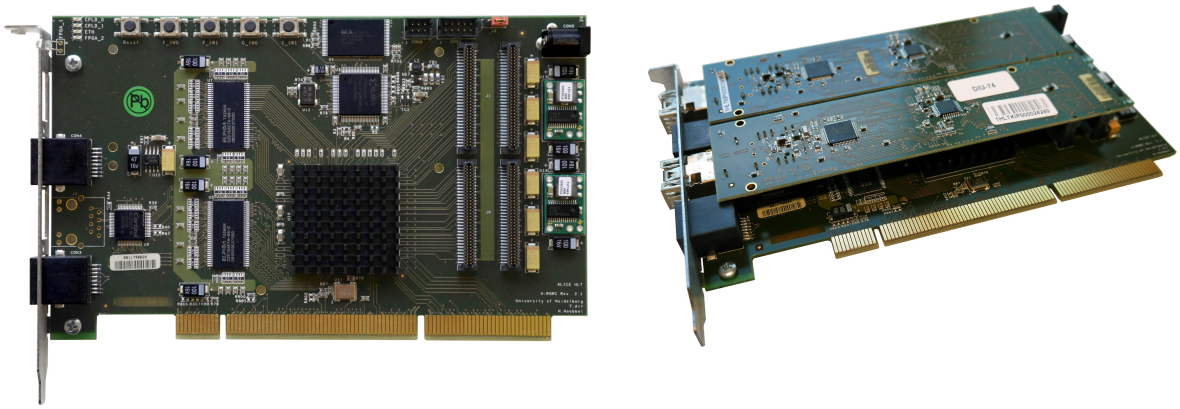

Figure 1.13: HLT Read-Out Receiver Card in Run 1.

The HLT Read-Out Receiver Card (H-RORC) [Alt 17] used during Run 1 utilizes the pluggable DIU/SIU modules described above as the interface to the DDL. Figure 1.13 shows the board without (left) and with the DDL modules attached (right). While the board itself looks similar to the D-RORC, it is different in both its architecture and functionality. The H-RORC serves two functions in the HLT: it is used both at the input interface of the HLT to receive detector data from the DAQ system and at the output interface of the HLT to provide the HLT decision and compressed detector data back to the DAQ system. These are two contrary dataflow directions. At the input interface, detector data is received with a DIU module and transferred to the host machine. At the output interface, the compressed HLT data is fetched from the host machine and transmitted via an SIU module. The HLT makes use of the H-RORC FPGAs to pre-process experimental data immediately in hardware. A specialized processing core placed directly into the data stream between the DDL and the interface to the host machine provides high-speed data processing capabilities with an only marginally increased readout latency. A central component of the HLT in Run 1 in this regard was the TPC hardware cluster finder algorithm implemented in the H-RORC FPGA that pre-processes raw TPC data. This saved significant CPU processing power in the HLT. The Run 1 cluster finder is described in detail in Section 4 and in [Alt 17].

Due to the online processing of detector data in hardware, the H-RORC had to implement a bigger FPGA than the D-RORC. Additionally, the HLT requires on-board storage to replay detector data for a full-chain testing, verification, and characterization. Real or simulated detector data can be loaded into this memory on the H-RORC and replayed as if it were coming from the optical links. This provides the option to operate the full HLT as a standalone system for

testing, benchmarking or debugging at fully controlled conditions. It is independent of the LHC state and does not require the detectors or the DAQ system to provide the corresponding data.

The H-RORC is equipped with a Xilinx Virtex-4 LX40 FPGA. 128 MB of on-board DDR-SDRAM are available for data replay purposes. The H-RORC uses PCI-X at 133 MHz as an interface to the host machine. This provides a measured payload throughput of 935 MB/s. Up to four FPGA configuration images can be stored on the board, with one of them loaded automatically on power-on. The DDL connectivity is achieved by connecting up to two DIU or SIU DDL modules to the CMC connectors at the rear end of the board. Around 240 of these boards were in use in the HLT during Run 1, handling 425 DDLs for raw detector data from the DAQ system and 28 DDLs for the transport of the HLT results back to the DAQ system. Up to two H-RORCs were installed per HLT node.

1.4 ALICE High Level Trigger

The goal of the HLT is to reduce the data rate to the permanent storage system without affecting the contained physical information. The original concept of the HLT is described in detail in [ALI 04], its implementation for Run 1 is explained in [Alt$^+$ 06, Ric$^+$ 07]. The HLT in Run 1 was designed to reduce the raw data rate from the detectors of up to 25 GB/s to a rate that the DAQ system is able to store. This was around 4 GB/s for Run 1 [Car$^+$ 14]. Any additional data reduction directly translated into the option to record more physics data with the same overall storage quota accredited to the experiment. The focus of the data reduction is clearly on the Time Projection Chamber detector, because its data makes up more than 80% of the total experimental raw data. Data reduction can typically be achieved with three methods or a combination thereof:

- Triggering: selecting events to be stored based on an online reconstruction of physics observables.
- Selection: storing only relevant parts of an event or extracting data from a region of interest.
- Compression: reducing the size of the optionally triggered or selected experimental data by applying lossy or lossless data compression algorithms.

Triggering and selection turned out to be no constructive methods for heavy-ion physics in the HLT [Kol 12]. However, data compression provided the required data reduction by a factor of more than four.

The HLT performs online data reconstruction and compression. A unique feature among the LHC experiments during Run 1 was the usage of hardware accelerator devices like FPGAs and GPUs for the data processing. GPUs were used to accelerate the reconstruction of collision fragment trajectories, called "tracking". With this approach, each GPU saved around one additional server node in the HLT cluster for an equivalent event processing rate [Roh$^+$ 12]. The raw detector data from the Time Projection Chamber was processed with a cluster finding algorithm already in the H-RORC FPGA. This cluster finder on its own reduced the data size on average by a factor of around 1.2 [Alt 17] in Run 1 while saving significant amounts of CPU processing power in the HLT. In combination with software-based format optimization of the cluster properties and a lossless Huffman compression algorithm, a data reduction factor greater than four was achieved in Run 1.

For each event, the HLT sends a decision together with the compressed and reconstructed experimental data to the DAQ system. The DAQ system evaluates based on this decision which parts of an event should be stored or discarded. The HLT decision is a bitmask of all detector DDLs. This bitmask defines the set of sub-events to be stored for a given event. The HLT data compression is applied by deselecting the raw TPC links in the link mask and selecting the HLT output link that carries the compressed TPC data instead. For quality assurance reasons, 1% of

the raw data is always stored together with the HLT compressed results by selecting all links in the HLT decision bit mask.

The HLT can be operated in different modes. Mode A means the HLT is disabled and does not even receive a copy of the detector data from the DAQ system. Mode B means the HLT is active and the HLT decisions are stored but not evaluated by the DAQ system. In this mode, both the raw detector data and the compressed data from the HLT are stored. This mode is typically only used when the HLT results need to be verified by the Offline analysis algorithms. This mode cannot be applied in high-rate scenarios because adding the HLT results on top of the raw data volume increases the data rate to the storage system even more. Mode C is the target operation mode: The HLT receives a copy of the raw detector data from the DAQ system and the DAQ system discards the raw detector data in favor of the HLT compressed data based on the HLT decision.

The HLT computer cluster is based on commodity server and network hardware. At the end of Run 1, the HLT consisted of around 225 nodes interconnected with an InfiniBand QDR network. The main detector data input and output interface was realized with the H-RORC boards installed into 117 of the nodes and connected to the DDL network via optical fibers. The servers containing the H-RORCs were called Front-End Processors (FEPs) and had a larger form factor for the installation of the two FPGA boards in PCI-X sockets. In contrast to the FEPs, Compute Nodes (CNs) could be integrated more densely. They were responsible for performing event reconstruction and partly contained GPUs as accelerators. A couple of additional infrastructure and portal nodes provided the core services and external interfaces to the cluster. Over the time of initial installation, commissioning, and operation during Run 1, as well as in combination with hardware replacements and successive improvements, the HLT cluster became a heterogeneous system with a number of hardware models and devices.

1.4.1 HLT Data Transport Framework and H-RORC Integration

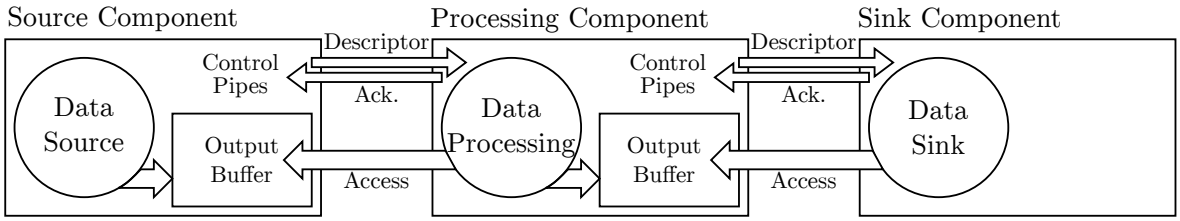

Figure 1.14: Publisher-subscriber model.

The data transport in the HLT is realized with a custom data transport framework [Ste 04] based on the publisher-subscriber model. A network of components distributed across the cluster handles the data aggregation, distribution, processing, and input/output handling for the whole system. The framework is implemented in a way that publishing components provide a data descriptor to their subscribers with information on where to find the corresponding data block in the publisher's output buffer. The subscribed component can directly access the publisher's output buffer to retrieve the data, process it, and place the results into its own output buffer for downstream subscribers. An acknowledgment channel in the reverse direction notifies publishers as soon as all subscribers are done with a given data block and the corresponding memory in the publisher's output buffer can be freed or reused for other data. A sketch of this scheme is shown in Figure 1.14. Source components provide input data into the system. They are not subscribed to other components but receive their data from different input interfaces, mainly the readout of the DDLs via the H-RORC FPGA boards. Sink components form the endpoints of the data processing chain. For the HLT output links, such sinks handle the transfer of the HLT decisions and the processed data via the H-RORCs and the DDLs to the DAQ system. The data transport framework provides a number of components to deploy the processing across the full cluster. Merger components collect and merge corresponding sub-events from different input sources or

processed results of the same event from different components. Scattering components distribute the work across several processing components. Bridge components transparently handle the distribution of tasks and data across the network. The same processing components can be instantiated multiple times for parallelism and higher processing rates. The actual data processing is achieved by running AliRoot [ALI] components as processing elements in the HLT framework. AliRoot is the ALICE Offline processing framework. Therefore, the same processing components can be run in the HLT online environments and in the Offline reconstruction frameworks.

Each data processing component, each source, sink, merger, and bridge is a separate Linux process. All processes are controlled from a hierarchical structure of *TaskManagers* that follow a state machine to start and stop the processes, initiate the subscription process to build the processing chain, and make them start processing data. A set of general HLT configuration files defines the chain of components to be instantiated, the list of machines that should be used, and a list of RORC channels providing data from the DDLs. The sub-events from the detectors do not necessarily carry information about their origin. For this reason, special care has to be taken to ensure that all DDL fibers are plugged correctly and the framework knows exactly where to expect data from which link. The actual HLT configuration with the list of processes to be run on each machine is built on-demand by taking general run and readout parameters into account. These are received from the Experiment Control System at the start of each data-taking run.

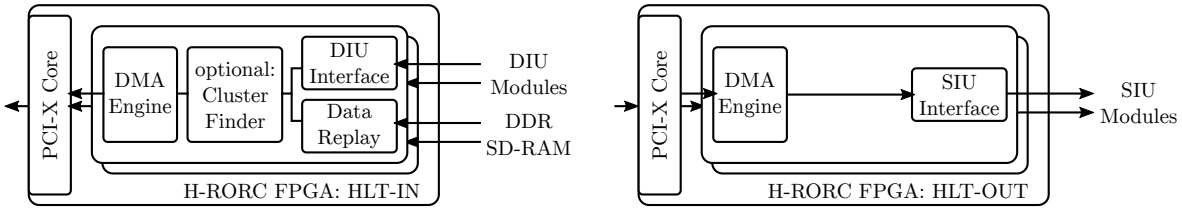

Figure 1.15: Schematic overview of the data flow in the H-RORC firmware variants.

The RORCs are integrated into the HLT data transport framework as data source components to receive detector data from the DAQ system (HLT-IN) and as data sinks for the HLT results to be sent to the DAQ system (HLT-OUT). A schematic drawing of the dataflow and the high-level building blocks of the H-RORC firmware variants is shown in Figure 1.15. The data transport from the RORCs into the host RAM is done with Direct Memory Access (DMA): The RORC writes into the RAM of the host machine without direct CPU involvement. The RORC is provided with bus addresses of available RAM pages and writes the detector data to these pages. It notifies the software framework whenever a new sub-event was transferred. The data transport from the host RAM to the RORC for the HLT output case works the other way around: the software framework places the processed data into RAM pages accessible by the RORC and provides a descriptor for this data block to the RORC. The RORC fetches the data block from host RAM and sends it via the DDL to the DAQ system.

The DMA buffer allocation on the host systems in Run 1 was realized with the "BigPhysArea" kernel patch set [Mid 03] that separated physically consecutive memory from the operating system early at boot time and provided access to it. The PSI device driver [Pai 07] handled both the virtual address to bus address translation for this memory for the H-RORC access and the software access to the buffers and H-RORC-internal registers.

DMA transfers with the H-RORC were implemented using two buffers per readout channel. The H-RORC wrote the sub-event received via the DDL into an *Event Buffer*. Once a full sub-event was transferred, the H-RORC placed an entry into a separate *Descriptor Buffer* containing information about the corresponding data block in the Event Buffer. The memory locations of the descriptors were known because all descriptors have the same size and started from a static offset. The readout software polled the Descriptor Buffer for new entries and eventually handled the data in the Event Buffer. The readout of the input DDLs and the handling of the HLT output channels was implemented with separate and independent DMA channels in the firmware. Each DMA channel and, therefore, each DDL was handled with its own process in the

HLT data transport framework, the *RORCPublishers*. The data received from the H-RORCs was either raw detector data as received via the DDLs or detector data that was pre-processed with detector specific algorithms already in the H-RORC FPGA.

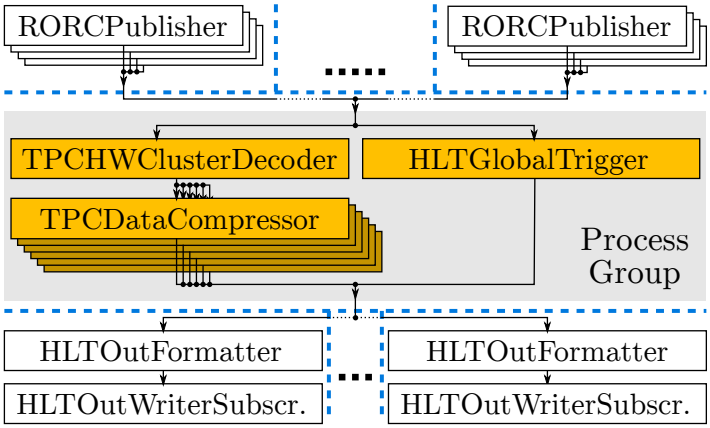

Figure 1.16: Example of an HLT framework chain for TPC data compression. The orange boxes indicate AliRoot tasks running as framework components.

Figure 1.16 shows an example of an HLT chain for the TPC data compression. 216 *RORCPublisher* components control the readout of data from 216 TPC DDLs from the RORCs. The raw TPC data is already processed in the FPGA with the cluster finder algorithm. The RORC DMA Event Buffers are at the same time the output buffers of the *RORCPublisher* component in the data transport framework. The sub-events read out from all 216 *RORCPublishers* are aggregated with merger components and handed over to one of the compute nodes. The software processing is implemented with AliRoot components. The *TPCHWClusterDecoder* parses the received data and performs some data format adjustments and coordinate transformations as preparations for the actual data compression. Multiple instances of the *TPCDataCompressor* component share the workload to compress the clusters with a Huffman data compression technique. The compressed results together with the HLT decision are sent to one of the *HLTOutFormatter* components to assemble the output data block according to a predefined data format. The subscribed *HLTOutWriterSubscriber* component makes the output data block available for the RORC and handles the interaction with the FPGA. The RORC finally sends the compressed clusters and the HLT decision to the DAQ system via the DDL.

1.4.2 Hardware Cluster Finding in the H-RORC

The cluster finder algorithm and its implementation in the H-RORC for Run 1 was a major part of the work of Torsten Alt [Alt 17] and is also further described in Chapter 4. The hardware cluster finder searches for space charge clusters created by collision products passing the TPC volume and calculates their properties. The cluster finding in the FPGA is applied in two dimensions in the pad-time-plane for each row of TPC readout pads. Both dimensions are handled separately. The hardware cluster finder is often referred to as a (1D + 1D) algorithm: The data is processed first in the time direction and only resulting signals are additionally processed in the pad direction. This is in contrast to the Offline reference software implementation which implements a real 2D cluster finder by taking both dimensions at the same time into account. By fitting the measured charge distributions in the time and pad directions with a Gaussian distribution, the clusters can be located in the TPC volume with an accuracy below the intrinsic size of the readout pads and the ADC sampling time granularity. The HLT hardware implementation uses an algorithm based on weighted sums that proved to provide an equivalent precision to a full Gaussian fit approach and can be mapped efficient onto an FPGA. The clusters found by the hardware cluster finder are then characterized by their position and width in the time and pad direction as well as by

their total and maximum charge values. The correlation of clusters in the third dimension across rows, partitions, and sectors is part of the tracking process performed on GPUs and CPUs.

The cluster finder was implemented as a streaming processor in the H-RORC firmware in Run 1. It was integrated between the DDL interface and the DMA engine and was designed to be able to handle the data rate coming from the TPC front-end electronics via the DDL. The data processing in hardware only added a minor additional latency to the readout path in the H-RORC while providing significant savings in terms of required CPU processing power for the HLT. Two cluster finder instances were integrated into the H-RORC firmware, one for each detector link. They were operated with a clock frequency of 133 MHz. A number of parameters to the algorithm were configurable at runtime via software. The hardware implementation fit exactly what was possible with the H-RORC FPGA both in terms of resource usage and clock frequencies and has proven its performance in the operation during Run 1.

1.5 Motivation and Goals for this Work

After the end of the Long Shutdown 1 in early 2015, the LHC machine started again with delivering particle collisions to the LHC experiments at an increased energy of 13 TeV [CER 14] for the second LHC run period, Run 2. The integrated luminosity for heavy-ion collisions in ALICE was expected to accumulate to $1\,nb^{-1}$ during Run 2. This is ten times the integrated luminosity from Run 1 [Tie 15]. During 2016, the LHC reached its design luminosity and even exceeded it by 40% [LHC$^+$ 16]. The shutdown period between the first and the second LHC run period was used in ALICE for a number of installation, consolidation, and upgrade works [Ron 15]. Multiple detectors were extended or completed with installation works that were not possible before the start of Run 1. The increased LHC energy and luminosity, as well as the upgrade of several detectors, projected a significantly increased load on the ALICE online systems for Run 2. The TRD detector was completed and a firmware upgrade of the front-end electronics roughly doubled its readout bandwidth per DDL. During Run 2, the TPC as the biggest detector in ALICE replaced its Readout Control Unit (RCU) with a new version that provided roughly double the bandwidth compared to the Run 1 setup [Alm$^+$ 13]. This resulted in a maximum bandwidth of around 80 GB/s from the TPC to the DAQ and HLT systems. The DAQ team replaced a number of their LDC and GDC machines with new hardware. The bandwidth to the permanent storage system was increased from around 4 GB/s in Run 1 to 7 GB/s in Run 2 [Div 16]. The HLT replaced all of its Run 1 hardware with new servers, network infrastructure, and GPU accelerators in order to be able to handle the data rates and processing requirements in Run 2.

A central part in these consolidation works on the online systems were the Read-Out Receiver Cards in the DAQ and HLT systems. The Run 1 D-RORCs and H-RORCs were both designed for operating two DDLs per board at 2.125 Gbps per link. Link rates above this value were hardly possible due to hardware limitations. Parts of the D-RORCs and all H-RORCs had a PCI-X interface to the host machine. This interface was mostly obsolete at the end of Run 1 and was hardly available in recent server machines, or only in legacy hardware. This was particularly challenging for the HLT where state-of-the-art GPU and server hardware was foreseen. The Run 1 H-RORC FPGA firmware with the hardware cluster finder required almost all usable FPGA resources. Therefore, it was hardly possible to extend the hardware processing tasks with this board. It was very clear that the H-RORCs could not be reused in Run 2 at all. Integrating the existing H-RORCs into a new HLT was impossible for detectors with an increased readout link rate and highly infeasible for all remaining detectors due to the PCI-X host interface. The same was true for all D-RORCs that required a detector readout with a link rate above 2.125 Gbps. This applied to all 216 DDLs from the TPC, 36 DDLs from the TRD, and 28 DDLs from the HLT. In total, this is more than half of the detector DDLs in the DAQ system. For detectors that stayed at 2.125 Gbps, the DAQ system could reuse the existing PCI-Express D-RORC variant.

One aspect of this work was to provide a common hardware platform for ALICE in Run 2 that could be used as a replacement for the H-RORCs and D-RORCs in the DAQ and HLT systems.

This device had to support the DDL at higher link rates and had to come with an interface to its host machine that is well supported by current and upcoming server generations. The hardware platform should require as little adjustments as possible to existing experiment infrastructure that is reused in Run 2. This applied mostly to the fiber installation and the interfaces to other components and systems. For the DAQ system, it was strongly requested to be able to integrate the new hardware into the existing server and software installation in order to operate both the D-RORCs and the new hardware with the same framework. For the HLT, the hardware had to provide enough FPGA resources to implement and extend the hardware cluster finder for the increased TPC readout data rates. A compact form factor was preferred in order not to limit the compatibility with a broad range of server machines. An increased link density with more than two DDLs per board was highly welcome because it reduced the number of server nodes and boards required to handle all DDLs.

A second aspect of this work was to integrate the new hardware into the new HLT cluster and the existing HLT data processing framework. The integration of the hardware platform into the DAQ system was planned to be done by the DAQ group. The works on the HLT involved firmware and software developments for the operation of the RORCs in Run 2. The reuse-ability of existing Run 1 H-RORC firmware components depended strongly on the hardware platform. The obsolescence of PCI-X, however, suggested that, in particular, the interface to the host machine cannot be reused anyway. An overall firmware functionality similar to the H-RORC, with a data transport between the DDLs and the host RAM, had to be provided as part of this work. A primary goal of the firmware development was also to integrate the hardware cluster finder algorithm into the new readout board. The implementation had to be at least extended in order to support the increased link rates and the changing TPC front-end hardware as well as the data-taking conditions in Run 2. Any algorithmic improvement on top is highly beneficial for the experiment.

A third aspect of this work was related to the pre-processing of detector data already inside the RORC FPGA. The previous Run 1 hardware cluster finder was developed as part of a Ph.D. thesis over a period of several years and in a low-level hardware description language. It was changed and partly rewritten several times during its development due to changing requirements on the data processing algorithm and experiences with the detector. Using one of the typical low-level languages is a common choice for hardware description on FPGAs. Describing data processing algorithms at the the same level, however, introduces a considerable degree of complexity into both the firmware development process and its maintenance regarding changes to the algorithm or the implementation. Already small algorithmic changes can require major reworks on the hardware implementation with high development, verification, and maintenance efforts. In this work, an alternative approach for the firmware development was chosen to be evaluated that describes the hardware processing steps at a higher abstraction level. Dataflow computing is a promising High-Level Synthesis (HLS) approach that comes with an algorithmic description of the hardware functionality. With detailed performance numbers available for comparison from the low-level implementation, the hardware cluster finder could serve as an exemplary use case. The algorithmic hardware description had to be evaluated on its suitability to ease the development and maintenance of hardware-based data processing in the context of high energy physics readout applications.

The efforts to find a solution for a common hardware platform for the RORCs in the DAQ and HLT systems in Run 2 started around mid-2010. The second LHC run period restarted with proton–proton collisions in mid-2015. Until then, the new solution had to be developed, tested, integrated, installed, commissioned, and had to be ready for physics operation. The new hardware was then foreseen to be used in the ALICE DAQ and HLT production systems throughout Run 2 from mid-2015 until the end of 2018. The RORC is a crucial component in the ALICE online systems because all detector data from and to the HLT, as well as all data from the TPC and the TRD detectors to the DAQ system, have to be handled with this hardware platform. ALICE relies on a reliable new RORC hardware, firmware, and software for the successful data-taking in Run 2.

To summarize all of these individual aspects, the major goals of this work could be defined as the following:

- Development of a hardware platform to be used as the RORC for ALICE in Run 2. The hardware has to support the Run 2 upgrades of the detectors and processing algorithms while keeping compatibility with existing and reused infrastructure, hardware, and software from Run 1.
- Firmware and software developments to integrate the RORC into the HLT as well as its operation throughout Run 2. This includes the integration, maintenance, and extension of the hardware cluster finder.
- Evaluation of the hardware description at an algorithmic level using the dataflow approach. The hardware cluster finder serves as a reference for a high energy physics readout pre-processing algorithm with detailed performance numbers available from low-level implementations.

1.6 State of the Art, Available Solutions and Technologies

The evaluation of available solutions and technologies for a readout board for the ALICE DAQ and HLT systems in Run 2 was started already during Run 1 and conducted into the first long shutdown period (LS1). The definition of the target platform had to be taken early during LS1 with components available at that time to account for development, purchasing, production, testing, installation, and commissioning. As the overall system architecture of a PC-based detector readout has proven its suitability during Run 1 and a number of tools around this concept were strongly foreseen to be reused, there was no reason to question this for Run 2. The next RORC was, therefore, also planned as an FPGA-based plug-in board for server PCs with connectivity to the existing fiber installation.

1.6.1 Server Technologies

The concept of PC-based readout in which a server plug-in board transports the detector data from the serial optical link with optional data pre-processing into the main memory of the host machine remained untouched for ALICE in Run 2. For a maximum compatibility between the readout board and the host machine, the interface between both should be available in a broad range of server hardware. The de-facto standard interface for plug-in boards in PC hardware is PCI-Express. The parallel PCI or PCI-X standard used on previous RORC generations is hardly available anymore. A number of PC components from a wide range of applications are available as PCI-Express devices, most prominently GPUs and network adapters. PCI-Express is a point-to-point interface and is available in different variants. The number of lanes defines the number of differential link pairs that are aggregated to one logical link. The PCI-Express generation defines, among others, the signal rate per lane and the data encoding. The higher the PCI-Express generation number and the higher the number of lanes, the higher the overall bandwidth of the interface. Table 1.1 shows the currently available PCI-Express generations with bandwidth information and approximate dates of their announcement, the specification, and the first product releases.

A big advantage of the PCI-Express interface standard is the backward compatibility with previous generations of the interface. PCI-Express root ports and endpoints negotiate the set of common features and possible link rates at initialization time. As a result, a generation-1 device can be used in a generation-3 host and vice versa. This means for the RORC that the device can be used in any newer host machine as long as its maximum supported bandwidth is sufficient for its use case.

PCI-Express generation	Gen1	Gen2	Gen3	Gen4
Specification announcement	2002	2005	2007	2011
Specification release	2003	2006	2010	2017
First products	2003	2007	2012	
Encoding	8b10b	8b10b	128b130b	128b130b
Per lane bit rate (Gbps)	2.5	5.0	8.0	16.0
Per lane raw throughput (MB/s)	250	500	985	1969
x16 raw throughput (GB/s)	4.0	8.0	15.75	31.51
x16 payload throughput (GB/s)	3.5	7.0	13.5	27.0

Table 1.1: Evolution of the PCI-Express standard. Numbers from [PCI 17], payload throughput from own measurements and extrapolations.

PCI-Express as an interface between the RORC and the host machine handles the data transport between a PCI-Express endpoint implemented in the FPGA and a PCI-Express root port integrated into the host system architecture. Depending on the connection of this root port to the CPU sockets and the systems main memory modules, the PCI-Express connection may not be the only interface that defines the throughput between a RORC and its host. Main memory speed itself is usually not an issue for the expected data rates. However, depending on the host architecture, the chain of PCI-Express root port, IO hubs, CPU interconnects, and the memory controller could potentially limit the throughput to a value lower than the PCI-Express bandwidth.

1.6.2 PC-based Readout Schemes

A central aspect of the RORC integration is the data transport between the FPGA and the main memory of the host machine. There are typically two approaches to handle data transfer between a host and an FPGA-based PCI-Express endpoint. The first approach is to actively push data to the FPGA or pull data from the FPGA. This necessarily means that each transfer involves CPU activity. This approach hardly scales for high data throughput and is typically only used to read and write configuration and status registers on the device (“slow control”). The second approach is to instruct the FPGA to directly access the main memory of the host machine. This is called Direct Memory Access (DMA). DMA transfers from and to the FPGA do not require direct CPU involvement. The CPU is, in this case, typically used to control the dataflow and to provide memory location descriptors to the FPGA. This method is widely used for interactions with network devices, graphics cards, video capture devices, and others.

DMA data transfer requires that an endpoint device as the initiator of a transfer knows about host memory regions it is able to and allowed to access. This can be realized with descriptor lists containing bus addresses and lengths of main memory regions. The DMA transfer can then be implemented in different flavors. Typical package-based network applications allocate small contiguous buffers in the kernelspace and provide the descriptors to the network device. Upon packet reception, the kernel buffer is either copied or mapped to a userspace program. High-performance network applications try to avoid copying data between the userspace application and the device access or even between devices (“zero copy”). Large data blocks are not necessarily available as consecutive bus addresses due to the operating system’s memory allocation granularity, memory fragmentation, and virtual user address spaces. A DMA data transfer of physically fragmented memory regions can be realized with vectored I/O by building lists of memory descriptors (“scatter-gather list”) to be handled by the DMA master. This procedure is simplified if the host architecture uses an I/O Memory Management Unit (IOMMU). This device can map physically scattered main memory address regions into a single consecutive area in the device’s bus address space.

Implementing a DMA-capable PCI-Express endpoint in an FPGA requires to build the underlying bus address handling logic. Obviously, a DMA engine for a single consecutive memory area is easier to implement in firmware than one with support for vectored I/O. On one hand, the solutions developed for physically contiguous memory regions as used during Run 1 are hardly supported in recent operating systems anymore and are, therefore, hard to maintain. On the other hand, IOMMUs are not necessarily available in any server machine either. This means that there is no way around using a DMA engine with support for vectored I/O for Run 2. This implies an entirely new DMA engine in firmware, a new device driver, and a new device access library used by new low-level userspace tools.

On top of the low-level vectored I/O support, the data handling in the host machine can be implemented in different ways. One option is to allocate DMA memory blocks dynamically while running. This approach provides the highest flexibility in terms of the DMA buffer size. As long as there is free main memory in the host, new DMA memory blocks can be allocated. This requires to continuously push new descriptors to the RORC. However, this approach is hard to combine with the Publisher-Subscriber model of the HLT framework where subscribed components access a publisher's static output buffer at a dynamic offset. One way to overcome this limitation is to allocate a fixed-size DMA buffer once and add a memory management layer that keeps track of which regions are available for DMA transfers. By using this buffer as a ring buffer, both the software-side memory management and the firmware implementation become much simpler and can be reduced to single read and write pointers with a bit of comparison logic. A big advantage of this approach is that the descriptors for the full ring buffer region can be loaded into the FPGA at once. This reduces the interaction between the CPU and the FPGA to a minimum. A limitation of the ring buffer approach is that a single data block, which remains in the ring buffer significantly longer than others, may stall the readout process. This can be compensated by adjusting the ring buffer size to the maximum expected processing time of data blocks with respect to the input data rate.

The data transfer between the FPGA and the HLT data transport framework needs some kind of handshake. Source components in the data transport framework must not read from DMA memory regions before they are filled by the RORC. Sink components must not free or reuse the DMA memory before the data is fully fetched from the RORC. The notification of userspace programs about completed DMA transactions can be realized with interrupts or with polling. PCI-Express endpoints can send interrupt signals to the host system. The operating system pauses its current task, jumps to an interrupt handler to process the interrupt, and resumes its original task. The advantage of interrupts is that the CPU is notified of new data only if it is actually available. Especially in low-rate scenarios, this consumes only little additional CPU time. However, if the rate of interrupts is high, the data processing tasks are frequently interrupted and each interrupt requires a context switch to kernelmode. This can severely affect the data processing performance. For this reason, network device drivers typically limit their interrupt rate in high-load scenarios by disabling interrupts and switching to a polling mode. This load-dependent handling method is part of NAPI [The 16]. Polling on its own does not use interrupts but requires a user application to periodically check for new data. This means that a certain amount of CPU time is always spent to check for new data even if no new data are available. Polling provides a better performance in high-load scenarios and comes with considerable savings in terms of firmware and software complexity.

In a system like the HLT where the processing performance in a high-load scenario is crucial, polling was considered the appropriate solution already for Run 1. A slightly higher CPU utilization in low-load scenarios is acceptable and does not affect the processing power. A dynamic sleep mechanism avoids excessive polling for low event rates. The additional savings at the CPU time in low-load situations by implementing interrupts in combination with NAPI-like methods are in no relation to the increased complexity as well as the development and maintenance effort for software and firmware. This fact remains true for the DMA handshaking in Run 2.

1.6.3 Fiber and Optical Technologies

The connection between the front-end electronics near the detectors and the computing clusters of the DAQ and HLT systems is realized with optical links. More than 500 serial optical links are installed between the detectors in the underground experiment cavern and the DAQ system on the surface. The total fiber lengths vary between around 80 m and 130 m per link. During Run 1, all fibers used a link rate of 2.125 Gbps. Several patch panels are installed along the way from the detectors up to the counting rooms of the DAQ and HLT systems. They use the E2000 connection standard. The Run 1 RORCs and the DDL modules in the detectors used regular Simple Formfactor Pluggable (SFP) transceiver modules. These SFP modules come with an LC optical connection socket. The interconnection between the patch panels and the SFP transceiver modules was done with E2000-to-LC patch cables.

The general fiber installation as used in ALICE during Run 1 was not planned to be changed for Run 2. This means that at least the connections between the detectors in the experiment cavern and the counting rooms should remain the same. The uppermost part, consisting of the last patch panels in the counting rooms and the connection to the next RORC, was subject to evaluation.

Fiber Type	2 Gbps FC	4 Gbps FC	8 Gbps FC
OM2, 50/125 µm	300 m	150 m	50 m
OM3, 50/125 µm	500 m	380 m	150 m
OM4, 50/125 µm	>500 m	400 m	190 m

Table 1.2: Maximum channel lengths for different multi-mode fibers and link rates at 850 nm. Numbers extracted from [TE 14].

Optical fibers are divided into different categories based on their mode and bandwidth. This bandwidth in return limits the maximum reach of the signals at a given link rate. The existing fiber installation in ALICE uses OM2 multi-mode fibers. Increasing the link rate of these fibers for some detectors in Run 2 decreases the maximum supported fiber length. Table 1.2 shows an overview of typical fiber length limits for multi-mode fibers at different link rates for FibreChannel (FC). The DDL protocol uses the same encoding and comma characters as FibreChannel so these numbers can be treated equally for DDL. The efforts to increase the readout link rate to around 4–5 Gbps with the given fiber lengths approached the upper limit for OM2 fibers. Detailed calculations and measurements from the CERN EN-EL team confirmed the suitability of the existing installation for link rates up to 5 Gbps.

The Run 1 setup used SFP transceiver modules on the front-end electronics as well as on the RORCs in the DAQ and HLT systems. These modules are also available for higher link rates. A drawback of SFPs especially for the next RORC is their size and, therefore, the maximum link density per board. The usable area of the IO-brackets of PCI-Express cards can hold at most four SFPs per PCI-Express slot. Supporting more than four serial optical links on one RORC with SFPs requires occupying more than one PCI-Express slot. This would in return increases the size of the board and limit the compatibility with the host machines.

However, a number of parallel optical technologies are available that provide an increased number of links at a smaller footprint. Quad-SFPs (QSFPs) contain four bi-directional transceivers in a module only slightly larger than an SFP module. They are available for different link rates and from multiple vendors. Up to three QSFPs fit into the IO-bracket of a PCI-Express slot. A photo of SFP and QSFP transceiver modules is shown in Figure 1.17.

Dedicated transmitter and receiver components provide four to 12 uni-directional links per component. These components are often vendor-specific solutions. Mini- and MicroPod receiver and transmitter components with bigger market shares and even higher link densities are recent alternatives. They use similar concepts but became available only after the development of the next RORC was done. The dedicated receiver and transmitter components require a separate passive

Figure 1.17: Photo of two QSFP modules with four bi-directional links each and one SFP module on the right with one bi-directional link.

IO-bracket to connect the fibers from the outside to the components on the board. CXP is a connection standard for 12 bidirectional links using pluggable transceiver modules. It is partly used for InfiniBand interconnects, but mostly for electrical or active optical interconnects. Active optical solutions cannot be used in this case because they cannot be connected to the existing fiber installation.

Connecting parallel optical fiber solutions to the existing fiber installation in ALICE is possible in different ways. The interconnection of different connection standards simply comes down to having a cable or adapter assembly with different connectors or sockets on both ends. For the RORC this is possible with three methods:

- Break-out fibers from the parallel optical RORC transceivers to the E2000 simplex sockets in the existing patch panels. This enables the reuse of the existing patch panels. Break-out fibers are more expensive than cables with the same connector on both ends.

- Replacement of the patch panels to provide parallel optical connectivity already at the patch panel. This would simplify the connection between the RORCs and the patch panels. The savings on the cheaper fiber cables had to be weighted against the costs of the new patch panels.

- Passive switch boxes with parallel optical connectivity to the RORC on one side and LC connectors on the other side. The LC side would be connected to the existing patch panels. This solution enables the reuse of the patch panels and the LC-to-E2000 patch cords. However, this comes with yet another connector stage, increasing complexity and affecting the optical signal integrity. It also requires additional rack space.

The third option brings several disadvantages and was, therefore, not further considered. The remaining two solutions are technically equally suited. A decision can be made based on availability and price.

1.6.4 On-Board Storage

Different technologies for on-board storage are available with the typical trade-off between price, bandwidth, and capacity. FPGAs already provide on-chip storage in the order of a few Megabytes that can be used for fast memories or FIFO queues. External SRAM components can provide tens of Megabytes of storage with a comparable bandwidth. However, they need a lot of IO pins. Larger memories in the order of hundreds of Megabytes up to a couple of Gigabytes are available with external DRAM. DRAM can either directly be soldered onto the board in the form of dedicated PCB components like on the H-RORC or connected via pluggable modules. A typical DRAM solution for FPGA boards is DDR3-SDRAM at up to 1066 Mbps, either as components or as pluggable modules using SO-DIMM sockets. The interface speed is limited by the maximum clock frequency of the memory controller implemented in the FPGA. The number of DRAM components or sockets that can be connected to one FPGA is dominated by the number and speed of suitable FPGA IO pins. Flash memory chips can provide non-volatile storage commonly used for FPGA configuration files. Larger and cheap non-volatile storage can be connected to FPGAs via memory card sockets such as Secure Digital (SD) cards.

1.6.5 FPGAs and Commercially Available Solutions

At the time the evaluation for a new RORC hardware was started the most common FPGA devices were the Xilinx Virtex-5 and the Altera Stratix-IV. Both devices already provide significantly more FPGA resources than the FPGAs used on the Run 1 RORCs and were available on the market for quite some time. A new generation of FPGAs available as engineering samples at the beginning of the evaluation was the Virtex-6 series from Xilinx. These FPGAs can implement PCI-Express endpoints with up to eight lanes at 5.0 Gbps (PCI-Express generation-2) and serial links at up to 6.6 Gbps or at 10.3125 Gbps [Xil 10]. Mid-range sized devices come with around 20 serial transceiver blocks, the flagship models with over 70. The successor of the Stratix-IV, the Altera Stratix-V, was announced but not yet available on the market at that time.

Both Xilinx and Intel/Altera provide evaluation boards of their FPGAs with reference implementations for a broad range of interfaces and FPGA features. These boards usually cover a very wide spectrum of applications and are mostly meant for prototyping applications. If used for only one specific application, the boards usually contain a lot of functionality that is not required for this use case. The boards typically provide a number of general purpose inputs and outputs via mezzanine connectors. This makes them usable for many applications using either vendor-provided or custom pluggable add-on boards that come with the required interfaces. However, this also means that these boards are not necessarily optimized for the operation as a server plug-in board in a dense computing cluster.

The Xilinx ML605 Evaluation Kit[1] is an example for a Virtex-6 evaluation board that comes with a PCI-Express interface with up to four lanes at 5.0 Gbps or eight lanes at 2.5 Gbps. It includes one SFP socket and one DDR3 SO-DIMM socket. An FPGA Mezzanine Card (FMC) connector provides various additional input and output options. The board can be installed into a server PC but exceeds the maximum size of a full-width standard-height PCI-Express card. Connecting additional daughter boards via FMC further increases the space requirement of the setup.

More specialized FPGA boards are developed from third party companies like HitechGlobal[2], Nallatech[3], Bittware[4], and others. These hardware platforms are typically not meant as general purpose devices but rather to provide the maximum performance for a subset of features. ASIC prototyping solutions provide standalone multi-FPGA boards with generic IO and a focus on

[1] https://www.xilinx.com/ml605
[2] http://www.hitechglobal.com/
[3] http://www.nallatech.com
[4] http://www.bittware.com

large amounts of logic resources. Networking solutions come with high-speed serial links. Co-processor boards provide the maximum interface bandwidth to the host machine and keep the form factor of the board within the limits for server plug-in boards.

One example of a PCI-Express-based FPGA board is the HitechGlobal HTG-V6-PCIe. It provides an eight-lane generation-2 PCI-Express interface and up to 8 GB of DDR3 memory via one SO-DIMM socket on the back side of the board. Two SFP and two Ethernet sockets are available on the IO bracket. More serial optical links can be added with FMC add-on boards. However, these extensions make the setup exceed the PCI-Express form factor limits. FMC add-on boards are available with up to eight serial optical links using two QSFP sockets or with 12 links using one CXP socket. A low-profile PCI-Express board with an Altera Stratix-V FPGA and two QSFP sockets already on the board became available only during the development of the next RORC.

A general problem with commercial FPGA boards in the context of physics detector readout is the gap between the required bandwidth for the interface to the host machine and the bandwidth and number of the individual serial optical links. Commercial boards typically match the maximum bandwidth of the optical links with their PCI-Express bandwidth. This results in a smaller number of high-speed links. However, the serial link rates used in large physics detectors are often several years behind the state-of-the-art serial link technologies available with new FPGA boards. This is due to the long development times for detectors with lots of custom hardware and large experiments as a whole. The server machines that host the readout boards are typically much newer than the detector front-end electronics and can be replaced much more easily. A readout board as for ALICE in Run 2 requires numerous comparably slow serial links in combination with a state-of-the-art interface to the host machine. A ready-to-use commercial hardware platform to implement the RORC functionality for ALICE Run 2 was not available. Therefore, the next RORC had to be at least a modification or extension of a commercial product, or even a custom development.

FPGA board vendors typically provide reference designs with source code for their hardware. This usually covers basic access to on-board storage, simple PCI-Express input/output or a reference networking implementation. On top of this, Intellectual Property (IP) core generators like Xilinx' *coregen* or Altera's *qsys* create IP cores based on custom interface or hardware specifications. More sophisticated features like Direct Memory Access via PCI-Express are available from third party companies as commercial IP cores. A well-known supplier of PCI and PCI-Express DMA implementations for FPGAs is PLDA[5]. A PLDA DMA IP core is used in the D-RORCs since Run 1. Commercial DMA implementations are also available for newer FPGA architectures. The DMA engine of the H-RORC is a custom development. Custom protocols like the DDL or the link to the TPC Busy Box are not available commercially, but need to be developed or ported from previous implementations. The decision on whether to use a custom or a commercial solution depends on the suitability of the commercial core, as well as its performance and price, with respect to the effort and gain of a custom development.

1.6.6 Hardware Developments of other Experiments

The first long LHC shutdown period was used by all LHC experiments for maintenance and upgrade works. However, the different experiments all have different detectors, architectures, readout hardware, and software implementations. The LHCb experiment implements Ethernet-based readout with crate-based hardware [Hae+ 06]. Developments in this field cannot be used for the PC-based readout in ALICE. The CMS experiment uses a copper-based readout chain with FPGA boards in crate systems that transfer the detector data into a commercial network protocol. The CMS FPGA readout boards were upgraded during LS1 from Myrinet to 10 Gbps Ethernet [Bau+ 13]. Due to significantly different interfaces, these developments can also not be used for ALICE.

[5] http://www.plda.com

The ATLAS experiment has Run 1 readout hardware in place that is comparable with the ALICE hardware: the ROBIN card [Kug 09]. This board is similar to the ALICE D-RORCs and H-RORCs. It has a PCI-X interface to the host machine, handles three serial optical links at 2.0 Gbps, and contains on-board DRAM storage. The ATLAS ROBIN has similar problems as the ALICE RORCs with the obsolescence of the PCI-X interface towards the host machine. Additionally, ATLAS is expecting significantly higher load at the interface to the host machine for Run 2. At the time a hardware platform for ALICE was evaluated and decided there was no hardware decision published by the ATLAS team. The ATLAS experiment is finally using the hardware platform developed as part of this work. This is described in more detail in Section 2.4.3.

The Compressed Baryonic Matter (CBM) experiment [Abl+ 17] at the Facility for Antiproton and Ion Research[6] (FAIR) with its First Level Event Selector (FLES) [dC+ 11] plans a readout architecture which is similar to the ALICE architecture. The experiment was, however, in a very early stage around the time of the ALICE hardware decisions. CBM could use existing custom boards and commercial evaluation boards for small-scale readout prototypes and beam tests. Achieving the final link density and form factor was not crucially important at this stage. Final CBM readout hardware decisions were not expected within LS1. For these reasons, joining forces with other physics experiments with similar use cases for the development of the next ALICE RORC was not possible at that time.

1.6.7 Programmable Hardware Description Methods

Hardware description languages (HDLs) are textual representations of the structure and the behavior of integrated circuits over time. In contrast to common imperative software description languages, HDLs have to express both the concurrency and the spatial parallelism of hardware. The functionality of hardware can be described from different perspectives and at different abstraction layers. A commonly used distinction of these perspectives is the Y-chart presented by Daniel Gajski and Robert Kuhn in 1983 [Gaj+ 83] with independent behavioral, structural and geometric representations of the same hardware. Each perspective provides its own layers of abstraction, from circuit behavior to system behavior, from logic gate to system building blocks, from polygon or cell to multi-chip floor plans. Corresponding abstractions from each perspective build a common abstraction layer. With FPGAs as arrays of configurable logic blocks of different types, the lowest description layers are already hidden from the user by the FPGA vendor. FPGA-based hardware description starts at the register transfer level (RTL). The FPGA primitives of flip-flops, look-up tables, RAMs, DSPs etc. form the structural perspective. Register transfer logic dominates the behavioral branch. The structural branch describes interconneted RTL blocks. Vendor-provided tools are responsible to map the primitives onto the available logic resources and translate the mapped and routed design into a configuration file that can be loaded into the FPGA. This step, in combination with user-provided floor planning constraints, covers the geometric representation. The internal structure of FPGAs and the encoding of the configuration file is typically not available to users, which in return limits the hardware description and implementation options to the tools provided by the FPGA vendors. FPGA hardware description is currently mostly done at the RT level. However, there are several other approaches especially towards higher abstracting levels for hardware description.

1.6.7.1 VHDL and Verilog

The most common hardware description languages (HDL) in use are VHDL and Verilog. VHDL is influenced by Ada and Pascal and is slightly more prominent in Europe. Verilog has similarities to C and has its user base more towards the US. An example of a register instance with reset and clock enable port described in VHDL and Verilog is shown in Figure 1.18. Both languages were initially designed as hardware documentation and simulation languages and were only later used

[6] https://gsi.de/fair

```vhdl
-- VHDL
process (CK) begin
  if rising_edge(CK) then
    if RST = '1' then
      Q <= '0';
    elsif EN = '1' then
      Q <= D;
    end if;
  end if;
end process;
```

```verilog
// Verilog
always @(posedge CK)
  if (RST = 1)
    Q <= 1'b0;
  else if (EN = 1)
    Q <= D;
```

Figure 1.18: Register with reset and clock enable described in VHDL (left) and Verilog (right).

to create hardware from this description. This led to the fact that only a subset of the language constructs can be synthesized into hardware. VHDL and Verilog have a different syntax but share similar concepts and abstractions for the description of hardware parallelism. A hardware developer that knows one can easily also use the other. Both languages are targeted towards describing hardware at the register transfer level. They provide a cycle-accurate behavioral description of synchronous or combinatorial logic elements. HDL designs typically consist of a hierarchy of modular entities that communicate with each other via interconnected input and output ports. This forms the structural perspective. The cycle-accurate description also means that the user controls exactly which logic operations are performed in which clock cycle. This gives ultimate control for high-performance processing blocks or low-level interfaces. VHDL comes with a few higher-level elements than Verilog, like records or custom types, but is also often criticized for being a rather verbose language. The majority of FPGA designs are currently described in VHDL or Verilog or a mixture thereof. The FPGA board vendors provide their reference designs and IP cores in VHDL or Verilog. Both languages can directly be used with the FPGA vendor tools to build configuration files for FPGAs. VHDL and Verilog are additionally well supported by various HDL simulators to verify the hardware description before implementing it in an FPGA.

Designing data processing algorithms is typically done at an algorithmic level, without explicit mapping of operations into clock cycles or onto device primitives. Mapping these algorithms onto hardware by describing them at the register transfer level with VHDL or Verilog is possible but easily becomes very complex. The developer has to manually partition the algorithm onto the register transfer level. Simple logic and arithmetic operations up to integer adders can still be described with a few lines of VHDL or Verilog. Any operation that is more complex than this is typically handled by instantiating specialized vendor-provided IP cores. Depending on the core, it may have a latency of several clock cycles, or may not accept new data on every clock cycle. The user has to make sure to match latency differences and to implement flow control between interconnected cores to guarantee valid results. The usage of pre-generated IP cores makes it hard to keep an implementation generic with respect to changes in the algorithm. A small change in the algorithm can require rewrites of several parts of the hardware implementation. While the native VHDL and Verilog description is mostly independent of the target device, the IP cores are specifically built for one FPGA model. Transferring the same algorithm to another FPGA model requires to re-generate all instantiated IP cores and to manually handle possibly changing latency characteristics. While implementing complex algorithms with VHDL or Verilog is technically possible and proven multiple times in the past, it comes with significant efforts for both the initial development, the verification, the maintenance of the implementation, and the modification of the algorithm. With FPGAs growing exponentially in size with each new generation, hardware description at the RTL HDL level can hardly keep up with technology. It becomes harder and harder for developers to efficiently exploit the possibilities of new FPGAs with VHDL or Verilog.

1.6.7.2 SystemVerilog and SystemC

```
// SystemVerilog
always_ff @(posedge CK)
  if (RST = 1)
    Q <= 1'b0;
  else if (EN = 1)
    Q <= D;
```

```
// SystemC
void p1() {
  if (RST.read()) { Q = 0; }
  else if (CE.read()) { Q = D;  }
}

SC_CTOR(dff) {
  SC_METHOD(p1);
    sensitive_pos << CK;
}
```

Figure 1.19: Register with reset and clock enable described in SystemVerilog (left) and SystemC (right).

SystemVerilog is a *"unified hardware design, specification and verification language"* [IEE 13 I]. It is an extension of Verilog and brings object-oriented programming concepts into the verification environment. SystemVerilog extends Verilog by providing, among others, new data types like multidimensional arrays or enumerated types and interfaces to group or compact the module connectivity description. For verification, it brings classes, assertions, and integrated code coverage measurements. These features make SystemVerilog an interesting alternative to VHDL or Verilog, only its adoption in vendor tools was slow in the past. Figure 1.19 on the left shows a register instantiation in SystemVerilog. This code is almost identical to the Verilog example shown above, but this example does not use any of the additional features provided by SystemVerilog. Recently, the XILINX Vivado and the INTEL/ALTERA toolchains support subsets of SystemVerilog also for hardware synthesis. The latter also generates SystemVerilog code when compiling OpenCL code for FPGAs. SystemVerilog is, like Verilog, limited to the register transfer level for hardware synthesis.

SystemC is a C++ class library to describe hybrid hardware and software systems. It has similarities to VHDL and Verilog but brings much more flexibility due to the underlying C++ language. Even though it is a C++ class library, it is often viewed as a language on its own. It can describe hierarchical models with structural interconnections, the scheduling and synchronization between them, and the passing of time [IEE 12]. Part of the SystemC standard is Transaction-Level Modeling (TLM), which models communication channels between digital systems at a high level. The SystemC description can be compiled to executable binaries that provide fast simulation environments. SystemC defines a subset of its language to be synthesizable for hardware [Acc 16]. This enables the language to be used to actually implement SystemC descriptions in FPGAs. An example of a register equivalent to the VHDL/Verilog description above is shown in Figure 1.19 on the right. Even though SystemC provides high-level system descriptions, its synthesizable subset operates still at the register transfer level. This means that SystemC can mitigate the limitations of VHDL and Verilog only towards usability, simulation, and verification but not towards a synthesizable hardware description at an algorithmic level.

1.6.7.3 High-Level Synthesis

High-Level Synthesis (HLS) is the *"process of automatic generation of hardware circuit from behavioral descriptions"* [Cou$^+$ 08]. The behavioral description in this context refers to the electronic system. This is beyond the capabilities of the descriptions methods for FPGAs shown above. The synthesizable subset of the HDLs described above provide behavioral description options only for combinatorial elements, not systems. The input to HLS tools is typically an *"algorithmic description of design behavior that does not contain structural information "* [Gaj$^+$ 94]. The rapidly growing capabilities of hardware devices require higher-level description approaches. Hardware description at the algorithmic level does not contain explicit timing information of

the individual processing steps. It specifies the order of operations but leaves their exact timing implementation to the HLS tools. This makes the hardware description much more compact, more flexible, faster to develop and easier to maintain compared to the traditional development at the RT level. An HLS compiler has no problem with generating a hardware implementation with thousands of pipeline stages. However, a human developer at the RT level will likely write much more lines of code, will require significantly more time to achieve the same result, and will possibly make mistakes on the way. As a result, HLS makes it easier for newcomers to get started with hardware development while easing the cumbersome and error-prone manual steps in RTL hardware design also for experienced designers.

This step from RTL to HLS hardware description is similar to the transition from assembly languages to higher level languages like C, C++ or Java in the software development world. Even though the designer has to give up control over every single instruction passed to the CPU, the gain in productivity easily justifies the higher level approach. For high-performance components or low-level applications, assembly languages can still be used directly integrated or embedded in libraries for use in higher-level description frameworks.

First efforts in researching and developing HLS approaches for hardware development go back to the 1970s and 1980s with a focus on improving the ASIC design process. However, up to the early 2000s, these HLS developments did not succeed as commercial products [Mar$^+$ 09]. Around this time, also FPGA hardware development tools started to adopt HLS approaches. Up to now, a number of tools are available as commercial or academic products to enable HLS hardware development for FPGAs. Numerous studies compared these different approaches and confirmed the benefits of the HLS concept [Geo$^+$ 11, Mee$^+$ 12, AA$^+$ 14, Nan$^+$ 16]. Almost all HLS tool variants share a common set of approaches to translate a behavioral description into hardware. These steps are only briefly mentioned here but described in more detail for example in [Cou$^+$ 09] and [Car$^+$ 16]. The compilers create a formal model from the specification. This model contains data and control dependencies between the different operations. At this stage, a number of optimizations are already applied. This includes, among others, loop optimization and loop unrolling, constant extraction, bit width analysis, and removal of unreachable code. The HLS tools then take care of scheduling, allocation, and binding. Scheduling is the process of distributing the processing steps across clock cycles and to make sure that parallel operations are latency-wise matched. Allocation selects the hardware blocks to implement the given operations, memories, or interconnects in the FPGA. Binding maps the scheduled operations onto the allocated hardware resources. It determines the number of units required as well as the resource sharing of operations with common hardware blocks. These steps can already include architectural properties of the target device to decide for example between registers or block RAM, fabric math or DSP blocks, and so on.

Pipelining is an important aspect to maximize the throughput of the design. The Initiation Interval (II) is a commonly used metric to describe how frequently a logic block can be provided with new input data. The optimal value for the maximum throughput is $\mathrm{II} = 1$, meaning that all data dependencies are resolved and the design is fully pipelined. It can accept new data on each clock cycle. The HLS compilers typically come with a device-specific library of RTL or netlist building blocks that are used to map the extracted control and dataflow graph to the target device architecture. HLS tools often generate RTL hardware descriptions in one of the common HDLs that are then processed by the existing vendor tools for RT level hardware design. This step keeps the option to mix HLS with existing RTL HDL code.

A big portion of HLS tools follows a C-to-hardware approach in different variations by both restricting the C language to a synthesizable subset and by extending it with statements to control the hardware implementation. The language has to be extended with constructs to describe concurrency, timing, arbitrary data types, and communication between elements. A common restriction of the HLS tools is that language features like recursion, dynamic memory allocation, system calls, and alike are not supported in hardware for obvious reasons. The selection of a well-known language like C as HLS language makes it easier for a broad range of developers to get started. On one hand, a common language for mixed hardware/software systems promises a more

dynamic partitioning of processing steps. On the other hand, a sequential language like C also makes it easy to design inefficient hardware. As soon as the C-like hardware description contains data or control dependencies that are not resolved by the user or cannot be resolved by the tools, the compilers have no choice but implementing the described logic with a state machine or von Neumann-like structures. In this case, the FPGA implementation will likely lose performance comparisons against CPU or GPU implementations, simply due to the significant differences in their clock frequencies. So even though HLS can greatly ease the firmware development efforts, it still requires programmers that are well aware of the underlying architecture and how HLS works. An experienced software designer without explicit hardware understanding will not be able to produce efficient hardware implementations with HLS. These concerns are also elaborated in more detail in [Edw 06].

The most prominent HLS solutions for FPGAs at the time of this writing are Vivado HLS (Xilinx) and OpenCL (Xilinx and Altera/Intel). Besides the FPGA vendor provided tools, there are a couple of well known commercial and academic solutions. A few of these HLS tools are briefly described here.

Vivado HLS[7] originates from AutoESL's AutoPilot. It accepts C, C++, and SystemC as HLS input languages and provides options to describe hardware at different abstraction levels as untimed, partially timed or fully timed [Zha$^+$ 08] models. The generated RTL code can be exported to be used as IP core in different projects. Vivado HLS brings a number of libraries to implement streaming interfaces, mathematical operations or video and signal processing tasks via dedicated C functions. Directives instruct the compiler on how to implement certain functionality. This can be done via pragmas directly in the code or via separate directive script files. Apart from unrolling loops, pipelining loops, and inlining functions, Vivado HLS directives can control the latency of functions or influence the implementation of memories. In combination with its AXI bus interface, HLS-generated hardware can be directly connected to CPUs, either as soft-cores in the FPGA or as dedicated ARM processors like on Xilinx Zync[8] platforms. Vivado HLS is clearly targeted towards Xilinx FPGAs with support for devices from the 7-series or newer. The Vivado HLS design flow, as well as an overview of its directives, is described in detail in [Sch$^+$ 16] and in [Xil 17].

OpenCL was originally designed by Apple Inc. and is now developed by Khronos Group[9] [Khr 09]. It is a framework for writing parallel software that can be run on numerous devices like CPUs, GPUs, DSPs, and recently also FPGAs. OpenCL is well known for its GPU adoption in HPC applications. OpenCL on FPGAs is in the meantime supported by both Intel/Altera and Xilinx. The latter embeds its OpenCL implementation into a package called SDAccel[10] in combination with C and C++ hardware synthesis tools. OpenCL is based on a heterogeneous architecture where a host processor is connected to one or more accelerator devices. Processing units running in the accelerator devices are called "kernels". The host processor orchestrates the dataflow between the kernels and the accelerators. The kernels are described with the C programming language. Acceleration is achieved by data parallelism within kernels as well as task parallelism by instantiating multiple instances of kernels for different parts of an input dataset. Data structures can be local to kernels, accessible by a group of kernels, or globally accessible for all kernels and the host process. Data exchange between kernels is typically handled via group or global memory. For the integration in FPGAs, the OpenCL implementation is extended to address FPGA-specific features [Sin$^+$ 16]. One example is the "channel" concept as interconnect between kernels which is implemented with FIFOs. This can greatly improve the performance of OpenCL kernels chains in an FPGA implementation. Special attributes and pragmas are available to control the implementation of the OpenCL kernels in the FPGA. The tight coupling between the host processor and the accelerator device in the OpenCL programming model requires a connection between the host and the device. The FPGA implementation is, therefore, clearly targeted towards SoCs or PCI-Express FPGA devices. The OpenCL FPGA framework

[7] https://www.xilinx.com/hls
[8] https://www.xilinx.com/soc
[9] https://www.khronos.org
[10] https://www.xilinx.com/sdaccell

comes with a generic implementation of a host API and a device driver. From firmware-side, it provides a PCI-Express endpoint, a memory controller, and an OpenCL infrastructure for kernels. The interfaces of an accelerator device towards the host, to the global memory or to network interconnects are defined with a board support package (BSP). This package is typically provided by the accelerator device vendor. The OpenCL FPGA compiler creates RTL HDL code to be used with the regular vendor toolchains. The kernels can be emulated on a CPU together with the host application. This provides a fast simulation environment for the hardware implementation. OpenCL is currently an actively used candidate in the HLS community and is also being evaluated for high energy physics applications [Sri 15, Fär+ 17]. An implementation of the HLT cluster finder with OpenCL was realized by Tom Zügel as a master's thesis [Züg 17].

LegUp[11] [Can+ 11] is developed at the University of Toronto as an open source HLS tool. It is based on C and is using the LLVM compiler infrastructure. It can generate RTL code for pure hardware platforms or hybrid CPU/FPGA systems. While the generated Verilog RTL code is vendor agnostic, the better performance is expected for Altera FPGAs. LegUp is described in more detail in [Can+ 16].

ROCCC[12], the Riverside Optimizing Compiler for Configurable Computing [Hal+ 10], is an open source HLS tool developed at the University of California, Riverside. It translates a C subset to VHDL and is device independent. ROCCC sees itself as a hardware accelerator and not as a general-purpose HLS framework. It distinguishes between modules and systems with different supported subsets of the C language. Modules are computational functions implemented in hardware. Systems control the dataflow and can instantiate modules. ROCCC is described in more detail in [Naj+ 16].

Bluespec System Verilog (BSV) is a commercial HLS toolset from Bluespec Inc.[13] for ASICs and FPGAs. It is based on the SystemVerilog language syntax. BSV uses a different approach for higher-level hardware description compared to the other HLS tools mentioned here. It does not try to move hardware description closer to software concepts or languages, but builds on the hardware concepts known from VHDL or Verilog. Where other HLS tools try to hide hardware complexity like clock cycles and concurrency, BSV explicitly exposes this to the user. Unlike other HLS tools, BSV is meant for RTL design experts. The hardware behavior is described with guarded atomic actions that are implemented as concurrent cooperating state machines [Nik 04]. The behavior of modules and interfaces is defined with `<guard, action>` pairs. The the guard is a boolean function that triggers the action. More details on BSV and code examples are available for example in [Nik 08] and [AA+ 16].

1.6.8 Dataflow Description of Programmable Hardware

Parallelism and concurrent computation are a major path to increase the processing power of electronic systems. Traditional von Neumann architectures achieve parallelism by deep pipelining of instructions, data parallelism like vector instructions, as well as multi-threaded or multi-core execution. Scaling these architectures to higher parallelism reveals two fundamental problems: latency and synchronization. CPU instructions, especially memory access and communication calls, have a latency that depends on the current system state. This latency scales with the number of parallel computing entities. With one instruction depending on the result of its predecessor, this limits the system performance. In order to mitigate the latency problem, hardware level synchronization methods like caches, prefetching, out-of-order execution, branch prediction, and others are applied. The effort to implement such synchronization solutions scales also with the amount of parallelism.

A different approach towards highly parallel systems is dataflow computing [Sil+ 99, Ian 90]. Dataflow computing, in general, goes back to the 1970s as an alternative concept to the extension of the von Neumann computing paradigm to multiprocessor architectures. Dataflow instructions

[11] http://legup.eecg.utoronto.ca/
[12] http://roccc.cs.ucr.edu/
[13] http://bluespec.com/

are executed as soon as all of its operands are available. There is no list of sequential instructions or a program counter. As a consequence, multiple enabled instructions do not interfere with each other. This concept provides explicit synchronization across all parallel computing entities. In order to exploit this architecture, the computing task has to be described as a dataflow graph (DFG) [Den 74, Kav+ 86]. A dataflow graph is a directed graph that describes data movement along its path. The graph consists of two types of nodes: links and actors. Links receive data from a single actor or transmit data to one or more actors. Actors describe data operations. The data packets traveling along links are called tokens. An actor is enabled ("fired") as soon as all of its input links contain a token.

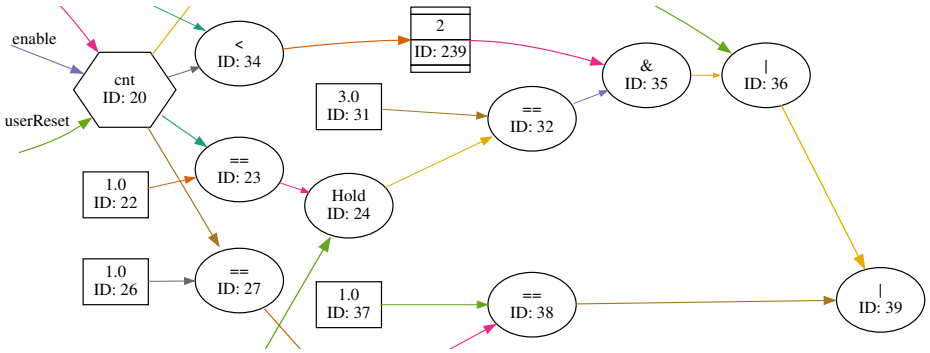

Figure 1.20: Visualized output from the data flow framework described in Chapter 5 showing parts of the dataflow graph from one processing kernel.

A graphical representation of a dataflow graph is shown in Figure 1.20. In order to describe a dataflow graph, a language with a single-assignment rule is required [Cyt+ 91]. This rule forbids the reassignment of variables once their value has been assigned. It ensures that the algorithm can be represented by a dataflow graph.

Dataflow architectures are traditionally classified as either static or dynamic. The static architecture fires a node only if all input tokens are available and no token exists on the output link. The dynamic architecture implements the links as buffers that can contain multiple tokens. Various flavors and mixes of these categories, as well as hybrid dataflow / von Neumann architectures, were developed in the past [Hur+ 07]. However, no commercially successful system could evolve. A limited number of functional units, memory bandwidth limitations and scheduling of available tokens across nodes restricted the practical performance of dataflow systems [Sil+ 99].

In the context of FPGAs, however, the dataflow approach is a very promising candidate for high-level synthesis. Most of the HLS tools described in the previous paragraph actually already build some form of dataflow graph during their synthesis step among other things. The concept of a static dataflow graph with actors and links can directly be mapped onto the FPGA architecture. All operations are implemented in a chain as a parallel data pipeline and are executing concurrently. The dataflow compiler ensures that the data dependency is resolved and the latency between different processing steps is matched.

The main vendor for a dataflow-based HLS framework for FPGAs at the time of this writing is Maxeler Technologies[14]. Maxeler provides a number of commercial dataflow computing platforms as a combination of a host system with one or more dataflow engines (DFE). The DFEs are custom FPGA boards with comparably large on-board DRAM buffers and an interface to the host system via PCI-Express or InfiniBand. DFEs can have their own network connection as data input or output interface and can be directly connected to other DFEs. A host software starts and orchestrates the dataflow to and from the DFEs similar to the OpenCL approach. A high-level host software API hides all low-level PCI-Express, InfiniBand or DMA memory operations to communicate with the DFE from the user. The behavior of the DFEs is described with a dataflow language called "MaxJ", an implementation of the OpenSPL[15] programming paradigm using a

[14] https://www.maxeler.com
[15] http://www.openspl.org

Java-like syntax [Max 13]. SPL stands for Spatial Programming Language and lets designers describe the hardware behavior in space instead of time [Bec+ 16]. Single processing entities are called kernels. Multiple kernels can be interconnected within and across DFEs. The topology of data streams between the host, the on-board RAM, the network interfaces and the kernels is defined with a manager instance. The spatial representation of the dataflow graph is optimized and translated into RTL and netlist blocks by a compiler with a library of device specific building blocks. The generated code is then processed by the FPGA vendor toolchain to create an FPGA configuration file. A programming IDE based on Eclipse[16] in combination with a software-based DFE emulator provide a rapid development and simulation environment.

Even though other HLS tools may be able to achieve comparable results with an appropriate hardware description, a key feature of the MAXELER approach is its strict compiler. This compiler enforces a description of the algorithm that is fully compatible with the dataflow approach. Where other solutions accept circular data or control dependencies and implement this description with low-performing state machines, the MAXELER approach requires the user to adjust the algorithm and its implementation. This dataflow approach is not at all meant to magically translate existing sequential code from slow software to fast hardware. It rather forces the user to rethink and reformulate the algorithm in a way that can be implemented as a high-performance hardware solution. More details about this concept as well as code examples and performance results are available in [Mil+ 15].

1.7 Approach

The goals of this work are defined in Section 1.5 as the development of the hardware platform for the Read-Out Receiver Card for ALICE in RUN 2, the integration and maintenance of this platform in the High Level Trigger, as well as the evaluation of higher-level hardware description approaches. All works are framed by the experiment schedule to have a thoroughly tested and reliable system installed in the production systems well before the start of RUN 2.

The selection of the target hardware platform starts with the definition of the requirements as well as the evaluation of available technologies using literature and reference hardware. Production-ready software and firmware developments are crucial already at the evaluation stage in order to draw conclusions on the scalability of approaches and solutions. PCI-Express-based firmware developments strongly depend on software interfaces to test and validate the hardware behavior. Hardware evaluations, as well as firmware and software developments for the basic functionality, are, therefore, done at the same time.

In parallel to the actual hardware development, prototype systems are set up using commercial FPGA boards with reduced feature sets. This step is important to evaluate and prepare different conceptual and technical aspects that will be part of the final system. Firmware components that are confirmed to be working in the prototype systems are then used to create a hardware test suite for the verification of the final hardware. A well-defined and automated test procedure is prepared for all boards to ensure that only fully tested boards are installed into the ALICE production systems. All tasks related to the development of the hardware platform, the coordination with industry partners for the PCB layout, the production of prototypes, as well as the administrative steps towards a large-scale hardware production have to happen in parallel to the firmware and software developments for the ALICE production systems.

The development of the final firmware variants required for the operation of the hardware in the HLT production system, as well as the integration into the HLT software environment, starts as soon as a working prototype hardware is available. This process is planned in multiple steps. The most important aspect to enable tests of the hardware platform in the HLT framework is the reliable data transport between the optical links and the host machine. This can be realized with a reduced number of serial optical links and with the RUN 1 link speed of 2.125 Gbps at

[16] https://eclipse.org/

the beginning. Once this is there, the support for higher link rates, as well as the full number of parallel links, is added.

Another important aspect is the integration and extension of the hardware cluster finder algorithm in the HLT firmware. With the upgrade of the TPC readout electronics, this implementation has to be able to handle double the input data rate compared to Run 1 in combination with different running conditions. The integration of the hardware cluster finder is also planned in two steps. An implementation with the cluster finder as used during Run 1, but ported to the new hardware, already enables tests with the existing detector front-end electronics. Extensions to the cluster finder, as well as the support for the higher link rate, are added in a second step. The implementation for the increased TPC link rate can only fully be tested with the availability of the corresponding Readout Control Unit 2 (RCU2). Retrospectively, this turned out to be not the case at the start of Run 2. Ideas to extend the Run 1 cluster finder algorithm for an improved HLT physics performance exist and can be evaluated and implemented as soon as the existing algorithm is running stable with the RCU2.

All firmware developments start with VHDL and Verilog. The vendor-provided reference designs as well, as all already existing and reusable firmware modules from the Run 1 hardware, are available in these languages. This means that a basic readout framework for the new hardware will be prepared with the low-level hardware description languages in any case. However, the limitations of this approach were already pointed out in Section 1.6.7. In parallel to the HDL firmware development, the dataflow description approach is evaluated using the HLT cluster finder as an example. In a first step, the processing core is implemented on a vendor-supported hardware platform and with the provided dataflow framework. The implementation of a detector readout chain with the dataflow approach is investigated in a second step. This requires a combination of detector- and experiment-specific protocols available as HDL implementations in the dataflow approach. One solution could be the integration of ALICE-specific protocols, such as the DDL, into the dataflow framework. Another option is the integration of a processing core developed in the dataflow framework into the existing HDL firmware environment. The options for the integration of a dataflow implementation of the readout chain strongly depend on the results of this evaluation.

As soon as the hardware, firmware, and software is integrated into the HLT, this framework has to be maintained throughout Run 2. A primary goal is to reduce the amount of "expert knowledge" required to operate the boards in the cluster. Repeating tasks have to be automated and integrated into the cluster management system in order to avoid human errors. Manual interventions have to be reduced to an absolute minimum and must not be required for the regular data-taking operations. Firmware and software fitness for an automated operation like this requires sanity checks and validation tasks that go far beyond what would be needed to operate the same hardware in a small-scale standalone lab setup. This level of automation is gradually added over time. Additionally, detector characteristics and changing data-taking conditions require a continuous adjustment of the data transport, processing steps and the monitoring parameters.

Chapter 2

C-RORC Hardware Platform

2.1 ALICE RORC Hardware Requirements for Run 2

The ALICE RORC for Run 2 had to be compatible with the existing detector front-end electronics as well as the existing fiber installation. It had to be possible to replace the previous H-RORCs and D-RORCs with the new board for all detectors. On top of this, increased readout link rates had to be supported for detectors that upgraded their front-end electronics during LS1 or Run 2. The Time Projection Chamber detector doubled its readout bandwidth compared to Run 1 and the Transition Radiation Detector increased its readout link rate from 2.125 Gbps to 4.0 Gbps. Support for higher link rates at the same time enabled to increase the bandwidth for the transport of HLT-processed detector data back to the DAQ system without the need for additional fiber connections. In the DAQ system, the new RORC had to be able to be operated in a mixed setup together with existing D-RORCs that were reused from Run 1 for some detectors. The new RORCs had to replace all previous H-RORCs in the HLT for Run 2.

An increased number of serial optical links per board was highly welcome in order to reduce the overall number of boards and host machines that are required to handle all DDLs. A hardware platform with 12 serial optical links provided the opportunity to have six DDLs from the detectors and six copies of these links to the HLT per board in the DAQ system. The HLT could then use up to 12 links per board to receive the detector data from the DAQ system. Assuming full link utilization per RORC, the increase from two to 12 links per board could reduce the overall number of required boards both for the DAQ and the HLT systems by a factor of six compared to Run 1. The number of links per board was strongly motivated by the TPC readout architecture. Six DDLs from the TPC cover one full detector sector. This provides data locality for the data processing and the data aggregation already in the RORC host machine. The link rates that can be used with the new RORC had to be configurable. The final link rates from the detectors were not decided at the time the RORC hardware development was done. This fact required the link rate not only to be switchable but also freely adjustable within the supported range of the transceivers. The option to have the link rate also configurable at runtime enabled to use the same firmware variant for several detectors. The experiment's general fiber installation was planned to be reused from Run 1. The possibility to connect the optical links of the new RORC to the existing patch panels in the counting rooms of the DAQ and HLT systems had to be ensured for the chosen optical link solution.

The interface to the host machines had to be compatible with current and upcoming server generations to avoid interface incompatibilities. It had to provide sufficient bandwidth margins to handle the raw detector data input bandwidth from the serial optical links. With up to 12 optical input links at 2.125 Gbps or higher, the RORC required a PCI-Express bandwidth well above 25 Gbps. For detectors with higher link rates, it was acceptable to reduce the number of DDLs per RORC. This also relaxed the data transport bandwidth requirements between the nodes in the cluster. A PCI-Express generation-2 interface with eight lanes is at the upper end of

what FPGAs at that time could deliver. This interface provides 40 Gbps PCI-Express bandwidth, which is sufficient to handle the expected input data rates. PCI-Express does not guarantee a certain latency or bandwidth between host and device. If the host system is busy with other tasks, the PCI-Express root port may stall data transfer from its endpoints for short periods of time. For this reason, the RORC had to provide a certain capability to buffer incoming data until the interface to the host machine is ready to accept new data. Data buffering can be realized with in-FPGA memories like block RAMs or FIFOs, if the required buffering capabilities are small enough, or with on-board storage like SRAM or DRAM for higher buffer capacities.

On-board DRAM was a requirement for the HLT application. The possibility to replay detector data from the RORC's on-board storage into the HLT system as if it were coming via the optical links proved to be a highly valuable tool for commissioning, testing, and benchmarking the HLT system already in Run 1. For realistic replay conditions, the effective read bandwidth from the on-board DRAM had to be at least equal to or above the aggregated maximum data input bandwidth from the optical links. On-board DRAM was not required for the DAQ application.

The RORCs in the DAQ system needed an LVDS connection to the Busy Box via Ethernet cables also in Run 2, similarly as on the D-RORCs. The firmware implementation could, however, be adjusted to handle more than one DDL per LVDS pair.

Increasing the throughput of the RORC with respect to the higher detector data link rates for Run 2 was expected to require more FPGA resources than the previous implementation. Resource usage calculations, therefore, had to account for more than just six times the logic resources available in the H-RORC. Additionally, the increased readout link rate for the TPC already suggested significant changes also to the hardware cluster finder implementation in the HLT RORC firmware. This means that the RORC had to provide more than six times the logic resources to implement the functionality of up to six H-RORCs into one FPGA. Any additional resource margins provide the possibility to also extend the hardware data processing algorithms in the firmware.

Besides the hard technical facts that needed to be fulfilled, there were a number of features that could improve the integration, the handling, and the maintenance of the RORC installations. The RORC needs a reliable procedure both to configure the FPGA at the initial power-on from an on-board storage and to support the upgrade of the stored firmware image. While accessing a single board with a programming cable is easily possible if things go wrong, doing the same in a cluster environment with boards installed in several dozens of machines is not feasible and has to be avoided. Optimally, a "known-to-be-working" firmware is stored on the board once and is kept in storage during firmware upgrades. This provides a backup solution with a configuration that can at least be used to deploy a new firmware image if an upgrade fails. Providing storage for more than one configuration image on the board is possible. Software access to the on-board configuration storage can be realized via PCI-Express. However, communicating with the board via PCI-Express is not possible if its configuration failed. For this reason, a secondary control path to the board would be convenient to monitor or control the FPGA configuration process.

Format	max. height	max. length	link widths
Standard height, half length	111.15 mm	167.65 mm	x1, x4, x8
Standard height, full length	111.15 mm	312.00 mm	x1, x4, x8, x16
Low profile	68.90 mm	167.65 mm	x1, x4, x8, x16

Table 2.1: PCI-Express add-in card sizes [PS 07].

The form factor of the RORC had to comply with the PCI-Express Card Electromechanical Specification (CEM) [PS 07] in order to be usable with a broad range of server machines. PCI-Express boards are allowed to come with different heights and lengths as shown in Table 2.1 and can span across up to two slots. The GPUs that were installed in the HLT are double-slot PCI-Express devices and require machines with appropriate PCI-Express slots anyway. However, keeping the RORC height at one slot increases its compatibility with more non-GPU/non-HPC

server machines and enables the boards to be integrated more densely. This is also relevant for the DAQ system where no GPUs are planned.

The running conditions and readout architecture of ALICE beyond Run 2 were not defined at the time this development was started. Given the technological progress over the recent years and the typical market lifetimes of electronic components, it was not reasonable to try to find or develop a hardware platform that could also be used for Run 3 ten years ahead. Nevertheless, this eventually happened in the end.

Unfortunately, none of the commercially available boards at that time were even close to the ALICE use case. This means that hiring a company to adjust an already available product to the ALICE requirements was not an option. The number of serial optical links on commercial boards barely exceeded four links, but with supported link rates way above the ALICE requirements for Run 2. Up to eight links were available only via add-on boards, but those made the setup exceed the PCI-Express form factor size limits. This was true also for custom FMC add-on boards, which could have also provided the required additional custom interfaces like the LVDS link via RJ45. Either way, this was not suitable for a production setup. A 12 link setup was not available at all. Changing the readout architecture to remove the need for the custom interfaces was not an option for Run 2.

For these reasons, the next RORC had to be a custom development again. In order to open the hardware also to other applications, it aimed to be as generic as possible while fulfilling all ALICE requirements. As a consequence, a central part of this work was the development of the Common Read-Out Receiver Card (C-RORC), a custom FPGA board to replace the D-RORCs and H-RORCs for Run 2 as a joint effort of the ALICE DAQ and HLT groups.

2.2 Hardware Features

A photo of the final board with a short explanation of its most important features is shown in Figure 2.1.

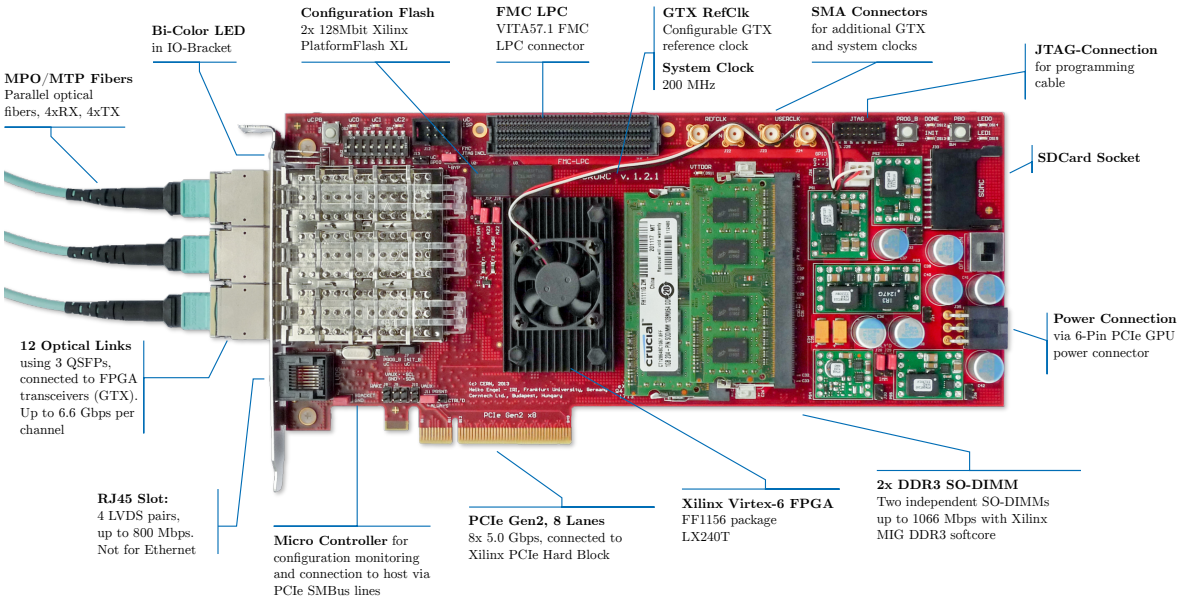

Figure 2.1: Photo of the final C-RORC hardware with brief descriptions of the most important hardware features.

The main component is a Xilinx Virtex-6 FPGA connected to 12 serial optical links towards the detectors and an eight-lane PCI-Express generation-2 interface towards the host machine. The optical interface is realized with three Quad-SFP modules. The FPGA configuration is stored

in on-board flash memories and a configuration controller provides additional controls over the configuration process via PCI-Express sideband signals. General-purpose on-board storage is optionally available via pluggable DDR3 SO-DIMM memory modules.

2.2.1 FPGA Selection

One of the first questions when designing a custom FPGA board is the selection of the FPGA brand and type. A lot of follow-up design decisions are dependent on this choice. Filtering the list of available devices by the requirements on the major interfaces as described in Section 2.1 narrowed down the selection to the two leading vendors: Xilinx and Altera. In mid-2010 when this development was started, the Xilinx Virtex-5 FPGA (65 nm) was available for quite some time and the first Virtex-6 FPGAs (40 nm) were available as engineering samples. On Altera side, the Stratix-IV FPGA (40 nm) was available since around 2008 with specifications comparable to or slightly below the Xilinx' Virtex-6. Its successor, the Stratix-V FPGA (28 nm) was only announced at that time. Both devices, the Stratix-IV and the Virtex-6, are able to deliver serial link rates sufficient for PCI-Express generation-2 and serial optical interfaces in the range of a couple of hundreds of Mbps up to above 6 Gbps. Link rates in the order of 10 Gbps were available only with special serial transceivers in the FPGAs, which were not compatible with lower link rates. A very important selection criterion was the availability of documentation, evaluation hardware, reference implementations, as well as experience and knowledge about these devices among the developers. Additionally, some portions of the VHDL code and especially the cluster finder were already available for Xilinx devices from the H-RORC project and other developments. All these facts combined led to the decision to use the Xilinx Virtex-6 FPGA as the target device.

Package	LX75T	LX130T	LX195T	LX240T	LX365T	LX550T	LX760T	SX315T	SX475T
FF484	✓	✓							
FF784	✓	✓	✓	✓					
FF1156		✓	✓	✓	✓			✓	✓
FF1759				✓	✓	✓		✓	✓
FF1760						✓	✓		

Table 2.2: Overview of available FPGA packages and parts from the Xilinx Virtex-6 product table [Xil 10]. The highlighted row shows the selected package and the blue checkmark shows the selected device.

Xilinx Virtex-6 FPGAs are available in different packages. Larger packages provide more user-programmable input/output signals. In addition, each package is available with a couple of options on the number of configurable logic elements available for the user logic. This enables the schematic design and the PCB layout to be prepared for a specific package based on input/output requirement while the actual logic size of the FPGA can be decided individually before assembly. An overview of the available Virtex-6 packages (in rows) with the corresponding part numbers defining their logic size (in columns) is shown in Table 2.2. The exact number of inputs and outputs for each package, as well as the number of logic resources available on each part, can be found in [Xil 10].

	Flip-flops	Look-up tables	Block RAMs	DSPs
H-RORC with 2x ClusterFinder	66%	73%	93%	31%
H-RORC without ClusterFinder	28%	34%	14%	0%

Table 2.3: H-RORC firmware resource usage on a Virtex-4 LX40 FPGA.

The estimation on the required number of configurable logic resources was done based on the existing H-RORC firmware design available for Virtex-4 FPGAs. The H-RORC firmware resource

usage is shown in Table 2.3. Given the required clock frequencies and the routing limitations of these devices on highly utilized designs, the H-RORC firmware with the cluster finder is close to the limit of what was achievable with the LX40T FPGA. Additional resources had to be accounted for a more complex DMA engine and two DDR3 controllers instead on one DDR1 controller. Savings were expected by the fact that not all 12 DDLs will be used for detectors with increased DDL rates. A reasonable estimation was to scale the device size with the number of DDLs. The H-RORC with two DDLs compared to the C-RORC with 12 DDLs makes a factor of six for the estimated resource requirements. Multiplying the Virtex-4 LX40T resources by this factor of six ends up at around the Virtex-6 LX240T FPGA. Technological progress from Virtex-4 to Virtex-6, including faster signal routing, increased look-up table width, and bigger on-chip RAM blocks, also provide a certain safety factor.

The FF1156 package or larger was required because smaller packages do not provide enough transceiver instances to operate eight lanes of PCI-Express in combination with 12 serial optical links. Choosing the FF1156 package provided flexibility in terms of the final part selection: both bigger and smaller devices compared to the LX240T are available and will fit onto the same PCB. The list of pin-compatible FPGAs in the FF1156 package is highlighted in Table 2.2. Another aspect is the FPGA cost: FPGA prices within the same package depend on the logic size. However, calculating the price per logic cell usually reveals that smaller devices are more expensive per logic cell than medium-sized devices. At the time the FPGA selection was done, the LX240T turned out to be also a cost-efficient solution compared to bigger and smaller parts.

In terms of FPGA speed grade, a classification on how fast a certain part can be operated internally, there was not much choice. The medium speed grade (-2) was at least required in order to have support for PCI-Express generation-2 at 5 Gbps and DDR3 at 1066 Mbps signal rate. The highest speed grade could have made it easier to fulfill timing requirements, but it did not provide additional features and came with higher costs.

2.2.2 PCI-Express Interface

The data interface between the host machine and the FPGA board is realized with PCI-Express. In order to be able to transfer all incoming data from 12 serial optical input links at 2.125 Gbps to the RAM of the host machine, a bandwidth of at least 25.5 Gbps plus some margin was required. This could be achieved by using an eight-lane PCI-Express generation-2 interface providing a total of 40 Gbps bi-directional bandwidth. PCI-Express uses two uni-directional differential pairs for each data lane. The PCI-Express standard consists of several protocol layers. A detailed description of these layers can be found in [PS 09]. The lowest layer, the physical layer, is further divided into an electrical and a logical part. The electrical part of the physical layer takes care of the serialization and deserialization of the data stream. The logical part of the physical layer applies an 8b10b encoding to ensure a DC-balancing of the serial data stream as well as data scrambling to avoid repeating patterns inducing electromagnetic interference (EMI). The distribution of data across several parallel lanes is also provided by the physical layer. On top of the physical layer is the data link layer. This layer orchestrates the sequence of data packets to be transmitted, takes care of handshaking protocols and data acknowledgment, as well as flow control credit handling. The transaction layer provides an interface for sending and receiving read or write requests, data completions, interrupts, or configuration requests.

Implementing a PCI-Express endpoint in a Virtex-6 FPGA is possible with the help of integrated Xilinx PCI-Express hard-blocks together with the FPGA serial transceivers and an IP core wrapping all components into an easy to use user interface. The distribution of the PCI-Express related FPGA building blocks across the different protocol layers is shown in Figure 2.2. The electrical part of the physical layer and portions of the logical part of the physical layer are done with the regular FPGA transceiver blocks. These are the same kinds of transceivers that are used for the serial optical links and can also be used for any other serial application. A 100 MHz reference clock for clock and data recovery is provided by the host machine via the PCI-Express connector. It is cleaned and up-converted to 125 MHz and 250 MHz with an external jitter

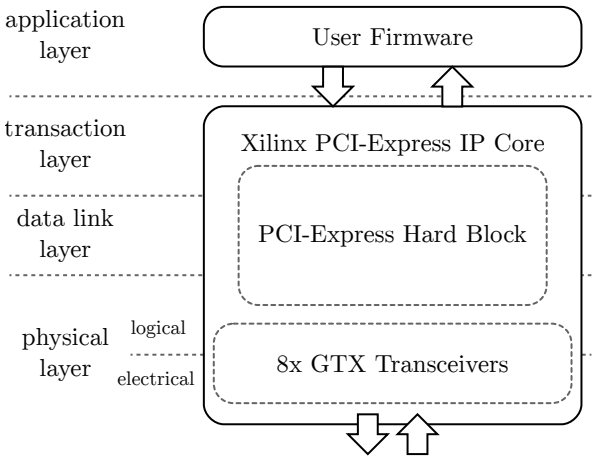

Figure 2.2: Distribution of PCI-Express related FPGA components across the protocol layer stack for an eight lane interface on a Virtex-6 FPGA.

cleaning component on the C-RORC PCB and routed to the reference clock inputs of the FPGA transceivers. The remaining parts of the physical layer, the data link layer and the transaction layer are shared across the PCI-Express hard-block and the Xilinx PCI-Express IP core using regular FPGA resources. The selected Virtex-6 FPGA provides two PCI-Express hard-blocks. It can operate up to two PCI-Express root ports or endpoints, each with up to eight lanes at the generation-2 link rate. As the bandwidth of one eight-lane generation-2 endpoint was sufficient for the ALICE application, the host interface could be kept simple and the second hard-block remained unused.

The Xilinx PCI-Express IP core provides an interface to the user firmware at the transaction level. The user firmware is responsible for assembling transaction-level packets according to the PCI-Express specification and reacting to incoming requests from the host machine. At this level, the user can issue read or write commands from the FPGA to the host RAM and can access the remote-side memory via low-level bus addresses. Apart from the Xilinx IP core for PCI-Express, there are commercial IP cores available that provide direct memory access via easy-to-use FIFO interfaces. These cores typically contain the low-level Xilinx core in combination with functionality to hide all transaction-level protocol handling from the user as well.

Figure 2.3 shows a measurement of the maximum throughput from the FPGA endpoint to the host RAM. The data points labeled with "payload only" show the aggregated rate of user data being transferred into predefined regions of the host RAM for 12 parallel readout processes. The readout scheme and the block size dependency are described in more detail in Section 3.4. The data points on the payload throughput including the transaction-level protocol headers can be calculated from the measured values because the size of the headers per transaction-level packet is known. On top of this, there is a certain overhead from the control flow and the credit handling on the data link layer. This is hidden from the transaction interface but is, nevertheless, going over the link. Comparing this usable bandwidth on the transaction interface of the Xilinx PCI-Express IP core with the physical limit of the link (shown as a dotted line) confirms that the PCI-Express interface can be used nearly up to the physical limit of the link and is suitable for the targeted use case.

In order to make this board distinguishable from other PCI-Express devices and to avoid device driver incompatibilities, the C-RORC is registered with a unique PCI device ID (`0x01a0`) at CERN using the CERN PCI Vendor ID (`0x10dc`). The board identifies itself in host systems as shown in Figure 2.4. The serial number is the unique device ID of the FPGA and enables to precisely identify each board in the cluster.

The PCI-Express connector contains not only the differential signals of the individual lanes and the reference clock but also provides a number of sideband signals. An illustration on the sideband

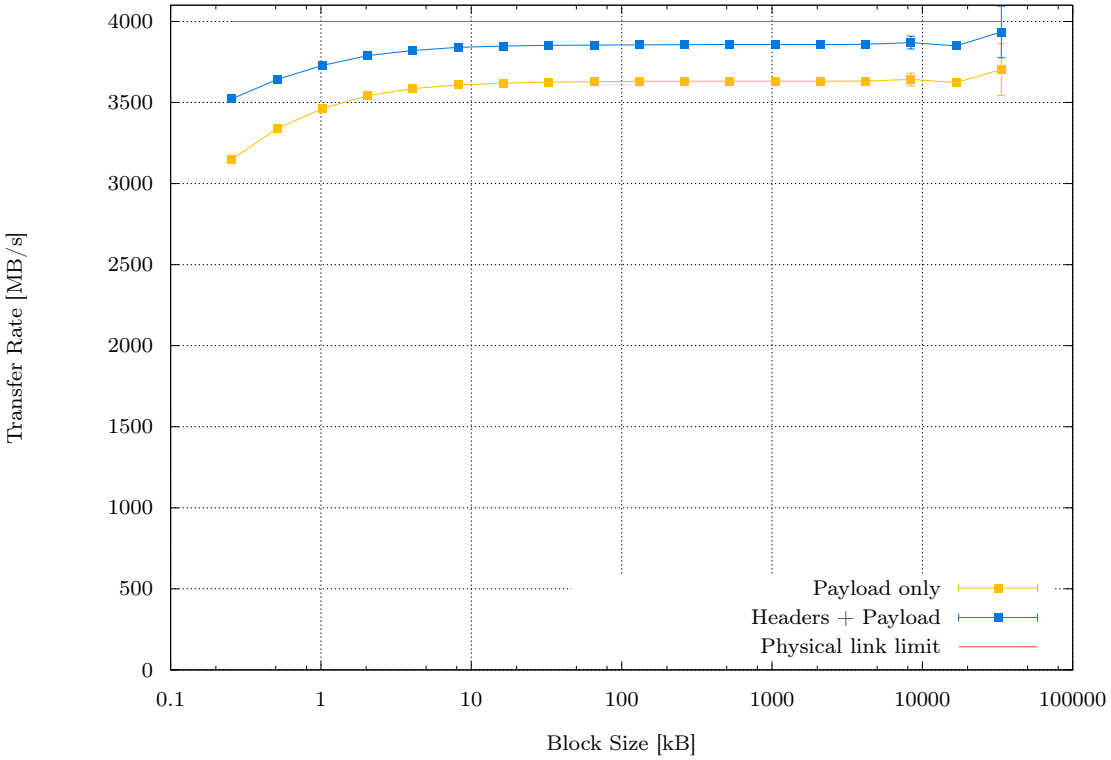

Figure 2.3: C-RORC maximum PCI-Express throughput from device to host.

signals that are used on the C-RORC is shown in Figure 2.5. The presence detection of PCI-Express boards is realized with a weak-high control signal from the mainboard to the add-in card. By pulling this signal to ground, the add-in card can inform the host machine that a card is inserted and can give hints about the maximum number of lanes it supports. The list of PCI-Express devices connected to a system is evaluated shortly after the mainboard is booted up but long before an operating system starts. This means that an FPGA-based PCI-Express device has to get configured as soon as the host machine is powered on in order to be properly detected. PCI-Express hotplugging is a method that allows devices to be added to or removed from the system while running. This is very attractive for FPGA applications because it enables to load a different FPGA firmware without having to reboot the machine. However, PCI-Express hotplugging assumes the add-in card to be physically removed from the system. Hotplug events are triggered by changed presence detection conditions. In order to address this limitation, the C-RORC optionally supports to control the state of the presence-detect signals from the FPGA. Given appropriate hotplugging support from the mainboard, this enables to trigger presence detection states as if the board was physically removed from the system and plugged in again.

Another set of sideband signals are the System Management Bus (SMBus) clock and data lines. SMBus [SBS 00] is a two-wire bus based on I^2C [Phi 01]. It is mostly an extension to I^2C with stronger restrictions but also slightly different in some details. Nevertheless, the two-wire interface (TWI) blocks as they typically come with microcontrollers can be used to implement an SMBus slave. In this case, this is realized with an Atmel ATmega[1] microcontroller and provides an alternative connection to the board even if the PCI-Express link is down. The role of the microcontroller is described in more detail in Section 2.2.7. SMBus via PCI-Express is an optional part of the PCI-Express specification. This means that mainboard vendors can choose if an interface to these signals is integrated into the system. For this reason, the SMBus signals are additionally available on a pin header to support a cable-based connection to the C-RORC if SMBus via PCI-Express is not available.

[1] http://www.atmel.com/products/microcontrollers/avr/megaAVR.aspx

```
# lspci -s 83:00.0 -v
83:00.0 Signal processing controller [1108]: CERN/ECP/EDU Device 01a0
        Subsystem: CERN/ECP/EDU Device 01a0
        Flags: bus master, fast devsel, latency 0, IRQ 38
        Memory at f8000000 (32-bit, non-prefetchable) [size=32M]
        Memory at fa000000 (32-bit, non-prefetchable) [size=4M]
        Capabilities: [40] Power Management version 3
        Capabilities: [60] Express Endpoint, MSI 01
        Capabilities: [100] Device Serial Number 00-93-46-ac-38-dc-05-80
        Kernel driver in use: uio_pci_dma
```

Figure 2.4: PCI identification of the C-RORC in the host system.

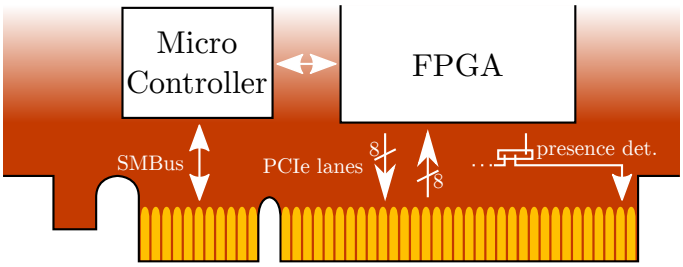

Figure 2.5: PCI-Express data lanes and sideband signals and their connection to the FPGA and the microcontroller.

2.2.3 Serial Optical Links

The main requirement on the serial optical links is the ability to read out up to 12 optical detector links. The first generation of the DDL as used during Run 1 is operated at 2.125 Gbps. The Source Interface Units (SIUs) on the front-end electronics use 850 nm SFP modules connected to OM2 optical fibers with 50 µm core and 125 µm cladding diameter. The fiber lengths from the detectors to the DAQ system vary between around 80 m and 130 m. They have up to five patch panels in the signal path and end up on patch panels with the E2000 connection standard in the DAQ counting room. The connection between the DAQ system and the HLT is using the same type of patch panels but is with up to around 30 m significantly shorter. The C-RORC solution had to be compatible with the already installed detector readout equipment in terms of link rate, optical characteristics, and physical connectivity. On top of this, an upgrade of the readout link rate had to be possible without changing the fiber optical installation. Table 1.2 in Section 1.6.3 shows the maximum possible link rates for OM2 fibers with a laser wavelength of 850 nm. The maximum cable lengths of the existing ALICE fiber installation between detectors and DAQ system of around 130 m result in a maximum link rate of around 5 Gbps. This fits well with the upgrade plans of TPC and TRD for Run 2. A higher link speed is possible between DAQ and HLT due to the significantly shorter fibers. A 10 Gbps link is neither planned nor possible with the current fiber installation for Run 2.

The implementation of the serial optical links on the C-RORC consists of two main aspects. In a first step, the received serial optical signal has to be translated into a serial electrical signal. In a second step, this serial electrical signal is decoded and converted to a parallel electrical bus. Data transmission works the other way around. A parallel electrical bus is serialized in a first step and translated into a serial optical signal in a second step.

The translation from optical to electrical signals is usually performed with optical transceiver modules. The sending units on the detectors use regular SFP modules for this. The main problem with SFP modules on the C-RORC is the space they need on the board and the IO-bracket. A single-height PCI-Express card can host at most four SFP modules. It is possible to have SFP slots both on the top and the bottom side of the PCB. However, this would then

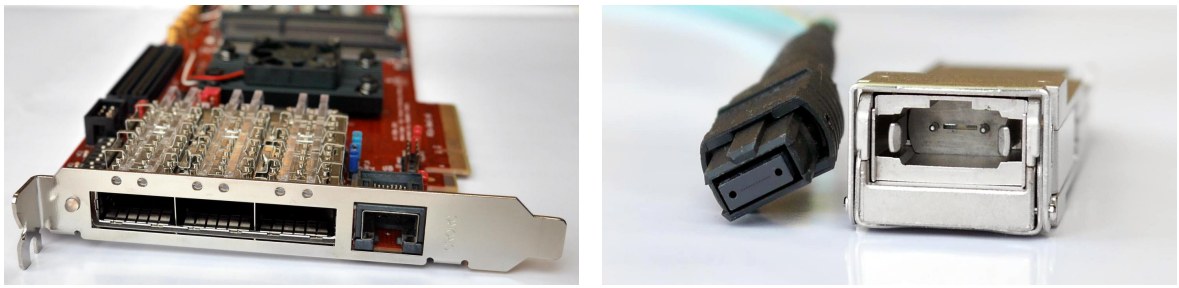

Figure 2.6: C-RORC optical interface. Left: IO-bracket with the three QSFP module slots. Right: Optical interface of a QSFP module and the according parallel optical cable.

occupy two PCI-Express slots and still only give eight serial optical links in total. In order to get the requested link density on the C-RORC, a parallel optical solution was required. Section 1.6.3 describes available technologies. The choice was made in favor of Quad-SFP (QSFP) modules. QSFPs enable to place three modules on the IO-bracket resulting in 12 links while leaving still some space for the also required LVDS connection. A photo of the C-RORC IO-bracket is shown in Figure 2.6 on the left. The main reason for using QSFPs was their availability from several vendors and their flexibility. Retrospectively, QSFPs also turned out to be the most future-proof variant of the available choices. QSFPs as available at that time typically supported link rates from around 2 Gbps to 10 Gbps. The optical interface is using the multi-fiber push on (MPO) standard. The more commonly used term MTP is a brand name registered by US Conec for "a high-performance MPO connector engineered for better mechanical and optical performance" [US 15]. The MPO variant used for the QSFPs has 12 fibers arranged horizontally in the center of the connector. A photo of an MPO fiber and a QSFP module is shown in Figure 2.6 on the right side. The mechanical pins left and right of the fibers are used to ensure proper alignment between the fibers of the QSFP module and the cable. The four outer fibers on each side are used for sending and receiving. The inner four fibers are not connected to the QSFP module. Break-out fibers are available to connect QSFPs to a number of different connector standards. This includes the LC standard for direct SFP interconnects and E2000 to connect to the existing patch panels. QSFP modules and MPO fibers are available with multi-mode lasers running at 850 nm and with 50/125 µm core/cladding as required to connect to the existing SIUs. Typical QSFP modules with 4× 10 Gbps apply the 40GBASE-SR4 specification for Ethernet at 40 Gbps. In this context, the average optical launch power (2.4 dBm) is higher than the maximum allowed optical input power into typical SFP modules used for Gigabit Ethernet or FibreChannel applications (-1 or 0 dBm). For this reason, sufficient attenuation from either the fiber installation itself, explicit attenuation components, or special QSFP modules with reduced optical output power are required to connect QSFPs with SFP modules.

The QSFPs convert the serial optical stream into a serial electrical stream and the other way around. There are two possibilities for the serialization and deserialization of the electrical signals: connecting dedicated external transceiver chips to the FPGA or using the FPGA-internal transceiver blocks to handle the lowest physical level of the protocol. The DDL protocol internally operates on a 16 bit data bus. The data stream is encoded into a 20 bit parallel bus using an 8b10b encoding before serialization. Predefined 10 bit characters are used to recover the clock and control the data stream. TLK2501 transceiver components from Texas Instruments connected to the FPGA were used in the past on the DIU and SIU modules as well as on the D-RORCs. These transceiver chips come with two parallel 16 bit data interfaces plus a few control signals towards the FPGA. They hide a lot of technical details about the serialization and deserialization from the user and provide an easy-to-use interface. The chips are also available for different link speeds. However, they come with two major disadvantages: their pin count and a limited flexibility.

Two 16 bit data buses need 32 FPGA IO pins. Taking the control signals and clocks into account results in 42 IO pins per optical link to be connected to the FPGA. Having 12 optical links

would already occupy in the order of 500 of around 600 available FPGA IO pins to be used only for the serial links. The high number of parallel data buses would additionally make the PCB layout much more complicated. In terms of flexibility, the external transceivers can usually be operated in smaller ranges of link speeds compared to the FPGA transceivers and have low-level link parameters statically set.

Using the FPGA internal transceivers to handle the lowest physical layer of the serial link protocol enables to use almost any link rate between a few hundred Mbps and around 6.6 Gbps. The serial link encoding and the control flow remains fully configurable and settings can even be changed at runtime. The PCB layout to connect the QSFPs to the FPGA is comparably simple and requires only one differential signal pair per link per direction. The only disadvantage of this method is the huge number of configuration options of the FPGA transceivers coming with their flexibility. The user has to set and understand all options in order to operate a custom link protocol reliably.

Serial links have no dedicated clock signal from the sender unit that is transferred in parallel to the data stream. Instead, the receiver has to recover the sender clock from the data stream. This is achieved by aligning a Phase Locked Loop (PLL) on the receiver side to the logic transitions of the incoming data stream. These PLLs are part of the FPGA transceivers. The data stream has to contain a minimum and a balanced number of low-to-high and high-to-low transitions. Data encoding schemes like 8b10b encoding and scrambling algorithms help to ensure this. To get the PLL to lock onto the data stream it has to be running already close to the target frequency. A local reference clock oscillator is used together with clock multipliers and dividers to feed the PLL. The reference clock frequency and the PLL multiplication and division factors define the target link rate. The number of possible target link rates for a given reference clock frequency is limited by the number of allowed combinations of multiplication and division factors. Different reference clock frequencies are required to support a wider range of possible link rates. Table 2.4 shows an overview of the link rates and the corresponding reference clock frequencies that are used with the C-RORC. This table also makes clear that a reference clock oscillator which is configurable to different clock frequencies is a key aspect of making the hardware suitable for a wide range of applications. With a fixed-frequency oscillator, and even with oscillators with four selectable frequencies, it would not have been possible to use the board in all applications that it is used now.

Link rate (Gpbs)	Reference clock frequency (MHz)	16 bit parallel bus clock frequency (MHz)	Application
2.0000	100.00	100.000	ATLAS S-Link
2.1250	212.50	106.250	DDL1
2.5000	125.00	125.000	PCI-Express generation-1
3.1250	156.25	156.250	DDL2 (TPC, CTP)
4.0000	200.00	200.000	DDL2 (TRD)
4.2500	212.50	212.500	DDL2 (TPC, planned)
4.8000	240.00	240.000	GBT
5.0000	250.00	250.000	PCI-Express generation-2
5.3125	212.50	265.625	DDL2 (DAQ/HLT)

Table 2.4: Some serial link rates with typical reference clock frequencies, resulting user clock frequencies on a 16 bit interface and application examples. A number of reference clock frequencies are required to cover all use cases.

The C-RORC has a reference clock oscillator connected in a way that all 12 transceivers can use this clock to feed their PLLs. The frequency of the reference clock oscillator is starting up with 212.5 MHz after power-on but can be reprogrammed to any frequency between around 10–1400 MHz from the FPGA via I^2C. This enables the board to be used for any serial link rate between a couple of hundred Mbps up to around 6.6 Gbps, covering the full supported range of the FPGA transceivers. All 12 transmitters can use the same user-side clock for sending while each receiver can have its own independently recovered clock for receiving data. A pair of SMA

connectors and a set of clock inputs from the FPGA Mezzanine Connector (FMC) can provide additional external reference clock inputs if needed. At the protocol level, this setup can run the DDL protocol at any link rate within the limits mentioned above. It uses a 16 bit wide data path to the transceivers and makes use of the built-in 8b10b encoding blocks. Any other link protocol is also possible with an 8(10) bit, 16(20) bit or 32(40) bit data path using the built-in 8b10b or a custom (bit widths in brackets) data decoding/encoding scheme.

2.2.4 General-Purpose On-Board Storage

General-purpose on-board storage is a requirement for the HLT application. Simulated or previously recorded detector data should be loadable into the board for data replay purposes to enable standalone full system tests. The ability to buffer incoming data from the optical links in on-board memory is not a requirement. Measurements on commercial FPGA boards have shown that typical PCI-Express dead times can be compensated with the available on-chip memory and external storage is not necessary for this purpose. This is described in more detail in Section 3.9. The previous H-RORC had four DDR SD-RAM components with 256 Mbit each. Depending on the firmware type, 32 to 64 MB of on-board storage per input link was available for data replay. The C-RORC had to provide at least a comparable amount of on-board storage per link for data replay that can deliver the same bandwidth as 12 DDL inputs. For the DAQ application, on-board storage was not required at all.

Typical on-board storage solutions were DDR3 SDRAM and QDR SRAM modules. The SRAM solution can easily fulfill the bandwidth requirements but comes with a comparably low storage capacity per component. DDR3 SDRAM is slower but still sufficient in terms of bandwidth and can provide a couple of gigabytes of on-board storage. It is available as individual components to be soldered directly onto the PCB or via pluggable dual in-line memory modules (DIMMs). These provide access to a number of memory components via a standardized interface. DIMMs in different form factors are already a common solution for memory in servers, laptop computers, and desktop PCs. The mobile-oriented DDR3 small outline DIMM (SO-DIMM) could be connected to the C-RORC FPGA. This pluggable solution came with three advantages for the C-RORC:

- Only the appropriate connector had to be foreseen at the board design and production phases. The capacity of the memory modules to be used could be decided later on for each application at installation time and could be changed at any time by replacing the modules.
- The memory only needs to be purchased and installed on boards where it will actually be used. The DAQ group did not have to pay for memory that was not used in their application. Less memory could be installed on the HLT boards where not all input links were used.
- The same type of modules is used for laptop computers so there was a large market with a number of modules sizes and the prices are comparably low. This automatically maintained a set of compatible modules whereas a specific component on the C-RORC PCB could easily become obsolete.

The C-RORC has two DDR3 SO-DIMM sockets. At the time of this writing, the maximum module size available is 8 GB so up to 16 GB of on-board storage could be installed on the C-RORC. In order to comply with all layout requirements, a solution with two slightly overlapping SO-DIMM modules could be found using two sockets of different heights. The overlap enabled shorter trace lengths to the second connector while keeping the impact on heat dissipation low. A photo of the overlap is shown in Figure 2.7. Placing one or both SO-DIMMs on the back of the PCB was not an option because this would have violated the maximum component height of the PCI-Express specification. The final solution was verified by the PCB layout company with signal integrity simulations at the PCB level for different module types and sizes.

A DDR3 controller IP core is provided by Xilinx. With this core, the modules can be operated at up to 1066 Mbps for single-ranked modules and 606 Mbps for dual-ranked modules. The

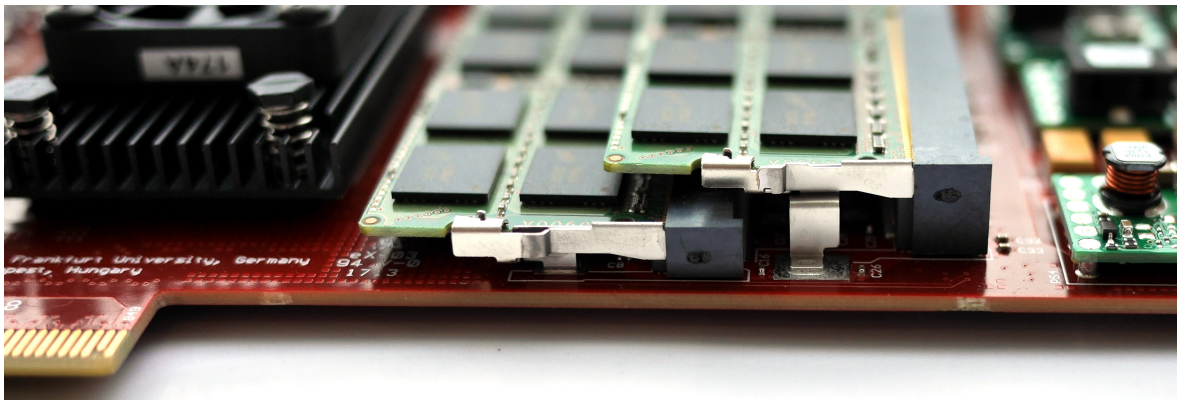

Figure 2.7: C-RORC DDR3 SO-DIMM modules with slight overlap.

controller for the Virtex-6 FPGA is fully implemented using the regular FPGA resources so it can be adjusted to different modules. A drawback of this solution is, however, that all module-related timing parameters have to be provided to the controller at firmware compile time. This means that changing the module type requires adjustments in the firmware. Data throughput measurements for different module types and access schemes are shown in Section 3.7.1. The serial interfaces of the SO-DIMM modules are also connected to the FPGA. This enables to read timing or temperature parameters from the modules.

2.2.5 Power Supply and Power Distribution

FPGA boards usually require a number of supply voltages for the FPGA itself and for the different components on the board. The Virtex-6 FPGA needs a 1.0 V core voltage for the internal operation of the device. The configuration interface is using 2.5 V. The interfaces to other components on the board typically use 1.5 V or 2.5 V standards. 1.5 V is also the main supply voltage for the DDR3 memories. The serial transceivers in the FPGA need another two analog supply voltages which have to be comparably clean. The QSFPs and a couple of status, control, and I^2C related signals require 3.3 V. The configuration storage flash memory chips additionally need a 1.8 V core voltage. On top of this, there are a few reference voltage generators for the signal termination and analog-to-digital converter references. An overview of all voltages on the C-RORC is shown in Figure 2.8. The blue boxes indicate switching regulators that convert the 12 V input to the corresponding voltages. A second stage of linear regulators shown in yellow creates low-noise supplies for the transceivers and the flash core voltage. The current ratings on the conversion modules are the maximum current each module can deliver. The currents required on each supply were estimated with the help of the Xilinx Power Estimator[2] tool for the biggest FPGA from the chosen package with all serial links at the maximum rate and a significant portion of the available logic cells actively switching. On top of this comes a generous safety margin towards the limit of what the individual modules can actually deliver. Retrospectively the power distribution network for the finally used FPGA is pretty much overdesigned but it can handle whatever FPGA fits onto the PCB.

All voltages required for the operation are generated from one common power input. Typical voltages available in server PCs are 12 V, 5 V, and 3.3 V. There are two options to power a PCI-Express device in a server: using the power connections provided via the PCI-Express socket or using a dedicated power cable from the power supply. The PCI-Express connector has 12 V power lines from the host foreseen as the main power source for the devices. A 3.3 V auxiliary line is optionally available but primarily meant for power management tasks. It cannot provide the required currents. A typical cable-based powering solution is to use the same type of PCI-Express

[2] http://www.xilinx.com/products/technology/power/xpe.html

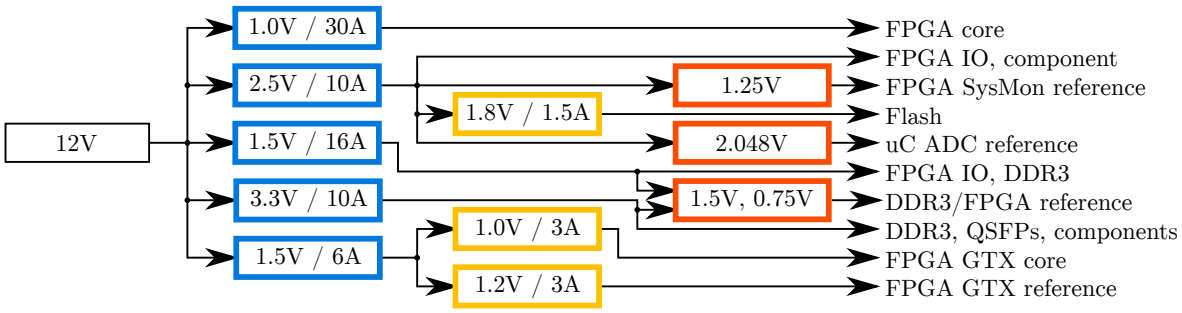

Figure 2.8: C-RORC power distribution network. Blue boxes show switching regulators, yellow boxes indicate linear regulators, red boxes are reference voltage generators.

power cables as for GPUs. These cables provide 12 V over three or four wires and can deliver more current than what is specified for the PCI-Express socket.

The selection of whether to use the socket or the cable-based solution was made based on the current consumption estimations and complexity considerations. The PCI-Express socket can deliver up to 25 W via the 12 V lines from the host machine at power on. A 16 lane PCI-Express device can then be reconfigured to a high-power device later on. This allows it to draw up to 75 W [PS 07]. 75 W is clearly enough for the C-RORC but 25 W can be at the short end. In order to ensure compatibility, a two-step power-on sequence would have had to be implemented: A base firmware that is guaranteed to never draw more than 25 W to be loaded initially, a configuration step towards the high-power settings, and finally the enabling of the extended functionality. This clearly introduces a complexity into the start-up process that can easily be avoided by using the cable-based solution. GPU power cables are the obvious choice. The HLT nodes coming with GPUs installed have these kinds of cables available anyway.

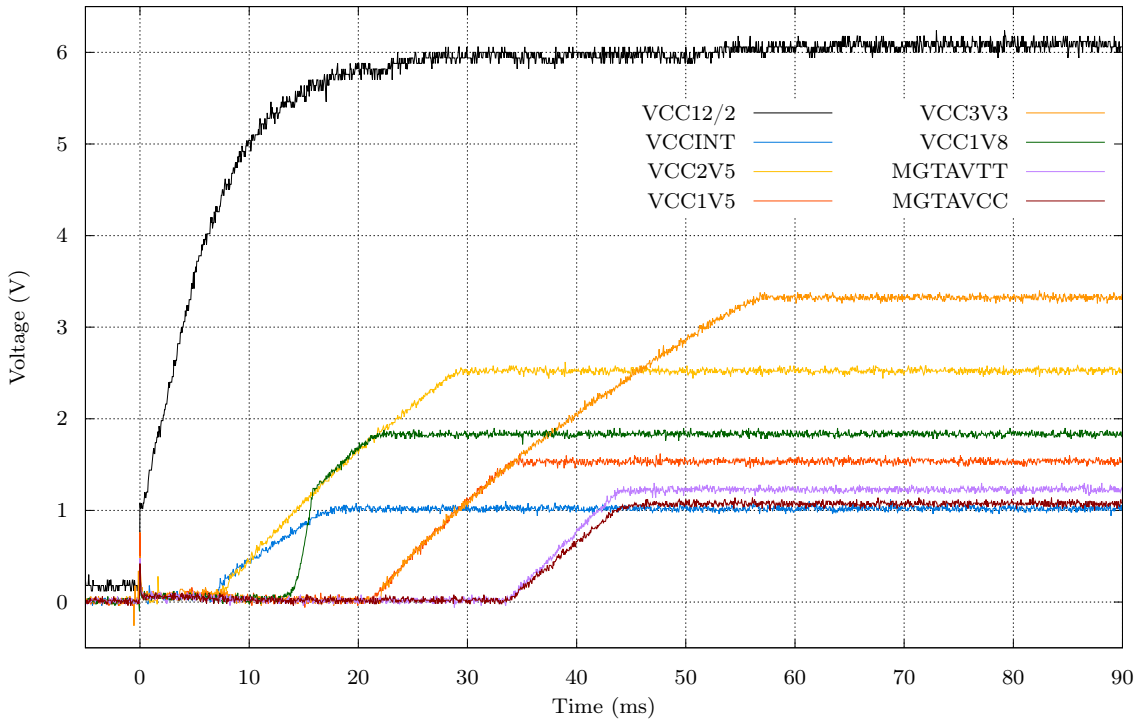

Figure 2.9: C-RORC power-on sequencing of the different supply voltages.

The FPGA comes with some constraints on the power-on sequence of the different supply voltages. Most importantly, the 1.0 V core voltage should be the first to be up. The 2.5 V configuration rail may rise at the same time or later. The remaining IO and transceiver voltages can come up at any time after the core voltage [Xil 14]. Figure 2.9 shows an aggregated graph of the C-RORC power-on sequence of all supply voltages. Each voltage was measured separately together with the 12 V input voltage and all measurements are drawn into the same graph. The 12 V input is scaled down by a factor of two for readability reasons. The core voltage rail (blue) and the 2.5 V rail (yellow) are the first to rise a couple of milliseconds after the input supply voltage is applied. The core voltage is stable at approximately 18 ms after power-on. The 2.5 V rail reaches its target value after around 30 ms. The 1.5 V (red) and 3.3 V (orange) rails are delayed with voltage supervisors to start the rise-up only around 15 ms after the input supply exceeds around half its target value. The transceiver voltages at 1.06 V (dark red) and 1.2 V (purple) are generated with linear regulators from the delayed 1.5 V rail and rise accordingly after the source supply is stable. Due to experience with previous Xilinx FPGAs, the transceiver supply voltage is adjustable by a potentiometer on the C-RORC within a certain range around the recommended values given in the datasheet.

2.2.6 FPGA Configuration Modes and On-Board Configuration Storage

The C-RORC FPGA is SRAM-based, so its configuration is volatile. This means a configuration file has to be loaded into the FPGA whenever the board is powered on and it gets lost as soon as the board is powered off. This configuration can come from either a remote storage using a programming cable or it can be stored in a dedicated on-board memory. The cable solution is using the JTAG [IEE 13 II] standard. It is a common method during the firmware development phase and when working with single boards, especially because it also comes with in-system debugging capabilities. It is, however, not suitable to manage the configuration of a number of boards installed in a computing cluster. For this case, the FPGA configuration has to be available on the board and the FPGA should automatically get configured whenever it is powered on.

The FPGA can be configured from on-board memory in two ways. In slave mode, the FPGA is mostly passive and expects configuration data to be provided from external components. This could, for example, be a microcontroller or a Complex Programmable Logic Device (CPLD) reading data from any kind of on-board memory and sending it to the FPGA. In master mode, the FPGA is directly connected to the on-board storage and actively controls the memory read access itself. The configuration data interface to the FPGA can be used as a serial or parallel bus depending on the required speed and the data interface of the storage memory. This master mode is very attractive because it does not require any additional external components in the data path that need to be programmed and the FPGA has direct access to the on-board storage also after configuration.

FPGA	LX130T	LX195T	LX240T	LX365T	SX315T	SX475T
Config file size	43.7 Mbit	61.6 Mbit	73.9 Mbit	96.1 Mbit	104.5 Mbit	156.7 Mbit

Table 2.5: Uncompressed configuration file sizes of all FPGAs compatible with the C-RORC.

The selection of the components for the C-RORC FPGA configuration is mostly defined by the size of the FPGA configuration file and the timing requirements from the PCI-Express interface. The PCI-Express specification comes with strict requirements on when a device has to be able to respond. PCI-Express endpoints have to be “ready to receive Configuration Requests within 100 ms of the end of Conventional Reset” [PS 09]. In the case of the initial power-on, the end of the conventional reset means the PCI-Express reset is released. The PCI-Express power-on specification gives 100 ms of stable power until the reset signal may go inactive [PS 07]. The ATX specification for power supplies guarantees another 100 ms of stable power until this condition is signaled to the system. The PCI-Express root port, therefore, cannot release its reset before this condition is fulfilled. For the C-RORC, this means that in the worst case the FPGA has to

be fully configured and running at around 200 ms after the input power is stable. The C-RORC power-on sequence shown in Figure 2.9 makes clear that the power distribution network itself already needs around 30 to 40 ms after the initial rise of the external supply voltage until the FPGA configuration process can actually start. Typical configuration file sizes for all FPGAs compatible with the C-RORC PCB layout are shown in Table 2.5. Given the remaining time and the configuration file sizes, this requires configuration data rates of several hundred Mbps. This is not possible with serial configuration protocols and is challenging for asynchronous parallel flash memories. Page-based read modes improve this situation but may still require more than 200 ms in total. Third-party flash memories offer synchronous read modes that can deliver the required data rate. However, the synchronous read mode is not enabled by default and cannot be enabled by the FPGA so an additional CPLD or microcontroller would be required.

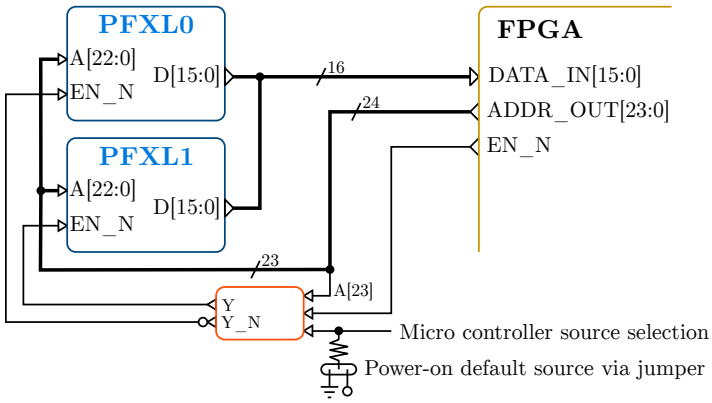

Figure 2.10: Arrangement of two PlatformFlash XL flash memories for multi FPGA configuration file storage.

The solution chosen for the C-RORC is to use XILINX PlatformFlashXL (PFXL) memory chips for the FPGA configuration. These flash memories provide 128 Mbit storage capacity and start up in a synchronous burst read mode that can continuously provide 16 bit of data at a frequency of up to 50 MHz. This enables to push the full 128 Mbit into the FPGA in the given amount of time. The memory chip can directly be connected to the FPGA in the master mode and, therefore, save additional components and complexity from the configuration interface. The XILINX tools support writing to this flash chip via the programming cable without the need of a custom FPGA design. 128 Mbit storage capacity is sufficient to hold full configuration files for all compatible FPGAs except the biggest one (SX475T). However, if this FPGA would be chosen there is still the option to use a compressed configuration file format or implement a two-stage configuration process.

In order to enable more than one configuration image to be stored in the on-board memory, a second 128 MBit PlatformFlashXL instance is added in parallel. An artificially introduced most significant address bit on top of the address lines going to both flash chips selects one or the other chip. Apart from the chip-enable signals, all data and control signals are shared between both flash memory chips. The individual chip-enable signals make sure that only one flash memory chip is active at a time. A sketch of the arrangement of both flash chips towards the FPGA is shown in Figure 2.10. This enables to always keep a "known-to-be-working" firmware image on the board while updating the primary firmware. The source memory chip to be used for initial power-on configuration can be selected by a jumper. The flash chips are accessible from the user firmware after the FPGA is programmed. This enables to update the firmware via PCI-Express. The methods to ensure a safe upgrade procedure are described in detail in Section 2.2.7.

A measurement of the actual configuration time of the C-RORC with the LX240T FPGA is shown in Figure 2.11. The black curve shows the 12 V input power supply level. The values from this rail are scaled down by a factor of two for readability reasons. The yellow curve shows the 2.5 V supply in the same way as shown in the power-on sequencing in Figure 2.9. *INIT_B* drawn in blue is an FPGA configuration signal that indicates the start of the actual configuration process.

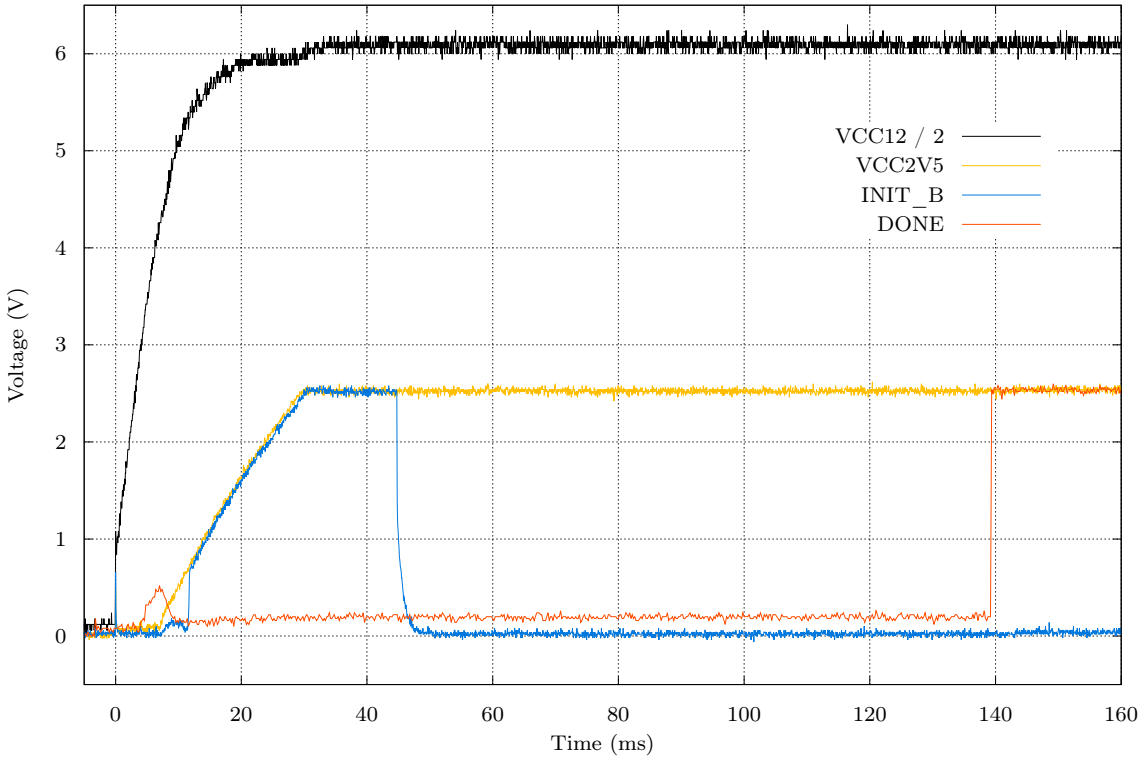

Figure 2.11: C-RORC time measurement from power-on to FPGA configuration done.

This signal is driven low at around 45 ms. The flash chips are enabled as soon as *INIT_B* goes low and start to push data into the FPGA. The red curve shows the level of the FPGA *DONE* pin, indicating the end of the configuration process and the release of the internal global reset at around 140 ms. The actual FPGA configuration process, therefore, takes a little bit less than 100 ms. Together with the ramp-up times of the different voltage rails, the FPGA is in any case running before the 200 ms limit described above. The same is true for any other but the biggest FPGA from Table 2.5. However, this one would require different tricks anyway. The chosen FPGA configuration solution for the C-RORC proved to fulfill the timing requirements of the PCI-Express standard. The FPGA is in any case configured and running before the PCI-Express root port expects feedback from the device.

2.2.7 Fail Safe Configuration Manager

The C-RORC FPGA is unconfigured when the device is powered on. A configuration file is automatically loaded into the FPGA from an on-board storage. In the case of a final and stable firmware image available at the board production time, this image could be programmed once into the on-board storage of all boards using a programming cable. However, as an FPGA firmware is rarely final at the time of the initial hardware deployment and one of the key device features is its reconfigurability, a way to conveniently upgrade the firmware image is required. The configuration can be changed at any time with a programming cable. However, this is typically only suitable for single boards or small lab setups. A larger-scale firmware upgrade process without the need of a programming cable can be achieved if each firmware contains the corresponding interfaces to load a new configuration file into the on-board storage via PCI-Express. Once a new configuration was written to the on-board storage, it will automatically be loaded after the next power cycle. Any following firmware revisions can be flashed using the interfaces of the previous version. This is how upgrade procedures are foreseen for typical commercial FPGA boards. However, this process makes three important assumptions:

1. It assumes an error-free configuration process. This needs a reliable configuration data transport into the on-board storage and from the storage to the FPGA on power-on. This is usually not an issue for well-tested boards.
2. It assumes thoroughly tested firmware images. The new firmware has to provide at least a fully working PCI-Express interface and access to the on-board storage to load the next configuration. This can be an issue for early developments or for time-critical fixes.
3. It assumes no human errors. Cluster administration tools provide means to configure all nodes of a cluster from a central point with only little effort. Flashing a wrong file into a number or even all FPGAs of a computing cluster can be a single typo. This can be a serious issue in high-workload or time-pressure situations.

Whenever a configuration process fails, when the new firmware image does not work as expected, or when the user accidentally flashes a wrong file, the path to switch to another firmware image may be unavailable. If a faulty firmware is loaded, FPGA boards typically have to be accessed with a programming cable to temporarily load a working image or to directly correct the image in the on-board storage. Accessing a PCI-Express board in a server machine with a programming cable requires to dismount the hosting server from the rack and to power it on a test stand with open chassis. If a faulty firmware is flashed into dozens or even hundreds of FPGA boards installed in server PCs, this can become a very costly operational intervention of the whole cluster.

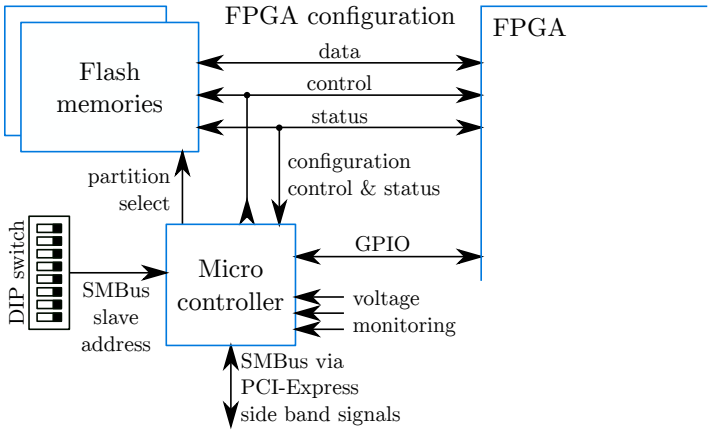

Figure 2.12: Schematic representation of the configuration manager interfaces.

For this reason, the C-RORC is equipped with a configuration manager and additional on-board storage for at least a second "known-to-be-working" firmware image. This concept provides monitoring of the configuration process itself and enables to re-trigger the FPGA configuration process with another firmware image. A secondary communication path to the board enables to control this process from the host machine even if the PCI-Express interface is down. This is realized with a microcontroller. An overview of this system is shown in Figure 2.12. The status and the control signals of the FPGA configuration logic are additionally connected to this microcontroller. This enables to select the configuration data source in the on-board storage and to trigger a reconfiguration of the FPGA. The Two-Wire-Interface (TWI) of the microcontroller implements an SMBus endpoint using the corresponding sideband signals via the PCI-Express connector as described in Section 2.2.2. A dip switch defines the SMBus slave address. A general-purpose interface between the FPGA and the microcontroller enables communication between both. This makes it possible to set up a watchdog implementation in the microcontroller that triggers a reconfiguration if a specific FPGA configuration condition is not reached within a given amount of time. The integrated analog-to-digital converters in the microcontroller additionally provide a coarse-grained monitoring of multiple supply voltages on the board. The firmware of the microcontroller itself can be programmed from the FPGA or with a dedicated programming cable. The microcontroller configuration is non-volatile and needs to be programmed only once after the board production. Another benefit of this approach is that the host machine does not

need to do a power cycle to load a new firmware image. The microcontroller can trigger the reconfiguration while the host machine is running.

The two flash memory chips are divided into four regions in total using the two most significant address bits. The highest bit selects one or the other flash chip and the second highest bit selects the higher or lower half of the corresponding memory chip. The memory region selected with these two bits is used as the data source for the FPGA configuration process. There are different levels of source selection priorities. The FPGA is the only component that can directly drive these two address bits. This can be done by enabling a chained configuration or using the revision-select features during the FPGA configuration file creation. The second access priority is the microcontroller. It can select whether pull-up or pull-down resistors are active on these signals. This assignment controls the address line levels if the FPGA does not actively drive the signals. The third level is the default pull-up or pull-down setting being applied if neither the FPGA nor the microcontroller are driving the signals. This is controlled with jumpers. This third level jumps in whenever the board is powered on because the FPGA is still unconfigured and the microcontroller itself needs several milliseconds until it is running while the FPGA configuration process may already have started. If the board is already powered on, the microcontroller can trigger a reconfiguration of the FPGA. By overriding the default pull-up/pull-down settings set via the jumpers, a source partition is selected from the microcontroller. After pulling an FPGA reset, the configuration process starts from the selected partition.

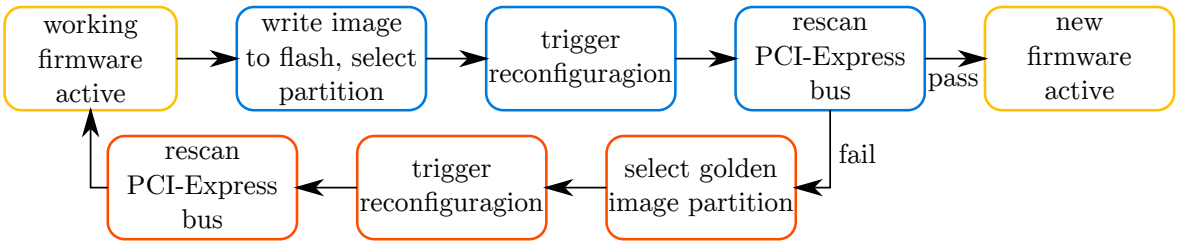

Figure 2.13: State diagram of the firmware update process.

With this controller in place, a firmware upgrade process with a fall-back solution to a working firmware image can be realized even if an upgrade fails. Figure 2.13 shows a state diagram of the firmware upgrade process. The procedure starts by writing a new firmware image into one of the unused flash memory partitions via PCI-Express while running a firmware image with at least a working PCI-Express and flash interface. By communicating with the microcontroller, the newly filled flash partition can be selected for the next reconfiguration. The host machine triggers the FPGA reconfiguration by sending the appropriate commands to the microcontroller. The FPGA is reset during the reconfiguration process. This means that also its PCI-Express endpoint temporarily goes down and the corresponding device gets marked as unavailable in the host system. A PCI-Express communication with the FPGA is not possible at this stage. The microcontroller provides status information on the configuration state of the FPGA. Once the configuration process is completed, the host rescans the PCI-Express bus to re-initialize the link. This re-enables the PCI-Express communication with the FPGA. If the new firmware works properly, the firmware update was successful. If anything went wrong, if the new firmware does not behave as expected, or if the PCI-Express link does not come up again, the upgrade process can be reverted. By selecting the flash partition of the known-to-be-working firmware image and triggering another FPGA reconfiguration, a working firmware is loaded into the FPGA. After another bus rescan, this firmware is accessible from the host machine and the upgrade process can be restarted with another image.

The triggering of the microcontroller actions and the verification of the functionality of the new firmware may happen completely inside the microcontroller. This could be done based on timeouts, watchdogs, and communication with the FPGA, or by accessing the microcontroller from host machine software.

The combination of a multi-file on-board storage and the PCI-Express sideband access to a configuration controller has proven to work reliably on the C-RORC. The fine-grained software control on the FPGA configuration process enables a firmware upgrade methodology with fall-back solution that can be integrated into the cluster management tools. The possibility to change the firmware configuration while running additionally provides quick turnaround cycles for tests.

2.2.8 Miscellaneous Features

The synchronization of the event information between the DAQ system and the trigger system across the detectors using the Readout Control Unit (PHOS, EMCAL, FMD and most importantly the TPC) is realized with the Busy Box. A communication link between the DAQ RORCs and the Busy Box using a custom data protocol transmits event identifiers. The Busy Box checks whether all RORCs have received the corresponding fragments of an event and pauses the trigger if not. The interface between the RORCs and the Busy Box is required to integrate the C-RORC into the existing trigger and DAQ readout system. Its implementation on the C-RORC is based on the previous D-RORC solution. It uses an LVDS buffer to protect the FPGA IO pins and an 8p8c modular connector for twisted pair cables, also commonly known as "RJ-45" or "Ethernet jack". The LVDS buffer is a newer generation of the same kind of component that was used on the D-RORC. It can run at higher speeds compared to the previous one. The main difference to the D-RORC is the number of LVDS connections. The D-RORC used two bi-directional differential pairs via one 8p8c connector towards the Busy Box and two DDLs. An obvious choice on the D-RORC was to use one LVDS pair per DDL. With the C-RORC handling six DDLs in the DAQ application, the same approach would require three 8p8c sockets. The possible solutions, in this case, were to switch to a denser connector that fits into the C-RORC IO-bracket and to change the cabling between the DAQ system and the Busy Box, or to adjust the firmware to allow multiple DDLs to share the same LVDS connection and keep the existing cabling. Fortunately, the firmware adjustments were possible. The C-RORC has one 8p8c connector for the LVDS connection towards the Busy Box. It uses two pairs of LVDS signals to handle the event synchronization for six DDLs.

A number of general-purpose input/output pins (GPIOs) are available via an FPGA Mezzanine Connector (FMC). The low-pin-count variant of this connector as used on the C-RORC provides 34 differential or 68 single-ended user-defined signals. The FMC enables to connect a wide spectrum of commercial or custom daughter boards to the C-RORC, opening up the range of possible interfaces to the board. One limitation is, however, that the fast serial link foreseen by the FMC standard cannot be used with the C-RORC, because all serial transceivers of the FPGA are already taken for PCI-Express and the 12 DDLs. A particular application that was considered when designing the board was the ability to connect a clock jitter cleaning device via FMC and provide the recovered clock as a reference clock to the serial optical link transceivers. This was mostly motivated by the idea to use the C-RORC also as a prototype for the CBM First Level Event Selector Interface Board (FLIB). The FMC daughter boards can optionally be configured with the same JTAG chain as the FPGA. Some additional GPIOs are available via 2.54 mm pin headers. They are meant as an easy method to connect a cable for a serial port, a simple board-to-board interconnect, or for debugging.

Light emitting diodes (LEDs) as status indicators are available at various places on the board and can provide some information about the board state at a glance. The status of the power distribution network and the FPGA configuration is visible if the board is operated outside a server or in a server with open chassis. A user-configurable bi-color LED in the IO bracket provides the possibility to spot a certain board within a whole rack of servers. The light from two LEDs behind each QSFP is directed to the IO-bracket via light pipes and can be used to encode the status of the serial optical links.

A heat sink with a fan is attached to the FPGA. This is required to provide an appropriate cooling of the FPGA in a range of environmental temperature and air flow conditions. The heat sink has a low height in order to comply with the maximum component height for single-slot PCI-

Express devices. In environments with only a weak air flow like desktop PCs or when operating the board outside a PC, the fan is necessary to move the hot air away from the FPGA. In high-performance server chassis with a strong air flow, the heat sink alone is usually sufficient and the fan on top is typically not be needed. A strong air flow orthogonal to the FPGA fan may even impose mechanical stress on the fan, reducing its mean time between failure. For this reason, the fan on the C-RORC FPGA can be controlled by firmware. A low-side transistor switch controlled via a regular FPGA output signal can switch the fan on and off. By providing a pulse-width modulated output on this signal, the rotation speed of the fan can be controlled. The fan itself comes with a tachometer signal that indicates the current rotation speed. With the help of the FPGA-internal temperature sensor, this system can be used to implement an autonomous FPGA temperature control circuit.

An SD-Card slot was added to the C-RORC because it can provide some valuable features and is cheap in terms of the number of signals, the PCB area, and the component costs. It can be used to provide large non-volatile on-board storage or easy access to per-board settings. If a soft-core processor is implemented in the FPGA, a root file system on an SD-Card would be a natural choice.

There are five independent I^2C chains available for status and control communication of the active components on the board controlled by the FPGA. The frequency of the transceiver reference clock is configured via I^2C. Both DDR3 SO-DIMM sockets share a common chain where each slot is assigned a fixed address range. This is particularly helpful to read out the DDR3 timing parameters from unknown or re-branded SO-DIMM modules without a detailed datasheet available. All three QSFP modules share another chain. The individual modules are selected with sideband signals. Another chain is dedicated to the FMC interface. The fifth chain is connecting the SMBus signals coming via the PCI-Express connector also to the FPGA. It allows also the FPGA to act as an SMBus device. This is the only chain that can also be used in slave mode. All others are meant to be controlled by one or more I^2C master cores implemented in the FPGA.

Hardware status and diagnostic information are available from various sources. The FPGA already brings a sensor for its die temperature that is accessible from the firmware. The QSFP modules provide temperature information via their I^2C interface. Temperature readings from the DDR3 SO-DIMMs are also available via I^2C if the installed module comes with an appropriate sensor. The different supply voltages on the board can be monitored with built-in analog-to-digital converters by the FPGA (core and auxiliary voltage) and the microcontroller (most of the remaining supply voltages). While these monitors are too slow to capture spikes or operational glitches, they still provide valuable information on the general status of the power distribution network and can detect static voltage mismatches. An identification and distinction of different boards and setups can be realized with unique device IDs that are pre-programmed into each FPGA, flash memory chip and QSFP module already by the vendors.

2.3 From Concept to Large-Scale Production

The collection of the requirements, the concept, and the schematic board design were done as part of this work. The PCB layout and the first prototypes were produced by Cerntech Ltd.[3], Budapest. The series production was conducted via an international tender. The time it took to transform the concept via a working prototype into a large-scale hardware platform to be installed into the experiment's production systems made up a considerable share of this work.

[3] http://www.cerntech.hu

2.3.1 Custom Board Design Challenges

Designing custom FPGA boards was a common process in the previous years, not only for physics experiments. The market for commercial boards was small and the applications for FPGA boards were specialized anyway. The chances to find a suitable hardware for a "unique" application were low. A custom board could provide exactly what was required for a given application. The complexity of the boards and the FPGA technology was at a level that was manageable with individual developers or small teams. The data rates and the clock frequencies of signals between the FPGA and its peripherals were low enough so that rule-of-thumb electronic design approaches and a bit of experience provided good results.

With increasing data rates, clock frequencies, and FPGA capacities, this approach gets harder. Recent FPGAs come with several thousand pins that need to be connected properly. More and more functionality in the components yields to more and more pages of datasheets, user guides, and reference designs that need to be read and understood in order to design a custom board. Additionally, the documentation of announced or engineering sample components changes over time until the device is finally released and thoroughly tested. Documentation changes have to be monitored closely. The increasing functionality leads to more and more components that require a configuration. Developing a firmware for a power distribution network controller in order to simply power-up a board is becoming common practice. The internal trace lengths between the FPGA IO buffers and the PCB pads start to play a significant role for the signal integrity at high signal rates and require trace-length matching on the PCB. High signal rates with densely packed PCBs require signal integrity simulations. Power regulators come with their own calculation tools to configure them optimally for a given load capacity and inductance. This again depends strongly on the types of capacitors that are used. Needing more than ten different supply voltages on a board is not uncommon. Complex power distribution networks and increasing currents require power integrity simulations. Increasing the number of signals and power rails on a board increases the number of PCB layers required. With an upper limit on the board thickness given by the PCI-Express connector, increasing the number of layers requires thinner layers, thinner traces, and better dielectric material to maintain the required signal impedance. Signal and power integrity simulations require knowledge, training, and the budget to afford the simulation tool licenses.

The time required to work through all details in order to compile a schematic design, develop a PCB layout, and produce boards clearly scales with the amount and expertise of the available human resources. Projects typically start with the design process as soon as engineering samples of the target FPGA are available. Depending on the available resources, the next generation of hardware may be available before the first pieces of a custom hardware are produced. The selected components have to be analyzed carefully to estimate the availability for both the first prototypes and for future production batches over the lifetime of the board. Minimum order quantities can play an important role in component availability if the required amount does not exceed this value. Once the hardware is produced, the high signal rates again make hardware debugging more and more difficult. High-speed signal measurements are only meaningful at the point where they are sampled by the receiving component. Especially for highly integrated components, this point is not always easily accessible with an oscilloscope probe. SDRAM debugging may require in-circuit measurement sockets, PCI-Express debugging may need bus analyzers, and the generally increasing signal rates require measurement equipment with appropriate bandwidth and sample rates. All in all, it gets more and more complex and expensive for individuals or small teams to design custom FPGA boards within a reasonable timescale or budget to debug hardware problems if occurring.

The C-RORC developed in this work is no exception to these challenges. The FPGA contains 1156 IO pins, the PCB hosts a total of 632 items of 113 different components. Fitting 16 PCB layers below the maximum allowed thickness for PCI-Express connectors that can be produced reliably was a tough challenge for the PCB layout and production companies. Thousands of pages of user guides were studied for the schematic design. During the time required for the schematic entry, the PCB layout, and finally the first hardware production, the next generation of FPGAs became

available that would have eased the effort on the firmware development considerably. Switching to the new series of FPGAs, however, would have required to restart the design process more or less from scratch. Optical transceiver components that were part of the technology evaluation process, disappeared from the market before the C-RORC was finally produced. The first active components on the C-RORC hardware were announced to be end-of-life at the end of 2017. The C-RORC hardware proved to be working reliably so luckily no expensive hardware debugging equipment was required.

2.3.2 Firmware Prototyping on Commercial Boards

Board	PCI-Express	Optical Links	DDR3	LVDS	Flash
Xilinx ML605	Gen1 x8 / Gen2 x4	1 (8 via FM-S28)	1x 800 Mbps	no	PFXL
HTG-V6-PCIe	Gen2 x8	2 (8 via FM-S28)	1x 1066 Mbps	no	BPI
C-RORC	Gen2 x8	12	2x 1066 Mbps	yes	PFXL

Table 2.6: Comparison of FPGA boards used for firmware development.

The firmware development was started early already during the hardware requirement analysis phase. The suitability of hardware had to be proven in a realistic environment, not the constructed scenarios of the FPGA feature marketing claims. The firmware developments were conducted in parallel to the hardware design. Two different commercial hardware platforms with add-on boards were used for this process. The evaluation hardware and the differences to the C-RORC are shown in Table 2.6. Initial tests were done with the Xilinx ML605 evaluation board hosting a Virtex-6 FPGA with the same amount of logic resources as the C-RORC FPGA but with a lower speed grade. This board provides a PCI-Express interface with eight lanes at generation-1 or four lanes at generation-2, one SFP port for one serial optical link, and one DDR3 SO-DIMM socket. Dozens of further hardware features are available but not needed for this use case. With the schematics and the board design files available, the ML605 served as a guideline and reference implementation for parts of the C-RORC design. First tests with the PCI-Express endpoint were possible and the DDR3 controller could be evaluated. The board hosts the same flash memory chip for the FPGA configuration as the C-RORC and enabled to study the FPGA power-on timing and behavior. However, both the PCI-Express interface bandwidth and the DDR3 bandwidth are below the maximum values supported by the Virtex-6 FPGAs. The PCI-Express bandwidth on the ML605 is also not enough for the HLT use case. The serial optical link on this board is not compatible with the ALICE DDL modules because an appropriate reference clock to operate the transceiver at 2.125 Gbps is not available. The ML605 was a valuable starting point but not suitable for a realistic hardware qualification on its own.

A second hardware platform used for firmware development is the HTG-V6-PCIe from Hitech-Global[4]. It provides the maximum PCI-Express bandwidth possible with the Virtex-6 FPGA. The board offers the same amount of FPGA resources as the C-RORC and the ML605 but in a bigger package. It has two serial optical links using SFP modules and hosts one DDR3 SO-DIMM socket capable of the maximum DDR3 bandwidth supported by the FPGA. The HTG board contains a different parallel flash memory chip that is larger but also slower than the configuration solution of ML605 and C-RORC.

Neither of the commercial boards is suitable to evaluate parallel DDL connections with its on-board features. However, both boards provide FPGA Mezzanine Connectors (FMC) to extend the I/O capabilities of the boards. An FM-S28 FMC add-on board from FasterTechnology[5] contains two QSFP sockets for eight serial optical links. A configurable reference clock generator provides the option to configure these eight links to 2.125 Gbps as required for the DDL. The add-on board can be used with both the ML605 and the HTG board to implement eight DDLs.

[4] http://www.hitechglobal.com
[5] http://www.fastertechnology.com/

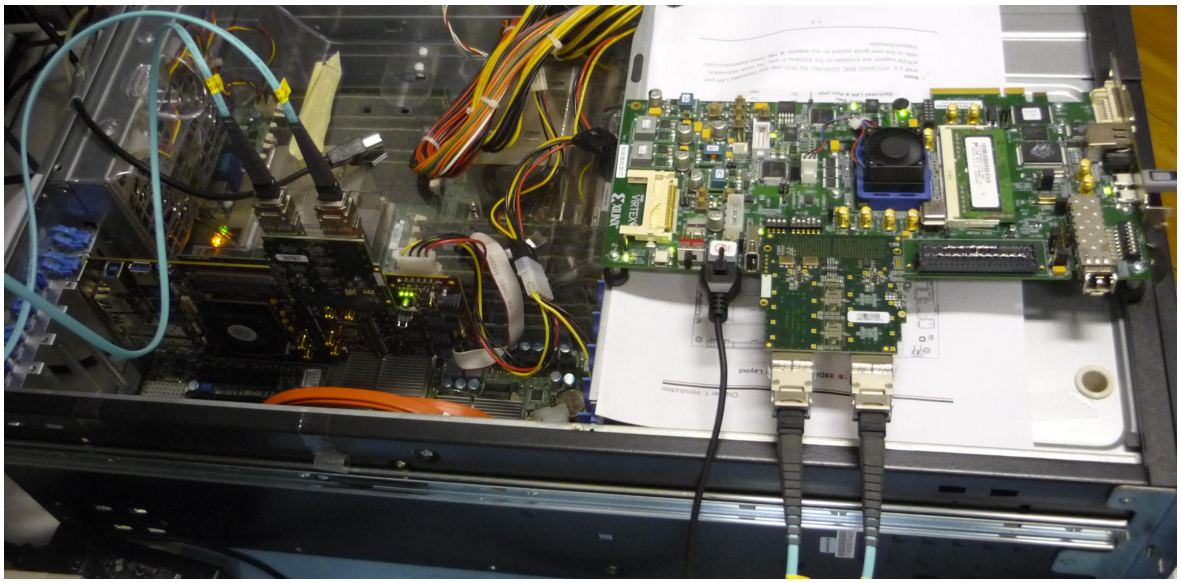

Figure 2.14: Readout prototype setup with the ML605 board (right) as data generator and the HTG board (left) as a RORC installed into a server machine.

This hardware provided the option to both test the parallel DDL connections and to get experiences with QSFP transceiver modules. Optical and electrical QSFP loopback adapters enabled functionality and bit error rate tests of the FPGA transceivers at different link rates. Interconnecting the two QSFP modules with a parallel fiber provided a first real hardware test of the DDL protocol using Virtex-6 FPGAs with QSFP modules. Using two FM-S28 add-on boards enabled to interconnect the HTG and the ML605 board with eight DDLs. The ML605, in this case, was used as data generator and the HTG board as RORC prototype to transfer the generated data to the host RAM via PCI-Express. A photo of this setup is shown in Figure 2.14. The setup with the FMC add-on board clearly exceeded the height of server machine and required it to be operated with open chassis. The SMBus connection via the PCI-Express sideband signals was tested with a standalone microcontroller board. By soldering wires to the PCI-Express SMBus pins of a Riser PCI-Express extension card, these signals could safely be connected to the microcontroller board without tampering with a PCI-Express device or the mainboard.

With the first C-RORC boards available, the HTG platform was used as a reference implementation to confirm the correct behavior of the C-RORCs. The same firmware description code could be used for the C-RORC and the HTG board with slightly different command-line arguments for the firmware synthesis steps. The compatibility with the DDL protocol in a first step was confirmed with the help of an H-RORC. By connecting the C-RORC to the DIU module of an H-RORC a constellation very close to the ALICE setup was achieved.

The software development was started also early in parallel to the firmware development with the commercial hardware. All PCI-Express related operations require a device driver and at least some low-level controls to communicate with the board. All runtime board configuration steps, status queries, as well as data readout operations are triggered by software.

2.3.3 Hardware Prototypes, ATLAS and Mass Production Support

The schematic design of the C-RORC with all components, interconnects, and interfaces was done as a part of this work. The PCB layout process to translate the schematic design into a stack of PCB layers with traces, vias, component footprints, and silkscreen documentation that can be manufactured was done by a company. After the schematic design of the board was finished in early 2011, a contracting process was started for the PCB layout. A number of quotes were requested from companies to estimate the layout costs. An international tender process for

the PCB layout including a detailed technical description of the requested works was prepared together with the DAQ group and the CERN procurement service. The first PCB layout contract that was awarded to the winning company in mid-2011 had to be canceled shortly after because it turned out the company could not fulfill the requirements. The PCB layout for the C-RORC was taken over by CERNTECH, who already produced the D-RORCs and the DDL modules for ALICE in RUN 1. In cooperation with CERNTECH, the decoupling of the power distribution network was refined and a couple of components and hardware aspects were adjusted to ease the PCB layout process. Throughout the layout process, a regular exchange with potential PCB manufacturing companies took place in order to ensure that the chosen PCB layer stack can actually be produced reliably with the given constraints and process variations. CERNTECH also coordinated the production of the first prototype boards. A batch of four C-RORCs was available by the end of 2012 and a second series of 20 boards were ready by mid-2013.

After the first C-RORC boards were available and the test results were positive in all aspects, the ATLAS experiment raised interest in the hardware platform. ATLAS finally decided to use the same hardware for the upgrade of their readout system for RUN 2. The ATLAS requirements are similar to the ALICE constraints and come with roughly the same schedule for hardware procurement, installation, and commissioning. The ATLAS application for the C-RORC hardware is described in more detail in Section 2.4.3. A common large-scale hardware production brought benefits for both parties. With a total number of 380 C-RORCs required for RUN 2 production systems, the ATLAS share of 200 boards more than doubled the original ALICE requirements. This had a considerably positive effect on the price per board for both experiments. Additionally, ATLAS provided knowledge, experience, and manpower for the administrative steps as well as the hardware tests. Only minor changes to the hardware design were done between the prototypes and from the prototypes to the large-scale production. The form factor of the power cable connector was changed to be identical to GPU power connectors. A power switch was not required anymore and was replaced by wire bridges. The model of the heatsink and fan on the FPGA was changed to a lower profile combination to stay well within the height limit for single-slot PCI-Express devices.

The production of the 380 C-RORCs for ALICE and ATLAS was done with the help of the CERN procurement service. A number of administrative steps were required for the hardware procurement processes based on the expected total contract volume. This included a departmental request as well as a market survey followed by an invitation to tender. The market survey was to identify potential contractors within the CERN member states. This process provided a number of documents to interested companies. A technical description defined the requested work items. A technical questionnaire had to be filled by the companies to judge their ability to fulfill the technical requirements. A list of qualification criteria was created to ensure that the company has a sufficient size, expertise, qualification, and certification for the job. The responses from the market survey were jointly analyzed to compile a list of companies to be invited to the actual tender. The invitation to tender again required a couple of contractual and legal documents and included a technical specification as well as design files necessary to place a bid. After the bidding phase, all received bets were compared. A contractor was selected based on the price and the compliance to the specification. The procurement service then awarded the contract. A process like this from the initial contact with the procurement service up to the contract awarding takes four to six months. The preparations for the C-RORC series production market survey were started in early 2013. The contract with the selected production company was awarded in August 2013.

The coordination steps required after the contract is awarded are often underestimated. The contract had foreseen two production steps. In a first step, 20 per-series boards had to be produced and provided to CERN for an acceptance test. Once approved, the same tools, materials and processes should be used for the production of the remaining 360 boards. A detailed power-on checklist, as well as a test bench with a server machine, peripherals, test firmware, and software, were provided to the contractor for on-site hardware tests. This setup is described in more detail in Chapter 2.3.4. The FPGAs were purchased by CERN and provided to the production company. A detailed component and PCB stack-up analysis by the awarded company was only done after

the contract, even though the company signed a full compliance with the CERN-provided stack-up in the tender document. This is a crucial part because each PCB manufacturer guarantees the final dimensions and electrical properties only with certain tolerances. Stack-up variations, as well as alternative stack-up parameters, have an impact on the trace impedances and may need trace width adjustments. This, in return, can affect overall the functionality. As a result, the PCB stack-up negotiations delayed the production of the pre-series boards by a few months. Cerntech, as well as experts from ATLAS, greatly helped with these negotiations. A suitable configuration was found in October 2013. Each manufacturer has a set of preferred component suppliers or suggests to use alternative components depending on order quantities. Additionally, the C-RORC bill of materials was defined almost a year before the actual assembly. Market changes with respect to the availability, the lead times of certain components, as well as the available stock required switching to alternatives for several components. With each hardware change to be approved by CERN, comparing component specifications from different vendors was a tedious task.

The 20 pre-series C-RORCs were available at CERN in January 2014. These boards were thoroughly tested for their general hardware functionality and then distributed between ALICE and ATLAS for application-specific tests. These hardware tests were successful with all test cases and the production of the large batch of boards was approved. The remaining 360 C-RORCs were delivered to CERN in August 2014. The HLT server nodes arrived approximately at the same time so the C-RORCs could be installed into the production system shortly after. The commissioning phase of the HLT production system started in autumn 2014.

2.3.4 Hardware Verification

The C-RORC hardware verification was a crucial part for a large-scale hardware deployment. Developing production grade firmware for known-to-be-working hardware is already hard, operation and development for potentially malfunctioning devices can be a nightmare. The hardware verification process ensures that all components on the board work properly and behave as expected. This is to detect hardware related malfunctions introduced by the PCB production, assembly errors, or from defective components. The firmware development on the commercial boards as a reference implementation for the C-RORC was already described in Section 2.3.2. While the behavior of a single board in a lab setup can be analyzed in detail manually, this approach does not scale to hundreds of boards. A well-defined test plan is required for each board to pass. In order to reduce turnaround times, the hardware production contract had foreseen hardware tests to be conducted already at the contractor site. Boards that fail these tests were not accepted to be sent to CERN. These on-site tests brought strong additional requirements on the test setup. The tests had to be prepared in a way that a person without a deeper understanding of the hardware could conduct them and draw conclusions on the malfunctioning part of the hardware in case of errors. Additionally, remote access to a test system in a corporate network to assist a test engineer in case of problems was highly unlikely. This required a much more careful firmware and software design, as well as a release-grade implementation quality, and went way beyond what would have been done for a local test bench. The aim was to cover a large fraction of functionality checks in only a few and simple to use applications. All this had to be prepared with detailed documentation for each step.

The hardware production contractor was provided with a checklist of tests to be conducted as well as a test system. This test system consisted of a PC, peripherals, firmware, drivers, and software to test each board with easy to use commands. All results had to be documented. The test procedures for the on-site tests at the contractor's premises were developed as part of this work and in collaboration with the ATLAS and ALICE colleagues. Each board was analyzed in three major test steps:

1. A power-on checklist
2. A standalone board test

3. A board-to-board interconnect test

The power-on checklist defined a procedure to sequentially enable and measure the different supply voltages on the board. This was done with the board on a table and connected to a lab power supply. The currents and visual indicators on the boards like LEDs were closely monitored and documented with each step. The clock frequencies of the oscillators on the board were confirmed with an oscilloscope at test points on the PCB. In the last step, the FPGA and one flash memory chip were programmed with an FPGA configuration file using a programming cable. This firmware image contained the hardware verification functionality for the following tests.

- ❏ check PCI-Express link status and BAR configuration
- ❏ check FPGA fan status
- ❏ check status of system clock and transceiver clocks
- ❏ run serial link bit error rate test at default link rate on all links
- ❏ I^2C access and sanity checks to reference clock, QSFP, DDR3, FMC
- ❏ check status of DDR3 initialization and pattern generator / checker
- ❏ FMC pin verification with loopback adapter
- ❏ check status of LVDS link and pattern generator / checker
- ❏ access to microcontroller configuration interface from the FPGA, configuration via FPGA and test access to the microcontroller via SMBus from host software
- ❏ check status and simple access tests on both flash memory chips
- ❏ short DMA-to-host data transfer test

Figure 2.15: List of tests performed by the automated test program for a C-RORC installed in a test server.

The standalone board was is carried out with the board installed into a PC. Peripheral components like DDR3 RAM modules, QSFP modules, fibers, and loopback adapters were installed on the board. All following steps were software-based tests. The test operator had to run only a single test program and all following tests were executed automatically. A brief overview of the steps is shown in Figure 2.15. In a first step, the automatic FPGA configuration process and the state of the PCI-Express link were verified. The FPGA fan and internal clocks were confirmed to be running. All optical links, the transceiver reference clock, and the QSFPs were initialized. The DDR3 module was initialized and an automatic test procedure was started that wrote predefined patterns to the memory and read them back. The connectivity of the FMC interface and the LVDS interface was tested with special loopback adapters. Figure 2.16 shows the FMC adapter. The FMC loopback adapter connects pairs of GPIO signals with 0 Ohm resistors. This adapter was provided by the ATLAS team. It was developed by Andrea Borga at NIKHEF, Netherlands. By iteratively driving one pin at a time from the firmware and reading the state of all other FMC GPIOs, the connectivity of all FMC GPIOs could be confirmed. Reading an I^2C temperature sensor on the loopback adapter ensured the I^2C connection via FMC works as well. The LVDS loopback adapter was a simple Ethernet socket connecting the differential RX and TX link pairs. The firmware checked continuously that the received signal matches the sent pattern. The configuration microcontroller was programmed via the FPGA from software. The flash memory chips were accessed from the software to confirm the option to reprogram the flashes. Reading the unique flash serial numbers confirmed that the flash selection mechanisms worked and both flashes responded correctly. A short DMA device-to-host transfer test made sure that the PCI-Express interface was stable. A pattern transmission test via the optical links at a fixed link rate automatically running in parallel to all previous tests confirmed the basic functionality of the serial optical links.

The last test sequence, the board-to-board interconnect test, required two boards to be installed into the PC. The two boards were interconnected with three QSFPs using all 12 serial optical links. This test verified the optical connectivity at 2.0, 2.125, 4.25 and 5.0 Gbps link speed. Each

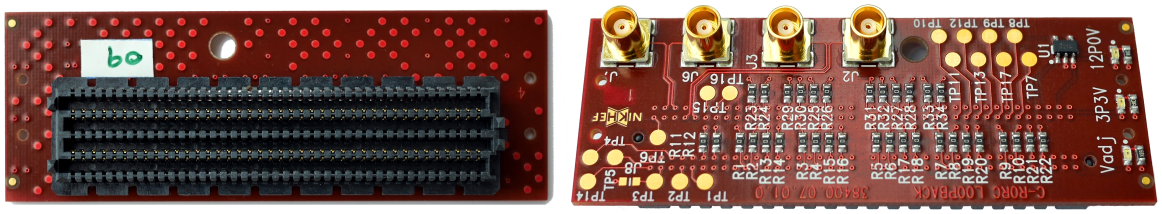

Figure 2.16: Bottom and top view of the FMC loopback adapter for the C-RORC hardware tests developed by ATLAS at NIKHEF.

link rate was tested for a couple of seconds. The firmware-based status signals and error counters were evaluated right after. This test ensured that both the reference clock oscillator and the transceivers in combination with their analog supply voltages worked reliably across different operating conditions.

All C-RORC boards that were produced have passed these tests. Two of the 380 boards initially failed the board-to-board interconnect test. These two boards could be fixed by replacing their FPGA. Up to now, no C-RORC hardware failure is known from any production or development system.

2.4 C-RORC Applications

Even though the board was primarily designed for the two ALICE applications in the DAQ and HLT systems, the board design was kept as generic as possible to encourage the usage of the hardware beyond its initial scope. During its lifecycle, the board found its place in various production and development systems across different experiments.

2.4.1 ALICE Data Acquisition

The C-RORCs in the ALICE DAQ system are used for all detectors that increased their readout link speed compared to Run 1. This is namely the Time Projection Chamber (TPC), the Transition Radiation Detector (TRD) and the Central Trigger Processor (CTP). In addition, the data transfer from the HLT to the DAQ system is also realized with C-RORCs on both sides. A total of 60 C-RORCs are used in the Run 2 DAQ system, replacing more than 300 D-RORCs. For detectors that continue to use the same link speed as in Run 1, the existing PCI-Express D-RORC boards are reused in Run 2. The overall density of the system could greatly be improved in the transition from the D-RORC with one input link and one copy to the HLT to the C-RORC with six detector DDLs and six copies to the HLT. The mixed installation of C-RORC and D-RORC boards in the DAQ system required a seamless integration of the C-RORC into the existing hardware and software infrastructure. From the software side, the same data acquisition software framework was used to control both the C-RORCs and the D-RORCs. The optical connectivity of the C-RORC proved to be compatible with existing and new front-end boards and with the D-RORC for the DAQ-to-HLT interconnect.

The C-RORC firmware and software for the DAQ application was developed independently from the HLT firmware by the DAQ group. To ensure compatibility with the existing D-RORC infrastructure, firmware blocks and software interfaces were reused where possible. The DMA engine of the DAQ firmware has to handle the data from a maximum of six DDLs because the other six DDLs are used for sending a copy of the detector data to the High Level Trigger. For this reason, it was sufficient to use the eight PCI-Express lanes at generation-1 link speed. This already gave a raw bandwidth of 20 Gbps from the device to the host. The number of DMA channels required together with the link speed enabled to use the same commercial DMA endpoint implementation as in the D-RORC firmware. This again simplified the integration on the software side. The DAQ C-RORC firmware uses the LVDS interface to communicate event

flow control information with the TPC Busy Box. The usage of the DDR3 SO-DIMM modules was not foreseen in the DAQ application.

2.4.2 ALICE High Level Trigger

The C-RORC is used in two different dataflow modes in the HLT, replacing all previous 240 H-RORC boards with 74 C-RORCs. The form factor and the interfaces of the C-RORC, together with the computing power of recent server PCs, enabled to drop the separation of nodes into input nodes ("FEPs") and compute nodes ("CNs") from Run 1 but use identical machines throughout the whole cluster. At the data input interface of the HLT, a copy of the experimental data from the detectors is received from the DAQ system with C-RORCs using up to 12 DDLs per board. The remote link partner on the DAQ side is a D-RORC or a C-RORC depending on the detector. The raw data is then transferred from the HLT C-RORC into the host RAM via DMA for processing within the HLT. For the TPC input links, the raw data is pre-processed already inside the FPGA with the cluster finding algorithm sketched in Section 1.4.2 and described in detail in Chapter 4. The on-board memory is used to replay generated or previously recorded detector data as if it were coming via the optical links for testing, verification and characterization purposes. The second C-RORC operating mode is the output interface of the HLT. The HLT-processed and compressed detector data is provided back to the DAQ system using C-RORCs. For this mode, the data is fetched from the host RAM by the C-RORC and sent to the DAQ system via the DDL. The HLT application is described in detail in Chapter 3.

2.4.3 ATLAS Readout System and Region of Interest Builder

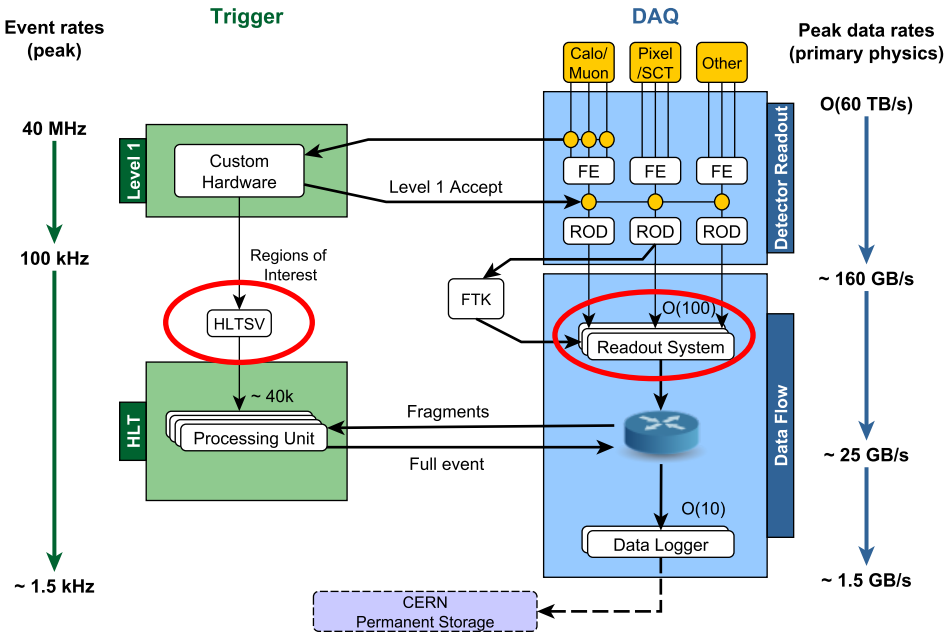

Figure 2.17: ATLAS Run 2 readout architecture. The positions of the C-RORC hardware is indicated with red circles. Image from ATLAS TDAQ ROS Team [PV 17], modified.

The ATLAS experiment consists of several of detector systems that are read out with custom front-end electronics (FE). Parts of this data is fed into a trigger system to make a fast decision on whether to store or discard an event (level-1 trigger). The detector data is transferred into the readout system (ROS) where a High Level Trigger (HLT) further reduces the event rate. Full events are built there and are sent to permanent storage. An overview of the ATLAS readout architecture for Run 2 is shown in Figure 2.17. The ATLAS experiment in Run 1 used around 1600 optical fibers to read out detector data from the readout drivers (RODs) to the

readout system (ROS). These links used the S-Link protocol [vdB+ 97] with a bandwidth of around 160 MB/s per link. The detector data was stored in the buffers of custom PCI-X-based readout boards, the Read-Out Buffer INterface (ROBIN) boards [Kug 09]. The ROBIN features three S-Link interfaces as well as a PCI-X interface to the host machine and 64 MB of on-board storage per optical link. The level-1 trigger decisions were used to construct a region of interest (RoI) of physical detector locations to be read out. Only data matching this region was read out from the ROBINs into the ROS. Detector data that was not part of the triggered region was not transferred to the host machine but discarded already in the ROBIN. The ATLAS Trigger and Data Acquisition (TDAQ) system received an upgrade during LS1 to meet the requirements for Run 2 in terms of enhanced detector capabilities, additional readout links, as well as increased event rates and data sizes. A limiting factor was the amount of spare ROBIN hardware, the rack space required for additional ROS servers, as well as the obsolescence of the PCI-X interface standard in recent server machines. The ATLAS experiment decides to use the C-RORC hardware platform for the upgrade of their readout system. The 12 serial optical links of the C-RORC enabled to reduce the number of servers required to read out all S-Links and increased the density of the readout system by a factor of four. The PCI-Express interface provided an up-to-date host connection compatible with recent and future server hardware. Up to 16 GB of on-board DDR3 RAM enabled to keep the RoI-based readout approach. The main conceptual difference to the original ROBIN implementation is in the firmware and its integration into the host system. The Run 1 ROBIN used a hard-core PowerPC processor in the FPGA that was responsible for the on-board memory management. The C-RORC FPGA does not have this feature. For this reason, this functionality was moved to the host machine. The C-RORC hardware in the ATLAS experiment is called "RobinNP", where "NP" stands for "no processor" for this reason. The application of the C-RORC hardware in the ATLAS readout system is described in detail in [PV 14].

The C-RORC hardware is used in a second application in the ATLAS experiment: the region of interest builder (RoIB). The ATLAS RoIB in Run 1 was a 9 U VMEbus-based crate system consisting of multiple input cards and one builder card to inject the target RoI into the readout system. Despite being a stable and reliable system in Run 1, the VMEbus implementation was lacking flexibility and could not handle the target level-1 trigger rate expected for Run 2. The system was replaced by the C-RORC hardware platform. This simplified the readout architecture and provided the previous functionality with a single PCI-Express board. The existing RobinNP firmware was reused for this use case. The RoIB application is described in detail in [Abb+ 16].

2.4.4 G-RORC - ALICE O^2 Prototype Readout Solution

The third LHC run period, Run 3, is scheduled to start in 2021. The ALICE experiment extends its physics program to exploit the potential of the LHC machine with heavy-ion collisions for studies of Quantum Chromo Dynamics (QCD). High statistics and high-precision measurements are required to achieve these goals. A low signal-to-noise ratio, as well as complex probes, make the existing triggered detector readout concept inefficient. For this reason, the ALICE detectors need to be upgraded for Run 3. A detailed description of the physics program and the detector upgrade plans can be found in [Abe+ 12]. The detectors in the central barrel will implement a trigger-less continuous readout scheme. This greatly increases the data rate to the readout boards. The lack of a trigger also changes the way of synchronization and clock distribution across the different front-end electronics. The readout link replaces the DDL with the Gigabit Transceiver (GBT) project [Mor+ 07]. GBT can provide radiation tolerance as well as a synchronous and deterministic clock distribution via the uplink fiber of the readout link from the readout system to the detectors. GBT operates at a symbol rate of 4.8 Gbps. With the readout bandwidth requirements from the detectors of around 1.1 TB/s, more than 8000 GBT links are planned for ALICE in Run 3. A few detectors will continue to use the DDL protocol with the Run 2 RORC.

A common Online and Offline (O^2) computing cluster will combine the DAQ and HLT online systems with the Offline grid computing infrastructure. This cluster is planned to consist of

around 250 First Level Processor (FLP) nodes that receive the experimental data from the detectors via the GBTs. Around 1500 Event Processing Nodes (EPNs) are foreseen to perform event building, event reconstruction, compression, and quality assurance before the data are written to the permanent storage system. The upgrade of the O^2 system is described in detail in the O^2 technical design report [Bun+ 15]. The Common Readout Unit (CRU) will be the successor of the C-RORC for all detectors that use the GBT protocol in Run 3. The CRU hardware, firmware, and software infrastructure that will form the detector readout solution in the common ALICE O^2 system.

In the preparation of the ALICE upgrade for Run 3, various detector teams characterize and develop their detectors, front-end electronics, and readout chains with beam tests and lab setups. These works have to be conducted already in parallel with the development of the CRU and the O^2 system. This also means that the CRU infrastructure itself is not immediately available for the detector tests and an intermediate solution is required. The C-RORC hardware is widely available and is able to implement the Run 3 readout link protocol GBT at 4.8 Gbps. For this reason, the DAQ group developed a prototype readout system based on the C-RORC hardware and the existing DAQ firmware and software stack: the G-RORC [Cos+ 16].

The G-RORC firmware implementation reuses large parts of the DAQ C-RORC firmware as used in Run 2, replacing the DDL interface with a GBT FPGA implementation. With only the optical link protocol changed, major parts of the DAQ readout software infrastructure are immediately reusable as well. The G-RORC firmware supports up to eight GBT links and uses the eight PCI-Express lanes at the generation-1 link speed. A continuous readout of all eight GBT links is not possible because the GBT bandwidth of the eight links exceeds the PCI-Express bandwidth of the G-RORC firmware. However, on-board FIFO buffers enable to capture the data from all eight links in a triggered or gated readout mode.

A limitation of the C-RORC hardware for the GBT readout is the lack of clock synchronization support across devices. The deterministic-latency mode of the GBT can only reliably be implemented if no more than one C-RORC is used. in addition, this board has to provide the reference clock to the front-end electronics. A synchronization of a readout systems with more than one C-RORC or with an external reference clock would require additional hardware. This was not necessary until now.

A Detector Data Generator (DDG) firmware provides arbitrarily generated detector data for tests of the readout chain. It also contains eight GBT links that are fed from individual firmware pattern generators. The C-RORC hardware with the DDG firmware can be operated in a standalone mode without installing the card into a server machine.

2.4.5 T-RORC - ALICE TPC Beam Test and SAMPA Mass Test System

The TPC RORC (T-RORC) is a fork of the HLT Run 2 C-RORC firmware to implement GBT readout similar to the G-RORC shown in Section 2.4.4. The T-RORC firmware provides 12 GBT links with deterministic link latency in its receive path. Based on the HLT firmware, it comes with the full PCI-Express bandwidth supported by the hardware. The existing HLT device driver and the low-level device access library are reused with a slim beam test readout software. A major difference to the G-RORC is the TPC specific GBT data decoding implemented already in firmware and a software-controlled GBT slow-control adapter (SCA) interface. The firmware and software infrastructure is tailored towards the readout of the TPC front-end cards (FECs) for Run 3. The firmware and software development for this setup was supported as a side project as part of this work. The DDL implementation in the firmware was replaced with the GBT FPGA core. The GBT clocking architecture was adjusted to support all 12 optical links, including the deterministic link latency and the GBT wide-bus mode. Firmware and Software were developed in strong collaboration with TPC electronics team. Stefan Kirsch developed and integrated the GBT SCA core and provided large parts of the beam test software. Sebastian Klewin developed and integrated the TPC data decoding in firmware. This code was primarily developed for the TPC user logic in the CRU and could be tested with this system for the first time in hardware.

Torsten Alt, Lars Bratrud and Christian Lippmann furthermore took care of system testing, integration and debugging.

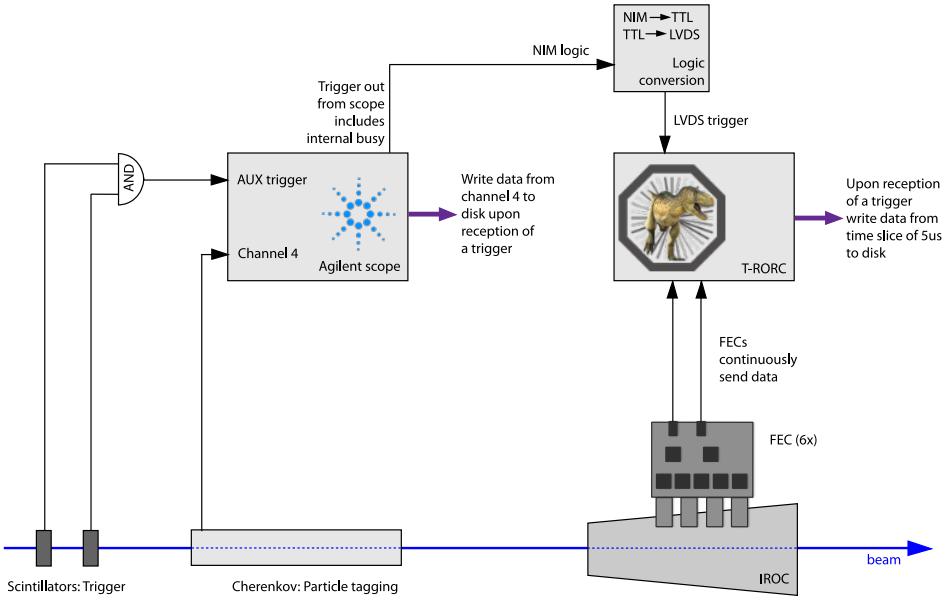

Figure 2.18: T-RORC setup from the TPC beam test in May 2017. Six FECs with a total of 12 GBTs are read out with one T-RORC. Image: Christian Lippmann, used with permission.

The T-RORC firmware and software infrastructure was used in the beam test to characterize a first TPC Inner Readout Chamber (IROC) for Run 3 in May 2017. A schematic of the beam test setup is shown in Figure 2.18. Six FECs with a total of 12 GBT links were installed on an IROC and read out with one T-RORC. With the overall GBT bandwidth exceeding the PCI-Express bandwidth, a triggered readout mode was used. The C-RORC LVDS inputs were used to trigger the readout process from all 12 GBTs. Scintillators in combination with a Cherenkov-based particle tagging and recording mechanism realized with an oscilloscope provided the trigger signals to the readout system. Upon a trigger, 5 µs of continuous detector data were read out from the 12 GBTs via the T-RORC and stored to disk. The recorded data was then analyzed by the TPC Offline experts for detector physics performance aspects.

The same firmware and software are also planned to be used for the mass-testing of the TPC SAMPA chips in Lund, Sweden. The SAMPAs are the custom ASIC chips for the upgrade of the TPC front-end electronics for Run 3 [Bar+ 16]. Five SAMPAs are mounted on each FEC.

Another application of the T-RORC firmware and software is the evaluation of a soft-error mitigation concept for FPGAs in radiation environments using the GBT SCA [Oan+ 15]. Communication with a front-end electronics board containing a GBT SCA is immediately available without firmware development efforts using the T-RORC environment.

2.4.6 Other Experiments

The NA61 experiment [Abg+ 14] at CERN will reuse the ALICE TPC hardware [Lá+ 15] after Run 2. With the C-RORC-based RCU2 readout system available from the ALICE production systems, the NA61 team will also reuse C-RORCs as well as ALICE firmware and software for their detector readout.

The PANDA experiment [Bri+ 06] at FAIR, Darmstadt, is evaluating a PCI-Express-based detector readout option [Rei+ 17]. The developments are based on the C-RORC hardware together with a fork of the HLT firmware and low-level software stack, adjusted to read out 12 optical links at 1 Gbps per link.

2.4.7 Discussion of other possible Applications for the Hardware

During the hardware development phase, the C-RORC was considered to be used also as a **FLES Interface Board (FLIB) prototype** [Keb+ 13] for the CBM experiment at FAIR, Darmstadt. At that time, the serial link protocol for the CBM readout was a custom development relying on a deterministic link latency. In order to achieve that, a dedicated jitter-cleaning circuitry to recover an external reference clock was required. The C-RORC does not bring this functionality by default and does not need it for its main purposes. However, a reference clock input from the FMC interface is available for this use case. The jitter cleaner for the external reference clock could be provided with the help of a custom FMC add-on board. With changes and new developments in the CBM readout plans, the target link speed on the FLIB moved towards 10 Gbps and the deterministic latency was not necessary anymore on the FLIB. The CBM experiment finally switched to commercial prototyping hardware based on Xilinx 7-Series FPGAs [Hut+ 17]. Nevertheless, the FLIB firmware was originally forked from an early HLT C-RORC implementation and the same kernel module is used for both applications.

The C-RORC can directly be used as a **co-processor accelerator**. The PCI-Express interface provides a bi-directional bandwidth of around 3.5 GB/s to move data to or from the device. The hardware provides two DDR3 memory interfaces with a total sequential read bandwidth of above 16 GB/s or up to 16 GB of storage if needed. The DDR3 performance is evaluated in more detail in Section 3.7. The pluggable QSFP modules do not need to be installed if the optical links are not required and, therefore, save invest costs when unused. With a bi-directional optical link bandwidth of around $(12 \cdot 6)$ Gbps $= 72$ Gbps, even a multi-board interconnect is possible. A co-processor application actually developed for the C-RORC hardware is a cluster finder evaluation implementation described in detail in Section 4.2.2.

The serial optical links enable the C-RORC to be used as a **network interface card**. The links support 1000BASE-SX with QSFPs using 850nm lasers for multi-mode fibers. Four Media Access Control (MAC) hard-blocks in the FPGA can implement four 1 Gbps Ethernet links. More interfaces require additional MAC implementations in the FPGA fabric. 10 Gbps Ethernet is not possible with the C-RORC.

The C-RORC hardware is also suitable as a **general-purpose FPGA board**. With the QSFP and DDR3 modules pluggable, only peripherals that are really required have to be purchased. The power supply via external cable instead of via the PCI-Express connector enables to operate the hardware also as a standalone device outside a server machine. The FMC connector provides additional functionality or interfaces with various commercial or custom add-on boards. An SD-Card slot offers large non-volatile storage that would be necessary, for example, for a Linux operating system running on a soft-core CPU in the FPGA.

2.4.8 Additional Firmware and Hardware Feature Reuse

Apart from the reuse of the C-RORC hardware as described above, various parts of firmware modules and hardware concepts developed as part of this work also found their place in applications with different hardware platforms.

The DMA engine firmware developed for the HLT is also in use on Xilinx Virtex-6 and 7-Series FPGAs. The fluorescence correlation spectroscopy readout chain of Jan Buchholz [Buc 16] uses a very early version of the DMA engine and the software stack with the Xilinx ML605 evaluation board. A more advanced DMA firmware version was ported to the Xilinx Kintex-7 FPGA by Dirk Hutter for the readout of the CBM FLIB prototype [Hut+ 17]. A similar firmware port in combination with the HLT software stack is used with a Virtex-7 FPGA on a Xilinx VC707 evaluation board. This application uses Software Defined Radio (SDR) techniques in the context of the readout and stimulus of a microwave multiplexed thermal sensor as part of the work from Panos Neroutsos [Ner+ 17].

A unique C-RORC hardware feature that made it into another hardware platform is the usage of the PCI-Express SMBus sideband signals for the FPGA configuration control. It is now part of the ATLAS Front-End Link eXchange (FELIX) board [And$^+$ 16]. This board is based on a Xilinx Kintex UltraScale FPGA, supports 24 GBTs, and will be used in ATLAS for Run 3.

Chapter 3

HLT C-RORC Implementation

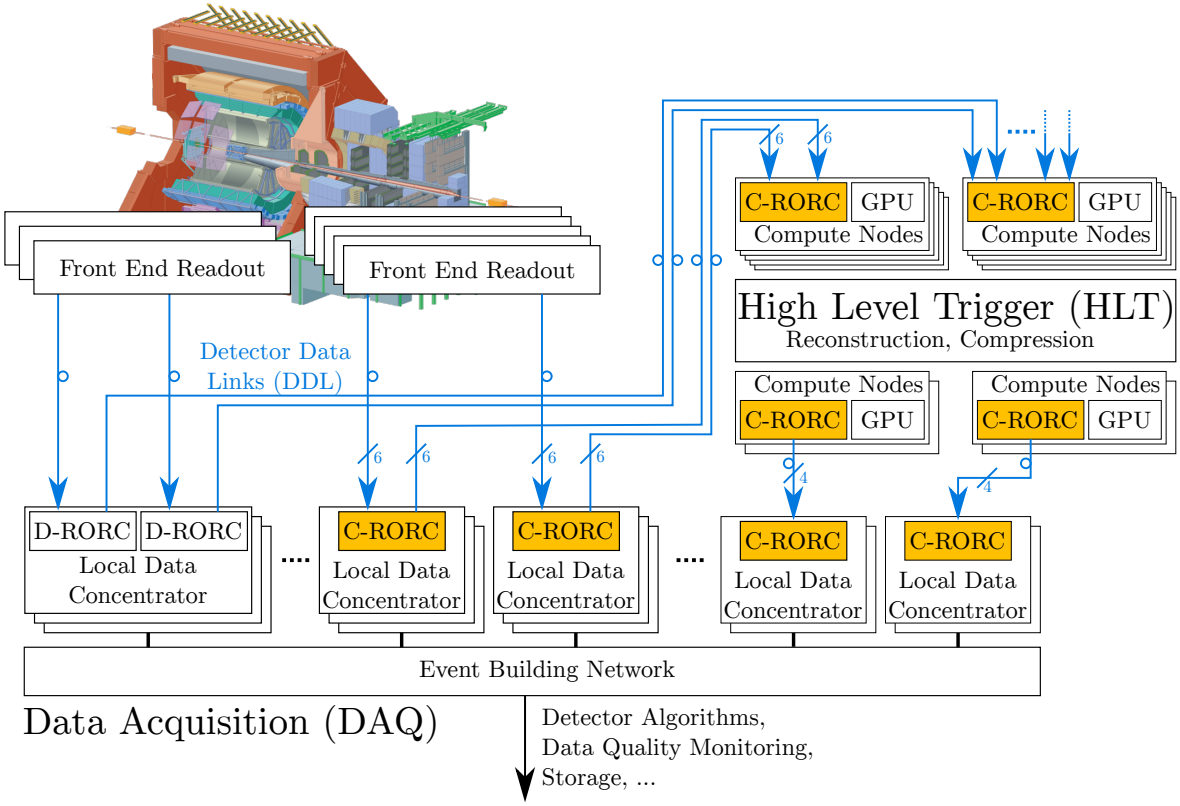

Figure 3.1: ALICE online readout architecture in LHC Run 2.

With the start of LHC Run 2, all previous Read-Out Receiver Cards in the HLT were replaced with C-RORCs. The DAQ system replaced all RORCs connecting to TPC, TRD and HLT DDLs with C-RORCs. For the remaining detector links the DAQ group continued to use the PCI-Express version of the Run 1 D-RORC boards. An overview of the Run 2 readout architecture with respect to the Read-Out Receiver Cards is shown in Figure 3.1. With the increased link density of up to 12 DDLs per board, one C-RORC is replacing up to six D-RORCs or H-RORCs. A configurable signal rate on the C-RORC enables the detectors to upgrade their front-end electronics towards higher link rates. Table 3.1 shows an overview of the distribution of detector links between the DAQ and the HLT system to the different HLT C-RORCs and their link speed for Run 2. The links with increased link rates are highlighted in bold.

Detector	#DDLs	#C-RORCs	DDL rate (Gbps)	Bandwidth (Gbps)
ITS-SPD	20	2	2.125	42.5
ITS-SDD	24	2	2.125	51.0
ITS-SSD	16	2	2.125	34.0
TPC	216	36	**3.125**	675.0
TRD	36	6	**4.000**	144.0
TOF	72	6	2.125	153.0
HMPID, PMD	20	2	2.125	42.5
PHOS, CPV	31	3	2.125	65.9
MUTRK, MUTRG	22	2	2.125	46.8
DCAL, EMCAL	41	4	2.125	87.1
FMD, T0, V0, AD, ZDC, TRG, ACO	9	1	2.125	19.1
HLT-OUT	28	8	**5.3125**	148.8
Sum	535	74		1361 in, 149 out

Table 3.1: Distribution of the ALICE HLT DDLs across the different detectors.

3.1 High Level Trigger in Run 2

The hardware of the HLT was completely replaced after Run 1. The HLT in Run 2 consists of 180 identical compute nodes plus around ten infrastructure machine. They are all interconnected with InfiniBand FDR (56 Gbps) for data transport and Gigabit Ethernet for cluster management. The server mainboards have two CPU sockets each and are equipped with Intel Xeon E5-2697 v2 CPUs with 12 cores per socket. Each server node has 128 GB of DDR3 RAM and hosts an Amd ATI FirePro S1000 GPU. All machines are of the same type. This makes a homogeneous system and greatly simplifies the cluster management. Another benefit is that the differentiation between input node, compute node, and output node with different hardware configurations and form factors like in Run 1 is not required anymore. All nodes can equally take all processing tasks. The servers acting as HLT input or output nodes are additionally equipped with a C-RORC, but also contain a GPU and can be used for reconstruction tasks like any other machine. The server chassis provides two separate trays of PCI-Express slots with independent fans and air flow channels. All PCI-Express slots in one tray are connected to the same CPU socket. For thermal, performance, and maintenance reasons, the C-RORC is installed into the tray connected to the second socket, while the GPU is installed in the tray for the first one. This reduces a possible bus congestion between C-RORC and GPU to a minimum. The setup provides individual thermal control via the fans and enables independent maintenance operations if necessary. The GPUs were already installed into the machines by the server distributor, the C-RORCs were installed into 74 machines in the context of this work. A photo of the cluster and this setup is shown in Figure 3.2.

All machines are distributed across 15+1 racks with 12 machines per rack and three racks sharing one InfiniBand switch. The DDLs are available from patch panels at the top of each rack. The distribution of DDLs across the racks, as well as the distribution of DDLs across the different C-RORCs with a strong focus on the capabilities of the HLT data transport framework was developed by Timo Breitner. Even though the C-RORC supports 12 DDLs per board, this dense configuration is not always the best performing solution. Critical parts of the HLT data transport framework are the merger components which combine sub-events from different DDLs for further processing. All data from one event on one C-RORC are merged already locally on the host machine before being published to the next stage. The higher the number of merger inputs, the lower the maximum event rate this merger component can handle. Another aspect is the network load: a solution with a moderate load on multiple network interconnects can provide a more reliable operation than a constantly high load on only a few. For these reasons, the number of DDL input streams per board is reduced to six links for the TPC and the TRD C-RORCs.

Figure 3.2: Left: Photo of the ALICE HLT at start of Run 2. Right: interior of a node. The GPU is installed in the left tray, the C-RORC in red on the right.

The DDLs are distributed in a way that all links from the same TPC sector are read out from the same C-RORC and similar metrics are applied to the TRD geometry. The distribution of the 20 SPD links across C-RORCs as 10+10 instead of 12+8 proved to be more efficient even though it required more and partially unused QSFP transceivers on the C-RORCs. The network aspect is the reason why the HLT output links towards DAQ use only four of the 12 available DDLs per C-RORC. Detectors with only a few DDLs share the same C-RORC if their link speed is identical. A key point here is that exploiting the full capabilities of one component in the system does not necessarily provide the maximum performance for the overall system.

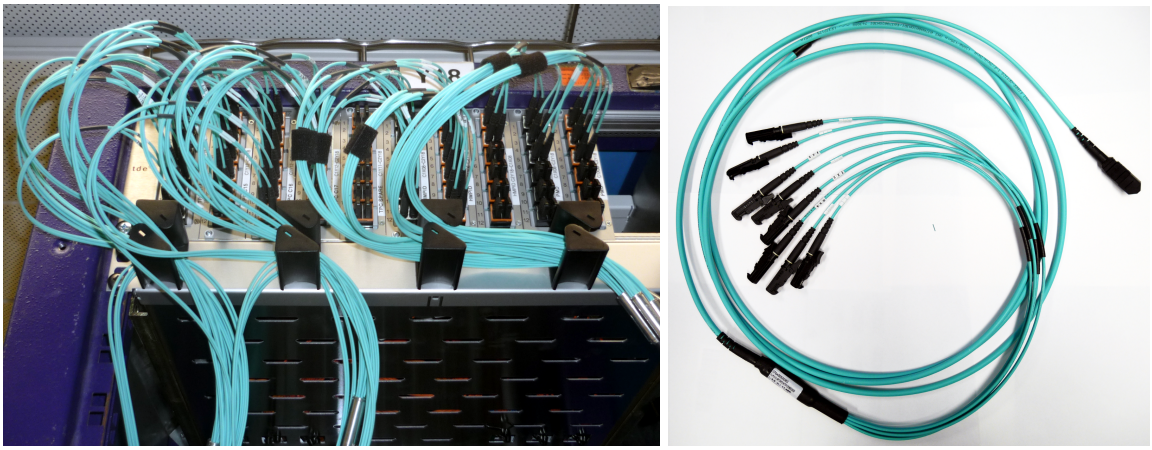

Figure 3.3: Left: DDL patch panel at the top of the rack. Right: break-out fiber from patch panel (E2000) to C-RORC (MTP).

A total of 74 C-RORCs are installed in the HLT production system. Their distribution across the different detector links is shown in Table 3.1. The connection to the fiber network for the DDLs is realized using break-out fibers and the existing patch panels at the top of the racks. The fiber installation at the end of Run 1 was using cables with one pair of fibers each. Several patch panels are on the way from the detectors in the cavern up to the counting room of the DAQ system. The patch panels use the E2000 connector standard. The connection between the DAQ and the HLT systems use the same kind of patch panels. The connection between the D-/H-RORCs and the patch panels was realized with LC-to-E2000 patch cords in Run 1, connecting one SFP on the RORCs with one pair of fibers at the patch panel. The C-RORC comes with QSFP modules using parallel MTP fiber connectors so the existing patch cables could not be reused. The solution for Run 2 was to use break-out cables with the C-RORC compatible MTP connector on one side and eight E2000 connectors for the patch panels on the other side. This decision and the purchase

of fibers was made together with the DAQ group. The alternative was to install MTP-based patch panels, but this option turned out to be more expensive. Additionally, the break-out fiber solution enables changes on individual links in case of problems without having to touch the internals of the patch panels. A photo of the patch panels and the break-out fibers is shown in Figure 3.3. The 74 HLT C-RORCs are in total equipped with 159 QSFP modules and the same amount of break-out cables to connect 507 input and 28 output DDLs to the HLT. 216 of these DDLs come from the TPC as the biggest contributor of detector data in ALICE. The aggregated input bandwidth from the TPC is around 675 Gbps. The aggregated output bandwidth of the HLT is around 149 Gbps.

3.2 Firmware Overview

While the hardware installation in the HLT is very homogeneous, the C-RORC firmware has to handle a mixture of different scenarios. The C-RORC is present in two major operating modes in the HLT. It is used at the data input interface of the HLT to receive a copy of the raw detector data from the DAQ via DDL. This mode is referred to as HLT-IN. An input firmware variant, HLT-IN-FCF, additionally contains TPC data pre-processing with the cluster finding algorithm in the FPGA and is described in detail in Chapter 4. The remote link partners to the HLT input C-RORCs are DAQ C-RORCs and DAQ D-RORCs with different link rates. The second major operating mode is the output data interface of the HLT. The C-RORC provides the processed and compressed detector data back to the DAQ system via DDL. This mode is referred to as *HLT-OUT*. In this case, the link partners are DAQ C-RORCs. The optical link protocol is the same for both modes. The link speed is detector dependent and configurable at runtime. The dataflow direction via PCI-Express is contradictory for both modes and the processing and data orchestration modules in the firmware differ.

Three different firmware images are deployed to the C-RORCs installed in the HLT cluster:

- HLT input firmware: receive raw detector data from the DAQ system via DDL at any rate up to 5.3125 Gbps. Push the raw detector data to the host RAM via DMA.
- HLT input firmware with data pre-processing using the hardware cluster finder: Receive raw TPC detector data via DDL at 2.125 or 3.125 Gbps, process the raw data and push the processed data to the host RAM via DMA.
- HLT output firmware: pull the HLT results from the host RAM via DMA and push them to the DAQ system via DDL at 5.3125 Gbps.

A simplified graphical overview focusing on the detector dataflow in these two major operation modes is shown in Figure 3.4. The upper part shows the HLT input firmware dataflow. Raw detector data is received via the serial optical links using the DDL protocol. Alternatively, raw data can come from the on-board DDR3 memory for testing purposes. In the case of the HLT-IN-FCF variant, the raw detector data is processed with the cluster finding algorithm. The results are then handed over to the DMA engine to transfer the data into the host RAM. This chain of components is instantiated once per serial optical link and provides independent readout paths for each detector link. The lower part of Figure 3.4 shows the HLT output firmware dataflow. The HLT processed data blocks are requested by the firmware from the host RAM via the DMA engine. The PCI-Express read completions containing the data are decoded and the data is sent via the optical link using the DDL protocol. This chain is again replicated once per serial optical link. All components in both chains provide configuration options and status outputs. These are connected to a dedicated slow-control bus accessible from software independent from any detector data transport or DMA activity.

In addition to the production firmware images, a number of firmware variants were developed for hardware tests or algorithm verification. This includes:

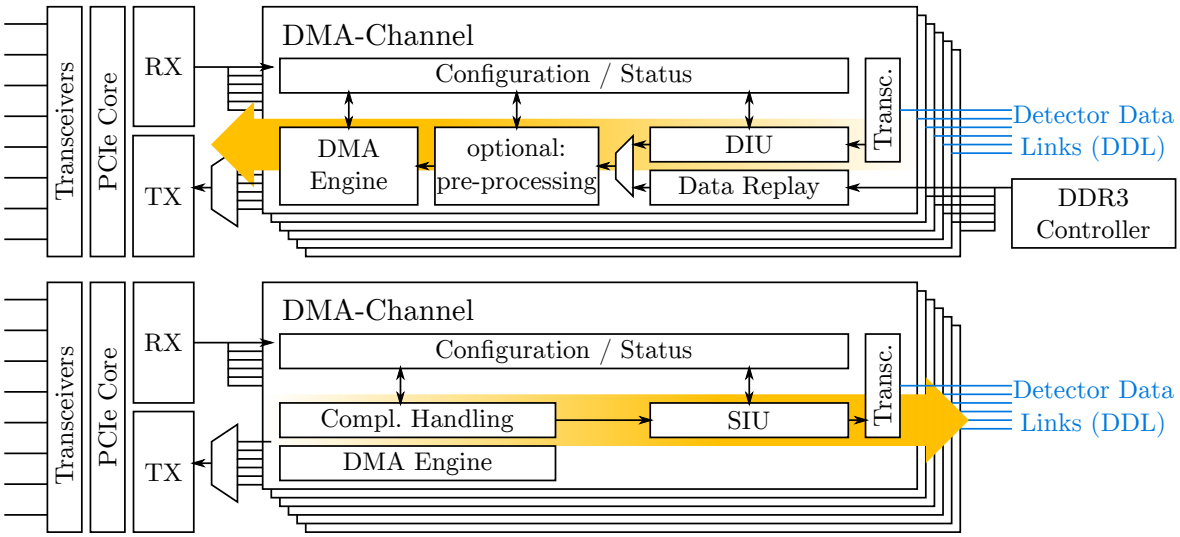

Figure 3.4: Overview of building blocks of the main HLT firmware variants. The detector dataflow direction is indicated with the yellow arrow. Top: HLT input firmware with dataflow from DDL to PCI-Express with optional pre-processing core. Bottom: HLT output firmware with dataflow from PCI-Express to DDL.

- HWTEST/IBERT: The HWTEST firmware image is used to validate the proper operation of a C-RORC at the hardware level. This is the firmware used for hardware tests at the manufacturer's site as described in Section 2.3.4. The IBERT image is a variant of HWTEST with a focus on low-level bit error rate tests and the characterization of the optical links.
- HWCF-COPROC: This firmware variant is a co-processor implementation of the hardware cluster finder, the data pre-processing core from the HLT input firmware. It is used to validate the processing algorithm and is described in more detail in Section 4.2.2.
- GBT/T-RORC: This firmware variant was originally developed as a proof of concept for the suitability of the C-RORC hardware to implement a GBT readout. It was later extended to the T-RORC platform described in Section 2.4.5.

Firmware type	Optical links	Link rates (Gbps)	Serial interface	PCIe endpoint	DMA-to-host	DMA-to-device	DMA channels	Preprocessing core	DDR3 controller(s)	Flash controller	Board diagnostics
HLT-IN	12	up to 5.3125	DIU	✓	✓	-	12	-	✓	✓	✓
HLT-IN-FCF	6	2.125 or 3.125	DIU	✓	✓	-	12	✓	✓	✓	✓
HLT-OUT	4	up to 5.3125	SIU	✓	-	✓	4	-	-	✓	✓
HWTEST/IBERT	12	up to 5.3125	LT/IBERT	✓	✓	-	12	-	✓	✓	✓
HWCF-COPROC	-	-	-	✓	✓	✓	12	✓	-	✓	✓
GBT / T-RORC	12	4.8	GBT	✓	✓	-	12	✓	-	✓	✓

Table 3.2: List of C-RORC firmware components used in the different firmware variants. Several components are shared by multiple variants.

Both the production firmware images and the secondary images share a number of common features, components, and interfaces. An overview of the features of the different images is

shown in Table 3.2. All firmware variants use the same PCI-Express endpoint implementation and connect to one or both of the two DMA engine implementations. The numbers of optical links and DMA channels to be used are configuration options at firmware build time. All variants provide the same interface to the on-board flash memory for the FPGA configuration files and come with the same board diagnostics. With identical interfaces on the input and output sides, the pre-processing core can easily and optionally be included as an IP core.

Firmware Variant	Look-up tables	Flip-flops	Block RAMs	DSPs
HLT-IN	64 %	44 %	43 %	0 %
HLT-IN-FCF (RCU1)	66 %	55 %	57 %	0 %
HLT-IN-FCF (RCU1 + RCU2)	63 %	47 %	62 %	6 %
HLT-IN-FCF (RCU2, extended)	51 %	42 %	79 %	6 %
HLT-OUT	16 %	12 %	13 %	0 %
HWTEST	39 %	29 %	27 %	0 %
IBERT	50 %	36 %	27 %	0 %
HWCF-COPROC	51 %	42 %	79 %	6 %
GBT / T-RORC	63 %	43 %	87 %	0 %

Table 3.3: C-RORC FPGA resource usage of the different firmware variants.

Table 3.3 shows the resource usage of the different firmware variants relative to the total number of resources available in the C-RORC FPGA. The numbers are extracted after the place-and-route step. The resource estimations done before the hardware development phase as described in Section 2.2.1 proved to be sufficient. The different HLT input firmware variants with the cluster finder are described with more details in Chapter 4. Each firmware variant has enough free resources to adjust or extend the functionality if needed.

A modular firmware architecture with well-defined interfaces between the different modules enables to integrate all firmware variants into the same source tree. A common source tree in return simplifies the maintenance and consistency of all firmware variants. All source files are tracked with a version control system and the source code revision as well as the date is automatically stored in a software accessible register in the firmware itself.

3.3 Detector Data Link (DDL)

The implementation of the Detector Data Link protocol in the HLT C-RORC firmware consists of three layers. The lowest layer is a device-specific physical and data link level using the FPGA-internal transceivers. The DDL IP cores for the Destination Interface Unit (DIU) and the Source Interface Unit (SIU) are mostly independent of the underlying device and implement a transport layer with a 32 bit data interface towards the user interface and a low-level 16 bit interface towards the transceivers. The DDL implementation ensures a reliable transmission of data blocks, transparently handles segmentation and provides flow control mechanisms. As the DDL is transparent to its payload, an application layer on top defines data properties. Each data block transferred over the DDL contains a Common Data Header (CDH) [Div 14] that enables to associate the data blocks from different detector links to the same event. The encoding of the payload after the CDH is detector-specific. The implementation of the detector data input and output interface of the HLT via the DDL protocol remains the same as designed for and used in Run 1. Keeping the existing approach enables to reuse a significant portion of software from Run 1 in both the DAQ and the HLT system.

The integration of the DIU and SIU IP cores into the HLT C-RORC firmware is sketched in Figure 3.5 for the DIU. The DIU and the SIU cores are provided and maintained by the DAQ group. This code is apart from a few FIFOs not specific to a certain FPGA and is independent of the actual link speed. On the link-layer side, independent 16 bit receive and transmit data

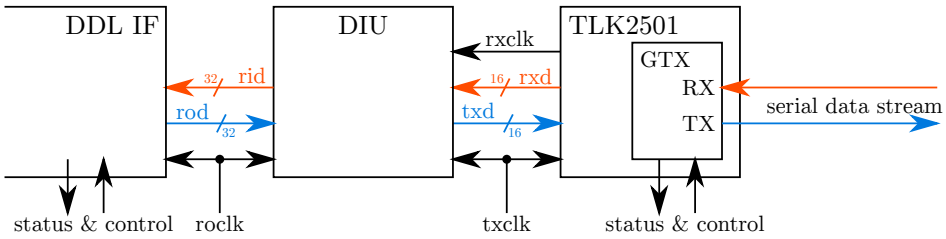

Figure 3.5: DIU integration into the HLT C-RORC firmware.

buses with their respective clocks are connected to the FPGA transceiver instances. The device-specific connector instance between the DDL and the transceivers labeled as "TLK2501" in the figure instantiates the FPGAs GTX transceivers. It provides additional information on the link status and controls compared to the previous H-RORC implementation with the DIU add-on boards. Low-level link status monitors enable immediate detection of data corruption at the link level due to faulty fiber interconnects or link-level connection issues. These errors can be differentiated from data errors on the DDL user interface.

On the transport-layer side, the DIU provides two 32 bit data buses, one in each direction, plus a few control signals. The interface clock has to be provided from the user logic and can be independent of any of the low-level sending or receiving clocks. The DDL interface code shown on the left of Figure 3.5 could partly be reused from the previous H-RORC firmware design for the basic data transport and command functionality. It was extended by a set of additional status signals, as well as word and event counters, to keep track on what happens on the link.

The implementation of the SIU in the HLT output firmware is equivalent to what is shown in the figure for the DIU. The link layer interface is using the same "TLK2501" module, providing exactly the same link status and control interfaces. The originally bi-directional data interface to the SIU on transport layer side has been modified to also provide the two 32 bit uni-directional data paths. As the main dataflow is in opposite direction, the only major difference to the DIU variant is the DDL interface code.

The DDL IP cores are mostly independent of the target link rate. The link rate is defined from the reference clock frequency in combination with the clock multiplication and division factors in the transceiver PLL. The reference clock frequency on the C-RORC hardware by default starts up with 212.5 MHz, suitable for DDL at 2.125 Gbps, 4.25 Gbps or 5.3125 Gbps. Other frequencies can be configured via I^2C. An I^2C master in the firmware and controllable from software enables to configure any reference clock frequency at runtime. The PLL multiplication and division factors can also be adjusted at runtime using the transceiver's Dynamic Reconfiguration Port (DRP). The configuration options of the reference clock oscillator and the transceiver parameters enable to set any DDL rate at runtime. The higher the link rate, the higher the clock frequency inside the DDL IP core and the connected user logic in the firmware. Therefore, the firmware implementation of transceivers, the DDL IP core, and the connected user logic have to be constrained to the maximum expected clock frequency already during firmware synthesis.

3.4 PCI-Express Interface

The PCI-Express interface is the central communication interface of the board. Detector data from the optical links have to be shipped to the host RAM via the PCI-Express interface. HLT-processed data have to be transferred from the host RAM to the C-RORC via the PCI-Express interface.

The detector data transport mechanisms in the C-RORC firmware use direct memory access (DMA) methods to read from or write to the host RAM without requiring CPU involvement in the data transport. This means that the FPGA can directly access parts of the host RAM to read data from or write data to. The data throughput using DMA is higher than using

programmed IO and requires only little CPU involvement. DMA is a common practice also for network devices, video capture cards or GPU processing. Since accessing the host RAM from the FPGA does not involve the CPU, the transfer goes undetected by the control software. However, further data processing steps have to be done in software, so the CPU at some point has to know about available data. For this reason, notification and synchronization mechanisms have to be implemented to enable data orchestration and data handling from the CPU side.

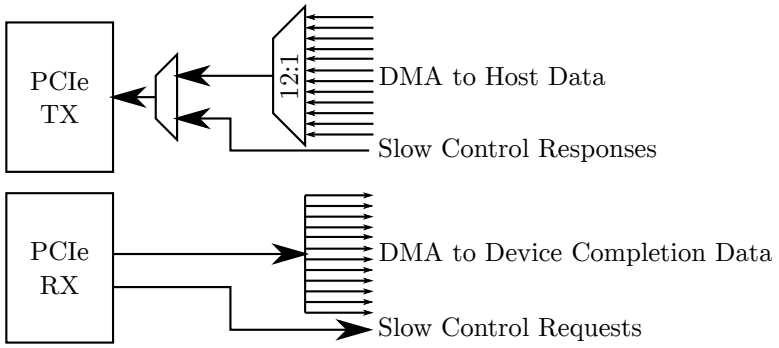

Figure 3.6: Data multiplexing and de-multiplexing of up to 12 DMA channels plus the slow-control interface from and to the single PCI-Express interface.

The PCI-Express implementation in the FPGA provides one receive and one transmit interface, each with a raw bandwidth of 4 GB/s. The detector data from up to 12 input streams have to be multiplexed onto the same PCI-Express transmit engine together with possible slow-control responses. Read completions from the host machine to the FPGA are distributed to the corresponding receiving channels. Additionally, slow-control read or write requests come via the receive interface. Figure 3.6 shows how the different data streams are multiplexed onto this single interface.

The PCI-Express protocol is packet-based, meaning that data sent and received via this interface is wrapped into a well-defined packet format with header and possibly trailer words. Packets cannot be arbitrarily big. The maximum packet sizes for different request types are negotiated between the device and the endpoint at the link initialization phase. The header and trailer information around the payload also lead to the fact that packets with small payloads are costly in terms of bandwidth usage because of their static protocol overhead. The maximum throughput can be achieved by sending and receiving only packets with the maximum allowed packet sizes.

Direct memory write access from the device to the host can be achieved by sending memory write request packets with the corresponding data in the payload to PCI-Express bus addresses associated with pages in the host RAM. The host RAM controller takes care of the actual write process. Direct memory read access from the host memory can be realized by sending read request packets to the root port containing a bus address of memory regions in host RAM together with a size. Replies from the memory controller via the root port to the PCI-Express device then contain the requested data in one or more read completion packets. PCI-Express uses a credit-based mechanism to distribute bandwidth across multiple devices. The PCI-Express device requests credit tokens for reading or writing from the root port before it is allowed to issue these commands. The credit-based design of PCI-Express and the independence of transmit and receive channels create an interface without guaranteed bandwidth or latency. The root port may temporarily throttle the data transfer from device to host at any time and requests in any direction do not have a fixed latency.

The PCI-Express DMA implementation in the HLT C-RORC firmware has to be able to handle the full input bandwidth from the serial optical links. This corresponds to up to 25.5 Gbps for 12 input DDLs of the first generation at 2.125 Gbps. The original plans contained the same bandwidth for six TPC DDLs with the RCU2 at 4.25 Gbps per link, the actual link speed of 3.125 Gbps now relaxes this requirement. The eight-lane PCI-Express interface provides a raw throughput of 20 Gbps at the generation-1 link rate and 40 Gbps at generation-2. Therefore, the

second generation is at least required to handle the raw input bandwidth but comes with some margins. The input data streams from all DDLs per board have to be multiplexed onto a single PCI-Express interface. Each DDL has to be able to be read out independently from any other and the HLT data transport framework has to know exactly from which DDL a certain data block is coming from. This requires that the data from each link is clearly distinguishable on the host side. Like during Run 1, each DDL is controlled by a separate readout process. This enables to start, stop, and operate the DDLs on a C-RORC independently. The data from each link is written into separate buffers in the host RAM. This requires a separate DMA channel per DDL in the firmware. The bandwidth required for each DMA channel is defined by the optical input bandwidth of around two to four Gbps, which is significantly lower than the overall PCI-Express bandwidth. For these reasons, the full C-RORC readout requires a firmware with support for up to 12 independent DMA readout channels that use the same PCI-Express interface. The need for vectored IO in each DMA channel was already explained in Section 1.6.2. Commercial DMA IP cores like the implementation from PLDA provide up to eight DMA channels. More channels are possible by attaching custom DMA firmware logic. However, if custom logic is required for 12 channels anyway, there is no need to use the commercial core in the first place. A custom DMA engine implementation saves licensing costs and provides a fully flexible solution for any number of DMA channels.

A slow-control bus to access a register file in the firmware enables arbitrary read and write access to the board without disturbing any data transfer. The slow-control bus access is by design controlled by the CPU. This bus is used by configuration and control software to keep track of the current firmware state. The slow-control interface is realized by mapping an address range assigned to the PCI-Express device into the hosts address space. Base Address Registers (BARs) in the PCI-Express configuration space define the addresses and sizes of these regions. Writing to these regions from software creates PCI-Express write request packets to be sent to the FPGA. Reading from this memory region triggers PCI-Express read request packets to the device. It is then up to the device to respond with a read completion packet. With this method, a fully user-configurable register file implemented in the FPGA firmware can be accessed from the host software, enabling status and control tasks of any kind.

3.4.1 DMA Device-to-Host Transfer Implementation

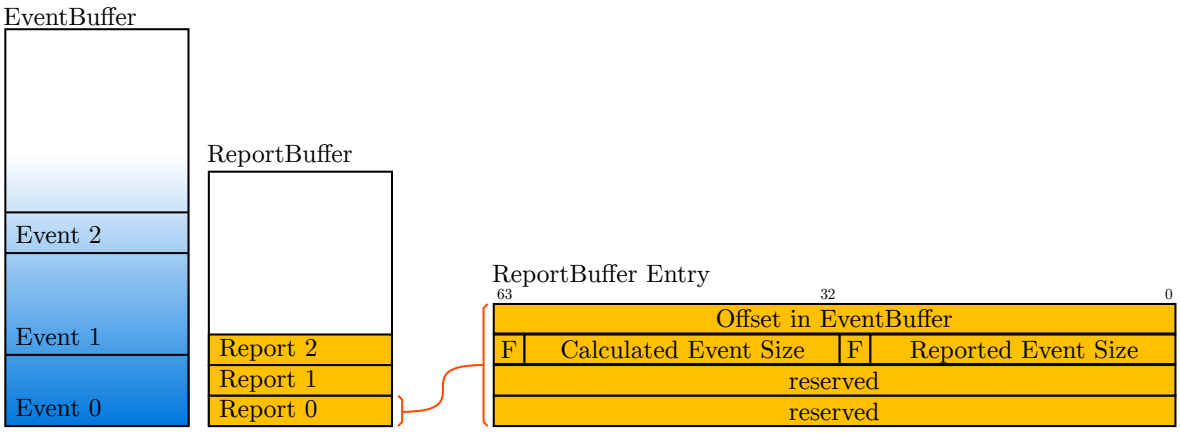

Figure 3.7: Dual buffer DMA transfer approach.

The DMA data handling concept on the C-RORC uses a similar approach as used in the HLT in Run 1. The implementation is, however, a completely new development. Due to the switch from PCI-X to PCI-Express and the support of vectored IO, none of the existing firmware modules or low-level software tools could be reused. Two DMA buffers in the host RAM for each DMA channel provide fast data transfer and synchronization. The detector data is directly written into the first buffer, the *EventBuffer*. A second smaller buffer, the *ReportBuffer*, is provided

with event descriptor entries of a fixed size each time a complete sub-event was written to the *EventBuffer*. This entry contains information on where the according sub-event can be found in the *EventBuffer* together with a couple of status flags and sizes. Figure 3.7 shows an illustration of *EventBuffer*, *ReportBuffer* and the fields of a *ReportBuffer* entry. Both buffers are used as ring buffers with read pointers and write pointers to handle buffer management. Both pointers are kept in firmware in the DMA engine and are accessible via software. The ring buffer approach greatly simplifies the memory management for both the software and the firmware side. The firmware ensures that the write pointer never overtakes the read pointer. This guarantees that no unprocessed data gets overwritten. The software updates the read pointer once a sub-event has been processed. This ensures that memory blocks belonging to fully processed events can be reused for new data. Possible address ranges within the ring buffers are static and known immediately after allocation. The dual buffer approach enables to fully separate the control flow and the data processing. A readout process controlling the data transport from the C-RORC to the host RAM only needs access to the *ReportBuffer* and can publish the *ReportBuffer* entries as descriptors to several subscribed data processing components. The data processing components only need access to the *EventBuffer* and do not need to know anything about the C-RORC. The FPGA DMA engine fills the buffers repeatedly from the beginning to the end. A potential drawback of the ring buffer approach shows up if a sub-event takes significantly longer to be processed than others. In this case, a single sub-event could stall the readout from the C-RORC to the host RAM even though there is usable buffer space available after the blocking sub-event. This situation can be avoided by choosing buffer sizes according to the maximum processing delays of the data. A 1 GB DMA buffer that is filled from a fully saturated DDL at 2.125 Gbps for example already gives a buffer time of at least four seconds until it would run full. The actual buffer time depends on the DDL saturation and varies across detectors and the detector occupancy for a given link. With 128 GB of host RAM per node, the blocking situation can easily be avoided even for 12 concurrent DMA channels. Typical DMA buffer sizes for C-RORC readout as used in the HLT throughput Run 2 are in the order of 1.0–1.5 GB.

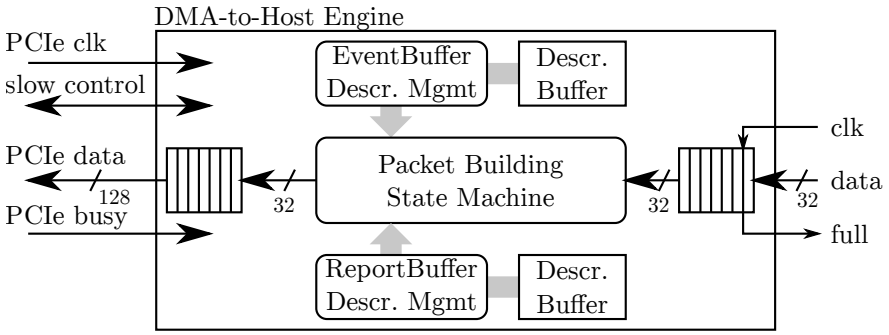

Figure 3.8: C-RORC firmware engine for device-to-host DMA transfers.

The implementation of the described DMA scheme in the HLT C-RORC firmware is sketched in Figure 3.8. This block is instantiated once per DMA channel in the firmware. The Xilinx PCI-Express endpoint IP core provides a 128 bit data bus and a couple of control signal in each direction at the transaction level layer. For PCI-Express generation-2, the core is running at 250 MHz and is providing the same bandwidth to the user interface as it has at the link level: 40 Gbps. If the transaction layer interface is saturated from the user firmware, some throttling is already expected by the fact that the lower-level protocol layers also consume some bandwidth. The effects of destination side transfer throttling are described in more detail in Section 3.9.

The bandwidth of the PCI-Express interface is much higher than the bandwidth of a single DDL. In contrast to typical commercial DMA engines, it is not necessary to implement a DMA engine that can saturate the PCI-Express interface from a single channel. Each DMA channel only needs to be able to process whatever it may receive via DDL. Only the packet multiplexing stage has to be able to saturate the PCI-Express bandwidth by efficiently scheduling packets from the different DMA channels. For this reason, the data path inside each DMA engine is reduced from 128 bit to 32 bit at 250 MHz. This still provides 10 Gbps of bandwidth per channel which is

way above anything the DDL will be able to provide and greatly simplifies PCI-Express packet assembly. Both the DDL protocol and the PCI-Express transaction layer protocol are based on 32 bit words anyway. The input to the DMA engine on the right of Figure 3.8 is a dual clock FIFO that enables to feed data to the engine with any clock frequency. At the output of the engine towards the PCI-Express endpoint, the 32 bit data stream is reassembled into the native 128 bit of the PCI-Express endpoint core interface and handed over to a packet-based 12:1 multiplexer. This multiplexer delivers the packets from all 12 DMA channels to the PCI-Express TX core in a round-robin fashion. Each DMA engine holds its own buffers for the memory descriptors. Each descriptor contains the bus address and the size of a region of memory pages in the host RAM that is prepared for DMA access. For each PCI-Express packet assembled in the packet building state machine, a new bus address is requested from the *EventBuffer* descriptor manager block. Whenever an event fragment is transferred, a packet containing the *ReportBuffer* entry is assembled using descriptors provided by the *ReportBuffer* descriptor manager. The descriptor buffers themselves are configured via the slow-control interface before the start of a readout and contain all bus address ranges belonging to this DMA channel. No additional descriptors are transferred while the readout is running. This reduces the background data transfer between the device and the host to a minimum and provides the maximum bandwidth for the device-to-host transfers. The maximum size of *EventBuffer* and *ReportBuffer* are given by the depth of the descriptor buffer in combination with the host memory allocation scheme and memory fragmentation. Using a vector-IO memory allocation approach creates a list of entries that can be fed to the DMA engine. Physically contiguous memory allocation schemes, such as provided by the BigPhysArea patch set, are equivalent to a descriptor list with only one entry. The same is true if the fragmentation of memory pages is hidden behind bus or device addresses managed with an IOMMU. This implementation with a buffer of descriptor entries makes the DMA engine completely independent from any memory allocation scheme on the host machine.

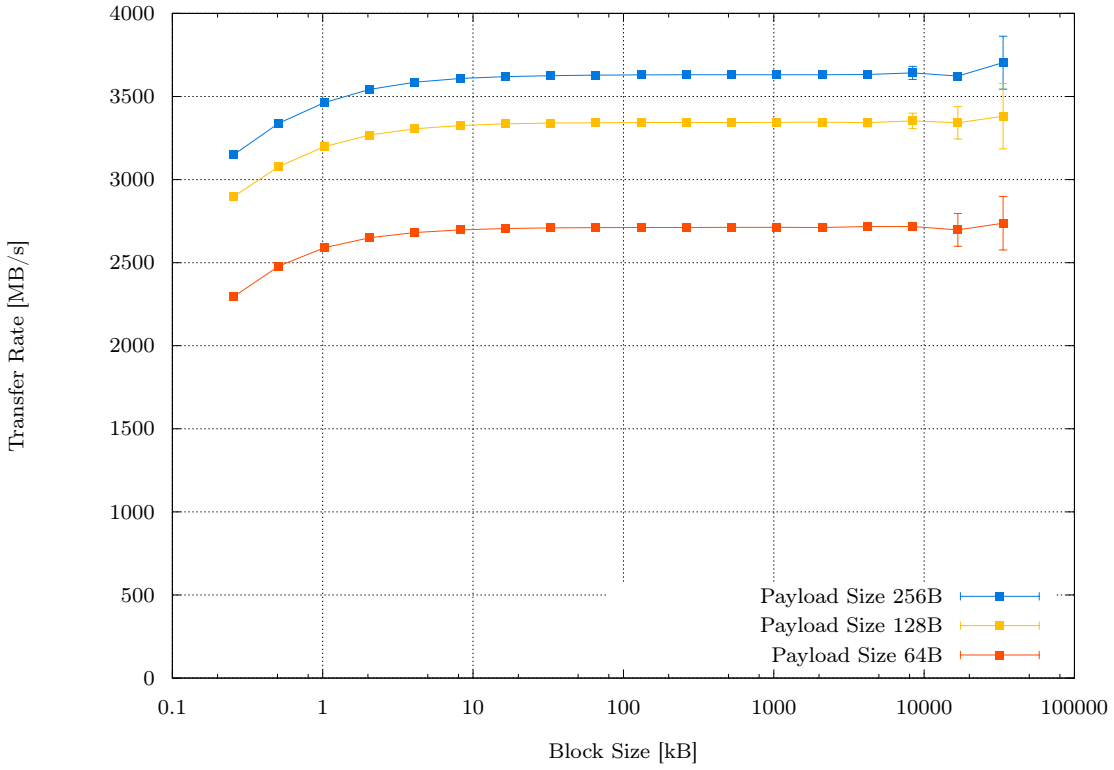

Figure 3.9: C-RORC DMA device-to-host transfer payload throughput into the *EventBuffer* for different block sizes and packet sizes.

A throughput measurement for DMA transfers from the device to the host using the described approach is shown in Figure 3.9. The measurement was done on a server machine in the ALICE

HLT cluster. Each measurement series is based on around 24 hours of readout benchmarks with varying block sizes. The throughput is measured once per second by accumulating the size of all completed block transfers in one second. The error bars indicate the mean squared deviation of the throughput measurements. The graphs show the accumulated throughput of payload data from 12 DMA engines into 12 *EventBuffer* instances for three different PCI-Express packet payload size limits. The maximum payload size supported by the XILINX PCI-Express core is 256 bytes. Pattern generator modules are used as data sources connected to the DMA channels. Each pattern generator can deliver up to 10 Gbps. This means that the total input bandwidth to the DMA engine instances is significantly higher than the PCI-Express bandwidth and does not affect the throughput measurements. The accumulated transfer rate from all 12 channels is shown on the y-axis. This value corresponds to the plain payload throughput transmitted into the *EventBuffer*. Packet headers and all writes to the *ReportBuffer* are not included. The effect of the packet payload size is immediately visible on the graphs. Each PCI-Express packet contains a header of 12–16 bytes. The smaller the payload size, the bigger the relative overhead for the headers per packet and, therefore, the lower the payload throughput. The block size on the x-axis is the number of bytes being written into the *EventBuffer* before a new entry is made in the *ReportBuffer*. A small block size means that only a few PCI-Express packets are written to the *EventBuffer* until a report entry is sent to the *ReportBuffer*. The fraction of write cycles to the *ReportBuffer* in relation to write cycles to the *EventBuffer* becomes visible in the graph as a slightly reduced overall throughput for small block sizes. With block sizes above around 1 kB, the overhead of the writes into the *ReportBuffer* becomes insignificant and the throughput into the *EventBuffer* is independent of the block size. With an accumulated throughput of around 3500 MB/s, each of the 12 channels provides around 290 MB/s. The data in the *EventBuffer* can only be counted as "transferred" as soon as the corresponding *ReportBuffer* entry is received. The variations at large block sizes are a result of the measurement method. With a rate measurement time base of one second and a block size of around 10 MB, each channel has a systematic uncertainty of ±1 blocks resulting in a throughput variation of around ±10 MB/s. A longer time base for the rate measurements coming with significantly longer measurement times would have avoided these effects, but the range with decreased precision is outside the ALICE use case anyway. The block sizes for physics events in ALICE in RUN 2 are well within the flat region of these plots: up to a couple of ten kilobytes per sub-event for proton-proton collisions and up to a couple of hundred kilobytes per sub-event for Pb–Pb.

3.4.2 DMA Host-to-Device Transfer Implementation

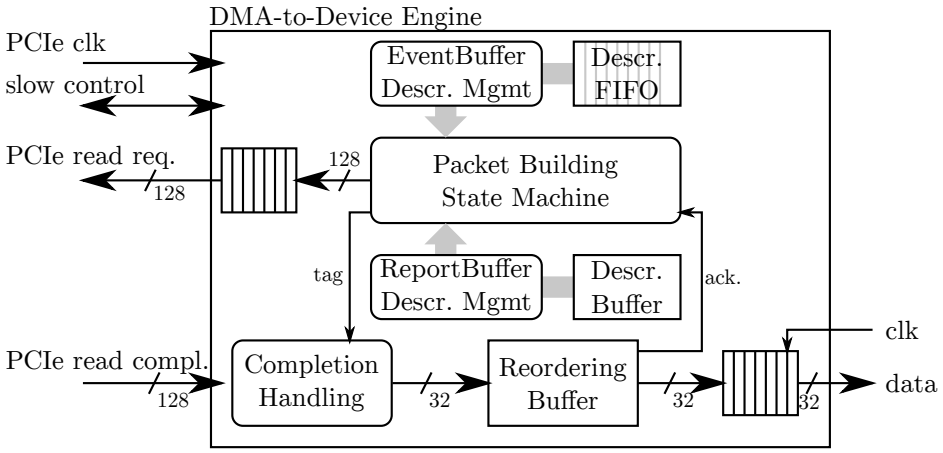

Figure 3.10: Schematic overview of the building blocks for the DMA host-to-device transfer engine implementation.

The DMA host-to-device transfer implementation is very similar to the device-to-host part and several firmware modules are equally used for both modes. A sketch of the implementation is

shown in Figure 3.10. In contrast to the device-to-host approach, the engine for host-to-device transfers assembles memory read requests instead of memory write requests. The requested data then arrives as memory read completion packets at the PCI-Express RX interface of the FPGA. A major difference to the device-to-host transfer approach is that the *EventBuffer* is not necessarily used as a ring buffer. The *EventBuffer* is filled by the HLT framework or any other software component and the filling scheme of the buffer is, therefore, controlled by this application. Keeping the publisher-subscriber model in mind it must not even be required that all data to be transmitted is located in the same *EventBuffer*. The DMA firmware implementation must not limit the use of several buffers or the way they are filled. For this reason, the only firmware requirement is that the data blocks to be transmitted via the C-RORC must be placed into memory regions with known bus addresses that are accessible by the C-RORC. This requirement slightly changes the *EventBuffer* approach compared to the DMA device-to-host transfer implementation: the descriptor RAM is replaced by a FIFO in this application. The DMA host-to-device transfer software provides descriptors with a start address and a block length to this FIFO. A data block may consist of one or more descriptors. The packet building state machine assembles the read requests based on these descriptors to retrieve the data from the host RAM. The maximum amount of data allowed to be requested with one PCI-Express packet is a parameter negotiated between the endpoint and the root port at the link initialization time. One or more PCI-Express read completion packets per request provide the data to the firmware DMA channel. Whenever a data block is fully received in the FPGA, a *ReportBuffer* entry is written in the same way it is done in the DMA device-to-host implementation.

The data throughput in the DMA transfer mode from the host to the device is strongly dependent on the latency from the read request to the read completion. Issuing a single read request and waiting until the requested data arrives would leave the bus unused for most of the time, resulting in a low data throughput. For this reason, it is desirable to be able to post several read requests at the same time. Individual requests can be identified with "tags". The same tag is present in the read completion packet. With this tag, the firmware can keep track of open requests and can correlate incoming read completions to the outstanding requests. A drawback of this approach is the fact that the completions are not guaranteed to arrive in the same order in the FPGA as the read requests were sent. This behavior is part of the PCI-Express specification. The DDL protocol, however, requires the data blocks to be in order. For this reason, received packets have to be buffered and potentially reordered in the firmware. Allowing up to eight parallel open read requests per DMA host-to-device engine instance turned out to be a reasonable compromise for data throughput compared to request tracking complexity and data reordering buffer requirements in the firmware.

Figure 3.11 shows the throughput measurements for the DMA host-to-device transfer implementation for different configurations. The y-axis shows the aggregated payload throughput from the *EventBuffer* into the FPGA of all channels in use. The block size on the x-axis is the number of bytes transferred until an entry is written into the *ReportBuffer*. The throughput measurements were done for two different maximum read request sizes (64 bytes and 128 bytes per request) and with one and eight parallel requests per channel. The 12 channel measurement shows the limits of the hardware. The four-channel measurement indicates the performance of the HLT use case where four DDLs are used per HLT output C-RORC. The measurements with only one open read request per channel at a time are mostly independent of the block size. These measurements are purely dominated by the latency from the read request to its completion. Comparing the throughput of around 500 MB/s and around 1500 MB/s with the bandwidth of four DDLs (green line) clearly shows that a single request approach does not fulfill the requirements of the HLT application. There is no way around posting several read request per channel at a time in order to efficiently use the available bandwidth. The DMA transfer throughput from the host to the devices for eight requests per channel exceeds the DDL bandwidth in all measured configurations. However, it has a much stronger dependency on the block size for small blocks compared to the DMA device-to-host measurements. The reason is that at the end of each data block the firmware has to wait for all open read requests to complete and has to flush the reordering buffer before starting a new block. This clearly limits the gain from the parallel read requests for small data

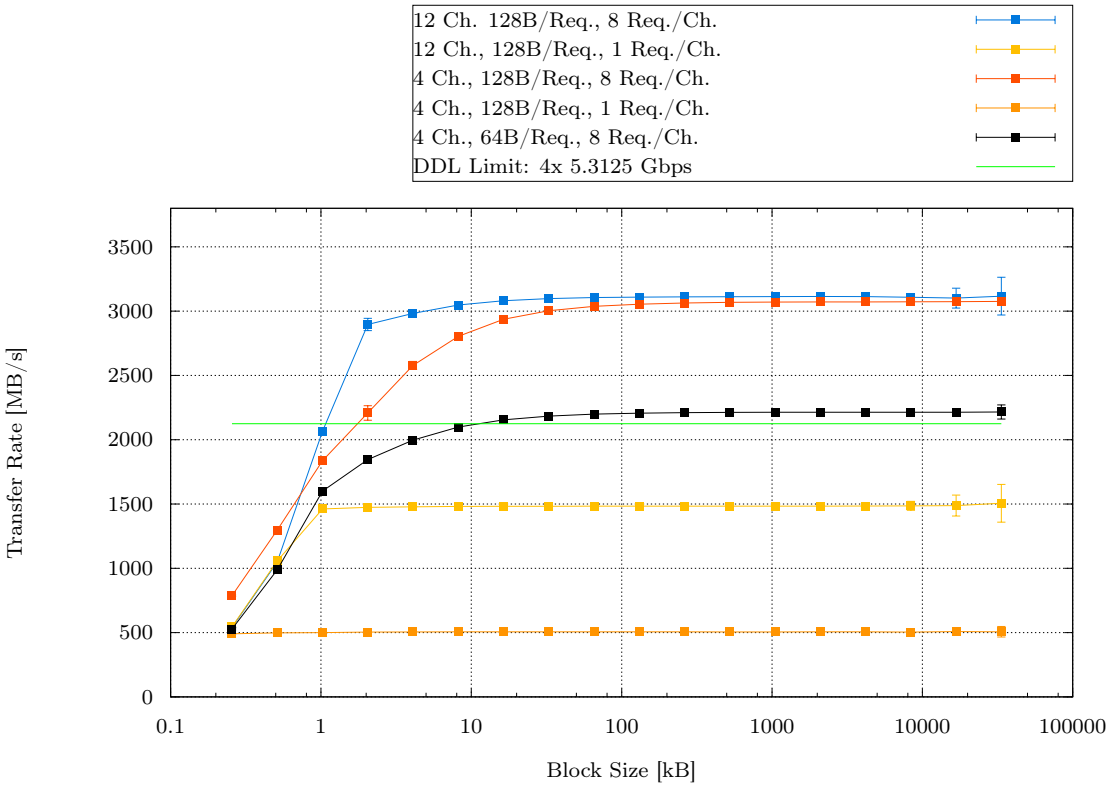

Figure 3.11: Throughput measurement results of the DMA host-to-device transfer implementation using different payload size limits and number of parallel read requests per channel.

blocks and directly correlates with the read latency of the bus. With block sizes above around 10 kB, these effects become insignificant and the throughput is mostly independent of the block size. The results are only slightly below the corresponding throughput measurements for the DMA device-to-host transfers.

The block sizes for the DMA transfers from the host to the device in the HLT are in general bigger than those for the DMA device-to-host case. This is due to the fact that each HLT output data block contains the compressed sub-events from all input links plus the reconstruction results. Typical block sizes are in the order of a couple of ten kB to a couple of hundred kB for proton-proton runs and up to several MB for Pb–Pb runs. Implementing the variant with four channels and up to eight open requests per channel with a maximum read request size of 128 bytes provides sufficient bandwidth margins to saturate four DDLs at 5.3125 Gbps each.

3.4.3 Host Architecture and Load Impact

All DMA throughput measurements in Sections 3.4.1 and 3.4.2 were done on dual-socket Ivy-Bridge machines as they are used in the HLT in Run 2. During the measurements, the machines were running no other CPU- or IO-intensive tasks apart from the measurement itself. No control was applied to where the DMA buffers were allocated in memory.

PCI-Express is credit-based, so if another device connected to the same root port is heavily utilizing the bus the DMA transfer from or to the C-RORC may be affected. The host RAM modules, as well as PCI-Express devices are physically connected to one or the other CPU socket on dual-socket systems. Both sockets are interconnected with a high-speed bus. The decision which memory is allocated on which socket is by default hidden from the user and done automatically by the kernel. A situation where the C-RORC is connected to one socket and the corresponding DMA buffers are allocated in the memory of the other socket creates additional

load on the socket interconnect and could reduce the usable the bandwidth from the PCI-Express into the RAM.

Early DMA device-to-host transfer throughput measurements on a machine with Nehalem architecture have shown a significant DMA performance impact of host buffer placement with respect to the C-RORC PCI-Express connection. The Nehalem architecture does not have the PCI-Express root port integrated into the CPU. Instead, one IO-hub chip per socket implements the PCI-Express root port. The hubs and the CPUs are interconnected with QPI[1]. The memory controller is contained in the CPU. Figure 3.12 shows the results of these measurements. The red curve is the reference measurement from the IvyBridge machine as shown already in Figure 3.9. The blue curve shows the throughput if both the C-RORC and all DMA buffers are located on the same CPU socket. The dataflow from the C-RORC via PCI-Express to the IO-hub and then via QPI to the memory controller is slightly slower compared to its IvyBridge counterpart. A significantly reduced bandwidth is measured if the DMA buffers are not allocated on the same socket as the C-RORC is connected to. In this case, the data has to be pushed via two QPI interconnects from the C-RORC to the host memory. The maximum throughput for this configuration is below the optical input bandwidth of 12 DDLs at 2.125 Gbps each. With this server type, an explicit pinning of all DMA buffers to the corresponding CPU socket would have been unavoidable in order to be able to handle 12 DDLs per board.

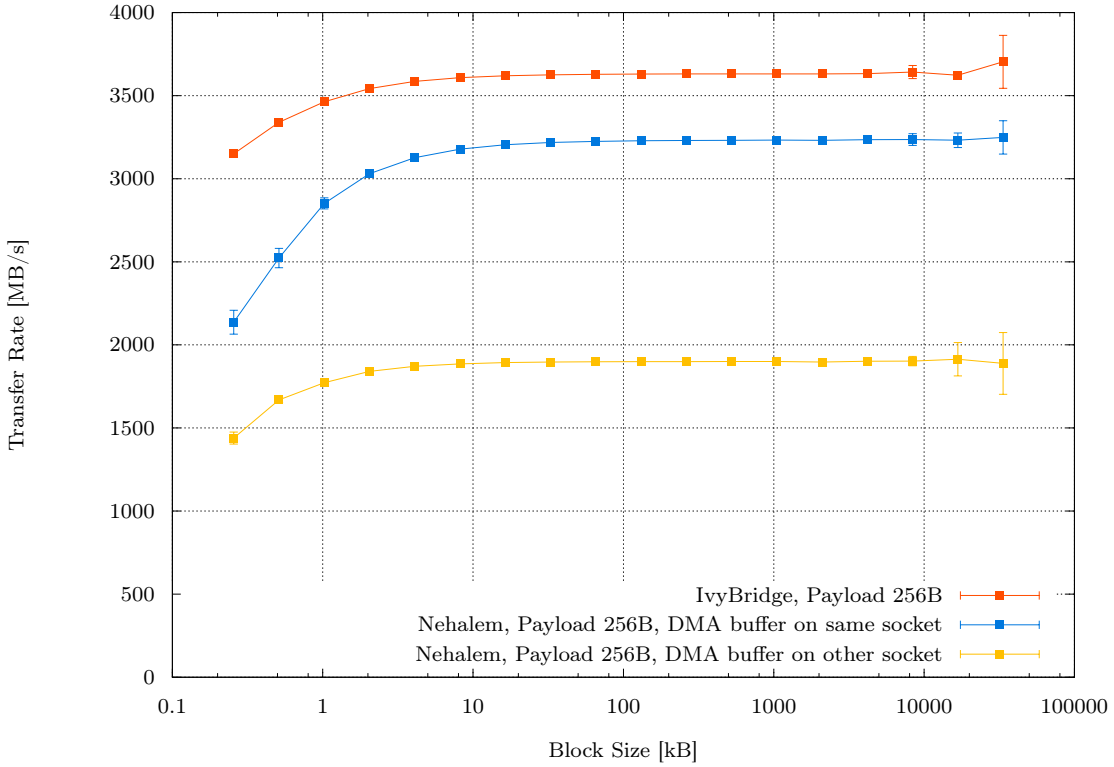

Figure 3.12: DMA device-to-host transfer throughput on a dual-socket Nehalem machine.

Luckily, none of these concerns affected the HLT due to the IvyBridge server architecture. This is mainly due to the fact that the maximum bandwidth a C-RORC can use is significantly lower than what the server hardware can handle. An eight-lane PCI-Express generation-2 endpoint cannot create a critical load on an IvyBridge system supporting a total of 32 lanes at generation-3 link rate per socket. The worst case situation with the C-RORC on one socket and the corresponding DMA buffers on the other does not show any reduction of the DMA bandwidth on these machines. Concurrent DMA activity on a second C-RORC or on the GPU while measuring C-RORC DMA throughput does also not affect the overall throughput of neither the device under test nor the

[1] Intel QuickPath Interconnect, a point to point processor interconnect

secondary device. Additionally, an artificial load on the CPU or the memory does not have an effect on the DMA bandwidth. Therefore, selecting IvyBridge machines for the HLT in Run 2 saved a lot of efforts to implement a software-based buffer-pinning support and optimizations to mitigate architectural limitations.

3.5 Host Software Interface

The PCI-Express firmware implementation on its own as described above only leads to the C-RORC being detected as a PCI-Express endpoint device in the host system, but nothing more. In order to interact with the card, a device driver and software using it is required. A prototype implementation of this device driver, as well as parts of the software layer to control the card, was done as part of this work. The final implementation of the kernel module and parts of the device driver library were done by Dominic Eschweiler as part of his Ph.D. thesis [Esc 16]. The concepts of the prototype device driver and library remained unchanged during this process. The same kernel module is also used for the First Level Event Selector Interface Board (FLIB) in the CBM experiment at FAIR, Darmstadt, and is planned to be used for the ALICE O^2 readout for Run 3. The integration of the C-RORC into the HLT data transport framework was done completely as part of this thesis.

3.5.1 Memory Allocation and Kernel Module

During Run 1 the HLT was using the PSI device driver [Pai 07] in conjunction with the BigPhysArea kernel patch set [Mid 03] to reserve large and physically contiguous memory regions for the RORC DMA transfers already at boot time. This memory was then dedicated to the H-RORC operation and could not be used by the operating system or other processes on the host. However, updating the operating system kernel always meant to also adjust PSI and the BigPhysArea patches to that new version. The memory regions reservable at boot time became smaller and smaller with each new kernel release and the effort to port the patches increased. The BigPhysArea patch set was finally discontinued in some early 2.6 kernels, so reusing this code for Run 2 was out of question. Control of the DMA buffers via PSI was realized with entries in the `/dev` file system[2] and `ioctls`[3]. For the C-RORC implementation, the `sysfs` tree [Moc 05] was chosen as an interface between the userspace and the kernelspace. This removes the need for custom `ioctls`, which happened to be problematic with PSI and kernel updates in the past. A minimal kernel module provides attributes in the sysfs tree that can be read, mapped or written from userspace programs. Allocating or accessing DMA buffers, as well as reading from or writing to the registers in the FPGA, is realized by simply accessing a file from a userspace program.

The PCI-Express endpoint memory regions defined in the PCI-Express IP core configuration in the FPGA firmware are mapped into the system address range when the device is initialized at boot time. With the help of the kernel module, these memory regions are then represented as files in the sysfs tree. A userspace program can `open()` and `mmap()` this file with the regular system calls. A write access to this map triggers a PCI-Express write command packet to be sent to the FPGA, a read access triggers a read request packet and provides the read completion data from the FPGA back to the userspace application. With this map, any interaction with the C-RORC can be done from userspace and does not require any device access methods in the kernel module.

The concept of ring buffers has proven to work well with the HLT data transport framework in the past and requires only little adjustments to the existing framework components. With the new DMA engine on the C-RORC, the DMA buffers do not need to be physically contiguous memory anymore. As a result, a replacement for the BigPhysArea patches was not necessary. C-RORC DMA memory can now be allocated as a list of memory fragments and by using the

[2] Linux directory with special files to allow software to interact with device drivers.
[3] Linux system calls for input/output operations.

standard Linux memory allocation subsystems. The DMA buffers are now contiguous only in the virtual address space. Different processes can map the same buffer into their own virtual address space. Access to the same data in these buffers can be realized by simply communicating offsets within these buffers between different processes. With the ring buffer concept being mapped onto linear memory fragments, the wrap-around point of the buffer needs to be handled carefully. A data block coming from the C-RORC can start near the upper end of the buffer memory, span over the end of the buffer and continue at the beginning. The previous software implementation used to compare the address and the size of the sub-event from the RORC against the buffer boundaries. In case of a wrap-around, it copied the parts before and after the wrap-around point into a separate buffer before publishing the event from there.

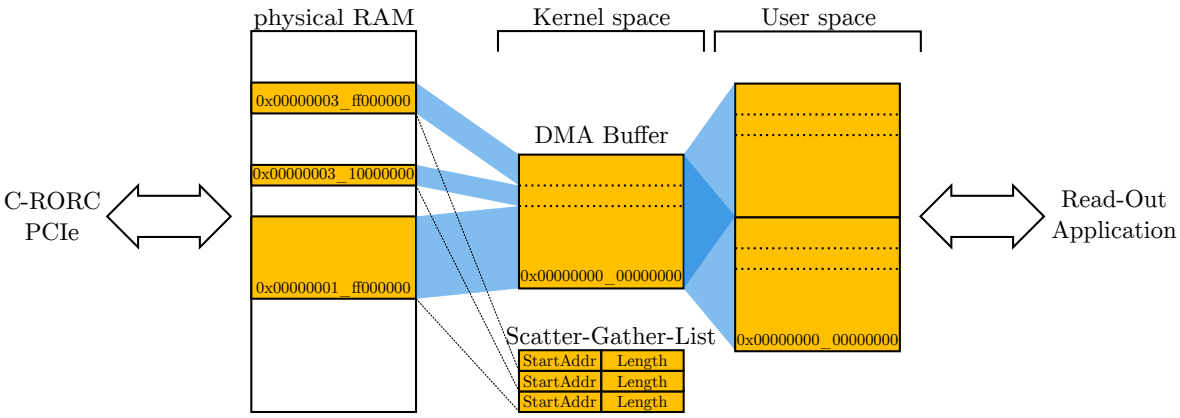

Figure 3.13: DMA buffer memory mapping to the userspace.

With the new implementation, this buffer wrap-around can be handled transparently. An illustration of the concept is shown in Figure 3.13. The same (possibly scattered) memory regions are mapped into the virtual userspace address range twice. The userspace application sees a memory map of twice the size of the physical buffer and each physical location can be accessed from two virtual addresses. If a data block spans across the physical wrap-around point of the buffer, the userspace application can still access it as a single virtually contiguous memory block and directly publish it to downstream HLT components.

DMA memory allocation itself can be done from the userspace or from the kernelspace. The kernelspace variant is close to the HLT Run 1 concepts and was available already in the first version of the kernel module. Additionally, it brings the wrap-around mapping feature described above at no cost. The userspace buffer allocation support was added by Dominic Eschweiler in a later revision of the kernel module and needs a manual handling of the buffer wrap-around. This is not used in the HLTs. The HLT C-RORC DMA memory allocation is done in the kernel module by iterating over `alloc_pages()` to reserve up to 1024 pages per call. This is the maximum number of pages supported by this function. With a page size of 4 kB, this corresponds to blocks of 4 MB of physically contiguous memory. A kernel API call maps the allocated pages for a specific PCI-Express device, which reveals the bus addresses to be used by this device to access the given memory region. This also means that a buffer is bound to the device it was allocated for and may be accessed via the given bus addresses only from this specific device. Access from CPU-side via the virtual addresses is unaffected by this limitation. Each of these 4 MB page blocks then has a page address, a bus address and a size to be added to a scatter-gather list. This list is provided to the C-RORC DMA engine. Repeated allocation of these 4 MB blocks does not necessarily provide physically neighboring bus addresses, but chances are good that at least some blocks are connected. By sorting the allocated blocks by their bus address and combining adjacent blocks to a single block with increased size, the number of list entries to be sent to the C-RORC DMA engine can be reduced. The chances for physically contiguous blocks depend on the overall memory fragmentation of the system. A freshly booted system is more likely to provide more connected blocks than a long-running system. A system with an active IOMMU internally maps all fragments into one contiguous bus address range.

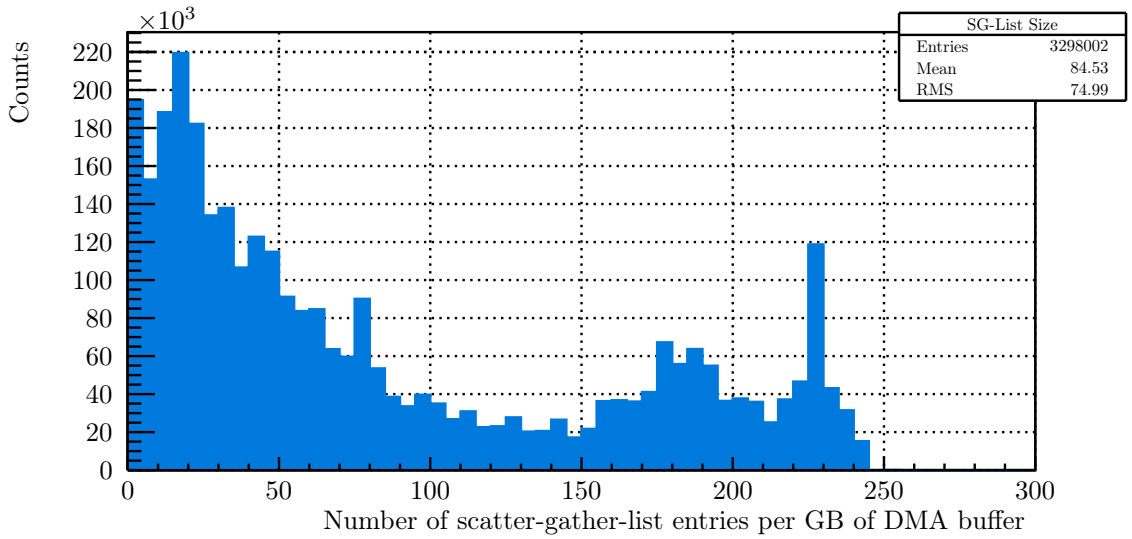

Figure 3.14: Distribution of the number of scatter-gather-list entries per GB of allocated C-RORC DMA memory during HLT operation from January to mid of July 2016.

Figure 3.14 shows a histogram of the number of list entries that were used throughout more than 3 million buffer allocations during the operation of the HLT in the first half of 2016. The x-axis shows the number of list entries per GB of allocated memory. An entry at 1 means that 1 GB (or more) of DMA memory is accessible as a physically contiguous memory region with one scatter-gather list entry. An entry at 250 means that none of the 4 MB blocks was next to another that could be merged. This represents the maximum possible memory fragmentation with 4 MB blocks. In a highly fragmented system, there is still the chance of not even getting 4 MB per allocation. In this case, the block size could in the worst case go down to one page per call, resulting in one scatter-gather list entry per 4 kB of DMA buffer space. A situation where the allocation of 4 MB as a block did not succeed was never observed in HLT. With this result, the maximum amount of memory per DMA channel in the firmware is limited by the descriptor buffer depth in the firmware times 4 MB. In practice, a descriptor buffer depth of 2048 entries resulting in a maximum of 8 GB DMA buffer per channel was more than sufficient. Furthermore, firmware resources are available to increase this buffer depth if necessary. The final kernel module, as written by Dominic Eschweiler, additionally supports userspace buffers, buffer placement controls, interrupt handling, and on-demand page mapping into userspace.

3.5.2 Device Access Library and Tools

The kernel module is only responsible for the allocation and the mapping of memory and IO regions into the userspace. All interactions with the C-RORC are done from userspace. In this context, the *librorc* library was developed in cooperation with Dominic Eschweiler. It is a C++ library that handles all accesses to the kernel module's `sysfs` files, either directly or by using the PDA [Esc+ 14] library. It provides a wide range of interfaces and calls to interact with the C-RORC on a structured and repeatable way. The *librorc* can equally be used by standalone C-RORC utilities and test scripts as well as the HLT data transport framework. The tools to manage the deployment and configuration of the different boards in the cluster use the same library. Having a common library for all C-RORC communication tasks enables to develop hardware-specific tools quickly and avoids redundant code.

The different classes in the library represent the firmware modules and their configuration options in the same way they are implemented in the dataflow between the optical link and the PCI-Express interface. Examples are the DMA engine, the cluster finder, the DDL interface or the data replay controller. Generator classes like the *EventStream* instantiating the firmware

classes provide a simple high-level interface to the sub-events from or to the FPGA for a given channel. They hide all operational steps to get the board initialized and running from the user. A schematic overview of some modules indicating the layer structure of the *librorc* library is shown in Figure 3.15.

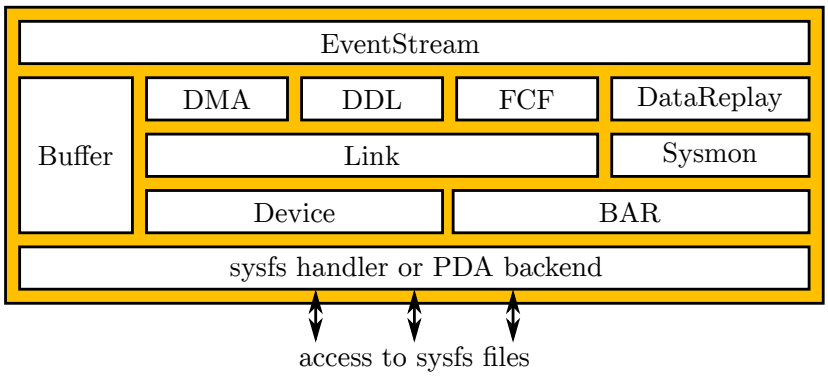

Figure 3.15: Overview of the *librorc* device driver library.

In order to operate the C-RORC, the dataflow into or out of the *EventBuffer* and *ReportBuffer* instances has to be orchestrated. Once the data transfer from the device to the host is activated, the DMA engine starts to write the sub-events into the *EventBuffer* and creates a *ReportBuffer* entry for each sub-event. The library keeps track of this process and provides a method to poll the *ReportBuffer* for new entries. Once a sub-event is fully processed, its buffer space in the *EventBuffer* and the *ReportBuffer* is made available to the DMA engine again by updating the software read pointers in the DMA engine. While sub-events are placed into the buffers in a predefined order, they are not necessarily freed in the same way. Different events may take a different time until the data is processed. For this reason, the library keeps track of received and freed events in the buffers and updates the pointers only if the tail of the list of published events changes. The readout process and the data processing can be decoupled completely if needed. The readout process only needs access to the *ReportBuffer* to check for new data and update the pointers. The subscribed data processing task only needs access to the *EventBuffer* because it receives the relevant descriptors from the readout process.

DMA transfer to the C-RORC works similarly. The event data is placed into a buffer accessible by the C-RORC. The library prepares a scatter-gather list of bus addresses and lengths for the given data block and provides this list to the DMA engine. The FPGA fetches the data according to this list and writes a *ReportBuffer* entry when done. From the software side, the *ReportBuffer* can again be polled to confirm the transmission of the data block. Once received, the *ReportBuffer* entry is freed again by updating the software read pointers in the firmware. The filling scheme of the source buffer, as well as the management of free space within, is up to the user process. It does not have to be used as a ring buffer.

Both data directions rely on a polling mechanism on the *ReportBuffer* instances. This may seem like a waste of computing resources but turned out to be the most efficient solution in terms of complexity versus gain. This decision was made already for the H-RORC during Run 1 and still holds true for the C-RORC in Run 2. The reasons are described also in Section 1.6.2.

The firmware implements a register file that is mapped into the PCI-Express endpoint memory region in the host system. The firmware provides PCI-Express access to it via the slow-control bus described above. This region is also exposed as a sysfs file into userspace by the kernel module. All runtime firmware configuration options, as well a large set of status signals, are mapped into this register file. The register file can be read and written with the library. A general region in this memory map provides status and configuration options for board-related functions. Each DMA channel additionally has an own set of registers for dataflow-specific status information and configuration options. Multiple processes can independently access the corresponding sysfs file at the same time and, consequently, the underlying register file. Sysfs

and the PCI-Express subsystem of the host take care of this. No custom access control or locking mechanisms are required. This also enables to start, operate, and stop any DMA channel in the firmware independently and from separate processes without interference across channels.

During the testing and integration phases, as well as continuing throughout the Run 2 operation of the boards in the HLT production system, a number of standalone tools were developed based on the *librorc* library. One aspect is a set of tools to test the full chain of firmware components from optical link to the DMA engine on a single board without the context of the HLT framework. All the hardware performance figures in this work were done with *librorc*-based tools. A second aspect is the analysis and triggering of hardware states as they may occur during operation that go beyond common occurrences or what automated error recovery mechanisms can handle. A third set of tools is used to operate all boards in the cluster reliably. One example is a dedicated reset tool that ensures a clean firmware state at the start of each run regardless of the previous firmware state. This enables a reliable start even if the readout chain crashed badly before. A full register file dump into an archive provides debugging and analysis capabilities also after a run was stopped. The cluster monitoring framework uses these tools to regularly query the device state and the cluster management uses *librorc*-based tools to ensure correct firmware versions on each board. The management and monitoring aspects are described in more detail in Section 3.10.

3.5.3 Integration into the HLT Data Transport Framework

The C-RORC needs to be integrated into the HLT data transport framework both as a data source and as a data sink. The data source components handle the detector data coming from the DDL and build the entry point into the data transport framework. They publish the data to the processing components in the HLT chain. These processes are called *CRORCPublishers*. The data sinks are the end points of the HLT chain and send the processed data to the DAQ system via the C-RORCs. They subscribe to the last processing component in the chain to receive the results. These processes are called *CRORCOutWriterSubscribers*. One data source or sink component is instantiated per DDL. An HLT input node with a C-RORC connected to 12 DDLs is running 12 independent data source processes.

The components handling the H-RORC communication during Run 1 were developed independently for source and sink. This is partly due to the fact that the corresponding firmware images for the HLT input and output were clearly separated as well. Both components were using the PSI API to handle DMA buffers and device access at a comparably low level. This led to a number of lengthy code fragments that were repeatedly present both within each component and between the components.

For the integration of the C-RORC into the HLT data transport framework, the hardware related code parts have been replaced completely. The *librorc* provides a compact abstraction level from bit operations at the firmware-level and significantly reduces the code size of the source and sink components. Linking the data transport source and sink components against *librorc* provides a higher-level control of the C-RORCs with significantly reduced code volume in the framework components. Additionally, a common hardware handler can be used for both the source and the sink components because the firmware implementations of both data directions are much closer connected now. The integration of the C-RORC also requires adding support for the new *librorc*-based DMA buffer type into the framework. Each framework component that produces data has an output buffer into which processed data is placed. The *CRORCPublisher* data source component allocates a *librorc* DMA buffer for receiving data from the board and uses this buffer also as the output buffer of the component. The subscribing downstream components, therefore, directly access the DMA buffer of this C-RORC channel to get the event data. For the data sink, the *CRORCOutWriterSubscriber* receives event descriptors from the *HLTOutputFormatter*. By making the output buffer of this processing component already a *librorc* DMA buffer, the *CRORCOutWriterSubscriber* can directly provide the bus addresses of event data blocks in this buffer to the C-RORC DMA engine without copying the data.

```
<DDL ID="768"><Node>tpca00</Node><RORCId>0</RORCId><Channel>0</Channel></DDL> <!--TPC HLT A00 RCU 0-->
<DDL ID="769"><Node>tpca00</Node><RORCId>0</RORCId><Channel>1</Channel></DDL> <!--TPC HLT A00 RCU 1-->
<DDL ID="840"><Node>tpca00</Node><RORCId>0</RORCId><Channel>2</Channel></DDL> <!--TPC HLT A00 RCU 2-->
<DDL ID="841"><Node>tpca00</Node><RORCId>0</RORCId><Channel>3</Channel></DDL> <!--TPC HLT A00 RCU 3-->
<DDL ID="842"><Node>tpca00</Node><RORCId>0</RORCId><Channel>4</Channel></DDL> <!--TPC HLT A00 RCU 4-->
<DDL ID="843"><Node>tpca00</Node><RORCId>0</RORCId><Channel>5</Channel></DDL> <!--TPC HLT A00 RCU 5-->
```

Figure 3.16: Example of DDL ID to node, C-RORC and DMA channel mapping. This block shows the DDLs of TPC sector A00.

A completely new feature added to the data source and sink components as part of this work is a continuous monitoring of the data stream in the FPGA from within the framework components. Status queries of the DMA engine, the pre-processing core, and the optical link are done once per second for each C-RORC source and sink component. With this monitoring in place, faulty optical links, buffers running full, and back pressure from or to the DAQ system can be detected quickly. By reporting unusual states into the same logging system as all other HLT components, the findings can easily be correlated with all other log entries to isolate the cause of possible problems.

Improvements have also been made to the handling of corrupt input data. Obviously corrupt data has previously been discarded in the source components. The mergers collecting the sub-events from each link, in this case, received data only from a subset of sources and could process the received sub-events only after a timeout expired. Corrupt events are now parsed more carefully already in the *CRORCPublisher*. If the corrupt event still contains a valid data header it is not discarded but truncated to its header only. By publishing only the header, the following merger component will receive a sub-event from all sources and does not have to wait for a timeout. This reduces the load on the mergers. For the processing of the event, it does not make a difference if the corrupt sub-event is completely missing or consists only of its header.

The sub-events arriving at the various C-RORCs all over the cluster do not provide information about the origin of the data from each link. For this reason, the framework has to know exactly which link on which C-RORC is connected to which part of each detector. The configuration of the HLT framework is done dynamically at the start of a run from a set of configuration files. A part of this configuration is a mapping of the DDL IDs to the corresponding host machines, C-RORCs, and DMA channels. An excerpt from this map is shown in Figure 3.16. The format of the mapping file was slightly changed compared to Run 1 because the H-RORCs and their channels were addressed differently.

Both the *CRORCPublisher* and the *CRORCOutWriterSubscriber* components support the standalone operation of the HLT for commissioning and local tests. The *CRORCPublisher* can be instructed via command line parameters to expect data to come only from the on-board memory instead of the optical links. In this case, the state of the optical link is ignored. The *CRORCOutWriterSubscriber* can be configured in a way to make the FPGA discard any data in the firmware right before it would be sent via DDL towards the DAQ system. With this setup, the full HLT can be tested with data coming from the C-RORCs, being processed and then push again into C-RORCs without the usage of the optical links. This makes the replay system independent of any operation of the detectors, the DAQ system or the LHC. The data replay functionality and its implementation in the firmware is described in more detail in Section 3.7.

3.6 Hardware-Software Co-Simulation Environment

A simulation environment is an essential part of an FPGA firmware project and takes a considerable fraction of the overall firmware development time. Turnaround cycles to "try things out" on the actual hardware are too long and the result is not necessarily conclusive. The Xilinx implementation tools are huge and take time to start. It can even take minutes until a simple VHDL syntax error is found. The synthesis of an FPGA bitfile can easily take a couple of hours.

Testing a firmware in hardware that means comparing output vectors from a set of known input vectors against a reference. If a mismatch occurs, the internals can still remain hidden inside the firmware. In-system debugging tools can help there, but they also come with long implementation times, may impact the timing of the design and have clear limitations when it comes to IO or multiple clock domains. An iterative approach to nail a firmware bug down on the actual hardware is time-consuming.

For this reason, a multi-layer simulation environment using the ModelSim [Men] HDL simulation tool from Mentor Graphics[4] was developed. All source files are checked for syntax errors using the ModelSim command line tools within seconds. Unit tests check a number of low-level and small system constellations on their basic functionality. Simulation models of peripherals, as well as data generator and data checker entities, provide the stimuli and reference data to cover this functionality. Simulation-only assertion statements throughout the code catch and report invalid firmware states. All unit tests can be run in parallel from a single command. This enables these checks to be run automatically from a continuous integration or nightly build system. Beyond the unit tests, there are several test benches for full system simulations of the different firmware variants. These environments can simulate the full firmware, from the serial link transceivers to PCI-Express interface, optionally including the DDR3 controllers and memory models, flash memories and I^2C or SPI peripherals.

The main problem of test coverage, however, remains or even gets worse. For small unit tests, it may be possible to cover all code parts by either a sufficiently large or a specifically selected set of test vectors. This is out of scope for a full system simulation. The challenge is here to select simulation input vectors that match the actual inputs of the hardware device in its final operating conditions. With the PCI-Express interface as the main communication structure, the system simulation inputs inevitably get close to what the *librorc* is doing internally: read from and write to the device. VHDL provides tools to create stimuli for tasks like that. However, with higher level functions on top of simple reads and writes things easily get pretty complicated. Higher level simulation frameworks like UVM[5], SystemC, SystemVerilog etc. can help to keep the stimuli readable. However, if things do not work in hardware two questions arise:

- Do the simulated conditions and stimuli sequences really match what is happening in the hardware?
- Is the error on the firmware or on the software side?

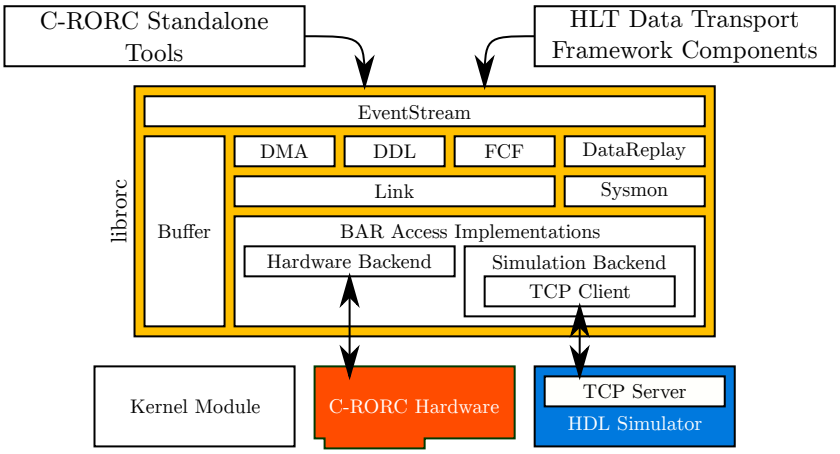

Figure 3.17: FLI software layer stack.

In order to tackle both questions at once, a different approach was taken in this work. With the help of the ModelSim Foreign Language Interface (FLI) [Men 12] a hardware-software co-simulation environment was created. FLI provides a C API to access information inside the

[4] https://www.mentor.com
[5] Universal Verification Methodology

HDL simulator. This covers information about the simulation itself up to reading signal values and driving signals to chosen values. In this case, the PCI-Express interface in the simulation environment is hooked to this C API. This means that PCI-Express read and write requests from both the host side and the endpoint side can be handled in C. On top of this, a TCP server is implemented in this C code. This provides dynamic stimuli coming from external tools. The stimuli can come from any other application. In this case, the *librorc* is providing the stimuli. An overview of this setup is shown in Figure 3.17.

The lowest level of hardware access in the *librorc* has two backends, out of which one is selected at compile time. The library either accesses the hardware directly or sends a hardware access request to the HDL simulator instance. With this setup, the same software tools can be used for simulation and hardware operation, depending on whether the library was compiled for simulation or for hardware use. A read request to the FPGA via the *librorc* is translated into a TCP packet for the simulation server. The TCP server drives the PCI-Express signals in the HDL simulation according to the request. The answer from the firmware under test is sent back to the *librorc* simulation backend via TCP and is presented to the caller as a response to the original request. Even PCI-Express DMA simulation is possible with this setup. Real DMA buffers are allocated on the simulation host system. The *librorc* simulation backend catches DMA write requests from the simulated device and goes through all DMA buffers in the system to check if any of them matches the bus address from the write request. If the destination is found, the simulation backend copies the payload from the write request into the given buffer location. The data block from the simulated device is actually written into the real DMA buffer. The user application will see the DMA data appear inside the DMA buffers just like it would when operating the actual hardware. The same applies to DMA read requests. The simulation server searches for the corresponding data in the real DMA buffers and provides it to the simulated device in the form of PCI-Express read completion packets. The simulation does not even need a C-RORC installed in the simulation server machine. The DMA buffers can be allocated and mapped for any PCI-Express device in the system.

With this simulation environment, the exact same host software can be used to operate the C-RORC as a hardware device in a server and to perform the same steps with a simulation of the HDL firmware sources. This makes sure that the same sequence of steps is performed on the hardware and in the simulation without maintaining an independent stimuli sequence for the simulation. In case of unexpected behavior, both the simulator and the host code can be paused and analyzed. Running the host software step-wise in a debugger like *gdb* while watching the internals of the firmware in the simulator enables to quickly identify whether issues are on the software or the firmware side. With this setup, it was even possible to run the final *CRORCPublisher* together with a small set of framework component with the simulation version of *librorc* and have all hardware access simulated.

3.7 DDR3 Memory Interface and Data Replay

In the HLT application, the on-board DDR3 memory of the C-RORC is used for data replay purposes. A data replay functionality was already available in the H-RORC implementation during Run 1. Unfortunately, due to the different on-board memory architecture and the increased number of links per board, only concepts could be reused and the system had to be re-implemented from scratch for the C-RORC.

Previously recorded, simulated or arbitrarily generated data can be loaded into the C-RORC memory and replayed as if it were coming via the optical links. Data replay helps to test the system at several levels, from a single link firmware verification via a single C-RORC node test up to the simultaneous replay on the full cluster. With a user-configurable event rate, almost any input data condition can be created artificially. Even though any HLT component can be tested on its own and options exist to feed the framework also with data from disk without the need of the C-RORCs, none of these methods provide a situation so close to the actual operating

conditions of the full system as data replay from the C-RORCs can do. By stressing the full system with high event and/or data rates, the limits of the HLT framework can be tested and analyzed beforehand and without risking reducing the data-taking efficiency of the experiment. Injecting corrupt or invalid data into the chain provides methods to test and verify the error handling capabilities of the framework.

On the output side, the firmware can be configured to discard any data just before they would be sent over the DDL. In conjunction with the data replay in the HLT input firmware, this enables to operate the full HLT cluster as close as possible to the experiment use case with user-defined conditions, but still completely independent from the optical links, the DAQ system, any detector or the LHC state.

3.7.1 DDR3 Memory Controller

The data replay on the firmware side is realized with the Xilinx Memory Interface Generator (MIG) [Xil 11], implementing one or two DDR3 memory controllers in the FPGA. This core provides a low-level command-based read and write access to the DDR3 SO-DIMM modules installed on the C-RORC. With parts of the controller running at 533 MHz for a 1066 Mbps interface, this clearly comes close to the limits of what the FPGA can handle. The C-RORC hardware is prepared in a way that two controllers can be instantiated independently. The controller itself has to be tailored to the target DDR3 SO-DIMM module already at compile time. The different timing parameters to operate DDR3 components and SO-DIMM modules have to be static at compile time. This means that the firmware has to be built for a specific set of timing parameters. Replacing a DDR3 module with a different model may require a new firmware image. The set of timing parameters for a given module is only rarely available from a datasheet. However, in most cases, the parameters can be read from the modules themselves. The DDR3 SO-DIMM modules provide an I^2C interface to an EEPROM that contains these values.

DDR3 controller resource usage	Single-ranked, 2 GB (LUTs / FFs)	Dual-ranked, 8 GB (LUTs / FFs)
with data mask, with re-ordering	5617 / 6466	5800 / 6587
with data mask, no re-ordering	5151 / 6188	5337 / 6309
no data mask, with re-ordering	5519 / 6328	5702 / 6464
no data mask, no re-ordering	5055 / 6051	5239 / 6186

Table 3.4: FPGA DDR3 controller resource usage for different modules and controller configurations.

The Xilinx memory controller provides a command-based 256 bit data interface. Each read or write command provides 512 bit of data from or to the memory. There are several additional configuration and optimization options available, depending on the target use case. An optional data mask can provide write access to smaller data blocks without the need for a read-modify-write cycle on a full 512 bit line. Optional command reordering increases the performance for non-linear memory access patterns. Enabling these options has only a small impact on the FPGA resource usage of the memory controller as shown in Table 3.4. Additionally, the resource usage differences between the single-ranked and the dual-ranked module controllers are minimal. However, the selection of controller options can impact the timing performance of the design during implementation.

The maximum supported controller bit rate per data line is given from the capabilities of the FPGA in combination with the Xilinx memory controller. For a single-ranked DDR3 SO-DIMM module, the limit is 1066 Mbps. This mode imposes the highest demands on the timing requirements in the FPGA design. Another commonly used mode is 800 Mbps, which provides

a reduced bandwidth but also relaxes the timing situation in the firmware. Dual-ranked modules typically provide a larger memory capacity, but are limited to 606 Mbps.

Throughput (MB/s)	DDR3-1066 (read / write)	DDR3-800 (read / write)	DDR3-606 (read / write)
calculated raw bandwidth	8535 / -	6400 / -	4855 / -
sequential read	8305 / -	6230 / -	4290 / -
sequential write	- / 7675	- / 5685	- / 3460
sequential read after write	830 / 830	670 / 670	505 / 505
6 streams without command reordering:			
independent sequential read + write	840 / 840	695 / 695	505 / 505
independent random read + write	785 / 785	660 / 660	525 / 525
interleaved sequential read	3910 / -	3840 / -	2790 / -
6 streams with command reordering:			
independent sequential read + write	2150 / 2150	1710 / 1710	1195 / 1195
independent random read + write	1250 / 1250	1040 / 1040	735 / 735
interleaved sequential read	3910 / -	3840 / -	2790 / -

Table 3.5: DDR3 throughput measurements for different controller settings and use cases. Values are rounded to the nearest multiple of 5 MB/s with an error in the same order of magnitude.

Table 3.5 shows an overview of a number of DDR3 throughput measurements. These measurements were done on different implementations of the memory controller with different memory access patterns. The first row shows the theoretical maximum bandwidth of the interface by multiplying the bit rate per data line with the number of data lines to the module (64). The underlying state machine, which handles pre-charging and refreshing of the rows, takes a bit of bandwidth. As a result, single sequential read or write command streams provide a usable payload bandwidth slightly below the calculated raw values. This overhead is seen in measurements of the sequential read and the sequential write bandwidth in the second and third row of the table. Reading an address just after it was written reveals the full latency of the controller and reduces the overall throughput per direction to approximately a tenth of the available bandwidth. This is shown in row four.

The DDR3 memory interface on the C-RORC has to provide data for up to 12 input links from up to two memory controllers. This means that each controller has to be shared by several data links. The lower two blocks in Table 3.5 show measurements of access patterns mixing reads and writes from several different start addresses, with and without the command reordering feature enabled. The first two rows of each measurement represent an alternating sequence of reading from one address and writing to another on sequential and pseudo-random addresses. Six of these request generators are working on different address ranges at the same time. The third row in these blocks is an interleaved read request stream from six channels with each stream requesting its own address range linearly. Comparing the numbers for command reordering enabled and disabled clearly shows its benefit when the memory is both read and written at the same time. However, if data only has to be read, it does not provide additional performance. In the case of the HLT data replay application, the read operation from the on-board memory has to be able to provide at least as many data as the optical link can provide. Writing to memory is done beforehand and does not come with high performance requirements. The replay data usually is read from a harddrive anyway.

For the HLT application, the interleaved sequential read command stream from six different addresses without command reordering, as highlighted in bold, can be used for data replay. With six data channels sharing one memory controller, the firmware can provide the data for all 12 links on the C-RORC with two SO-DIMM modules installed. The TPC readout with the largest amount of links per detector and six links per C-RORC, therefore, only needs one

SO-DIMM module per board to provide all channels with replay data. The total read bandwidth per controller for all three speed variants is exceeding the maximum amount of data that could come via six DDLs, even at the upper limit of the link rate. With the target design limit of the DDL bandwidth, the single-ranked controller at 800 Mbps is sufficient. The 1066 Mbps variant provides additional read bandwidth, but this is not necessary and the increased FPGA clock frequencies make timing closure much harder. The 606 Mbps variant could also be used if the total required on-board storage is not sufficient with single-ranked modules.

The final setup in the HLT uses up to two single-ranked 2 GB modules per C-RORC operated at 800 Mbps. Each SO-DIMM is shared by six data links. More channels per controller would reduce the read bandwidth below the DDL bandwidth. Fewer channels per controller cannot provide data replay for up to 12 channels. A non-uniform distribution of channels to memory controllers would technically be possible but would bring detector-specific aspects into a generic firmware design.

3.7.2 Data Replay from DDR3 Memory

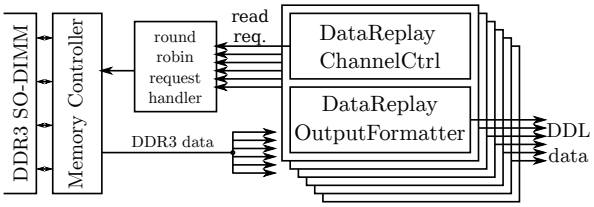

Figure 3.18: DDR3 data replay firmware blocks. Six data channels share one memory controller. The read requests are multiplexed onto the controller and the replies are sent to all channels.

The performance measurements from the previous section provide a setup with six data channels sharing one memory controller as the most viable solution. The fact that not all links are part of each run requires that the data replay implementation of each link can be controlled separately. This is possible by having a separate read command generator state machine per replay channel as part of a *ChannelController*. The read requests from all channel are multiplexed to the memory controller in a round robin way. The replies of the memory controller containing the payload of these read requests are in return delivered to all channels. Each channel decides if it is the requester of the given payload and either ignores it or presents the contained data as a DDL data stream as it were coming from the optical link. A sketch of this mechanism is shown in Figure 3.18.

With the multiplexing of read requests from six data sinks, up to six requests can be open at a time. The read latency of each request depends on the internal state of the memory controller and may vary. Each read response consists of 512 bit or 16 DWs of data. The firmware has to keep track of which data block corresponds to which read request. The solution chosen for this implementation is to treat each read response as a data packet and use the lowest DW as a data header. The encoding of the packet and the header are shown in Figure 3.19. The user has to make sure that the data written to the memory follows this format. A channel identification number defines the destination of the packet payload. A bit mask enables or disables parts of the remaining 15 DWs as payload. This is especially important at the end of a sub-event because not all sub-events have a size of a multiple of the payload size. An end-of-event flag makes the *OutputFormatter* insert an end-of-event status word as it would come via the DDL. Additionally, setting the DIU error flag enables to marks the sub-event as if it were corrupt.

Each *ChannelController* starts requesting data from a configurable start address and increments this address for each read response. Whenever the end-of-channel flag is set, the address is set back to the original start address. With this method, a continuous replay of a user-defined list of events is possible. Furthermore, the *ChannelController* offers the options to replay only one event, the full list once, a predefined number of events, or the full list continuously. With the

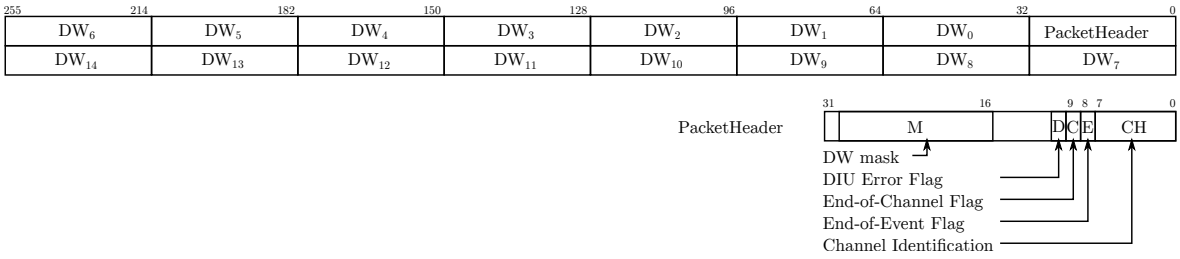

Figure 3.19: DDR3 packet data format for data replay.

configurable start address, each channel only needs a contiguous address range inside the memory. The partitioning of the available memory to the different channels is done by setting this start address from software. This enables to dynamically share the available memory between all channels.

The previous implementation of the data replay mechanism on the H-RORC kept a list of sub-events currently loaded into the RORC memory. While this is not exceptionally difficult to implement, it turned out to be not worth the effort anymore. The PCI-Express interface is fast enough so that it is usually easier to re-upload the desired set of events instead of carefully verifying a list of addresses and sizes. Additionally, checking lists of events from a couple of hundred input links for a cluster-wide replay is not practical anyway.

3.7.3 Cluster-Wide Data Replay and Framework Integration

Uploading sets of sub-events into the C-RORC on-board memory is independent of any readout application and can be done for each board at any time before starting the replay. A *librorc*-based data upload tool takes care of transferring the data into the board and manages the packet and header formatting. In order to use the data replay with the full cluster, corresponding sub-events from the different links for each event have to be present at the same position in the list of events on each board in the cluster. A bunch of scripts, run from a central point in the cluster, take care that each sub-event is loaded into the appropriate replay channel and that the order of events is the same on all links. All replay channels are then configured with the same target event rate and replay duration or options. Enabling the data replay is done at approximately the same time on each node for each board. The accuracy of the start time across nodes depends on the system time synchronization in the cluster.

The data source components in the HLT framework know from command-line arguments whether data will come from the optical links or from the on-board RAM.. When data replay is enabled, the state of the optical link is ignored and possible link errors are not reported. The number of events replayed into the chain typically does not exceed a few thousand. The limiting factor is the amount of memory available on the C-RORCs. Each sub-event contains an event ID in its header. This ID defines which sub-events belong together and is required for event building. The ID is also used as a seed for a pseudo-random load balancing algorithm that handles the distribution of events across the processing nodes. Using this feature with only the same few thousand sub-events repeatedly does not necessarily yield a uniform load balancing. For this reason, options exist to enumerate the event IDs in software while running instead of using the IDs contained in the data.

One important aspect is, however, that not all events contain sub-events from each detector. There are usually multiple trigger classes in each run, resulting in events with a contribution from only a subset of detectors. Different detectors participating in the same run may, therefore, have different event rates. Replaying events with mixed constellations of contributions from the participating detectors solely by event countes would lead to discrepancies. For this reason, the *CRORCPublisher* supports a special event type called *NOPE* events. This event type was already present during Run 1. The concept is to replay a small sub-event with known size and

content that cannot come from any detector during data-taking. This event is replayed whenever a link does not have a contribution for a given event, but other links do. The NOPE events are detected and discarded by the *CRORCPublishers*. All replay links in the cluster can, therefore, run with the same replay event rate at the firmware level, while the rate of events published into the processing framework is different for each detector depending on the raw data.

The framework can be tested for resilience by uploading corrupt sub-events to single replay channels or by flagging sub-events as if they were corrupt. A number of situations with bogus input data could be simulated and verified for proper handling in this way before they could have appeared unexpectedly during operation.

3.8 Parallel Raw Readout and Data Filtering

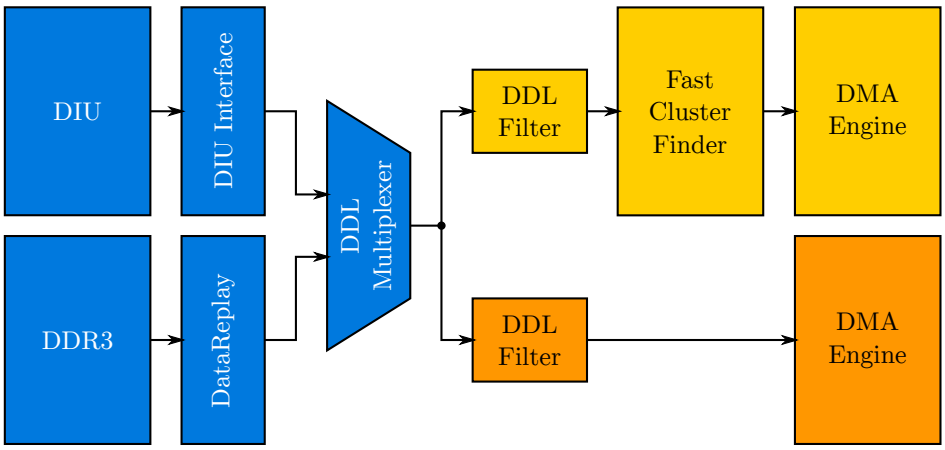

Figure 3.20: Firmware blocks for raw readout and data filtering.

The HLT input firmware for the TPC detector links contains the cluster finder as a hardware pre-processing step already inside the FPGA. The cluster finder is described in detail in Chapter 4. The raw data is received via the optical links and fed into the cluster finder. The processed results are handed over to the DMA engine to transfer them into the host machine. As the cluster finder replaces the raw detector data with the clusters found, the original raw data is not read out. This means that information on this raw data is available only indirectly via the resulting signatures in the clusters. In good operating conditions, there is no need to have the raw data. Additionally, the raw data is still available from the DAQ readout and can be stored in addition to the HLT results by choosing the appropriate HLT operating mode. This is done for tests and during commissioning but is not suitable for the data-taking operations. For HLT testing and commissioning purposes, the turnaround cycle of starting a run with raw data recording, stopping the run, waiting for the data to be on the grid, and downloading it again to analyze it is pretty long. For this reason, a secondary readout path from the C-RORC is foreseen in the HLT input firmware variant with the cluster finder as shown in Figure 3.20.

The HLT input firmware with the cluster finder primarily only needs six DMA channels because only six DDLs are connected to each TPC input C-RORC. Another set of six secondary DMA channels is available to this firmware variant to read out the raw data in addition. The incoming detector data is pushed into two readout chains. The primary chain feeds the data into the cluster finder and sends the results to the primary DMA engine. This is the DMA engine the HLT framework with the *CRORCPublisher* connects to. The second chain sends a copy of the raw input data directly to another DMA engine. This second engine is implemented in a way that it can be started and stopped at any time without interfering with the primary readout channel. A raw data analyzer process can connect to this channel while the primary channel is already running and can disconnect at any time. It can be configured not to stall the main dataflow in case it cannot keep up with the data or event rate.

The parallel raw readout may, however, impose a significant additional load on the system and the PCI-Express bandwidth of the C-RORC. The cluster finder output with a data reduction factor of around 1.5 plus the raw data for six DDLs at 3.125 Gbps exceeds the bandwidth limit of the PCI-Express link. In a high-load scenario, it may not even be possible to read out both the processed and the raw data in parallel. However, in many cases, the typical quality assurance algorithms do not need the full set of raw data but already work pretty well with a subset. For this reason, a filtering module is included into the firmware data path. This filter can be configured to forward the payload of only a subset of events. All other events are truncated to their header only and a status bit in the header is set to indicate that the payload was truncated. Not all detector links provide data with the same event rate during a run. As a result, a filtering scheme based on event counters, such as sending every n^{th} event, will not work. What can be used, however, is the orbit-ID contained in the event header. By applying a modulo bit mask against this ID, the same fragments from each event are either filtered or forwarded on all links on all C-RORCs in the run. With this method, the data rate across each data path can be reduced significantly while still getting the payload from the same events across all links. A typical monitor or quality assurance application would set a filtering mask on the secondary chain to receive only a subset of the raw data payload.

The parallel raw readout chain was used during the tests of the RCU2. The data sorting in the RCU2 firmware is defined with a configuration file that is loaded into the RCU2 at the start of a run. A data order mismatch in the readout of the RCU2 would result in an increased number of clusters after the cluster finder, but without strong evidence that this is caused by invalid order. Using the parallel raw readout, the order of the data from the RCU2 could be verified with a simple monitoring tool while the corresponding clusters were read out into the HLT framework. The system caught errors in the early tests of the RCU2 and enabled to quickly restart tests with a corrected RCU2 data order configuration.

3.9 Firmware Buffer Dimensioning

The dataflow in the HLT input C-RORCs consists of data reception via the serial transceivers and the DIU, data processing and transport to the DMA engine, either directly or via the cluster finder, and finally transmission into the host RAM. This is a chain of different processing components operating on different clock frequencies with FIFO buffers in between. Each component in this chain has a flow control mechanism in order to stop processing data if the buffer towards the next component is running full. If the dataflow stalls in one stage, the downstream buffers will gradually fill up and also stall these components. Finally, the DIU throttles data reception from the DAQ system, which in return forwards this signals to the detectors. This ultimately means that the experiment trigger is stopped and no more data is read out from the detectors. Therefore, a single busy readout channel in one of the C-RORCs in the HLT can stall a whole detector. This is obviously a situation that should be avoided to the greatest possible extent.

The maximum bandwidth of the DDL is known from the link rate subtracted by a small static protocol overhead. The components receiving data from the DDL can be implemented in a way that they are able to cope with this rate. Things get a bit more complicated with the cluster finder because its throughput is partly data dependent. The same amount of input data can contain more or fewer signals per pad or overall clusters. The processing time of the cluster finder depends on these numbers. Already with the cluster finder in Run 1, the algorithm could be implemented in a way that it can handle nearly everything that could come from the DDL without blocking the readout. The same was possible for Run 2 and is described in detail in Section 4.2.4. All components up to this point run with the DDL clock or with a clock derived from it. Therefore, the chain is automatically scaled when the DDL signal rate is changed. The next component in the chain is the DMA engine. It is running with the PCI-Express clock and is implemented in a way that each channel can handle nearly 1 GB/s. This is higher than the maximum data rate from the DDL. The next components in the chain are the PCI-Express packet multiplexers, merging packets from the 12 DMA engines onto the single PCI-Express interface. This final

interface is a crucial point in the whole chain. If the PCI-Express interface stops accepting new data, the DMA engines and all downstream components will see their buffers gradually filling up. While the PCI-Express interface at the physical level provides eight lanes with 5 Gbps signal rate each, the actual data transmission bandwidth on the transaction layer depends on a credit-based transfer negotiation between the root port and the endpoint devices. If the endpoint runs out of transfer tokens, it cannot send more data. The crediting mechanisms are buried deep inside the PCI-Express subsystem of the host machine and the PCI-Express endpoint implementation. They are mostly out of user control. For this reason, the PCI-Express interface on transaction layer may be busy at any time and for an unknown duration. While the average bandwidth, despite these effects, is still close to the theoretical limit as shown with the DMA throughput measurements in Sections 2.2.2 and 3.4.1, the firmware has to be able to handle temporary dead times.

The PCI-Express interface implementation in the C-RORC contains a transmission dead time monitoring component. This monitor keeps track of the longest consecutive transmission dead time observed on the PCI-Express interface. A graphical representation of the maximum dead time measurements is shown in Figure 3.21. This graph shows the maximum consecutive dead time measured in the HLT production cluster throughout all standalone, technical, and physics runs from January to November 2016. The values are shown for all ITS, EMCAL, and TPC C-RORCs in the cluster. The TPC boards were operated with six DMA channels per board. The boards reading out ITS and EMCAL links use up to 12 channels per board. The dead time monitor only increments its counter if the firmware has pending data to be transmitted and the bus is busy. For this reason, the C-RORCs with low data rates (e.g. SDD and DCAL1) typically only show little dead time. The measured dead times vary around 12 µs to around 43 µs, with a majority of boards around 20 µs. For a stall-free operation, however, the maximum value across all boards is relevant.

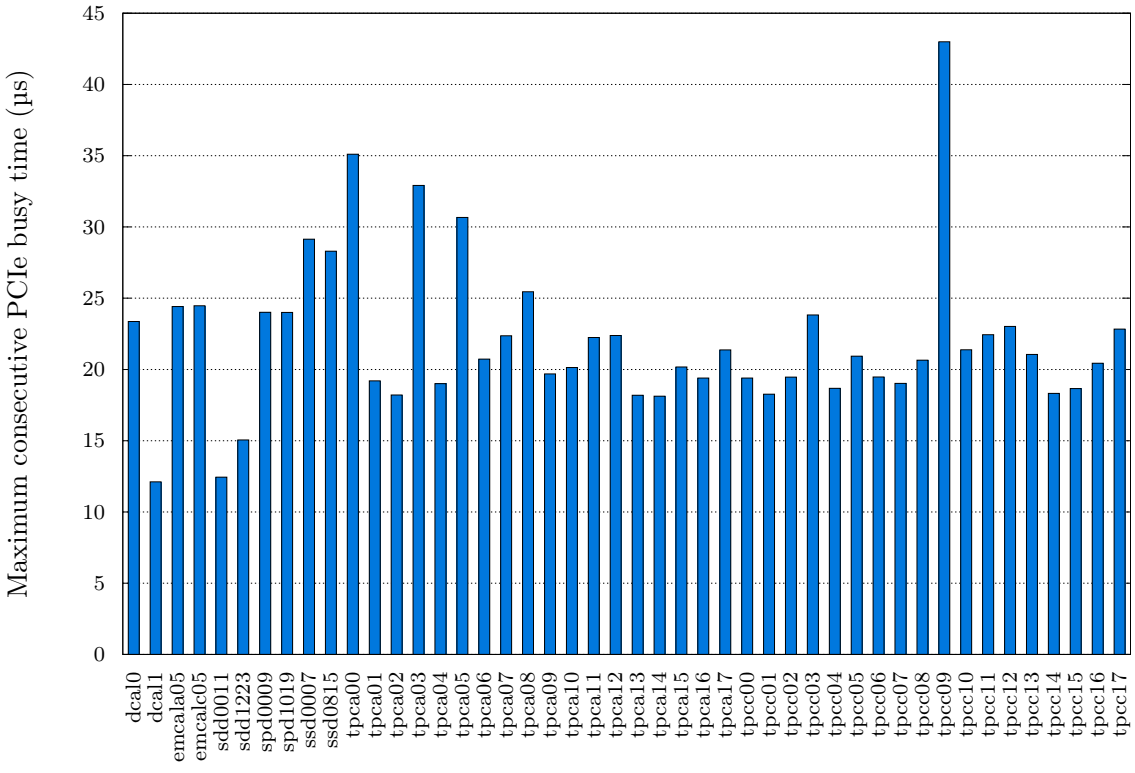

Figure 3.21: Maximum consecutive PCI-Express busy time on EMCAL, ITS and TPC C-RORCs between January and November 2016.

A simulation of the full HLT input firmware from the PCI-Express transaction-level interface to the optical links provides information about the buffering capabilities of the firmware. The PCI-Express interface is artificially halted during a readout simulation with 100% DDL bandwidth utilization. The time is measured from the moment the PCI-Express stops accepting new data until the DDLs stop to accept new data. The measurement is repeated for different buffer configurations in the data path. The results of these simulations are shown in Table 3.6. The base HLT input firmware has FIFOs in the PCI-Express interface, the DMA packet multiplexer, the DMA engine and in the DIU core. This configuration was already able to compensate up to around 13.5 µs of consecutive PCI-Express dead time until the DDL stalls. Comparing this number with the dead time monitor above makes clear that this implementation is not able to cope with the measured dead time during operation. For this reason, additional buffer stages were integrated into the firmware. By including another 1024 or 2048 entry FIFO stage, the maximum dead time the firmware can compensate increases to more than 32 µs or 51 µs. It is important to note here that the simulations assume mostly empty buffers in the firmware. If the buffers already filled up to a certain point when the interface gets busy, the time that can be compensated may be reduced significantly. The HLT operation with the additional 1024 entry FIFO still revealed rare situations with temporary DDL back pressure. No more back pressure was observed with the additional 2048 entry FIFO. Adjusting buffers in the C-RORC firmware for TPC readout with the cluster finder due to PCI-Express dead times was not necessary. The streaming implementation of the algorithm itself, with several internal FIFO stages, already provides sufficient decoupling to compensate the observed dead times.

Design Variant	Minimum PCIe dead time until a DDL stalls
HLT_IN, no additional FIFOs	13.5 µs
HLT_IN + FIFO with 1024 entries	32.8 µs
HLT_IN + FIFO with 2048 entries	51.2 µs

Table 3.6: Comparison of the minimum PCI-Express dead time until the first DDL stalls for different firmware variants in simulation.

3.10 C-RORC Management and Monitoring

The HLT cluster contains a total of 74 C-RORCs running one out of three different firmware variants, depending on its connection to the DDL network. Configuring or operating these boards manually is cumbersome and error-prone. Integrating a number of boards into a production environment like this requires far more effort than what would be necessary to operate one board as a standalone on-desk proof-of-concept solution. This starts by being able to execute the same hardware operations on all boards at once and still being able to spot a single failing instance. All software has to be designed in a way that it can catch and properly handle a broad range of possible error conditions. Automation was introduced to a wide extend to reduce the chance of operation errors. One key aspect to the cluster-wide operation is a reliable upgrade and verification method for the FPGA firmware version. Tightly coupled with this is an automated management system that knows which firmware has to be loaded on which C-RORC in the cluster. The state of each board is continuously monitored. Additionally, the correct cabling of the DDL network has to be ensured.

3.10.1 FPGA Configuration and Reconfiguration

When the C-RORC is powered on, it automatically configures itself from one of the on-board flash memory chips. In order to identify the currently running firmware, the build process integrates the firmware type, the `git` revision, and the date into the FPGA register file. By reading these

values via software, the exact firmware version is known and can be compared against a reference value.

In order to support an update of the firmware, the flash memory chips need to be accessible from software. All HLT C-RORC firmware variants contain an identical flash controller for this reason. Both flash memory chips are mapped into the PCI-Express device address space of the host machine to make them accessible via the *librorc*. A userspace application can read and write both flash memory chips. This already enables to write new firmware images into the flash memories. A new image can then be loaded with a power cycle of the host machine. However, it does not yet provide any fallback solution if something goes wrong.

The system gains resilience by activating the configuration microcontroller. A simple software state machine with an SMBus slave listening on the interface towards the host machine implements a register file in the microcontroller firmware. This register file can be read and written from the host machine via SMBus, independent of the state of the PCI-Express link. The server nodes installed in the HLT provide control of the PCI-Express sideband signals for SMBus via the IPMI controller. This interface can be used to check the configuration status of the FPGA, select a data source for the configuration, and trigger a reconfiguration of the FPGA. It also means that a reconfiguration from a different flash partition can be triggered if the initial configuration failed. The SMBus sideband signals can be used even if the main PCI-Express interface is down. This process does not require a programming cable to resolve the situation.

Both flash memory chips are in use on every C-RORC in the HLT. The first chip contains the HWTEST firmware that was programmed by the production company during the hardware tests. The second chip is used for the active HLT firmware variant. The firmware stored there depends on the DDL connectivity of the board. The power-on jumpers are set to automatically configure the board from this second chip. Whenever a firmware update is performed, the FPGA configuration file is written into this second flash memory chip. In case something does not work, the recovery image from the first chip can be loaded at any time to replace the image in the second chip. By rescanning the PCI-Express bus after each reconfiguration process, the newly loaded firmware can be initialized and host machine does not even have to be rebooted.

3.10.2 Automated Firmware Management

In order to reduce the number of manual interventions and, therefore, the error-prone steps, the management and the update of the C-RORC firmware images is integrated into the HLT cluster management environment. The HLT uses a cluster management system consisting of Foreman[6] for automatic deployment of the nodes, Puppet[7] for the configuration management and Zabbix[8] for the monitoring of important parameters like temperatures, network traffic, CPU load, and others. This cluster management system is described in detail in [Leh+ 17]. Each node in the cluster with a C-RORC is assigned a host group depending on the requested firmware variant. The target firmware type and revision for each host group are set centrally in a single configuration file. The HLT cluster management framework reads the current firmware type and revision from the C-RORC and compares these against the target values from the configuration. If the target firmware version does not match the currently running version, the C-RORC is automatically reconfigured. The cluster management tools automatically load the target FPGA configuration file from a central FTP server onto the node and run the C-RORC tools to write this image to the second flash memory chip. Once this is done, a reconfiguration of the C-RORC is triggered via the microcontroller over the SMBus connection. After a rescan of the PCI-Express bus, the new firmware version is again compared against the target version. A firmware update of all C-RORCs in the cluster is as easy as uploading the new firmware image to the FTP server and adjusting a single central configuration file that defines the target firmware revisions. The

[6] https://www.theforeman.org/
[7] https://puppet.com
[8] https://www.zabbix.com

firmware is then automatically deployed to all corresponding C-RORCs with the next iteration of the cluster management tool.

3.10.3 Hardware and Firmware Monitoring

Both the general state of the C-RORC hardware and the dataflow in each firmware variant are continuously monitored to detect possible problems as early as possible. Monitoring efforts at different levels work together to achieve that goal. Besides some self-regulating features, general metrics from all firmware variants are captured with the cluster monitoring tools. During operation, each firmware variant is also monitored from the readout process with metrics both asynchronous and synchronous to the data stream. During development, detailed monitoring tools were developed to investigate single board setups.

The C-RORC hardware comes with a number of on-board sensors and metrics in the FPGA or its peripheral components. These are also described in Chapter 2. The most important parameter is the temperature of the FPGA because an excessively high temperature could cause physical damage to the hardware. Each firmware image contains a temperature monitoring module that continuously reads the FPGA temperature and turns on the fan on the heatsink in case a certain threshold is exceeded. Operating the fan permanently when installed in a server machine is not recommended because the fan on the C-RORC is oriented orthogonal to the comparably strong air flow of the host machine. On top of this, the FPGA temperature and some PCI-Express link related metrics are periodically recorded with the cluster monitoring framework. With this system in place, C-RORC parameters be correlated with the general cluster monitoring metrics like rack, CPU, or GPU temperatures, CPU load, and others. A screenshot of the C-RORC temperature recording is shown in Figure 3.22. The boards running the HLT input firmware variants are typically in the order of around $60 - 70°C$. The HLT output boards with only four DDLs and without DDR3 controller show significantly lower temperatures in the order of $40 - 45°C$.

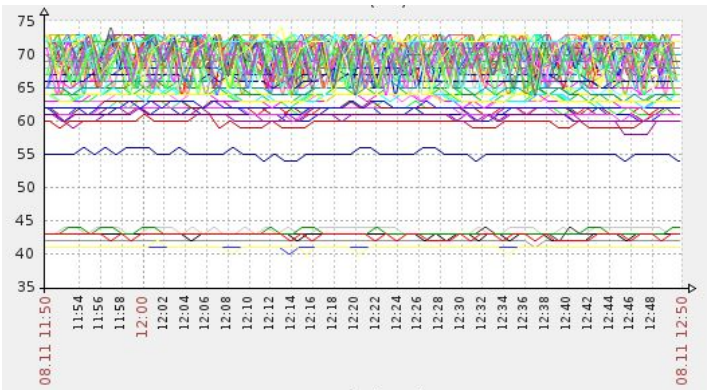

Figure 3.22: Zabbix screen shot of an overlay graph with the temperatures of all 74 C-RORC FPGAs in the HLT.

On top of the general hardware monitoring, there is a periodic firmware dependent check in the *CRORCPublisher* and *CRORCOutWriterSubscriber* processes responsible to receive data from or push data to the C-RORCs. Each of these processes handles data as well as status information from one single link. The status of each active link and the dataflow through the firmware is queried once per second from these processes. This enables to spot faulty optical links via link error counters, invalid DDL states, FIFO fill states and possible overflows or buffers running full. Errors are reported into the HLT framework logging system together with the logging information of all framework processes. This information is available is stored in a database for online monitoring or later inspection. A status dump of all firmware registers is stored in a file at the end of each run to document the final state of the firmware. This is especially helpful if a run was stopped due to an error condition. The link-level metrics and the general dataflow related

measurements are asynchronous to the dataflow, which means it is neither required nor possible to correlate a certain condition to a specific block of detector data.

Status data that belongs to a certain detector data block has to be integrated into the detector data stream. Two examples where this was done are the protocol error flags from the DDL and errors from the raw data decoder in the cluster finder. These flags indicate an error condition that belongs to a certain sub-event. For this reason, they have to be transported with the corresponding data block in order to be usable. The *ReportBuffer* entry has some bits available that can be used to transmit per-event information to the software part. The software can evaluate these bits and react on them. Sub-events with the DDL error flag set are treated very carefully and are possibly discarded or truncated in software. Cluster finder data decoding errors indicate protocol errors in the incoming data stream from the detector, the front-end cards, or the RCUs. They are used to monitor the fraction of corrupt input data. With this system in place, excessive protocol errors can be reported to the TPC team immediately while running.

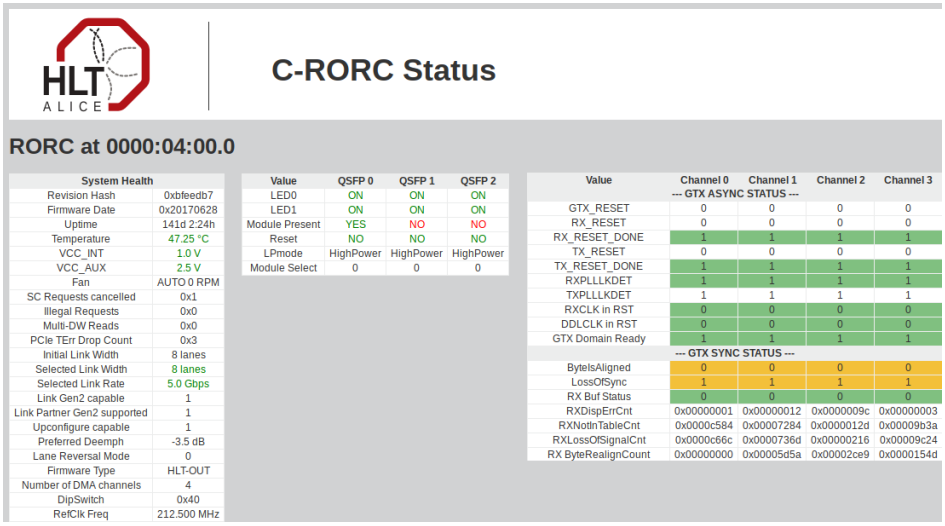

Figure 3.23: Screenshot of the C-RORC development web GUI showing the status of most internal registers, partly with color codes to quickly visualize possible problems.

During the firmware development phase, a web-based monitoring interface was built to quickly visualize the status of a board. A screenshot is shown in Figure 3.23. A large fraction of all software-accessible registers in the firmware is shown in a table. Colors indicate the state of different values. The graphical representation with colors provides an overview of the full firmware at a glance and enables to spot error conditions immediately. While this is a valuable tool for development, it clearly does not scale to an environment with C-RORCs installed in 74 different machines. Nevertheless, the tool is integrated into the cluster software repository and can be installed on the production machines at any time if needed.

3.10.4 Verification of the DDL Connectivity

For every DDL, it is crucial to know exactly where it is connected to. The sub-events transmitted via DDL do not necessarily contain information about their remote link partner. Corresponding data coming via different links can only be merged and subsequently reconstructed correctly if the origin of each fragment is validated. The fact that optical power is detected or both ends successfully speak the DDL protocol only means that there is a connection to some link partner, not to a specific one. The detector teams and the DAQ team achieve a validation of the link map by reading an identification number from the SIUs on the detector front-end electronics. The detector teams make sure the mapping of SIUs to the detector itself is correct. For the HLT, this method cannot be used. The reason is that the links from the DAQ to the HLT system are using DIUs on both ends of each link. DIUs do not provide link identifiers. The fiber interconnects

between the DAQ and the HLT systems are maintained from three different parties. The DAQ team ensures the connection between the DAQ C-RORCs, D-RORCs, and the patch panels in the DAQ counting room. An external contractor maintains the connections between the different patch panels. The HLT team connects the HLT C-RORCs to the patch panels in the HLT counting room. Retrospectively, all three parties had inconsistencies in the initial setup of the fiber installation. With more than 500 DDL pairs already on HLT side and roughly double the number of fiber pairs on the DAQ side, this is rather expected than surprising.

The DDL connection verification for the HLT could be realized with special command words sent via each DDL. It assumes that the mapping of DAQ DDLs to the detectors is verified. The DAQ system then sends the link identifier of each DDL to the HLT via the corresponding link. This works equally for the HLT input and output links. All HLT firmware images support capturing this specific status word, independent from the firmware state. The received value can be read via software without having to start a data-taking run. The links can be verified by checking if a link identifier was received and if the identifier matches the expected value from the HLT framework configuration. This method was used each time the fiber installation was touched or modified, when broken nodes were replaced, or after the balancing of the detector links was optimized.

Chapter 4

Hardware Cluster Finding

The hardware cluster finder is an FPGA-based data processing core to find clusters in the space charge distribution contained in raw TPC data. With a streaming implementation tailored to the input data rate, it provides high-performance data processing capabilities already in the C-RORC FPGA. Processing the data immediately in the readout FPGA introduces only a negligible additional latency compared to the raw data readout because the data is going through the FPGA anyway. The possible processing steps are only limited by the number of available FPGA resources or clock frequency requirements. The hardware cluster finder replaces the raw TPC data with the clusters found in the data before they are handed over to the DMA engine and transferred to the host machine. The algorithm was already developed for and integrated into the Run 1 HLT readout boards. This implementation is the starting point for the cluster finding in the Run 2 HLT with the C-RORC, as well as optimizations and extensions of the processing core during this work.

4.1 Cluster Finder Fundamentals and Run 1 Implementation

4.1.1 TPC Readout Architecture

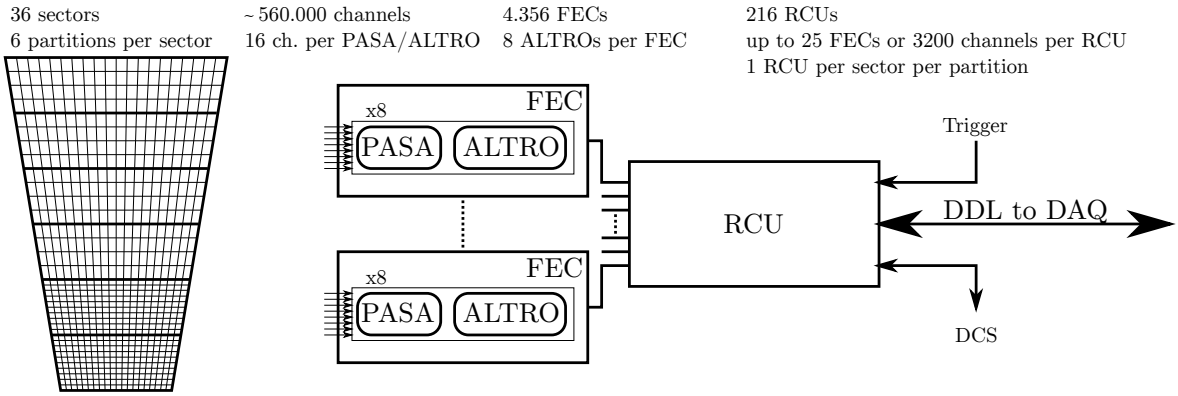

Figure 4.1: Sketch of the TPC readout architecture.

The TPC detector and its working principle was sketched already in Section 1.3 and is described in detail in [Del+ 00]. Collision products traversing the TPC volume create electron-ion pairs along their trajectory through the gas. A high electric field makes the electrons drift towards the TPC end caps where the electron distribution is recorded with readout chambers over time. Figure 4.1 gives a rough overview of the components in the readout chain from the arrays of pads in the readout chambers to the online systems. Front-end cards (FECs) located only some 10 cm away from the pads apply a pre-amplification and a shaping (PASA) of the analog detector signals.

These signals are then digitized and stored with a custom ASIC chip, the ALTRO [Bos$^+$ 03]. The ALTRO performs a baseline correction, tail cancellation, and zero suppression. Each ALTRO processes 16 channels and each FEC contains eight ALTROs, resulting in 128 channels processed by each FEC. The Readout Control Unit (RCU) [Gut$^+$ 05] then bundles the data from up to 25 FECs and provides the interface to the Data Acquisition system via the optical Detector Data Link. This chain creates a digitized projection of the space charge, created by the original particle trajectories, onto the end caps of the TPC. Characteristics of the gas and the pads in combination with the amplification and the shaping of the analog signals define the response of the detector in the time and the pad direction. The position of each pad is known from the detector geometry. From the raw data perspective, each space point inside the TPC volume is equivalent to a sector, partition, row and pad number of a specific channel. The position along the beam axis can be calculated from the timestamp of the charge sample relative to the trigger time in combination with the known drift time of the electrons in the gas. The local maxima in the raw data signals due to the projection of a particle trajectory onto the pads of a padrow are called "clusters" here. The cluster finder algorithm searches for these signatures and calculates their position and width in the time and the pad direction. The projection of charge distributed across multiple pads as well as the temporal sequence of signals enables to locate the clusters with a precision below the intrinsic resolution of the pad size and the timestamp granularity. Clusters are later correlated and assigned to "tracks" in the tracking phase in the HLT.

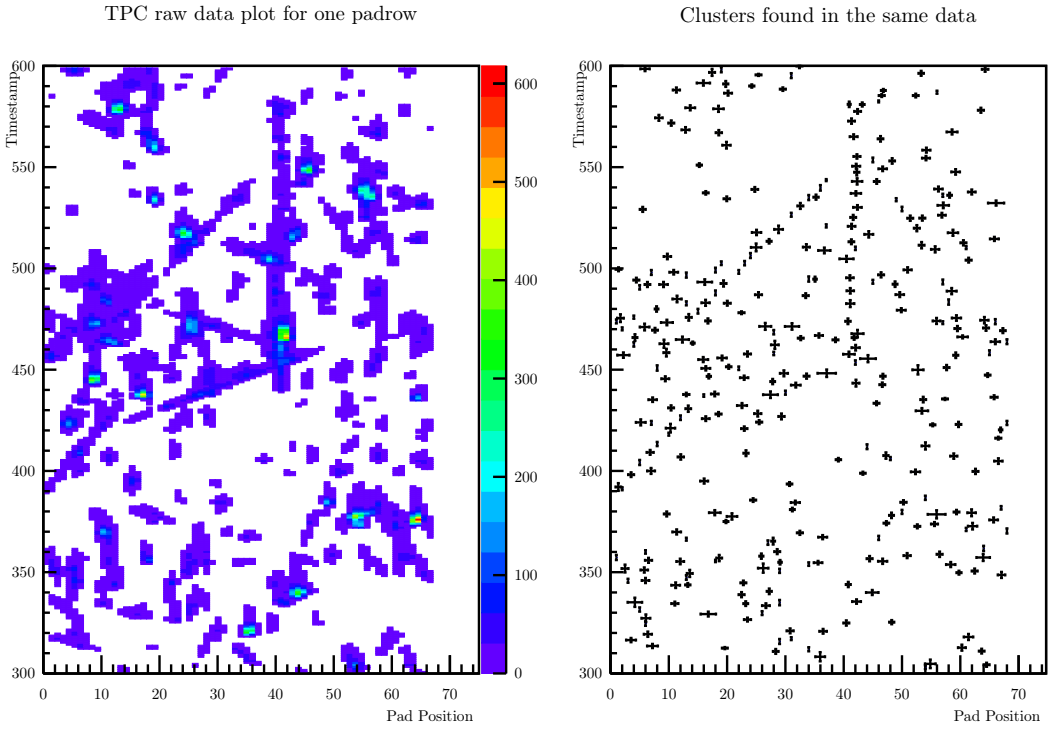

Figure 4.2: Left: TPC raw data from one row of one partition. The color indicates the charge. The timestamp range is reduced for readability reasons. Right: Cluster positions and widths in the pad and the time direction found by the hardware cluster finder in the same dataset.

The HLT receives a copy of the detector data sent to the DAQ system via DDL. The data received via one link contains all signals from all channels of one TPC partition. A total of 216 links cover all six partitions from each of the 36 sectors. The ADC samples from the detector pads are encoded in the DDL data. An important aspect for hardware-based online data processing of raw data is the ordering of this data as it is coming from the RCU. The first generation of RCU boards as used during Run 1 and up to the end of 2015 in Run 2 had limited buffering capabilities. For this reason the readout was done in two interleaved branches per TPC padrow:

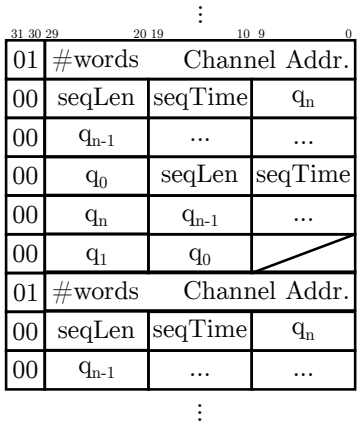

Figure 4.3: Cluster finder input data format. The figure shows a sample temporal sequence of 32 bit data words received via DDL with the TPC specific encoding of bit fields.

Each row of pads is divided approximately in half. With 68 pads in the most inner row of partition 0, pads 0–33 belong to branch A, while pads 34–67 belong to branch B. Interleaved readout means that the first samples come from pad 0, then pad 34, followed by pad 1, then pad 35, and so on. The second generation of RCU boards, installed in early 2016, has sufficient buffering capabilities to provide the data from all pads in numerical order. The update of the RCU boards and the implications to the HLT cluster finding are described in more detail in Section 4.2.3. The data from each pad is a temporal sequence of charge samples together with timestamps. Figure 4.2 on the left side shows an example of raw data. This is a sample of data received from the innermost row of the inner partition of one sector. The white areas in the graph indicate data regions that were already suppressed by the ALTRO as part of the zero suppression mechanism. The color indicates the charge value at the given pad and time as sampled by the ADCs in the ALTROs. The data in this figure is received in the RORCs from high to low timestamp values for each pad. One pad after the other is read out, eventually with the described interleaved pad readout with the first generation of RCU boards. Both RCU variants provide the data from one row after the other in incremental order.

The DDL interface to the DAQ and HLT systems is a 32 bit bus. This means that all data from the detectors is also a multiple of this width. Additionally, detector data over the DDL follows an event data format with a predefined header and a detector specific payload encoding. The detector specific encoding for TPC is shown in Figure 4.3. TPC charge samples and timestamps are 10 bit wide. Three samples are encoded into one DDL word. An RCU header word provides the number of words and the hardware address of the originating channel for the following data. The samples are then encoded into sequences with a given length, the timestamp of the first sample and the corresponding charge samples themselves. Several sequences can follow a common header. The channel address in the header word can be used to look-up the physical position of the corresponding channel in the detector. The data format is described in more detail in [ALI 11].

4.1.2 Cluster Finder Overview and H-RORC Implementation

The pad and the time response functions of the TPC provide characteristic signals of charge distributions inside the TPC volume in the pad and the time direction. The first step in the data processing is to find these signatures in the pad and the time direction in the raw data and to calculate their properties. For this process only data within the same row is relevant. Different rows are treated independently. Correlations of clusters across rows, partitions and sectors are done in a later stage in the HLT processing chain during the tracking phase. The clusters found during the cluster finding process typically span across a few pads and a few timestamps. This enables to calculate the positions and the widths of these distributions at a precision below the intrinsic resolution of the pad size and the timestamp granularity. Both functions can be

described with sufficient accuracy using Gaussian distributions [Alt 17]. The properties of the Gaussian distributions are derived from center-of-gravity and standard deviation calculations. This can be implemented efficiently in hardware in a streaming architecture. The underlying math consists of weighted sums of the charge samples with the pad number, the timestamp or their squared value as weighting factors. This gives the formulas shown in Equations 4.1 to 4.5 for the calculation of cluster properties. A more detailed analysis of how these formulas are derived is shown in [Alt 17]. The black terms are calculated in hardware, the blue terms later on in software. $q_{p,t}$ is the measured charge at the corresponding pad and time in the pad-time plane as shown in Figure 4.2. The summation is always done over all samples in this plane that contribute to a specific cluster.

$$Q_{tot} = \sum_{p,t} q_{p,t} \tag{4.1}$$

$$Pad = \frac{1}{Q_{tot}} \times \sum_{p,t} q_{p,t} p \tag{4.2}$$

$$Time = \frac{1}{Q_{tot}} \times \sum_{p,t} q_{p,t} t \tag{4.3}$$

$$\sigma^2_{Pad} = \frac{1}{Q_{tot}} \times \sum_{p,t} q_{p,t} p^2 - Pad^2 \tag{4.4}$$

$$\sigma^2_{Time} = \frac{1}{Q_{tot}} \times \sum_{p,t} q_{p,t} t^2 - Time^2 \tag{4.5}$$

The implementation of the hardware cluster finder is sketched in Figure 4.4. An input FIFO decouples the core from the optical link and enables the cluster finder to operate at a clock frequency higher than the DDL clock. The RCU Decoder decodes the incoming 32 bit data stream from the optical link into a serial sequence of charge samples. A serial stream of samples greatly simplifies the implementation of the cluster property calculation in the time direction because the temporal neighbors of a sample are at a fixed distance earlier or later in the pipeline. With up to three samples per DDL word, the RCU Decoder has to run at least with three times the clock frequency of the DDL interface, assuming an uninterrupted data stream from the RCU. The mapping between the channel addresses and the physical pad positions is available in a block RAM that is filled with the corresponding data before the readout is started. The channel address from the headers of the incoming data stream is used to look up the row and pad numbers for the following samples via this RAM. The look-up additionally provides the option to mask single channels and to apply a per-channel gain correction factor to the raw samples. This provides the option to compensate for differences in the charge sensitivity of the individual channels.

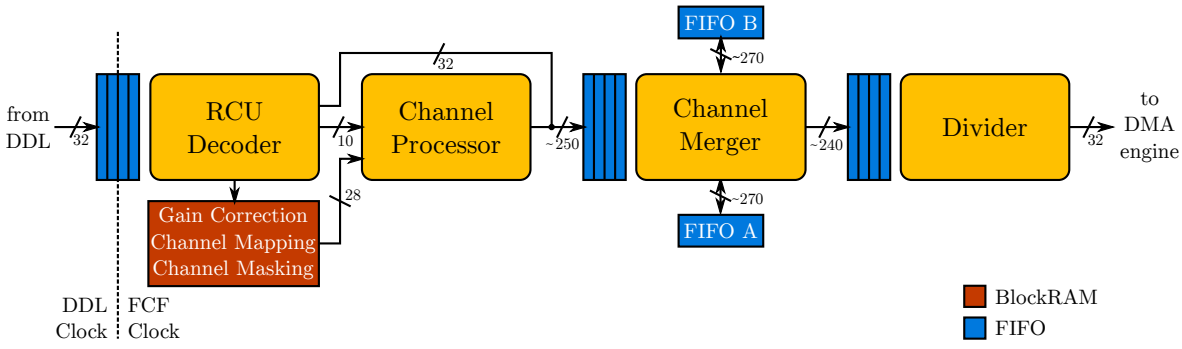

Figure 4.4: Dataflow through the hardware cluster finder building blocks from the DDL interface towards the DMA engine.

The way the data is provided from the RCU, and the fact that the weighted sums are additive across contributions from different pads, enables to process the data in two steps. At first, the

stream of samples is only analyzed in the time direction on a per-pad base to find peaks and pre-calculate properties. The resulting cluster candidates contain the weighted sums for all properties from only one pad in the time direction. This calculation is done in the Channel Processor. In a second step, the candidates found in the time direction are merged with close-by candidates from the neighboring pad in the Channel Merger. The Channel Merger is implemented as a state machine that compares the current input candidate to the output of a local FIFO queue. Merged candidates are placed into the local queue for comparison with further neighbors. The interleaved readout from two branches described above requires to handle the cluster candidates from the different branches independently. For this reason, two FIFOs are used inside the Channel Merger to correlate the data from neighboring pads, one for each branch. Completed clusters are pushed out towards the Divider. The way the Channel Merger works is described in detail in [Alt 17]. The output of the Channel Merger is the total charge from Equation 4.1 together with the weighted sums from Equations 4.2 to 4.5, plus a row number and a few flags. Up to this point, all calculations are done in fixed-point arithmetic and do not cut any precision.

For the calculation of the final time and pad values, as well as their deviations, they all have to be divided by the total charge of the cluster. This division is done in single-precision floating-point representation. A division is an expensive operation in hardware in terms of resource usage or throughput. For this reason, only a single but fully pipelined divider instance is integrated into the hardware design. This instance is used for all four divisions in a time-sharing mechanism. It implies that the Divider can process a new cluster only every fourth clock cycle. In combination with the output data format of 6 DWs, new input data is requested only every sixth clock cycle. This is sufficient for the Divider due to the architecture of the prior processing steps and a verification of this claim is shown later in Section 4.2.4. The subtractions of the squared mean values for the final deviations of the pad and the time values, shown in blue in Equations 4.4 and 4.5, are not done in hardware. They are done at a later stage in software to additionally save FPGA resources.

In order to handle overlapping clusters appropriately, the cluster finder can split sequences both in the time and the pad direction. This mechanism creates independent and localized clusters instead of a single but wide cluster. The Channel Processor detects local minima in the sequence of charges within a pad and splits the property calculation into two cluster candidates with different timestamps. The Channel Merger detects local minima in the sum of charges in the pad direction and also splits the cluster candidates into two clusters.

The hardware cluster finder contains a number of runtime configuration options and filters that enable to tune the cluster finding process without rebuilding the FPGA configuration. Peak detection in the time direction can be made more resilient to noise by requiring a minimal difference between successive ADC samples before treating the overall slope as rising or falling. Lower limits on the sum of charges in the cluster candidates after the Channel Processor and the clusters after the Channel Merger enable to discard low-signal clusters. Single pad clusters can be suppressed entirely. The temporal distance up to which cluster candidates from neighboring pads are merged is configurable.

The output of the hardware cluster finder is a list of data structures of six 32 bit words per cluster, framed by a Common Data Header and several RCU trailer words. The encoding of the cluster properties into this data structure is shown in Table 4.1. The originating TPC sector and partition are known via the DDL connection. A row number gives the row offset of a cluster in the given partition. The maximum charge and the total charge of the cluster are provided as fixed-point values. The mean cluster position in the pad and the time direction, as well as the intermediate widths in both directions, are provided as single precision floating-point value. The last two words are the weighted sums divided by the total charge, which are required to calculate the final widths. This leaves the subtraction of the squared mean values for the first software processing steps.

All cluster finder modules contain their own flow control logic to handle the data transport from the input FIFO towards the output FIFO of each module. Each module stops processing new input data if its output FIFO runs close to full. This ensures that no data is lost during processing,

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DW0	1	1	(uint6_t) Row						0	(fixed_t<11,12>) MaximumCharge																							
DW1	00		(fixed_t<18,12>) TotalCharge																													
DW2	(float32_t) Pad																															
DW3	(float32_t) Time																															
DW4	(float32_t) PadError																															
DW5	(float32_t) TimeError																															

Table 4.1: Hardware cluster finder output data structure for clusters.

also in a high-load scenario. Multiple FIFO stages within the algorithm provide a decoupling of the individual processing steps. If one FIFO runs full, the earlier stages will gradually fill up. However, it also means that the implementation can potentially stall the readout if the input FIFO of the cluster finder runs full. A full input FIFO makes the DDL stop accepting new data and ultimately stops the trigger for the given detector. This condition is typically called "back pressure" or "XOFF" and should be avoided to the greatest possible attempt.

4.1.3 Software Implementations

There are two conceptually different software implementations of the TPC cluster finder in ALICE. The Offline cluster finder works on full event information. It searches and fits Gaussian distributions onto windows of five time bins by five pads to calculate cluster properties. This implementation is the reference in terms of the physics performance. However, it cannot easily be implemented in an FPGA which led to the hardware implementation as described above for Run 1. The output of the cluster finder as implemented in the FPGA was compared against the reference Offline implementation by TPC and Offline experts to confirm its physics performance. These checks were done both at the start of Run 1 and again at the start of Run 2 before any raw data was discarded in favor of HLT compressed clusters. Additional verification runs were done in 2017 with the improved cluster finder variant described later Section 4.3.2. In order to run this comparison, a bit-wise equivalent software implementation of the hardware cluster finder, called the *HWCFEmulator*, can be run in the Offline framework without the need of an H-RORC, a C-RORC or the HLT framework. This implementation uses a software model of the streaming algorithm of the FPGA core and provides an output that is bit-wise identical to the hardware cluster finder.

Performance measurements of the H-RORC hardware implementation against the *HWCFEmulator* and another software implementation on different CPUs available from 2009 to 2013 were done by Torsten Alt as part of his Ph.D. thesis [Alt 17]. These measurements show an equivalent of 8.5 to 19.7 CPU cores per hardware cluster finder instance in order to achieve the same processing performance as the FPGA implementation. The measurements assume an input data stream of 160 MB/s as present in the HLT during Run 1.

4.2 Cluster Finding in the HLT in Run 2

4.2.1 Porting the existing Cluster Finder to the C-RORC

The cluster finder as used during Run 1 was implemented on the H-RORC FPGA. This board contains a Xilinx Virtex-4 FPGA which is two generations older than the Virtex-6 FPGA on the C-RORC. Due to advances in FPGA technology, embedded device specific components like block RAMs and DSP blocks changed from Virtex-4 to Virtex-6. For this reason, the existing code could not directly be reused on the C-RORC. All IP cores for block RAMs, FIFOs, multipliers, format conversions, and the division had to be re-generated for Virtex-6. Differences in the latency of the new cores compared to the previous variants were manually compensated with shift registers on the dependent signal paths. A general challenge for the implementation of the cluster finder in the C-RORC is the increased link parallelism per board. The C-RORC implementation for Run 2 requires six cluster finder instances per board to handle a full TPC sector, compared to two

instances in the H-RORC during Run 1. Any change in resource usage per instance translates into a factor of six resource usage change for the overall design. A major difference between the C-RORC implementation and the original code is the output interface. The H-RORC had a 64 bit PCI-X interface to the host machine. The output interface of the cluster finder was also designed as a 64 bit interface for this reason. The PCI-Express bus on the C-RORC internally works with packages in multiples of 32 bit words. Breaking the existing 64 bit interface down into two 32 bit words was one option. However, the input to the cluster finder is a stream of 32 bit words from the DDL and most of the cluster properties were 32 bit values as well. This motivated a rewrite of the output formatting module of the cluster finder to directly deliver the clusters as a stream of 32 bit words. This additionally makes the input interface of the cluster finder identical to its output interface. The identical interfaces make it easy to optionally include the processing core where needed, without changes to the surrounding firmware logic.

Resource Usage	original H-RORC impl. (Virtex-4)	initial C-RORC port (Virtex-6)	C-RORC DSP port (Virtex-6)
Flip-flops	7015	6677	5169
Look-up tables	6314 (LUT4)	5802 (LUT6)	4274 (LUT6)
DSPs	10	0	8
Block RAMs	39 (RAMB16)	23 (RAMB18/36)	23 (RAMB18/36)

Table 4.2: Cluster finder resource usage comparison for the initial H-RORC implementation and the two C-RORC variants.

Table 4.2 compares the FPGA resource usage for the original H-RORC cluster finder implementation to the initial C-RORC port of the design. The numbers contain all resources belonging to one cluster finder instance after the place-and-route step of the full firmware image. The table shows that the overall resource usage was reduced from Virtex-4 to Virtex-6 for all resource types. This has several reasons. One reason is the increased look-up table size with six inputs on Virtex-6 FPGAs compared to four inputs on Virtex-4 FPGAs. This enables to implement more combinatoric logic or more shift registers into the same amount of LUTs and reduces the overall number of elements required. The internal capacity of the block RAMs increased from 16 kBit to 36 kBit per instance. Each 36 kBit instance can as well be used as two 18 kBit instances, depending on the required width, depth, and clock conditions. This almost halves the number of block RAMs required. A significant difference in the resource usage is in the divider stage of the cluster finder. The Virtex-4 floating-point division implementation uses DSP blocks for this step. The Virtex-6 FPGA achieves the same operation without DSPs and with only a slightly higher number of flip-flops and look-up tables.

Exploiting the availability of DSP cores for the fixed-point multiplications in the Channel Processor step enables to further reduce the overall resource usage. Seven multiplication operations are done in this step. Six of them have input and output widths below the intrinsic width of the DSP cores and can, therefore, be implemented with one DSP block each. One multiplication exceeds this width and requires two DSP blocks. Using the DSPs for the multiplication saves around 25 % of look-up tables and flip-flops compared to an implementation without DSPs. The resulting resource usage is shown also in Table 4.2 in the rightmost column.

A positive side effect of the C-RORC cluster finder implementation is an improved floating-point implementation in Virtex-6 FPGAs. While the cluster finder output from the Virtex-4 implementation may differ from the software reference implementation by one least significant bit in the binary representation of the floating-point values, the results from the Virtex-6 implementation are bit-wise identical to the software model. This simplifies the verification of the implementation. The software reference results and the hardware results can now be compared with any binary `diff` tool in a data-format agnostic way.

4.2.2 Hardware Cluster Finder C-RORC Co-Processor Implementation

The correct operation of the cluster finder is crucial for the ALICE data-taking operation. In the target HLT operation mode, the raw TPC data is discarded in the DAQ system and only the compressed HLT clusters are stored. Errors in the cluster finder could, therefore, lead to wrong physics data being stored without options to recover. This means that the output of the cluster finder must be consistent and has to represent the contents of the original raw data at any time.

A verification of the hardware implementation is possible with different methods at different levels. The most details of the internal behavior of the cluster finder are available with a hardware simulation of the cluster finder implementation from the HDL sources. Every single internal bit can be traced and analyzed with HDL simulations. This is used to systematically verify individual modules using unit tests. Simulating a full cluster finder instance with all processing steps in this way is possible as well and is used to compare hardware and software processing results. However, the hardware simulation at this level is comparably slow. Simulating the processing of only a couple of hundred kB of input data in one cluster finder instance already takes several minutes. Verification at this level is only possible for some hundred to maybe a few thousand sub-events, whilst simulation times can easily go into the order of days. An automated HDL simulation environment with some sub-events that can be tested within a few minutes was used to catch regressions during development. A formal verification of the implementation is not possible due to the possible range of input vectors. The HDL simulation of selected cases cannot be used to verify a hardware implementation with enough statistics in a reasonable timescale. Using only HDL level hardware simulation is, therefore, not sufficient for a reasonable verification.

A test setup as close as possible to the final implementation could consist of two C-RORCs interconnected with a DDL. One C-RORC with the HLT output firmware could be used to read raw data from disk and send it via the DDL. A second C-RORC with the HLT input firmware including the cluster finder would be used to read the data from the DDL, process it and send the results to the host machine. The disadvantages of this approach are the hardware requirements and the control flow. Two C-RORCs plus transceivers plus optical fibers would be required for this interconnect. This setup is also not available in the HLT cluster. Additionally, only one C-RORC is typically installed in each node. The output side of the chain needs to know to which input file the current results belong to in order to compare the output with the corresponding reference file. Having this functionality distributed across two nodes makes the software part more complicated.

Another verification option is to use the on-board DDR3 RAM with data replay. A set of raw TPC sub-events is loaded into the RAM and then replayed. The amount of data that can be replayed at a time depends on the amount of on-board memory available per channel. Additionally, the upload interface to transfer replay data from the host into the on-board memory is significantly slower than the data paths between the host and the DDL. This is not a limitation for the HLT cluster-wide data replay, because the data is pre-loaded and then replayed continuously in this case. For the cluster finder validation, however, this is limiting the throughput considerably. While this method is technically suitable to verify the cluster finder implementation with much higher statistics than the HDL simulation approach, the modular structure of the overall firmware architecture provides an easier and more efficient solution.

By connecting the existing firmware blocks for the DMA host-to-device transfers, the cluster finder, and the DMA device-to-host transfers, a co-processor-like PCI-Express device is created to test the cluster finder. All required firmware blocks already exist from the HLT input and output firmware variants. A schematic drawing of the building blocks is shown in Figure 4.5. The host software loads the raw data into a DMA-accessible host buffer. The C-RORC fetches this data via DMA, pushes it through the cluster finder and provides the results back to a host buffer via DMA. The host software accesses this buffer to compare the results with a reference file or write it to disk. With this approach, the processing capabilities of the firmware are mostly limited by the disk read or write bandwidth of the host. This setup provides a method to quickly create or check large amounts of cluster finder results. Using the co-processor implementation

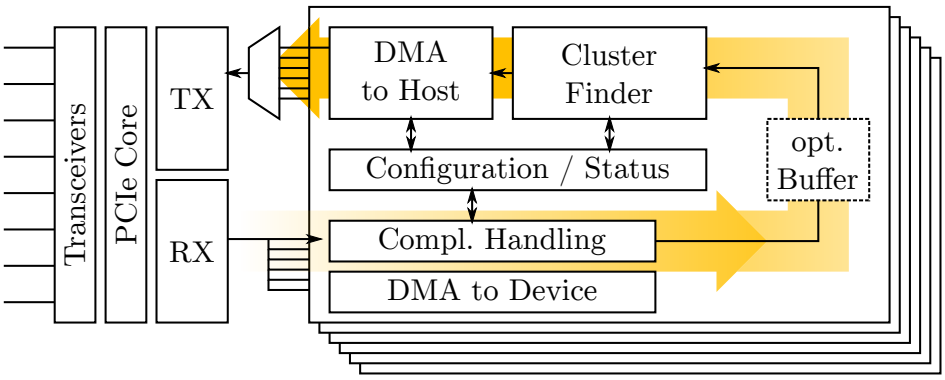

Figure 4.5: Building blocks of the cluster finder co-processor firmware.

provides a test data throughput which is orders of magnitudes higher than what is possible with the HDL simulation. This approach typically provides a throughput in the order of a couple of ten Megabytes per second. This enables larger datasets to be used for verification. The reference data can be obtained from the cluster finder emulator.

In order to characterize the processing of single sub-events, the same setup is used with a slightly different software. Instead of queuing up as much sub-events as possible, only one sub-event at a time is pushed through the chain. After each sub-event, the internal firmware performance counters are read back to get information on the processing characteristics. The optional buffer in the processing chain ensures that the input stream into the cluster finder is uninterrupted, which is crucial to measure busy times of the cluster finder for individual sub-events.

This co-processor implementation contains six instances of the cluster finder. By configuring each instance with the channel mapping information of one of the six TPC partitions, this firmware image can be used to process sub-events from any TPC DDL. A ZeroMQ[1] endpoint in the host software enables to provide lists of raw input files, reference files for comparison and target output files at runtime. Several hundred Gigabytes of raw TPC detector data from different physics runs from the last couple of years were used for the verification of the cluster finder. This combination of hardware, firmware, and software was used for the characterization and the studies of the different cluster finder implementations described in the following sections.

4.2.3 RCU2 Support

The TPC Readout Control Units (RCUs) pull the TPC raw data from the front-end cards (FECs) and send them to the DAQ system as described in Section 4.1.1. The first generation of RCU boards (RCU1) was installed into the TPC before the start of Run 1 and was used throughout Run 1. Data rate limitations with respect to the expected Run 2 conditions, as well as stability issues, lead to the development of the second generation of this board, the RCU2. The RCU2 project was started only after the development of the C-RORC and was, therefore, a strong reason to build the C-RORC as flexible as possible. The RCU2 was not yet available at the start of Run 2 mid-2015. It was installed into the TPC at the beginning of 2016. The data rate limitations of the RCU1 could be solved by two major changes in the RCU2. One improvement was to use the DDL protocol at a higher link rate, improving the bandwidth between the RCU and the DAQ system. The second major improvement was the change of the internal readout structure from two to four branches, doubling the bandwidth between the RCU and the FECs. Both changes have direct consequences for the HLT readout of the TPC links and especially the hardware cluster finder.

While the RCU2 hardware was designed to support link rates up to around 5 Gbps, the target link rate for the operation in Run 2 was changed several times throughout the development due to

[1] Distributed messaging framework, www.zeromq.org

technical difficulties. The final DDL rate was chosen to be 3.125 Gbps, providing roughly double the bandwidth of the RCU1 DDL implementation. For a raw data readout, only the link rate itself is relevant. The selected link rate of 3.125 Gbps is well within the supported range of the HLT input firmware for raw readout and would not need special adjustments. In combination with the hardware cluster finder, however, the situation gets more complex. The cluster finder has to be integrated in a way that it can handle the maximum data rate from the TPC in order not to stall the readout. The doubled detector readout bandwidth within the RCU2, as well as the increased DDL signal rate, result in a higher bandwidth for the TPC data into the HLT. The cluster finder implementation has to be adjusted to cope with the new data rates. Section 4.2.4 describes the challenges of scaling this implementation to the target link rate in detail.

The RCU1 used two parallel internal branches to read out the front-end cards. A fully sorted data stream was not possible due to resource limitations in the RCU1 FPGA. This led to the interleaved readout scheme described in Section 4.1.1. The cluster finder used with the RCU1 in Run 1 implements an independent handling of data from the two branches for this reason. The Channel Merger contains two buffer FIFOs, one for each branch. This enables merging of cluster candidates from neighboring pads even if the readout provides an interleaved stream of samples from both branches. The merger stage of the cluster finder is a critical point in terms of combinatorial complexity, data path width, and, therefore, the maximum clock frequency and the overall throughput. Cluster candidates from the merger input FIFO have to be correlated with partially merged clusters from the buffer FIFOs and potentially queued, discarded or pushed out. A data path width of around 250 bit in combination with data dependencies across different block RAMs limits the maximum clock frequency. Furthermore, the interleaved readout creates an increased number of clusters at the branch boundaries. This is due to the fact that corresponding cluster candidates from the other branch are not available in the merger at the same time. These clusters have to be merged later on in software.

The RCU2 doubles the bandwidth from the FECs by internally using four branches in parallel. An improved readout scheme in the RCU2 firmware makes sure that the extended parallelism translates into the required increasing data throughput. The RCU2 FEC readout parallelism with four instead of two readout branches also affects the HLT cluster finder. Switching from two to four readout branches in the RCU2 in combination with the previous interleaved readout scheme would have made the branch dependent data handling in the cluster finder even more complex. Interleaved readout from four branches required four independent buffer FIFOs in the Channel Merger. This would have increased the logical complexity as well as the resource usage significantly and would have decreased the maximum clock frequency of the Channel Merger. The effect of the increased number of clusters found near the branch boundaries would be even reinforced with four branches. Fortunately, the RCU2 FPGA has enough resources available to implement a branch merging step already in firmware. This completely hides the branched readout process from the HLT processing. The RCU2 provides a stream of raw samples fully ordered by time, pad and row. With this branch merging in the RCU2, the Channel Merger in the cluster finder does not need to distinguish between data from different internal readout branches anymore. The problem with the increased number of clusters found near the branch borders with the RCU1 automatically gets solved with this ordering as well.

Figure 4.6 shows an example of this effect. An increased number of clusters around the border between the two branches is clearly visible after the cluster finder with the RCU1 readout (left). Providing the same dataset to the cluster finder in the order as it would come from the RCU2 shows an even distribution of clusters across the pads. The total number of TPC clusters found in the HLT hardware cluster finder is reduced by around 1–2 % due to the branch merging in the RCU2. If only data from the RCU2 has to be handled, the cluster finder could be stripped-off the dual branch handling logic. This would reduce the resource consumption and improve the timing performance. However, this also brings some limitations to the overall HLT benchmark and standalone testing system using the C-RORC data replay functionality described in Section 3.7. The set of raw data samples for HLT tests until the end of 2015 only consisted of testing data from the RCU1. Especially Pb–Pb reference data for HLT stress tests are only available from physics runs in 2012 and 2015, all recorded with the RCU1. p–Pb data became available at the

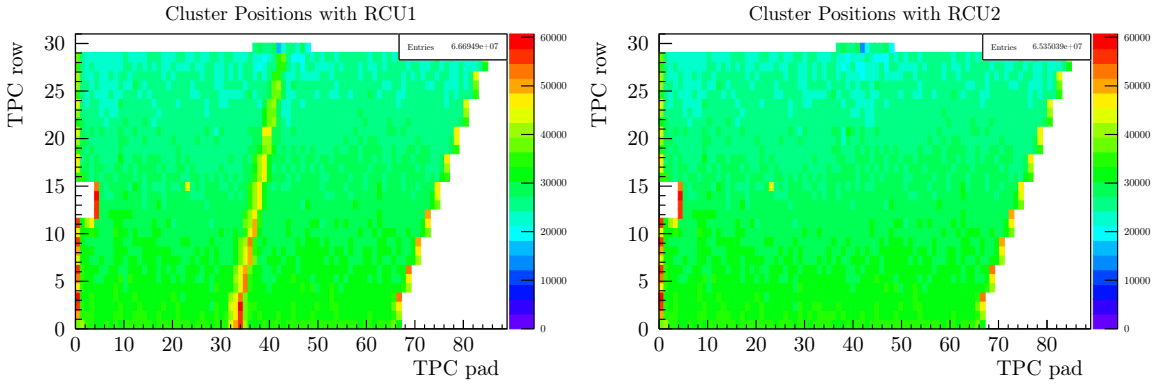

Figure 4.6: Distribution of cluster positions from RCU1 data (left) with an increased number of clusters near the branch boundaries and from the same data converted to the RCU2 data format (right). The graph superposes the cluster positions in the innermost partition from all sectors.

end of 2016 and the next ion run period is planned for the end of 2018. Therefore, a cluster finder implementation that can handle the data from both the RCU1 and the RCU2 was clearly preferred. This approach is more demanding in terms of resource usage and clocking performance. Nevertheless, it provides much more flexibility. It was possible to extend the cluster finder to support both formats. A runtime configuration option selects if data stream should be treated as RCU1 or RCU2 data.

4.2.4 Link Speed Scaling Analysis

Component	Initiation Interval (II)	Description
RCU Decoder	$\leq$ 3	Decode incoming data, apply calibrations, serialize 10b ADC samples.
Channel Processor	1	Find peaks in the sequence of ADC samples, select ROIs, separate overlapping ROIs, calculate cluster candidate properties.
Channel Merger	$\geq$ 3	Merge candidates from neighboring pads.
Divider	$\leq$ 4	Divide cluster properties by total charge.

Table 4.3: Breakdown of the throughput of the individual cluster finder processing steps.

The original cluster finder is tuned to be able to handle the data rates received from the TPC during Run 1. With the doubling of the readout bandwidth, the protocol changes, and improvements of the data quality with the RCU2 in Run 2, the throughput of the cluster finder implementation had to be scaled accordingly. The cluster finder is implemented in a way that it can handle burst data transfers at the maximum DDL data rate without stalling the readout. In order to achieve this, each processing step in the cluster finder has to fulfill this requirement or a derivative thereof on its own. A breakdown of the processing steps with respect to their throughput is shown in Table 4.3. The Initiation Interval (II) specifies how frequently, in terms of clock cycles, new input data can be provided to the module's input interface. The RCU Decoder step has to serialize up to three 10 bit ADC samples from each 32 bit input word and, therefore, requires up to three cycles per input. Header and trailer words can be processed faster because they do not generate output. The Channel Processor is a pure streaming processor. Its calculations depend on a window of samples before and after the current sample. This is implemented

as a pipeline and can accept new input data on each clock cycle. The Channel Merger is the only component in this chain that has a data-dependent processing time. It potentially has to buffer all previous candidates from one pad until the corresponding timestamp of the next pad is received or exceeded. The dependency between the data words from the input FIFO and the outputs of the merger FIFOs require at least three cycles to buffer the values before using them in calculations, plus any cycle required to partially or fully flush the merger FIFOs. The Divider comes again with a deterministic II. Each division depends only on the current input data. The reason why this step is implemented with an II greater than one is to save FPGA resources by sharing the same divider instance for all division operations.

The throughput of the RCU Decoder is sufficient as long as its clock frequency is at least three times the clock frequency of the 32 bit words from the DDL. The Channel Processor is fully pipelined. If it is running with the same clock as the RCU Decoder, it cannot stall the processing. The critical part in the chain is the Channel Merger. It needs at least three cycles per input word, possibly more depending on the data from the TPC. On the other hand, a cluster candidate can only be found up to every second ADC sample, simply by the definition of a peak. Real data sequences contain a peak less often. This relaxes the requirements on the merger throughput. The Divider throughput is again deterministic. While it is technically possible to operate different parts of the cluster finder with different clocks, the design remains simpler with one common clock for all modules. The ratio between the cluster finder clock (CF clock) and the clock frequency of the 32 bit input data bus from the DDL (DDL clock) is referred to as "clock ratio" in the following.

	DDL rate (Gbps)	RCU DDL clock (MHz)	RORC DDL clock (MHz)	DDL bandwidth (MB/s)	RORC CF clock (MHz)	clock ratio
Run 1, 2009–2012 H-RORC + RCU1	2.125	40.000	40.000	160.0	133.3	3.33
Run 2, 2015 C-RORC + RCU1	2.125	40.000	53.125	160.0	159.4	3.99
Run 2, from 2016 C-RORC + RCU2	3.125	78.125	78.125	312.5	312.5	4.00

Table 4.4: Clock frequencies of the DDL interface and the cluster finder for the different running configurations in Run 1 and Run 2.

Table 4.4 shows an overview of the clock frequencies at different stages that are used for the different readout constellations in Run 1 and Run 2.

The original H-RORC cluster finder implementation used during Run 1 expected up to 160 MB/s from the TPC RCUs. This value was defined from the interface to the DDL. Both the DIU on the H-RORC and the SIU on the RCU operated their parallel 32 bit data interface with a 40 MHz clock. The cluster finder was using a 133 MHz clock for all processing steps. This same clock was used for the PCI-X interface. It resulted in a clock frequency ratio of 3.3 between the DDL clock and the cluster finder clock. This clock ratio proved to be sufficient for the operation of the TPC in combination with the running conditions, the H-RORC, and its cluster finder throughout Run 1.

The C-RORC implements the DDL directly in the firmware, without using the pluggable modules. The interface between the DDL firmware module and the cluster finder in the C-RORC uses a clock derived from the optical link. With a link rate of 2.125 Gbps, this results in a frequency of 53.125 MHz at the 32 bit parallel DDL data bus. However, the DDL interface on the RCU1 side was still driven from the same 40 MHz clock as during Run 1. This means that the C-RORC only received an effective maximum data rate of 32 bit at 40 MHz even though its own interface has a higher bandwidth. The clock frequency for the cluster finder was chosen to be 159.4 MHz, three times the actual DDL clock (53.125 MHz). Natural clock conversion factors are easy to implement

with the on-board clock management blocks and the resulting frequency is low enough to avoid timing problems. The effective clock ratio with respect to what the RCU1 can actually send via the DDL increased to 159.4 MHz / 40.0 MHz = 3.99 for the readout of the RCU1 with the C-RORC in the first phase of Run 2.

With the switch to the RCU2 in early 2016, the link rate changed to 3.125 Gbps. The RCU2 can fully exploit the available bandwidth. This results in burst data rates of around 310 MB/s from the TPC per link. This is almost a factor two higher than the 160 MB/s from the RCU1. The link speed of 3.125 Gbps makes a DDL clock frequency of 78.125 MHz. A readout with a clock ratio of 3.3 as before requires a clock frequency of 257.8 MHz for the cluster finder modules. This is still within the range of reasonable clock frequencies for the given FPGA. Nevertheless, it already requires a timing-oriented design flow.

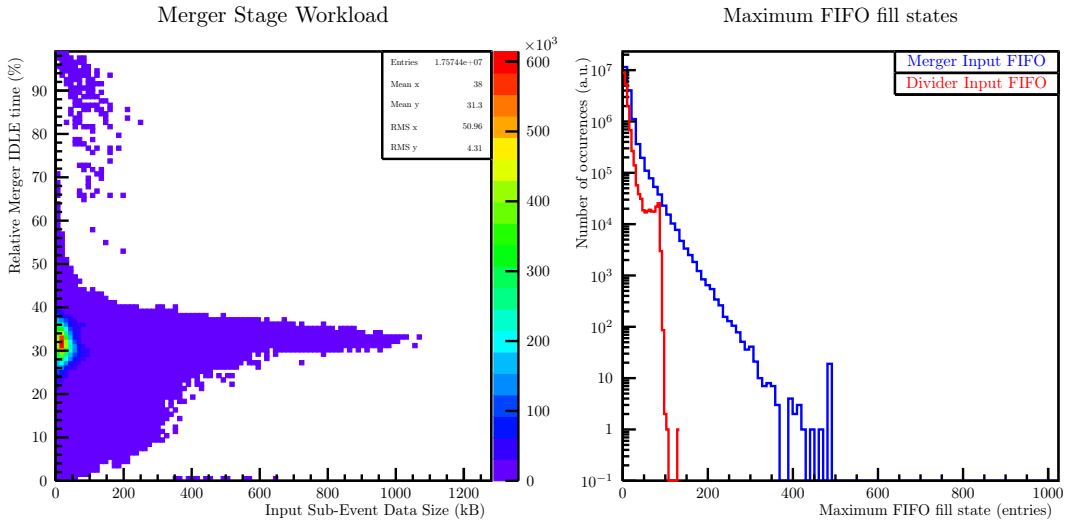

Figure 4.7: Merger idle time measurements and maximum FIFO fill states for a clock ratio of 3.3 as used with the H-RORC and the RCU1.

Unfortunately, a clock ratio of 3.3 turned out to be not sufficient for Run 2. Already the first physics runs with the RCU2 occasionally showed that the cluster finder FIFOs ran full and temporarily stalled the readout process. Measurements with real data from Run 2 confirmed this as shown in Figure 4.7. These measurements were done with the cluster finder co-processor implementation using a clock ratio of 3.3. One sub-event at a time was streamed through the cluster finder. After each sub-event, the monitoring parameters of the cluster finder were read out. This measurement method assumed a reasonable temporal spacing between successive sub-events, making sure that all buffers are empty again before the next sub-event arrives. While this is usually the case also in the actual application, it cannot be guaranteed in a high-load scenario. It means that the measurements possibly underestimate the actual load and the fill states due to successive events queuing up and arriving at the cluster finder while it is still busy from a previous sub-event. However, this measurement already confirms the observation. The dataset contains around 17 million sub-events with a total size of around 750 GB of raw TPC data from different runs between 2015 and 2017. The histogram on the left shows the relative idle time of the Channel Merger. The merger is considered idle if it is waiting for new data on the input interface. The measurement starts with the first data word of a sub-event and is stopped after the last word of the sub-event was processed. While the Channel Merger is idle in average around 31 % of the time, there are several individual sub-events that keep the merger busy most of the time. The histogram on the right shows the maximum FIFO fill state of the Channel Merger input FIFO and the Divider input FIFO after each sub-event. The Divider input FIFO can hold 512 entries and barely exceeds 100 entries. This means the cluster finder is not throttled from the divider or the PCI-Express readout. Looking at the Channel Merger input FIFO fill state confirms that this component is responsible for the reduced throughput. Its input FIFO can also

contain 512 entries with a programmable full threshold at 450 entries. This value is reached a couple of times with the given dataset. This means that the merger input FIFO ran full, stalling the previous processing steps and ultimately stalling the DDL readout.

There are several reasons why the clock ratio of 3.3 is not sufficient anymore with the RCU2 in Run 2. The TPC gas mixture was changed from Neon to Argon at the same time the RCU2 was installed. The new mixture yields a higher probability of noise above the zero suppression threshold. This leads to a higher number of peaks being found in the charge sequences, which in return increases the rate of cluster candidates towards the Channel Merger relative to the input data rate. Furthermore, the TPC improved over the last years, applying zero suppression more efficiently. The RCU2 with its improved readout scheme uses the available bandwidth much more efficiently than the RCU1 did. These facts required the clock frequency of the cluster finder to be scaled over-proportionally compared to the DDL rate change.

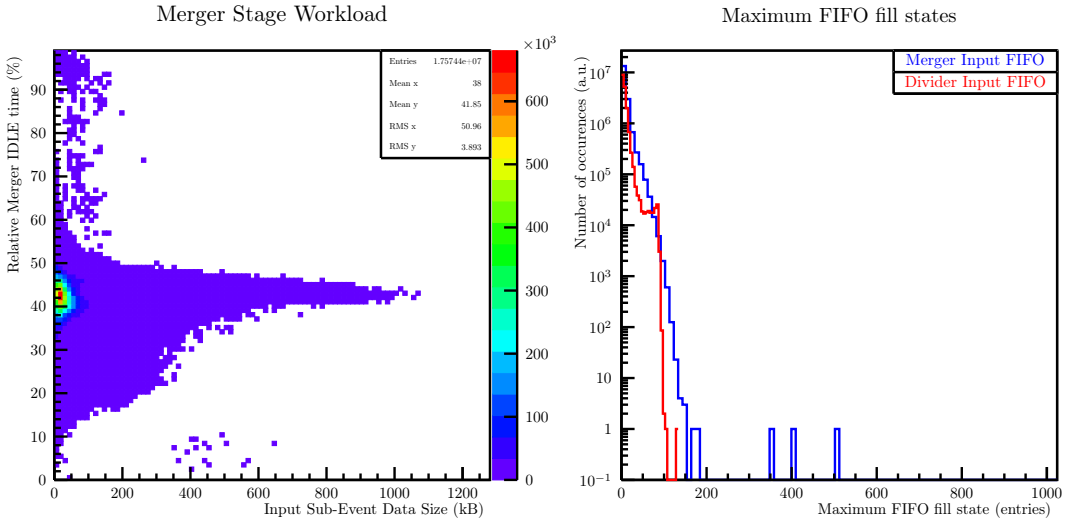

Figure 4.8: Merger idle time measurements and maximum FIFO fill states for a clock ratio of four as used on the C-RORC with the RCU2.

Figure 4.8 shows the Channel Merger idle time and FIFO fill state measurements with the same input dataset as above but with a clock ratio of 4. This ratio lifts the average Channel Merger relative idle time to around 42 %, which is around 10 % more than for the ratio of 3.3. The number of sub-events with an idle rate around 0 % is also decreased significantly. This change is also reflected in the maximum FIFO fill state measurements. The maximum fill state of the Divider input FIFO did not change compared to the previous measurement because both the Channel Merger and the Divider use the same clock in both cases. The maximum fill state of the Channel Merger input FIFO, however, is reduced significantly. This is because the input data rate into the cluster finder and, therefore, the number cluster candidates into the Channel Merger is reduced with the higher clock ratio. The input FIFO depth itself was also increased to 1024 entries. This means that the FIFO was never more than half full for the given dataset. The clock ratio of 4 in combination with increased FIFO depths also proved to be sufficient for the operation of the cluster finder with the RCU2 at 3.125 Gbps in the HLT production system. This variant was used in the HLT C-RORC firmware since mid-2016.

4.2.5 Common RCU1 + RCU2 Cluster Finder Implementation

A clock ratio of four in combination with a DDL clock of 78.125 MHz corresponds to a cluster finder clock frequency of 312.5 MHz. This is clearly at the upper end of what is possible with a Virtex-6 FPGA and six cluster finder instances per board. The need for backward compatibility with the RCU1 data format for the HLT data replay functionality makes this even more chal-

lenging. Implementing the cluster finder with these constraints in the C-RORC FPGA required a number of adjustments to the existing code.

The first processing step, the RCU decoding, was rewritten from scratch. The previous implementation was based on a state machine that reads the input FIFO and pushes out the up to three ADC samples contained in the input word. The state machine detected the input data type and processed header words and words with less than three ADC samples faster than others. This implementation achieved a high throughput but contained a dependency between the data received and the next action of the state machine. With the requirement of having a clock ratio of above three, a particularly high throughput of the RCU Decoder brings no gains for the overall throughput of the cluster finder. It may even increase the load on the Channel Merger. The RCU Decoder was rewritten in a dataflow manner. It processes new input data every third clock period, independent of the received data type, and writes out up to three ADC charge samples. This removes the data dependency between the input control logic and the data received. The decision on the number of samples to be pushed out can be pipelined. The same applies to the access to the calibration ROM. As a result, the RCU Decoder can be operated at a higher clock frequency compared to before.

The Channel Processor was already reasonably pipelined in its original implementation. It can handle one ADC charge sample per clock tick. In order to achieve the target clock frequency, it was extended in a first step with a couple of register stages. In a second step, parts of the Channel Processor were rewritten to implement an improved error detection described in Section 4.3.1 as well as a new peak finding algorithm described in Section 4.3.2.

The Channel Merger remains the critical part of the implementation. The branching and joining of comparably wide data paths from the input FIFO via the Merger FIFOs and also towards the Divider limit the maximum clock frequency. A number of register stages were added to relax the dependency of the maximum clock frequency on the placement of the different FIFO instances on the chip. The controller state machine was optimized to reduce the total number of states. A compile-time switch selects whether the support for only the RCU2 or both RCU versions is implemented.

The Divider was rewritten in the same way as the RCU Decoder. The original version was also using a state machine with a data-dependent number of clock ticks per input cluster. It was confirmed during the performance measurements that the division step is not the bottleneck in the chain of cluster finder modules. By rewriting it with a fixed number of ticks per input cluster independent of the data type, a control dependency could be resolved and the maximum clock frequency could be improved.

General adjustments were performed in all cluster finder modules. The existing reset scheme was evaluated and could partly be extended by further decreasing the reset signal fan-out. The depths of the FIFOs between the different processing stages were increased to extend the time from a full FIFO until the readout is actually stalled. This reduced the chances for readout stalls. The interfaces within and between the processing steps were implemented in a more generic way. This enabled to add new features to the cluster finder more easily. The firmware implementation is independent of the RCU version and can handle both data formats. The target RCU version number is set via software at runtime.

A first version of this implementation was successfully used in the HLT production system at the first physics runs with the RCU2 in mid-2016. Improved variants up to the final revision were gradually rolled out in the following days. Table 4.5 shows the FPGA resource usage for one cluster finder instance comparing the original cluster finder port for the RCU1 with the common RCU1+RCU2 cluster finder implementation. The increased number of flip-flops originates from the additional register stages for timing reasons. The number of look-up tables and DSPs barely changed because the algorithm and its combinatorial steps were not modified. The change in the number of block RAMs results from the increased FIFO depths. The reported maximum clock frequency after the synthesis step emphasizes the increased timing performance of the common implementation. All resource numbers were extracted after the place-and-route step

Cluster finder resource usage	initial RCU1 implementation	common implementation (RCU1 + RCU2)
Flip-flops	5169	8465
Look-up tables	4274	4250
DSPs	8	8
Block RAMs	23	25
post-syn max. frequency	223.0 MHz	358.0 MHz
implemented frequency	159.4 MHz	312.5 MHz

Table 4.5: Cluster finder C-RORC FPGA resource usage comparison of the initial port for the RCU1 readout and the extended common version supporting twice the input bandwidth. The Numbers are extracted after the place-and-route step.

with the design meeting the timing constraints of the target clock frequency. Six cluster finder instances running at 312.5 MHz supporting both RCU versions at double the initial bandwidth were successfully integrated into the HLT C-RORC readout firmware.

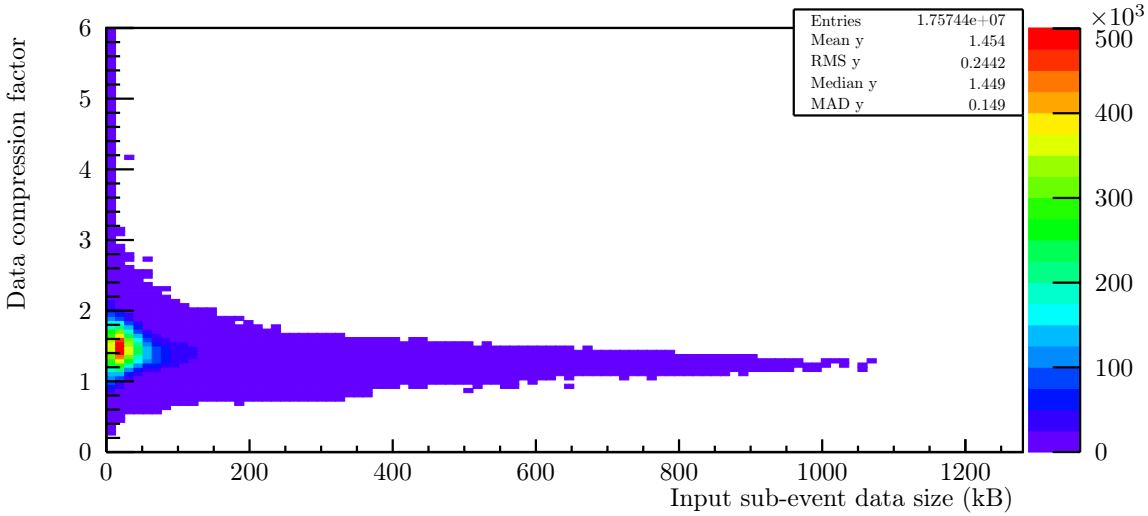

Figure 4.9: Distribution of the compression factors per sub-event of the cluster finder for around 17 million sub-events from different runs between 2015 to 2017. The cluster finding algorithm and parameters are the same as in Run 1.

One important aspect, besides the gains in the HLT CPU computing performance, is that the hardware cluster finder already slightly reduces the data volume before it enters the HLT data transport framework. This means, the hardware cluster finder already contributes to the overall data compression of the HLT. The output size of the cluster finder depends on the number of clusters that are found in the raw input data. Each output cluster has a fixed size of 24 bytes defined by the cluster parameters mapped into the output data format previously shown in Table 4.1. The DDL header and RCU trailer words add a constant offset of 40 and 36 bytes respectively for each sub-event. The cluster finder data compression factor, as the quotient of the input data size and the output data size per sub-event, therefore, depends on the TPC raw data. The ability to compress the input data volume from a technical perspective did not change since Run 1 because the cluster finder algorithm and its configuration parameters were not changed for Run 2 until mid-2017. The cluster finder compression ratio in Run 1 was measured to be in average 1.25 [Alt 17]. The actual compression ratio, however, depends on the input data. Figure 4.9 shows the distribution of data compression ratios for around 17 million sub-events in Run 2 from different runs between 2015 and 2017. Table 4.6 shows the mean and median compression ratios for the same dataset but is distinguishing between the different runs.

Run	Date	Beam type	RCU	Number of sub-events	Mean compression	Median compression
245683	Nov. 2015	PbPb, 6.4 TeV	RCU1	220k	1.35 ± 0.18	1.32 ± 0.08
246980	Dec. 2015	PbPb, 6.4 TeV	RCU1	377k	1.40 ± 0.20	1.38 ± 0.10
253461	May 2016	pp, 6.5 TeV	RCU2	626k	1.54 ± 0.15	1.53 ± 0.09
264347	Oct. 2016	pp, 6.5 TeV	RCU2	940k	1.57 ± 0.20	1.56 ± 0.13
271381	Jun. 2017	pp, 6.5 TeV	RCU2	1.11M	1.46 ± 0.27	1.47 ± 0.16
271768	Jun. 2017	pp, 6.5 TeV	RCU2	2.32M	1.46 ± 0.34	1.45 ± 0.20
271868	Jun. 2017	pp, 6.5 TeV	RCU2	1.21M	1.46 ± 0.24	1.47 ± 0.16
271912	Jun. 2017	pp, 6.5 TeV	RCU2	2.72M	1.43 ± 0.20	1.44 ± 0.13
271915	Jun. 2017	pp, 6.5 TeV	RCU2	1.65M	1.33 ± 0.14	1.33 ± 0.08
271962	Jun. 2017	pp, 6.5 TeV	RCU2	6.40M	1.47 ± 0.24	1.47 ± 0.15
all above				17.57M	1.45 ± 0.24	1.45 ± 0.15

Table 4.6: Cluster finder mean and median data compression factors for different runs. Statistics are based on the number of sub-events given in the table. Errors are RMS for mean and MAD for median.

The mean compression, in general, is higher than the Run 1 measurements and varies between different runs due to different beam types, experiment conditions, and trigger configurations. An important aspect is the distribution of the compression factors as shown in the histogram. While the mean compression factor is around 1.45, there is a wide spread of compression ratios per sub-event also to values below one. A compression ratio below one means that the data volume after the cluster finder is larger than its raw data input size for a given sub-event. This means that the cluster finder does not reduce the input data size in any case but may even increase the data volume under certain circumstances.

4.3 Cluster Finder Extensions and Improvements

4.3.1 Protocol Error Handling, Detection and Reporting

The detector data sent via the DDLs consists of a stream of 32 bit words with a common data header and a detect-specific payload encoding. In order to find cluster signatures in the raw TPC ADC samples, this 32 bit data stream has to be decoded in the cluster finder from a predefined data format. This also means that the cluster finding can only work properly if the raw data from the DDLs can be decoded correctly. With a highly complex and distributed readout system with thousands of components, partly operating in radiation environments, this cannot be guaranteed. Data and protocol errors can occur at different layers. They have to be detected and handled.

The optical data link between the SIUs, the DAQ RORCs and the HLT via DDL uses a CRC checksum to detect transmission errors. The DDL implementation in the C-RORC signals CRC errors to the user interface. An error at this stage means that the data block received in the DIU does not match the data sent from the SIU. Therefore, this block is definitely corrupt to some extent. During Run 1, DDL errors went undetected in the HLT. For Run 2, DDL errors are propagated to the readout software. They are handled and counted. Upon errors, the readout software tries to parse the data header of the corrupt data block. If the header looks reasonable, the sub-event is stripped to only its header. This way, the HLT data transport framework knows that there was a corrupt sub-event from a given DDL but the reconstruction does not have to process the corrupt payload. If the header cannot be parsed, the full sub-event is discarded. This method does not only apply to the TPC readout with the cluster finder but is also used for the raw readout of all other detectors. It helped to spot and fix a reset problem with the DDL between the DAQ and HLT systems.

The encoding of the TPC detector data over the DDL is already briefly described in Section 4.1.1. The protocol distinguishes between header, trailer and data words. Data words contain sequences of a length marker, a timestamp, and the corresponding ADC samples. Each header specifies the data origin and the total number of sequence samples for the following data block. The cluster finder relies on a proper decoding of this format to interpret the raw data correctly and extract the actual timestamps and ADC samples. A typical error condition is that the total number of samples in the header words does not match the actual number of samples that are following. Another possible error is that a sequence does not end before the next header word. The original cluster finder used in RUN 1 partly already detected these error conditions and handled them by ignoring the inconsistent parts of the input data. However, the information about protocol errors was not communicated to the readout process and, therefore, went undetected. Certain data words were silently ignored in case of errors. Additionally, the output of the cluster finder differed from the output of the software reference model in case of corrupt input data.

The cluster finder for RUN 2 was improved with extended error detection mechanisms. All protocol errors are now detected, counted, and reported to the readout application. Additionally, the error handling is identical now between the emulator and the hardware implementation. It ensures a consistent output for both the hardware and the software reference also for corrupt input data. This functionality was realized by adding additional error checks into the protocol-decoding firmware module and the peak finder. Additional error flags are carried along with the data throughout the cluster finder processing modules. Error flags in the DMA *ReportBuffer* entry are used to propagate the detected error conditions to the readout software. Warnings are presented to the HLT operator in case of excessive error rates on individual links. The functionality was verified with artificially corrupted data as well as actual detector data with knowingly high error counts. The output of the software emulator and the hardware implementation was compared using the cluster finder co-processor implementation described in Section 4.2.2. This method helped to detect faulty TPC channels already while running.

4.3.2 Improved Noise Rejection, Split Cluster and Border Cluster Tagging

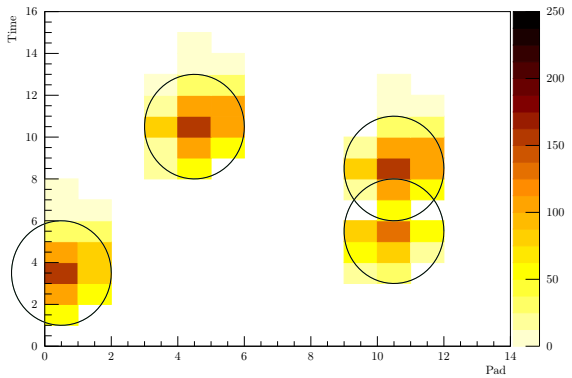

Figure 4.10: Sample TPC charge sequences. The cluster finder calculates the cluster properties based on the samples in a region of interest around the charge maxima.

The cluster finder is looking for local maxima in the charge sequences from a TPC pad plane in the time and the pad direction. The calculation of the cluster properties is based on all available samples in a region of interest around a maximum. A window of five samples across the peak value in the time direction is taken into account. This includes two samples before the peak, the peak value itself and two samples after the peak. In the pad direction, cluster candidates are merged if their temporal spacing is below a configurable distance. This value is defined by the TPC Offline group based on data analysis and is set to four time bins since RUN 1.

Figure 4.10 shows a possible charge distribution across the pads and the time. In the case of the cluster in the center of the graph, the samples from rows 3–5 and timestamps 8–13 would be used to calculate the cluster properties. The cluster finder uses an algorithm based on weighted sums to calculate these properties as described in Section 4.1.2. A systematic error in the calculation of the cluster properties is introduced if the signal falls partly outside the active detector region. The cluster on the bottom left clearly has some charge left of the leftmost pad in this row. This charge cannot be taken into the calculation. As a result, the position of this cluster will be slightly shifted to the right and its width and the accumulated total charge will be underestimated. A similar effect occurs if clusters are overlapping as shown on the right side of Figure 4.10. The cluster finder makes a cut in the time and the pad direction, calculating cluster properties only from the resulting share of the charge signals. In this case, the positions of the two clusters are slightly shifted towards each other and the total charge is not accurate because each set of samples contains charge contributions from the other cluster. The Offline cluster finder handles these cases by introducing virtual charges outside the active detector regions and by implementing a charge sharing mechanism instead of a straight cut to deconvolute overlapping clusters. This is not feasible with the data streaming approach of the hardware cluster finder. Implementing a comparable approach compatible with the existing architecture would require substantial changes to the implementation, if possible at all. It is, however, possible to tag clusters that contain charge samples from border pads and to tag clusters that are split in the time or the pad direction. This idea was suggested by the TPC Offline team. Algorithms sensitive to these kinds of effects can identify clusters based on these flags and exclude or correct them if needed. This information can be used to improve energy loss calculations relying on accurate total charge values of the individual clusters.

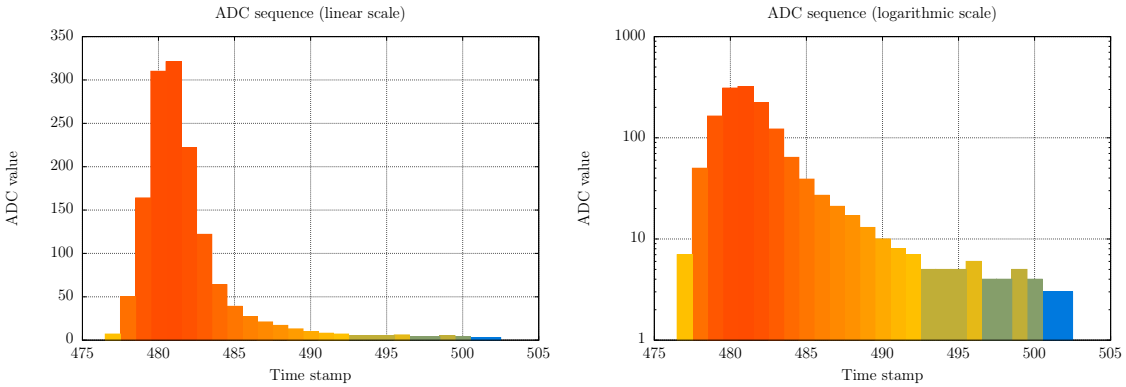

Figure 4.11: Random TPC charge sequence in linear (left) and logarithmic (right) charge scale. Besides the obvious main peak, the cluster finder may find local maxima due to noise in the slowly decaying ion tail towards higher timestamps.

Implementing the tagging of split clusters in the time and the pad direction in the cluster finder emulator revealed that a comparably large fraction of up to 30 % of clusters contain charge sequences with at least one split in the time direction. The reason for this is the peak finding algorithm in the time direction and its susceptibility to detector noise. The peak finder in the time direction compares the current charge value with the previous value. If the current value is larger, the charge slope is considered rising. If the value is smaller, the charge slope is considered falling. Otherwise, the previous slope value is kept. A peak is flagged if the slope changes from rising to falling. As a result, a charge variation of ± 1 can already trigger a peak to be found. Figure 4.11 on the left shows a random TPC charge sequence in the time direction from real detector data. The main signal is clearly visible at around timestamp 480. The sharp peak is followed by slower decay, the ion tail. While barely visible in the linear scale, the logarithmic plot of the same sequence on the right shows small charge fluctuations near the end of the sequence. The cluster finder as used until mid-2017 detects the samples at timestamps 496 and 499 as peaks as well. Additionally, they are only three samples apart. This means that neither of them can use the full five sample window to calculate the cluster properties, so both would be flagged as

split in the time direction. However, both of them originate only from noise. A higher gain in the TPC since 2015 as well as the change of the TPC gas mixture in 2016 raised the expected level of noise up to around ±1 least significant ADC bits.

In order to resolve this issue, the hardware cluster finder had to be made more resilient to noise in the temporal charge sequences. The pad direction is not that sensitive to this kind of noise because the cluster splitting in the pad direction is based on the total charge of the contributing temporal sequences. These sums already average out some noise effects. The previous cluster finder implementation already had an option to consider the slope of the incoming charge sequence as rising or falling only if the current and previous charge values differ by a configurable amount. Requiring a charge difference of at least two already filters out noise based on charge fluctuations by ±1. However, it also suppresses or shifts clusters with a slow signal rise or fall time. As a consequence, the noise filtering options of the previous cluster finder could not provide an efficient filtering for the changed circumstances.

A more advanced algorithm to filter out noise in the charge sequences was developed by David Rohr. The implementation of this algorithm in the hardware cluster finder was done as part of this work. This new peak finder algorithm takes a sequence of up to five charge samples into account and keeps track of the values of the last local minimum and maximum charge. Correlating all of these values provides the option to distinguish between a real peak and noise-induced fluctuations by building a trend across the samples. An optional low-pass filter before the peak finding step can further reduce the noise level. The actual cluster property calculation would still use the original charge samples in this case. Different variants of the algorithm with configurable parameters controlling the level of noise rejection were integrated into the software-based cluster finder emulator by David Rohr. Real datasets were used to evaluate the effects of the different options and to find viable parameter sets for the Run 2 conditions. A large-scale analysis using the data from full physics runs was done by the TPC Offline team. The results with the new options were compared with those from the Offline reference implementation and the previous cluster finder model. Dedicated physics runs, storing both the raw TPC data and the compressed HLT clusters using the new cluster finder firmware, were analyzed to confirm the hardware implementation.

An improved noise rejection in the cluster finder immediately has an impact to the number of clusters and the data volume. Fewer noise-induced peaks found in the temporal charge sequences lead to a reduced number of cluster candidates pushed into the Channel Merger stage. This results in an overall reduced number of clusters found in the same input dataset. Fewer clusters from the same input data volume increase the overall HLT data compression ratio and reduce the experiment's total data rate to the permanent storage. The smaller storage footprint allows the experiment to store more physics data with the same data storage quota.

Mean number of ...	Prev. peak finder	New peak finder	Rel. change
candidates	116.21 / kB	98.81 / kB	−15.0 %
clusters	27.34 / kB	21.42 / kB	−21.7 %
candidates per cluster	4.25	4.61	+ 8.5 %
clusters split in the time direction	18.97 %	3.46 %	−15.5 %
clusters split in the pad direction	13.67 %	14.32 %	+ 0.7 %

Table 4.7: Cluster finder statistics with the old and the new peak finder based on around 10 million sub-events (approx. 430 GB) from HLT validation runs in June 2017.

The effect of the improved peak finder is summarized in Table 4.7. This measurement is based on around 10 million random sub-events with a total raw data size of approximately 430 GB. They originate from physics runs recorded in June 2017 where both the raw data and the HLT-compressed results were stored. These runs are part of a set of runs that was used by the TPC Offline group to validate the new cluster finder for its operation in the HLT production system. Around 15 % fewer cluster candidates were found with the new peak finder in the same dataset compared to the previous cluster finder. With less fake cluster candidates, the chances

to merge neighboring clusters increased and resulted in around 8.5 % more cluster candidates per cluster. The decreased number of overall candidates in combination with the increased number of candidates per cluster reduced the total number of clusters found in this dataset by around 21.7 % compared to the previous peak finder implementation. Enabling the new peak finder for TPC data from 2016, when the Argon gas mixture with the increased noise was in use, resulted in a reduction in the number of clusters of even 32 % [Roh 17 VIII]. The reduced number of clusters also accelerated the online track reconstruction in the HLT. As anticipated above, the number of clusters marked as split in the time direction could be reduced by more than 15 %. A cluster is considered "split" in the time direction if at least two contributing sequences were split in the peak finder step.

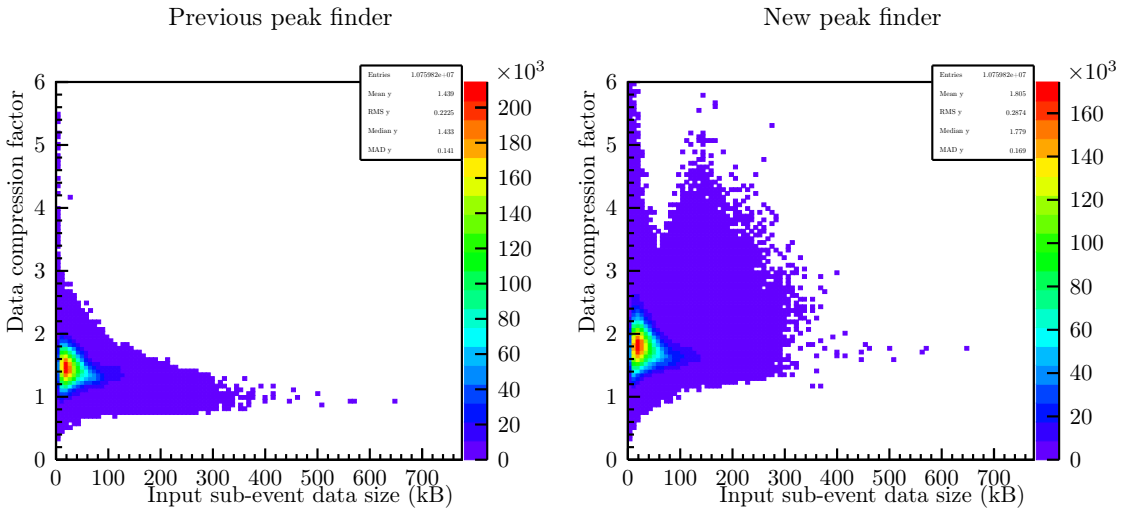

Figure 4.12: Cluster finder data compression improvements comparing the previous implementation (left) with the new peak finding algorithm (right).

The reduced number of clusters for the same input data size directly leads to an improved data compression ratio. Figure 4.12 compares the distribution of the data compression ratios for the previous and the new peak finding algorithm. The mean compression ratio raises from around 1.4 to around 1.8 and the overall distribution changes considerably especially for larger sub-event sizes. A lot of sub-events between around 50–400 kB input size achieve much higher compression ratios by filtering out the detector noise. Additionally, there are significantly fewer sub-events with a compression ratio below one.

Gradual improvements on the Huffman compression scheme and the fixed-point representation of cluster properties, in combination with the hardware cluster finder changes, increased the overall HLT compression ratio from around five to six during the first part of Run 2, to around seven starting mid-2017, and finally to above eight for 2018. Histograms with the HLT TPC compression ratios against the number of clusters per event are shown in Figure 4.13. The data from 2016 is processed with the previous cluster finder implementation. The measurements in 2018 contain the new cluster finder together with several improvements on the software side.

The hardware implementation of the improved peak finding algorithm, as well as the tagging of clusters, required substantial changes to the cluster finder. All parts of the cluster finder, the RCU Decoder, the Channel Processor, the Channel Merger, as well as the Divider instance had to be modified. The new peak finding algorithm could not easily be integrated into the previous peak finder implementation. While the new options are just another `if-else` block in the software reference implementation, this has more implications in hardware. The flow control and the flushing solution for the pipeline in the previous implementation did not scale to the increased pipeline depth required for the correlation of up to five samples and the last local minima and maxima. For this reason, the peak finder implementation was rewritten from scratch. The new implementation can still process the data with the previous peak finding

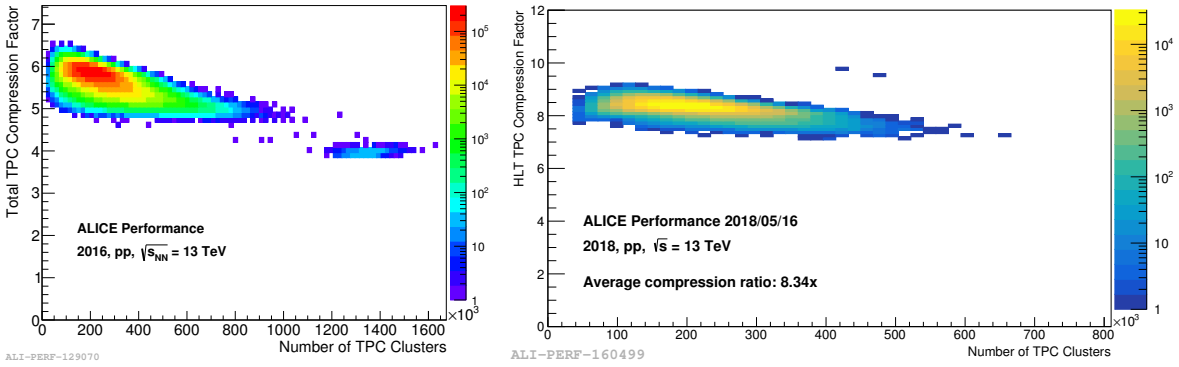

Figure 4.13: Total HLT TPC compression factor for proton-proton data from 2016 (left) with the previous cluster finder and 2018 (right) with the new cluster finder and an improved software compression scheme. Images: David Rohr [Roh 17 IV, Roh 17 V].

algorithm but also supports the new noise rejection options. The level of noise rejection, as well as the size of the window for building the charge trend, is configurable within a certain parameter range. The software model uses integer data types for the configuration parameters. Having the same value ranges in the hardware implementation is not reasonably possible with the available resources. For this reason, a subset of configuration parameter value ranges was selected for the implementation in hardware. The ranges are based on evaluations with the software reference implementation using real detector data. This is to keep the hardware implementation as simple as possible and the resource usage low. The previous peak finder is based on the current and the last charge value plus a configurable step size to flag peaks. The new peak finder takes a window of up to five charge samples plus the last minimum and maximum charges into account. The increased number of input signals makes the combinatorial aspect of the peak finder much more demanding. The bookkeeping of the latest charge minima and maxima introduces a data dependency that limits the possibility to pipeline the implementation. The resulting hardware implementation is, therefore, harder to integrate with the same clock frequency constraints.

Fortunately, the implementation of the border cluster tagging came with fewer implications. The information on whether a pad is a border pad could be integrated into the mapping table in the RCU Decoder and is simply passed along with the sequences and cluster candidates. The merger marks a cluster as border cluster if at least one contributing candidate had the flag set. One cluster finder instance does not always process full TPC rows near partition boundaries. Some rows are divided across partitions and, therefore, across detector links. For this reason, another flag for edge clusters near the physical end of the padrows was added in the same way. Clusters near sector edges tend to have a systematic shift in cluster position away from the edge. With the edge flag, this effect can be mitigated with additional corrections in hardware or software in a future implementation.

The implementation of the split cluster flags is a bit more complex. While the actual information about a split was available already, the tagging of clusters with this information has to take more aspects into account. The split cluster flag in the time direction requires that a certain number of contributing sequences are split in the time direction. This requires keeping track of the number of merged sequences per cluster in the Channel Merger. Additional data fields in the Channel Merger further increase the data path width to and from the buffer FIFOs and, therefore, further reduce the maximum clock frequency.

The improved peak finding algorithm, as well as the tagging of clusters as described above, was successfully integrated into the hardware cluster finder. Implementing the new features was particularly challenging due to the required clock frequency of 312.5 MHz. While the reduced number of clusters may enable a lower clock frequency again, sticking with the previous frequency, if possible, is a safe solution. A reduction of the clock frequency can then be evaluated by monitoring the Channel Merger workload during operation. The analysis of the hardware

	initial RCU1 CF port	common CF (RCU1 + RCU2)	improved common CF	improved RCU2 CF
Flip-flops	5169	8465	8953	8468
Look-up tables	4274	4250	4426	4006
DSPs	8	8	8	8
Block RAMs	23	25	28	24
post-syn max. Freq.	223 MHz	358 MHz	322 MHz	327 MHz
implemented Freq.	159.4 MHz	312.5 MHz	300.0 MHz	312.5 MHz

Table 4.8: Cluster finder C-RORC FPGA resource usage comparison of the initial versions and two variants with the new peak finder. Numbers are extracted after place-and-route.

cluster finder performance with recorded data was again done with the co-processor implementation. The co-processor implementation is realized with support for both the RCU1 and the RCU2 at 300 MHz cluster finder clock frequency. The implementation in the HLT input firmware at 312.5 MHz with support for both RCU versions comes close to the limits of the FPGA. In order to meet the clock frequency requirements with the new implementation, the support for the RCU1 data format was dropped at this stage. This change saves block RAM resources and improves the Channel Merger timing. The limitations of this approach for the data replay are not that important anymore because plenty of RCU2 data was recorded in the meantime and is available for replay tests. Improving the timing performance of the common implementation would have taken much more effort without any gains for the operation during detector readout. The stripped down version of the cluster finder operating at 312.5 MHz was integrated into the HLT readout firmware. Table 4.8 compares the resource usage of the improved cluster finder with the previous implementations. The first two columns contain the resource usage of the initial cluster finder implementation for the RCU1 and the common RCU1+RCU2 implementation as described in Section 4.2.5. The third column shows the resource usage of the common RCU1+RCU2 implementation including the new peak finder and the cluster tagging. The number of flip-flops and look-up tables slightly increased due to the higher complexity of the peak finder. The block RAM usage is higher due to wider data paths throughout the Channel Merger and an increased FIFO depth at the Channel Merger output. The latter simplifies the flow control and, therefore, increases the timing margin in the Channel Merger. The design shown in the third column could not be implemented with six cluster finder instances running at 312.5 MHz. It was not possible to meet the timing constraints for this variant. The implementation shown here is from the co-processor implementation running six instances at 300.0 MHz. The last column shows the cluster finder for only the RCU2 as used in the HLT readout firmware starting mid-2017. Dropping the RCU1 support saves around 400–500 look-up tables and flip-flops, as well as four block RAMs in the Channel Merger, compared to the common implementation. The reduced number of block RAMs makes it possible to implement this variant with six cluster finder instances at 312.5 MHz in the C-RORC.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DW0	1	1	(uint6_t) Row						B	(fixed_t<11,12>) MaximumCharge																							
DW1	S_P	S_T	(fixed_t<18,12>) TotalCharge																													
DW2	(float32_t) Pad																															
DW3	(float32_t) Time																															
DW4	(float32_t) PadError																															
DW5	(float32_t) TimeError																															

Table 4.9: Cluster finder output data structure for clusters including the new cluster flags highlighted in blue. "B" indicates border clusters, "S_P" and "S_T" mark clusters split in the pad and the time direction.

The flags for border clusters and clusters split in the time or the pad direction are implemented as three individual bits in the cluster finder output data format. Exactly three bits were unused in the previous output format definition. These are now used to extend the cluster finder with the new features without breaking compatibility with existing applications. Table 4.9 shows the

positions of the flags for the border clusters (B), the split in the time direction (S_T) and the split in the pad direction (S_P).

The new cluster finder implementation with the improved peak finder and the described cluster tagging is active in the HLT production system since August 2017. Additional improvements on the noise rejection and cluster tagging are being evaluated and can be integrated into the hardware cluster finder in a future firmware revision.

4.4 Cluster Finder Performance

4.4.1 Cluster Finder Physics Performance

The Offline software cluster finder is the reference in terms of the physics performance. However, it is more complex than the HLT hardware implementation and takes more effects into account. It performs a more precise unfolding of overlapping clusters, implements a per-pad gain correction, applies a baseline shift if needed, and can handle clusters at the edge of the active detector region more accurately. With the HLT replacing the raw TPC data with the compressed clusters, the Offline reference implementation cannot be used in the reconstruction chain. This makes it even more important to ensure that the mathematically simpler HLT approach does not result in a physics performance degradation. The TPC Offline group has performed detailed studies on the impact of the HLT approach. Dedicated validation runs storing both the raw TPC data and the HLT results were recorded and reconstructed both with Offline clusters and with the clusters found by the HLT approach. It was concluded that the HLT implementation has no negative effect on the physics performance and the new cluster finder even improves the quality compared to the previous HLT implementation. These studies were using the data from full runs and, therefore, have much better statistical significance than the measurements done at the cluster finder level shown in Section 4.3.2. The physics performance studies were carried out by the TPC Offline group using either the HLT clusters or the clusters found with the Offline cluster finder as a starting point. The results are briefly summarized here to confirm the effectiveness of the HLT hardware cluster finder changes described in Section 4.3.2.

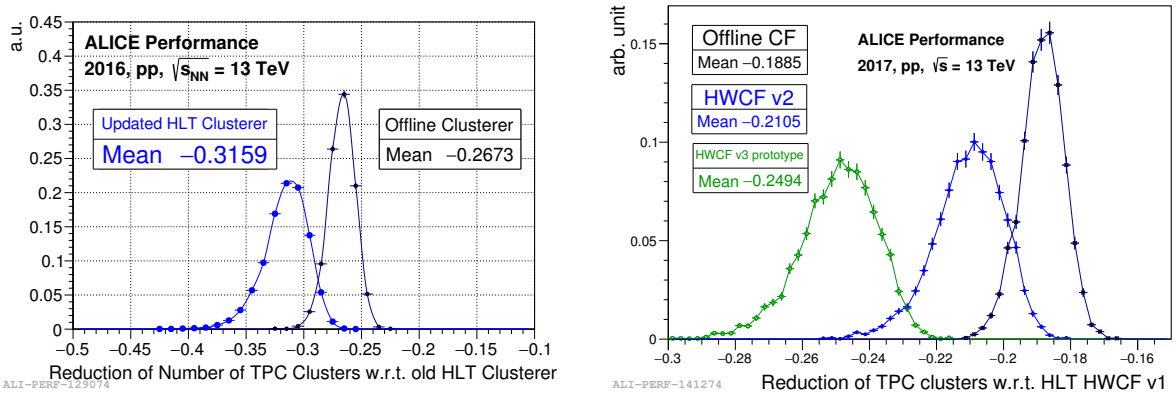

Figure 4.14: Reduction in the number of clusters found with the improved cluster finder and the Offline cluster finder in relation to the previous cluster finder. Left: 2016 pp data with the Argon gas mixture in the TPC. Right: 2017 pp data again with the Neon gas mixture. Measurements: TPC Offline group, Images: David Rohr [Roh 17 VII, Roh 17 VI].

Figure 4.14 compares the number of clusters found with the Offline cluster finder and the improved HLT hardware cluster finder with respect to the previous HLT cluster finder. The study was carried out with raw data from proton-proton collisions from 2016 where the TPC was using an Argon-based gas mixture (left) and with data from 2017 again with the Neon mixture (right). While the Argon mixture has advantages for several aspects of the TPC, it brings an increased level of noise to the raw ADC sequences. As a result, the improved cluster finder with the noise

suppression reduces the number of clusters found in the data from 2016 in average by around 31 %. With the Neon gas mixture in 2017, the reduction in the number of clusters is at around 21 %. The Offline cluster finder extracts around 27 % fewer clusters than the previous HLT hardware cluster finder for the data from 2016 and around 19 % less for data from 2017. This means that the improved HLT cluster finder slightly undercuts the Offline implementation in the number of clusters in both scenarios. The numbers derived from these analyses are consistent with the cluster finder standalone measurements shown above in Table 4.7. The additional cluster finder variant in green applies a more aggressive noise rejection. This results in even fewer clusters. This option is evaluated as a possible future extension of the hardware cluster finder but is not implemented in hardware at the time of this writing. With fewer clusters found with the HLT approach compared to the Offline cluster finder, it is important to make sure that no physics information gets lost.

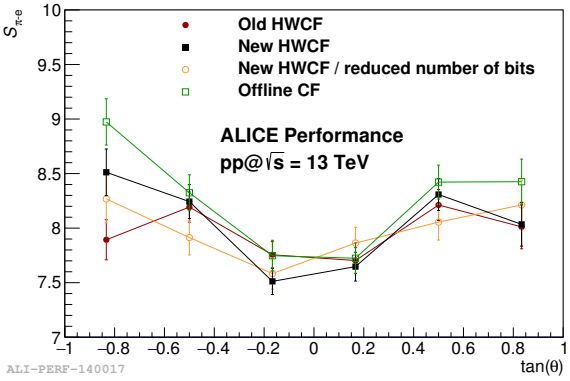

Figure 4.15: Particle separation power for electrons and pions for different track angles in the TPC volume using the $\mathrm{d}E/\mathrm{d}x$ of TPC tracks computed using HLT clusters or Offline clusters. The results from all variants are equivalent within the error ranges. This confirms that the total cluster charge calculated in the cluster finder is consistent. Measurements: TPC Offline group. Image: David Rohr [Roh 17 II].

Figure 4.15 shows the particle separation power of electrons and pions for different track angles in the TPC volume. Different particles deposit different amounts of energy in the TPC volume based on their charge and momentum. The energy loss ($\mathrm{d}E/\mathrm{d}x$) of charged particles traversing matter is described with the Bethe-Bloch equation. A given energy deposition along a trajectory enables drawing conclusions on the original particle type. The energy deposition in the TPC volume is proportional to the space charge recorded with the readout chambers. The space charge from a single track is derived from the sum of charges of associated clusters. A particle separation power consistent with the Offline cluster finder confirms that the number of clusters associated with a track, as well as the total charge values of the HLT clusters, are correct. The separation power of all cluster finder variants is equivalent to the Offline implementation within the errors.

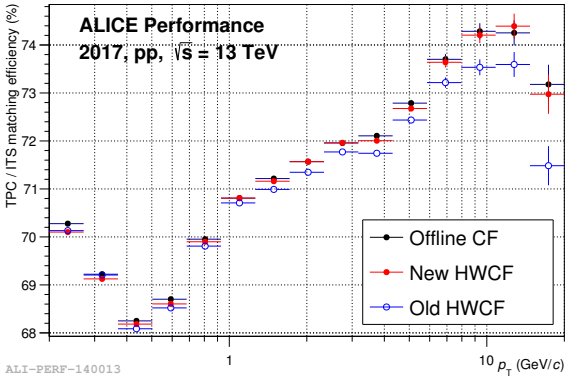

Figure 4.16: Matching efficiency of TPC tracks to ITS tracks in the Offline reconstruction using either HLT clusters from the previous cluster finder, the improved cluster finder or using Offline clusters as input. The improved cluster finder mitigates a deficiency of the matching efficiency especially towards high transverse momenta. Measurements: TPC Offline group. Image: David Rohr [Roh 17 III].

Figure 4.16 shows the matching efficiency of TPC tracks with ITS tracks using the HLT or the Offline clusters in relation to the transverse particle momentum (p_T). A comparable efficiency between using the Offline and the HLT clusters confirms that clusters with equivalent positions and widths are associated with the same tracks. The previous HLT cluster finder implementation had a deficiency in the matching efficiency especially for high–p_T ranges. This is significantly improved with the new HLT cluster finder, providing equivalent results for the efficiency with

the Offline and the HLT clusters. The filtering of the noise-induced clusters with the improved cluster finder, therefore, mitigates prior inefficiencies of the HLT approach.

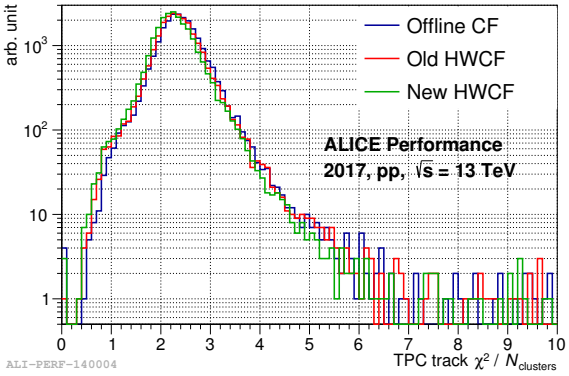

Figure 4.17: The distribution of TPC track χ^2-residuals using Offline track reconstruction with either HLT clusters or Offline clusters as input. This confirms that the cluster positions calculated in the cluster finder with respect to the reconstructed particle trajectory are consistent. Measurements: TPC Offline group. Image: David Rohr [Roh 17 I].

Figure 4.17 shows the distribution of track χ^2-residuals using the Offline track reconstruction with either HLT clusters or Offline clusters as input. The χ^2-residuals are a measure for the cluster positions with respect to the track position. The Offline cluster finder, the previous HLT cluster finder, and the improved HLT cluster finder provide equivalent results. This confirms that the cluster positions calculated with the improved cluster finder are comparable to the previous and the Offline implementation.

4.4.2 Computing Performance

216 hardware cluster finder instances are distributed across 36 C-RORCs in 36 HLT nodes to handle all TPC readout links. The firmware implementation is designed to handle the maximum amount of raw TPC data that could come via the DDL. With the RCU1, this corresponds to around 160 MB/s and around 312 MB/s for the RCU2. The cluster finder firmware architecture and its operating clock frequency are chosen in a way to ensure this to the widest possible extent. This process is described in Section 4.2.4. With the data dependent processing steps in the Channel Merger, this cannot be guaranteed under every possible condition. The present design has, however, proven its suitability with standalone performance tests with large raw datasets and, most importantly, during the operation in the HLT production system in Run 2. Additionally, continuous data replay measurements with a set of several Gigabytes of raw detector data were conducted. A set of raw TPC events was loaded into the C-RORC on-board RAM and played back continuously at the maximum DDL bandwidth. The system then adjusts to an average processing rate. However, the statistical significance of this method is limited due to the amount of on-board storage. The continuous replay has to run for a couple of seconds to settle to stable throughput conditions and buffer fill states. This, in return, limits the number of measurements that can be done within a reasonable timescale. Measurements were conducted based on around 1.6 million sub-events from a proton–proton run in June 2017. The results confirm that the target cluster finder processing rate is achieved. The replay measurements in combination with the running experience during Run 2 confirm that this cluster finder implementation can process raw TPC data at the maximum DDL bandwidth of 312 MB/s. The throughput of one hardware cluster finder instance can, therefore, be considered to be equal to the DDL bandwidth.

In order to quantify the performance benefit of implementing the cluster finding in hardware, these measurements have to be compared with software approaches. There are two software implementations available for comparison: the cluster finder emulator and the Offline cluster finder. Torsten Alt has also compared the Run 1 hardware cluster finder with a software implementation named “ALICE HLT Cluster Finder”. This implementation is an early version of the HLT algorithm. It only calculates the center of gravity and the total charge of the sequences. It does not perform peak finding or cluster width calculations [Alt 17]. This implementation is not analyzed in the performance measurements in this work because it is not used in the current reconstruction environment and comes with yet another algorithm.

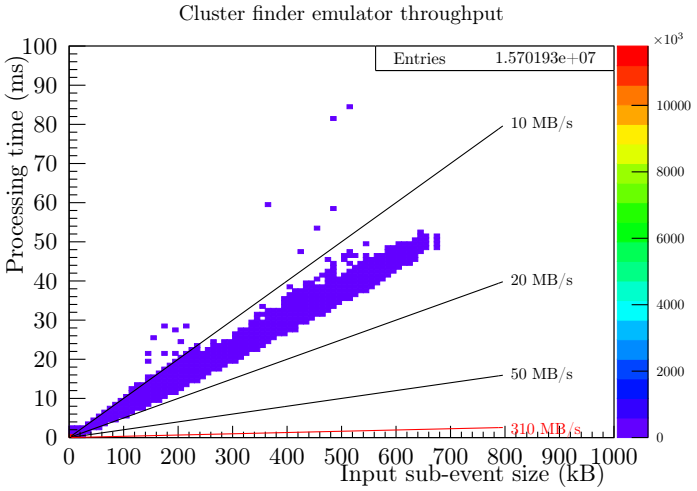

Figure 4.18: Cluster finder emulator throughput measurements using a single instance in an HLT chain.

The cluster finder emulator is the software reference implementation of the HLT hardware algorithm. Its output is bit-wise identical to the hardware implementation and and the internal structure is similar to the processing steps as implemented in hardware. The cluster finder emulator is used in the software model of the HLT in the Offline analysis chain. Designed as a hardware reference, this software is clearly not optimized for a high-performance operation on CPUs. Comparing the hardware implementation with its emulation model, therefore, has to be taken with a grain of salt. Figure 4.18 shows a measurement of the throughout of the cluster finder emulator with Pb–Pb data. The measurements were done using the HLT publisher-subscriber data transport framework on an HLT node. The cluster finder emulator is run as an AliRoot component between a data source and a data sink. A *FilePublisher* component reads the raw sub-events from files on the disk and places them to its output buffer in memory. From there, the sub-events are published continuously to the cluster finder emulator component. A dummy sink component receives the results from the cluster finder emulator. The execution time of the main `doEvent()` loop in the cluster finder emulator in relation to the raw input size is filled into a histogram. The file publisher approach ensures that the event data is available in a RAM buffer before being processed, so the measurements do not contain disk IO bandwidth or latency effects. A set of around 250k Pb–Pb sub-events was continuously sent through the chain for several hours. The processing time depended mostly linearly on the input size, with a few exceptions towards higher processing times. A single cluster finder emulator instance in this setup could handle in average around 12 MB/s. Therefore, a single hardware cluster finder instance in this environment is around 26 times faster than a single emulator instance.

The Offline cluster finder implementation is the reference in terms of the physics performance and uses a different algorithm based on Gaussian fits. This implementation is more complex than the HLT implementation and takes a lot more effects into account. The results of the full HLT processing chain are, however, nearly equivalent in terms of the physics performance as shown in Section 4.4.1. This makes both approaches again comparable to some extent. Besides the hardware cluster finder, the HLT chain contains a cluster transformation component that applies additional calculations, coordinate transformations, and data format optimizations to the cluster finder output. This component has to be taken into account as well for a fair comparison.

The Offline cluster finder processes raw TPC data at a per-event basis. This means that one input data block for the Offline cluster finder contains the raw data from all 216 TPC sub-events received over the 216 DDLs. The hardware cluster finder, on the other hand, processes the raw data at a per-sub-event level, handling one sub-event from one DDL at a time. The configuration of the hardware cluster finder depends on the TPC partition to which the source DDL is connected to. There are six TPC partitions, so a set of six individually configured cluster

finder instances can process any TPC sub-event from any DDL. In order to have a consistent setup for both approaches, a comparison on a per-node level is chosen here. One C-RORC with six hardware cluster finder instances is compared to whatever is possible with the Offline cluster finder on one HLT node. Both approaches process full TPC events. Different running scenarios of the Offline cluster finder were evaluated. The performance measurements of the Offline setup were done by David Rohr. They were conducted on an HLT compute node, once as a single-threaded application and once as a multi-threaded application with 48 threads. The 48 threads were chosen based on the number of available cores in the machine. Each HLT machine contains two Intel Xeon E5-2697 CPUs with 12 physical cores each and 24 virtual cores per socket due to hyper threading. The single-thread performance was additionally measured on a recent Intel Core-i7 6700k CPU to show the performance gains of using a newer CPU.

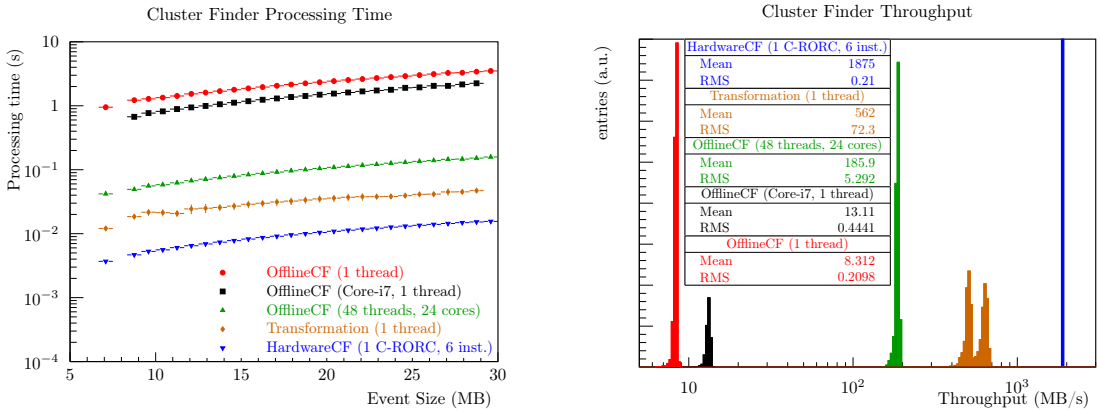

Figure 4.19: Comparison of the computing performance of the hardware cluster finder with different setups of the Offline cluster finder. The same measurements are shown in different ways. Left: Processing time in relation to the event size. Right: Throughput distribution.

The graph on the left in Figure 4.19 shows the processing time of the cluster finder variants in relation to the event size. For each cluster finder variant, the processing time depends approximately linearly on the number of clusters in the event. The measurement is shown in a logarithmic y-scale to cover the range of three orders of magnitude for the processing times. The graph on the right shows the same results plotted as a histogram of the individual data throughput values. These values are calculated as the raw input data size per event divided by the processing time for this event. The Offline cluster finder makes around 8.3 MB/s on an HLT machine when executed as a single thread. The same measurement on the Core-i7 yields around 13.1 MB/s. Running the Offline cluster finder with 48 threads in 24 physical CPU cores on an HLT node makes around 186 MB/s. This multi-threaded configuration is the maximum that can be reached on an HLT node. The hardware cluster finder achieves 1875 MB/s with a very sharp distribution. This throughput equals exactly six times the raw DDL input bandwidth of 312.5 MB/s. The narrow distribution of the measurements around that value confirms that the hardware cluster finder itself can handle the maximum input bandwidth for all sub-events contained in this measurement.

For a fair comparison between the Offline cluster finder and the hardware cluster finder, the cluster transformation component in the HLT chain has to be taken into account. This software component provides some additional calculations and coordinate transformations required to make the output of both approaches comparable. A single transformation component processes in average around 560 MB/s. In order to keep up with the maximum hardware processing rate, up to four transformation components would be necessary per C-RORC. In this measurement setup, one C-RORC plus four CPU cores provide an equivalent processing performance to 480 CPU cores or ten HLT nodes running the Offline cluster finder code.

Chapter 5

Cluster Finder Dataflow Implementation

The firmware efforts around the hardware cluster finder described in Chapter 4 make clear that a hardware processing core like this, integrated into the readout and data reconstruction chain of a large experiment, requires active developments. The experiment and its detectors continuously evolve, conditions change and demands increase. Each iteration forms an evaluation of possibilities for the successor system. The data processing chain repeatedly has to adapt to new situations. For processing steps in hardware, this means that these adoptions have to be applied at the firmware level. The resulting changes to the firmware can be divided into general throughput improvements to handle higher data rates and algorithmic changes to adapt to changing experimental conditions and to improve the physics performance or the compression. The algorithmic changes to the cluster finder are especially cumbersome and error-prone. They require extensive verification, especially when described in a low-level hardware description language. The algorithmic operations have to be manually partitioned and mapped into clock cycles, processing steps, and register stages with a hardware description at the register transfer level. A next iteration of the algorithm may require reworking large parts of the previous implementation, even for comparably small changes at the algorithmic level. The way to implement a processing algorithm in hardware is mostly straight forward for an experienced RTL hardware designer. However, the result is, by design, specialized for that particular case. The options for generic implementations are limited as soon as dedicated IP cores for buffers or mathematical operations are required. Most importantly, a development at that level is slow and the implementation is costly to verify and to maintain in terms of man hours.

The concept of dataflow description of programmable hardware was introduced already in Section 1.6.8. The developer can specify the processing steps at an algorithmic level in a dataflow description language. A compiler takes care of all the tedious steps to translate this into an RT level hardware description that is understood by the FPGA vendor's synthesis tools. A dataflow graph is generated and mapped onto the target architecture by the compiler. The tools align operations relative to each other, match the latency of different steps, prepare IP cores for buffers or mathematical operations if needed, and take care of all control flow tasks like streaming, data validity, and FIFO fill states. Having a processing core like the HLT hardware cluster finder implementation derived from a description at an algorithmic level would make both enhancements and the maintenance of the implementation easier, more flexible, and additionally more accessible to less experienced hardware designers.

In the context of this work, the Maxeler approach for a dataflow description of the HLT hardware cluster finder was evaluated. This processing core is a typical example for a high energy physics readout data pre-processing algorithm. A highly optimized, production-ready, and low-level hardware implementation exists from the HLT application and is available for comparisons.

5.1 Dataflow Computing Concepts and Features

Maxeler refers to its approach as "Maximum Performance Computing" (MPC) [Pel+ 13] and aims to provide the maximum performance for a given problem within the constraints of budget, space, or power. In order to achieve that, a computing platform is designed for the algorithm using a hybrid system of CPUs and dataflow engines (DFEs). The dataflow engines are implemented as custom PCI-Express or InfiniBand-based FPGA. Maxeler provides a dataflow development environment to describe both the host and the DFE implementation. The DFEs can be interconnected with each other to form a chain or a ring of computing entities. A CPU-based host system controls one or more DFEs. It coordinates the dataflow from and to the DFEs while it can be used for CPU-based calculations in parallel. Maxeler provides custom FPGA boards as DFEs integrated into desktop, server, or full-rack systems. These boards typically contain large FPGAs and include as much DRAM as possible with the given FPGA. The hardware used in this work are a MAX3 and a MAX4 DFE. The MAX3 is based on a Xilinx Virtex-6 FPGA with 24 GB of DDR3 memory. The FPGA on this board is from the same series as the FPGA on the C-RORC. This makes it possible to directly compare the VHDL implementation from Chapter 4 with the dataflow approach. The MAX3 is integrated as PCI-Express endpoint into a MaxStation, a regular desktop PC with the Maxeler device drivers and libraries installed. The MAX4 is a PCI-Express board containing an Altera Stratix-V FPGA and 48 GB of DDR3 RAM. The MAX4 is part of an MPC-X system, a rack chassis with up to eight interconnected DFEs that interface to the outside using two 40 Gbps InfiniBand ports. In this case, a separate host machine controls the dataflow from and to the DFEs in the MPC-X via InfiniBand.

The user-specified DFE description is divided into kernels and managers. The term "kernel" is used in a similar context as in OpenCL. A kernel contains a data processing pipeline that implements the user-specified functionality. Depending on the algorithm, the data processing can be distributed into multiple parallel or chained kernels to exploit the available resources in the FPGA and to maximize the throughput. Runtime configuration parameters can be provided to the kernels via "scalar inputs" or mapped memories from the host code. The data transport between the kernels is implemented with FIFOs in the FPGA. This automatically provides a decoupling of the processing steps in the different kernels. The partitioning of processing steps into one or more kernels is defined by the user. Managers instantiate and connect multiple kernels with each other and define the connections to host system, the on-board memory controller, and possibly IO interfaces to other DFEs or to network infrastructure. The interface to the host machine via PCI-Express, as well as the memory controller and inter-DFE connections, are provided by Maxeler and do not need to be developed by the user. With this in place, the manager description is simply a compact high-level structural representation of the interconnects between the kernels and the DFE interfaces to the host or the peripherals.

5.1.1 Maxeler Dataflow Description Language

The behavior of the kernels and managers in the DFE is described with MaxJ [Max 13], an OpenSPL [The 13] implementation using a Java-like syntax. The MaxCompiler translates the textual MaxJ dataflow description into RTL HDL code and netlists. These are then implemented by the FPGA vendor-provided synthesis tools. The MaxCompiler is not a Java-to-hardware compiler and attempts to compile existing Java code will most likely fail or be inefficient. Only the syntax and the object orientation are borrowed to describe the dataflow in a DFE. The standard Java conditional and loop operators are evaluated and unrolled only at compile time. The Java syntax is extended by additional keywords to describe a subset of conditional operations that can be implemented in hardware also at runtime. The MaxJ description is processed by the MaxCompiler to build a dataflow graph of each kernel. This dataflow graph is optimized in a second step and finally mapped onto an internal library of available low-level RTL modules. The latency of each module is known so MaxCompiler can optimize the overall graph for an optimal usage of delay stages. The vendor-specific IP core generators are automatically run by MaxCompiler to create the required building blocks for mathematical operations, FIFOs, or

memories. State machines are available both in the kernels and at the manager level. They can be used for fine-grained flow control or protocol handling. A state machine in a kernel can make it easier to analyze and control the dataflow compared to using counters or control streams. State machines in the manager can be used if a processing step cannot be implemented as a streaming kernel. Manager state machine implementations follow closely the way state machines are described in VHDL or Verilog, only with a different syntax. The inputs and outputs of a kernel are typically implemented with FIFO interfaces. The kernel pulls new input data whenever it is available in the input FIFO and the output FIFO has enough free space to push the result. The input and output interfaces can have almost any width, either as single vectors or as structure of multiple vectors. However, the wider the interfaces the more FIFO resources are necessary for the implementation. MaxCompiler can handle integer, fixed-point and floating-point data types, as well as any derived data structures, with user defined widths and precision. It also supports floating-point exceptions. The precision of mathematical results can either be derived from the input types automatically by the compiler or selected by the user. Casting from one data type to another instantiates the appropriate conversion logic. MaxCompiler strictly enforces the single assignment rule for dataflow processing and denies data dependencies that cannot be implemented as a streaming process. While this feels like a strong limitation when getting started, it ensures that the description can actually be implemented efficiently as a dataflow pipeline.

The MaxCompiler comes with its own simulation environment. The managers and kernels can be compiled into software models for a simulation on a CPU. The same host application that communicates with the real DFE is then instructed to interact with a simulation model of the DFE. These models behaves like the manager and the kernels in hardware. While this environment is obviously slower than working on the real hardware, it is still orders of magnitudes faster than RT level hardware simulation and comes with extensive debugging support that is not possible on the real hardware. Besides the simulation models for the CPU, the generated RTL HDL code can also be simulated in HDL simulation tools.

MaxJ provides a number of optimization and configuration options via dedicated classes to control the hardware implementation. The level of pipeline register stages is configurable per-kernel. This directly influences the implementation trade-off between resource usage and maximum clock frequency. The target clock frequency of each kernel can be set individually to scale with the level of pipelining or to compensate tick-level throughput differences between kernels. There are options to place kernels at specific locations in the FPGA. These options are equivalent to the floor planing steps available in the low-level FPGA design flow. Custom VHDL or Verilog modules can be integrated as kernels into the framework. An interface specification for custom logic helps to develop suitable HDL modules.

A particularly interesting feature in the context of this work are “standalone kernels”. This feature is not part of the regular Maxeler university program but access was generously granted by Maxeler for this work. A special build target creates a netlist for a single kernel with FIFO interfaces for input and output streams, as well as ports for scalar parameters. This netlist is independent from the Maxeler firmware framework and can be integrated into a custom firmware project. It enables the integration of processing blocks described with a dataflow language at an algorithmic level into a firmware environment based on RT level hardware description. Maxeler provides everything that is needed to use dataflow computing on their own hardware. At the time of this writing, there is no support for third-party or custom hardware platforms with the Maxeler dataflow framework. However, especially in high energy physics with various custom protocols and rigid clocking and IO requirements, commercial FPGA boards are often not suitable. Funding regulations and risk evaluations, on the other hand make, it hard to lock the hardware selection to a single vendor even if it is technically suitable. The option to use dataflow programming by integrating standalone kernels into a custom or third party hardware could open a completely new field for high-level hardware description in the high energy physics community.

The code in Figure 5.1 shows a simple MaxJ example for a kernel calculating $(x^2 + x)$ on a stream of 32 bit single-precision floating-point vectors. It can accept a new input datum on each clock cycle and also produces a result on each clock cycle. The input and output vectors and

```
1 class XSquarePlusX extends Kernel {
2   protected XSquarePlusX(KernelParameters params) {
3     super(params);
4     DFEVar x = io.input("input", dfeFloat(8, 24));
5     DFEVar result = x * x + x;
6     io.output("result", result, dfeFloat(8, 24));
7   }
8 }
```

Figure 5.1: Dataflow description for a kernel implementing $(x^2 + x)$ for a stream of 32 bit single-precision floating-point input vectors.

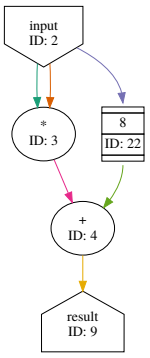

Figure 5.2: Corresponding dataflow graph.

data types are defined in lines 4 and 6. The data type of the actual mathematical operation in line 5 is derived from the input data types. In the dataflow description, all operations within this constructor happen in the same "tick". The concept of ticks is an abstraction layer on top of clock cycles and "counts" the number of execution cycles of the dataflow description. The same tick mark for different hardware operations in a kernel will be at different spacial locations in the internal pipeline at a given clock cycle. In a perfectly pipelined kernel with an uninterrupted input and output stream, the system will handle one tick on each clock cycle. As soon as static or dynamic flow control is applied to kernel input or output streams or an input is idle, clock cycles will pass without completing a tick. The compiler resolves all dependencies between the individual operations and builds a dataflow graph. This graph is optimized to construct a processing pipeline. The resulting dataflow graph for the example above is shown in Figure 5.2. The dependency between the result of the multiplication and the addition in the hardware implementation can clearly be seen. The multiplication of the input vector with itself (ID: 3) has a latency of eight clock cycles. The input vector is delayed with a shift register (ID: 22) by exactly this latency for the final addition step (ID: 4).

Implementing the same algorithm with a hardware description language at the RT level would be done with the same dataflow graph in mind. The multiplication and the addition operations would be generated with the IP core generator. The shift register depth had to be adjusted manually to the latency of the multiplication. The flow control logic to read from the input FIFO or to write to the output FIFO with respect to fill states had to be implemented as well. Apart from the initial time required to write this down, a big advantage of the dataflow approach is its flexibility. The input and output data types can be changed easily, additional operations can be added in no time. Doing the same with VHDL/Verilog requires to manually regenerate all IP cores and to adjust the delay stages with possibly changing latency values. For large kernels, the compiler approach is likely to pack the operations more efficiently into a pipeline than a human designer with separate sub-steps in mind and a modular development approach would do.

```
1 class LpfKernel extends Kernel {
2   protected LpfKernel(KernelParameters params) {
3     super(params);
4     DFEVar x = io.input("input", dfeUInt(32));
5     DFEVar result = (stream.offset(x, -1) + (2 * x) +
6                      stream.offset(x, +1)) / 4;
7     io.output("result", result, dfeUInt(32));
8   }
9 }
```

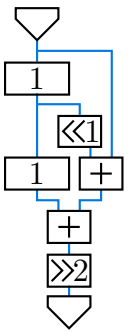

Figure 5.3: Stream offsets in dataflow description (left) and their schematic representation (right).

An important concept of the dataflow description is the `stream.offset()` operator. This operator provides access to a stream value at past or future ticks. Figure 5.3 shows an example of a weighted low-pass / smoothing filter over three samples. The description using the offset operator is shown on the left and the resulting pipeline in a schematic representation on the right. The algorithm correlates the stream value from the current tick with those from the previous and the next ticks. The implementation of the stream offsets in hardware is realized with shift registers. At the RT level, a designer has to implement and connect each register stage manually based on the number of past and future stream values that are required. The more register stages are required, the larger the textual RTL hardware description and the higher the complexity. In the dataflow description, the stream can be accessed at almost any offset without large impacts on the code size or the complexity. The compiler takes care of the process to generate, map, and connect the underlying pipeline stages. Depending on the stream width and the offsets, this is implemented with flip-flops, look-up tables or block RAMs in the FPGA. The maximum available offset range is only limited by the amount of FPGA resources available to implement the according shift register in hardware at the given timing requirements. Stream offsets provide an elegant way to correlate stream samples across past and future ticks while ensuring a compact hardware description and a pipeline implementation in hardware.

5.1.2 MAXELER Kernel Flushing and Host to DFE Communication

The dataflow kernels in the DFE are typically deep pipelines. Whenever a new datum is pushed into the pipeline, a new result from a past input datum is pushed out. While this concept works perfectly for a continuous stream of input data, it needs a careful handling of the start and the end of the data stream. The starting problem is trivial. The first valid output data word leaves the pipeline only a number of ticks after the first input datum is pushed in. This number is equivalent to the latency of the pipeline and known by the compiler. The stopping problem is a bit harder. After the last input datum is pushed into the pipeline and no further data are following, the pipeline is "stuck". Without further input, the data that is still in the pipeline from previous input ticks will not leave the kernel. MAXELER, by default, handles this case with run cycle counts. The number of kernel ticks that are required to handle a given amount of input data is provided to each kernel. As soon as the internal tick count reaches this run cycle count, the kernel stops accepting new input data and internally ticks the kernel with the appropriate amount of dummy data to flush the pipeline. The number of dummy ticks to flush a kernel pipeline is known from the total latency of the pipeline. Another option besides using the run cycle counts is to make the kernel flush itself depending on an input data condition. In this case, the user application has to be aware that the kernel will not accept new data after the flushing condition is met. A third option is to completely disable the automatic kernel flushing. In this case, the user application has to provide a sufficient amount of dummy data after the last valid data word in order to flush all remaining data words from the kernel. The user application has to decide up to which point the output data is valid or if it is the result of the dummy words.

All interaction methods between the host application and the DFE are wrapped inside a PCI-Express interface firmware module as well as a library and a driver stack on host side. This framework hides all buffer allocation, device communication, and DMA data transfer details from the user. A typical host application is `action`-based. An action can contain a pointer to an input data block, a pointer to a memory location for the results, as well as configuration parameters and data sizes. A DFE-based hardware acceleration of an algorithm can be realized with single function calls from a user application containing one or more of these actions. Once one or more actions are started, the software has to wait for those to complete before queuing new actions. The processing pipeline is flushed at the end of each invocation. In order for that scheme to work, the hardware implementation relies on run cycle counts for each kernel that are provided together with the *action*. Exact run cycle counts for each kernel in a chain, however, require the exact number of output data words for a given input size to be known for each kernel in the chain beforehand. While this is typically no problem for image processing algorithms, this

can be a strong limitation for feature extraction tasks where the number of "signals" in the data is not known in advance.

One solution for an algorithm with unknown output data size is to write the processing results into the on-board DRAM first and to send a descriptor containing DRAM addresses and sizes to the host application. The size of the descriptor is known and fixed even if the data processing produced no output data. The actual processing results can then be pulled from the on-board DRAM in a second step. This approach, however, makes the implementation much more complex than necessary. Additionally, it requires that the additional DRAM access latency and bandwidth do not affect the processing performance.

Another workaround for such applications is to use a low-latency API with lower-level device controls and user-controlled pipeline flushing mechanisms in the kernels. The manual flushing removes the need for the run cycle counts and the low-latency interface provides calls to continuously poll the DFE for new data. Unfortunately, this is only available directly via PCI-Express and does not work via InfiniBand. This approach can, therefore, only be used with the MAX3 board in the MaxStation, but not with the MAX4 boards in the MPC-X.

5.1.3 MAXELER Dataflow Tooling

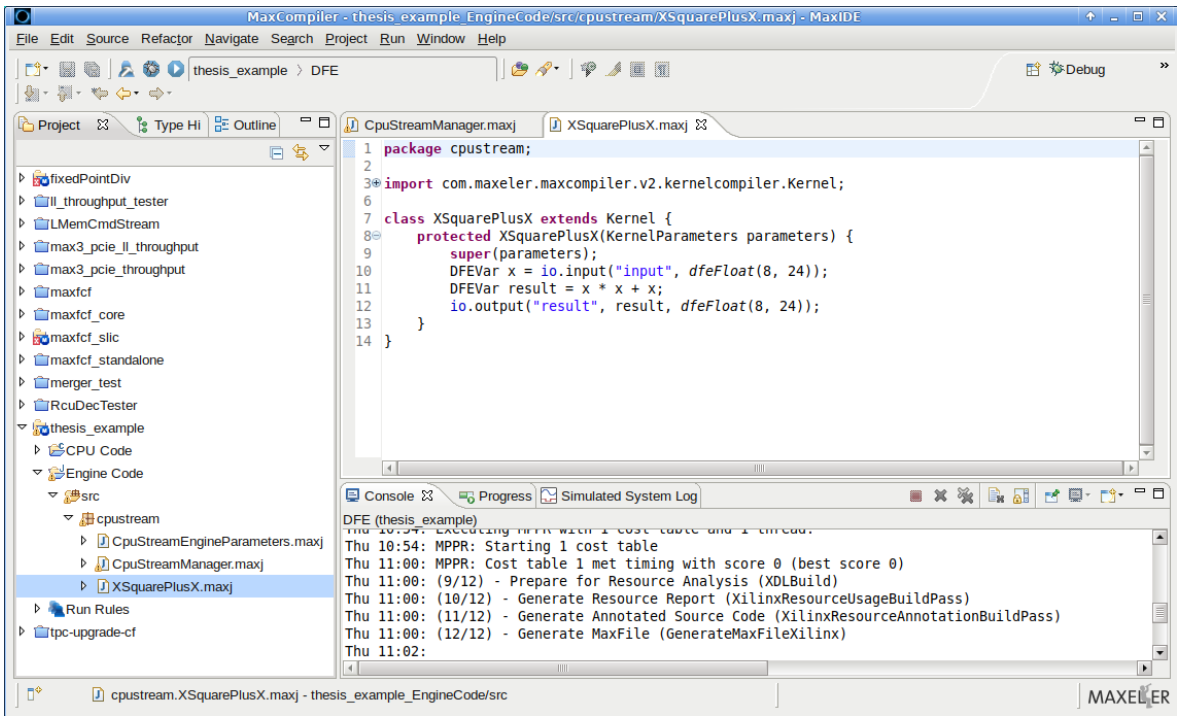

Figure 5.4: MaxIDE screenshot: combined host and DFE code development, simulation, compilation and debugging.

MAXELER provides an Eclipse[1]-based development environment, MaxIDE, for both the DFE and the host code. A screenshot of this IDE is shown in Figure 5.4. The host code can be described in C or C++ and the DFE code with MaxJ. The MaxJ class library bindings are pre-installed and usable out of the box. A project wizard helps to quickly set up a new project and can prepare a kernel, a manager, and a host application with a minimal example if selected. The IDE provides contextual help and documentation for MaxJ via the Eclipse auto-completion and pop-up functionality. "Run rules" allow the user to select the target device or the simulator for a given manager. The MaxIDE executes all required tools in the background. Compilation, simulation, execution, and debugging can all be controlled via the IDE. The compilation process first runs

[1] https://www.eclipse.org

the MaxCompiler and the corresponding IP core generator to create the target-specific HDL and netlist files. The FPGA vendor synthesis tools are then used to build an FPGA configuration file. The host application is compiled with the GNU Compiler Collection (GCC)[2]. Upon execution, it configures the DFE FPGA with the previously built configuration file and runs the specified user code.

The GNU Debugger (GDB) is integrated into the MaxIDE and can be used on the host application. Hardware debugging is possible at different levels. In the hardware simulation, dedicated `log` statements can print out kernel states while running. `Watch` statements can record the state of internal signals on each kernel tick or each clock cycle while active. A spreadsheet is filled with all watched signals on each tick or cycle. This file can be analyzed when the simulation is stopped. Kernels can be compiled with additional debug information. A command-line utility, `maxtop`, can be run while or after the DFE is active to retrieve information about kernel ticks, interface utilization, memory bandwidth utilization, and others, and helps to spot possible bottlenecks or dead locks.

```
1  LUTs   FFs   RAMs  DSPs : XSquarePlusX.maxj
2                         : class XSquarePlusX extends Kernel {
3                         :   protected XSquarePlusX(KernelParameters params) {
4                         :     super(params);
5    1     32    0.0     0 :     DFEVar x = io.input("input", dfeFloat(8, 24));
6  545    686    0.0     2 :     DFEVar result = x * x + x;
7                         :     io.output("result", result, dfeFloat(8, 24));
8                         :   }
9                         : }
```

Figure 5.5: Annotated source code with resource usage information.

In order to evaluate the final resource usage of the dataflow description in the FPGA, the MAXELER tools provide a detailed breakdown of the resources consumed for each line of dataflow code. Figure 5.5 shows the annotated source code of the above example kernel as produced by the tools. The example contains a 32 bit input register stage on line 5. The multiplication and the addition use 545 look-up tables, 686 flip-flops, no block RAMs, and two DSP blocks. The same kind of report is also available at the manager level. Additional compiler reports give information about the latency of the different kernels, the overall resource usage, and the timing margins after the place-and-route step.

MaxCompiler and the chain of FPGA vendor tools are also available as standalone command-line versions. This provides the option to run the time-consuming and resource-intensive synthesis steps separately in a powerful continuous integration or nightly build server.

5.2 Hardware Cluster Finder in Dataflow Description

The hardware cluster finder, as presented in Chapter 4, receives raw detector data from the optical links, processes the data, and sends the information about contained clusters towards the DMA engine. Given the MAXELER hardware and software environment, there are three options to evaluate a dataflow implementation of the cluster finder:

1. A co-processor approach using the MAXELER hardware and software infrastructure in addition to the C-RORC-based detector readout.
2. A full detector readout using MAXELER hardware and software instead of the C-RORC.
3. A dataflow cluster finder integrated into the existing C-RORC firmware, using MAXELER-generated firmware blocks in a custom hardware, firmware and software environment.

[2] https://gnu.org/

The most convenient approach, especially for the initial development and debugging, is to implement the cluster finder as a co-processor with the MAXELER hardware and software. This enables to use all benefits provided by the MAXELER development environment. Raw detector data is pushed into the DFE via PCI-Express and processed clusters are received back. For an integration into the HLT, however, this would require the DFE hardware in addition to the C-RORCs and the integration of the DFE software stack into the HLT data transport framework. The C-RORCs would implement raw data readout for all detectors. TPC raw data would additionally be pushed through the DFEs for cluster finding. This also means, that there are at least two additional PCI-Express hops for raw data before processed clusters are received: from C-RORC to host RAM and from host RAM to DFE. A dataflow implementation would require significant advantages over the existing approach to justify the additional hardware costs, the integration works of yet another software framework, and the additional PCI-Express data transfers.

The second variant, a full detector readout using MAXELER hardware, can reuse the kernels from the co-processor approach in a different manager and an own software integration. An add-on card to the MAX3 DFE exists that provides four serial optical links. These are primarily meant for Ethernet connectivity. Implementing the DDL protocol with custom HDL code integrated into the dataflow environment in theory provides the necessary tools. However, in practice, this solution is not viable. Using the DDL protocol at the required link rates is not possible without hardware modifications. The link density of the C-RORC is not achievable with this hardware or any other add-on board to the DFE. The form factor of the MAX3 plus the add-on board exceeds the maximum height for PCI-Express devices and will not fit into server machines. Additionally, the maximum PCI-Express bandwidth of the MAX3 is only around 65 % of the C-RORC bandwidth, as shown later in Section 5.2.2. With the hardware available at this time, it makes no sense to further follow this approach.

The third approach of integrating a dataflow-described processing core into the C-RORC firmware environment is, however, a good candidate. The C-RORC hardware platform exactly matches the ALICE requirements. The high-level hardware description would ease the development efforts for the algorithmic firmware parts. The standalone kernel feature mentioned above enables to extract the dataflow-described kernels from the MAXELER framework and integrate them into the custom hardware. Additionally, this comes without any modifications to the existing software infrastructure. In the optimal case, the existing RTL hardware cluster finder implementation can simply be replaced by a netlist created with the dataflow tools.

All three approaches share the same cluster finder description in the dataflow language. The approaches differ only at the manager level and on software side. The co-processor approach and the standalone kernel approach were evaluated in more detail.

5.2.1 Cluster Finder Dataflow Architecture

The cluster finder algorithm is divided into different dataflow kernels. Starting from a streaming HDL implementation as the reference, the general structure of the hardware implementation was very likely to remain the same. The functional blocks of the cluster finder are similar to these described in Section 4.1.2. The RCU Decoder parses and serializes the ADC samples contained in the DDL words. The Channel Processor provides peak finding, region of interest cuts and the weighted sums for the cluster property calculations in time direction. The Channel Merger buffers the stream of cluster candidates and merges signals from neighboring pads. The Divider performs the final division of the weighted sums by the total cluster charge. Interfaces between the dataflow kernels are implemented with FIFOs in the MAXELER framework. The number of kernels should be kept as low as possible. The more functionality is contained in a single kernel, the better the compiler can optimize the individual processing steps. Additionally, each interface between kernels uses FIFOs which require FPGA block RAM resources. A low number of kernels also keeps the resource usage lower. On the other hand, FIFO interfaces between kernels provide an efficient way to decouple data streams if needed.

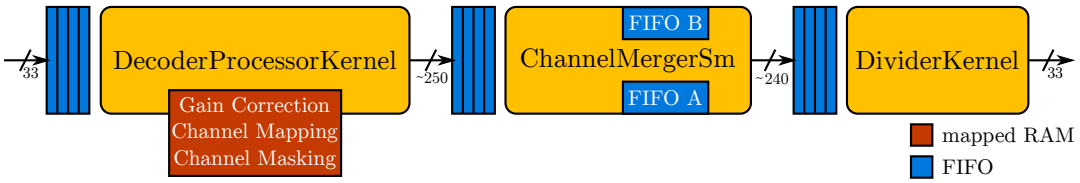

Figure 5.6: Cluster finder partitioning into dataflow kernels.

Figure 5.6 shows how the cluster finder algorithm is distributed across dataflow kernels. The functionality of the RCU Decoder and the Channel Processor are integrated into the same *DecoderProcessorKernel*. This kernel accepts new DDL input data words every third clock cycle to remove any data dependency between the input data and the control stream. This is done for the same reasons that are described in the discussion of the rewrite of the RCU Decoder for the common RCU1+RCU2 cluster finder in Section 4.2.5. The output of the RCU Decoder is a stream of ADC samples that also forms the input to the Channel Processor. Both are streaming processors. This means, there is no need for a decoupling FIFO stage between them. It enables both modules to be integrated into the same dataflow kernel, the *DecoderProcessorKernel*. The cluster finder configuration table containing row and pad information for each channel, as well as the gain correction factors, are integrated into the kernel as a mapped RAM. The configuration values can, therefore, be set at runtime in the same way as it is implemented in the VHDL/HLT variant. The Channel Processor part of this kernel exploits the `stream.offset()` operator described above to correlate charge samples from previous and future time stamps. This is used to detect peaks and to mark samples contributing to the region of interest around a peak. The HDL-equivalent as used in the VHDL implementation is to feed the intermediate signals of shift registers into a combinatorial logic element together with a considerable amount of flow control logic. The dataflow method is much more flexible, less verbose, and independent of the offset range.

```
1 // first multiplication stage
2 DFEVar GxC = gain * charge;
3 DFEVar TxT = time * time;
4 DFEVar PxP = pad * pad;
5 // second multiplication stage
6 DFEVar GxCxT = GxC * time;
7 DFEVar GxCxP = GxC * pad;
8 DFEVar GxCxTxT = GxC * TxT;
9 DFEVar GxCxPxP = GxC * PxP;
```

Figure 5.7: Fixed-point multiplications in the dataflow description. All data types and the latency of the different operations are handled by the tool.

The multiplications of `charge`, `gain`, `pad`, and `time` values can directly be written as such in the dataflow description as shown in Figure 5.7. `charge`, `time`, and `pad` are unsigned integers of 10, 10, and 8 bit. `gain` is a fixed-point number with one integer and 12 fractional bits. The dataflow compiler derives the bit-widths of the results automatically from the input types. The VHDL-equivalent requires individual IP cores for each multiplication and multiple shift registers to compensate the latency of the first multiplication stage for the inputs to the second stage.

The Channel Merger cannot be formulated as a dataflow kernel due to the data ordering of the TPC readout. This module has strong data and control dependencies given by the general TPC readout architecture. Reformulating the algorithm to achieve a full dataflow implementation is not possible in this case. In VHDL, the Channel Merger is implemented as a state machine with buffer FIFOs for up to two TPC branches. The same concept can be achieved in the dataflow context with a manager state machine, the *ChannelMergerSm*. Input and output FIFOs decouple the data dependent processing time of the *ChannelMergerSm* from the other dataflow kernels with

deterministic throughput. This is equivalent to the VHDL implementation. The description of a manager state machine with MaxJ is different from dataflow kernels but conceptually very close to VHDL or Verilog. With the VHDL Channel Merger in place, this code could be translated into an equivalent MaxJ manager state machine.

```
1 DFEVar dividend = dividend_fixed.cast(dfeFloat(8, 24));
2 DFEVar divisor = divisor_fixed.cast(dfeFloat(8, 24));
3 DFEVar result = dividend.div(divisor);
```

Figure 5.8: Core logic of the cluster finder *DividerKernel*.

The *DividerKernel* is again a pure dataflow kernel and implemented equivalently to its VHDL counterpart. This includes the decision to reuse the same divider instance for all four division operations in a time-multiplexed manner. The dataflow description is much simpler than the VHDL equivalent. The cast from fixed-point to floating-point, as well as the floating-point division, can simply be written as such and are independent of the input data type. The compiler takes care of all details about the bit widths, the IP cores and the latency of individual operations. The dataflow description of the core logic inside the *DividerKernel* is shown in Figure 5.8.

The input interface to the *DecoderProcessorKernel* and the output interface from the *DividerKernel* are the same as in the VHDL implementation from Chapter 4. The input interface is defined from the DDL implementation to consist of 32 bit data words plus a 1 bit end-of-event flag. The output interface could have been implemented with a wider bus to achieve a higher throughput of the *DividerKernel*. However, since the *DividerKernel* is not limiting the overall throughput, this brings no improvements, has barely an effect on the resource usage, but would make data formatting for the DMA transfer more complex. For these reasons, the same output interface as with the VHDL implementation is also chosen here. This additionally brings the benefit that both implementations remain well comparable.

An important aspect of the TPC hardware cluster finder in the dataflow description is the fact that the amount of output data per sub-event depends on the content and not the amount of the raw input data. This brings some challenges to the implementation of the kernel flushing strategy in the dataflow description. Detector data is always read out in multiples of sub-events via the DDL. The size of each sub-event is typically in the order of several Bytes up to a few Megabytes. The actual size of an incoming sub-event is not known beforehand and is only available once the full sub-event was received. The stream of data from the detector can stop after each sub-event. This means that the pipeline has to be flushed whenever a sub-event is completed and no other sub-events are following. The pipeline could be flushed unconditionally after each sub-event. However, not accepting new input data while the following sub-events already queue up would reduce the throughput. A flushing strategy that detects the end of a sub-event from the data stream is possible, but will stall the input queue and flush the kernel even if new input data is available. Using run cycle counts to flush the chain of kernels is not possible because neither the amount of input data nor the amount of output data per sub-event is known beforehand. Disabling the kernel flushing mechanisms is a good option but requires feeding dummy input data whenever the dataflow from the detector stops. Only by disabling the internal kernel flushing mechanisms and handling the end of the data stream manually, the implementation could reach the same throughput as the RTL variant.

5.2.2 Integration into the Maxeler Hardware and Software

The implementation of the TPC hardware cluster finder with the Maxeler-provided hardware and firmware is, in general, possible in two ways. A co-processor approach streams the detector data from the host through the DFE and back to the host. A direct detector readout using an add-on board with optical links could provide similar functionality as the C-RORC without the efforts to build the DMA part and the data processing part in VHDL. Section 5.2 already explained why the direct readout is not a viable option in this case due to hardware limitations. However, the co-

processor approach is possible and provides all benefits of the MAXELER approach. Additionally, the dataflow kernels from this implementation can be used in any of the other possible approaches as well.

The co-processor implementation uses the MAXELER software API to communicate with the DFE. A host application streams detector data via PCI-Express to the DFE and receives the processed results back. In order to evaluate the performance of this approach, the maximum PCI-Express throughput of the MAXELER hardware is measured for two devices, the MAX3 and the MAX4, as well as two interface methods, the action interface and the low latency interface.

DMA throughput is measured with a simple kernel forwarding data with a configurable input-to-output ratio. The DMA device-to-host throughput measurements are done with an input-to-output ratio of 1:1024. For each input word received, the kernel produces 1024 output words. The DMA host-to-device measurements use the inverse ratio of 1024:1. Only every 1024^{th} input word is sent back to the host. The `actions` interface call takes pointers to the memory regions for the input and the output data blocks as well as the configuration parameters and run cycle counts for the given input-to-output ratio. This function is called repeatedly. The input data rate and the output data rate are recorded while the input and output block sizes are modified.

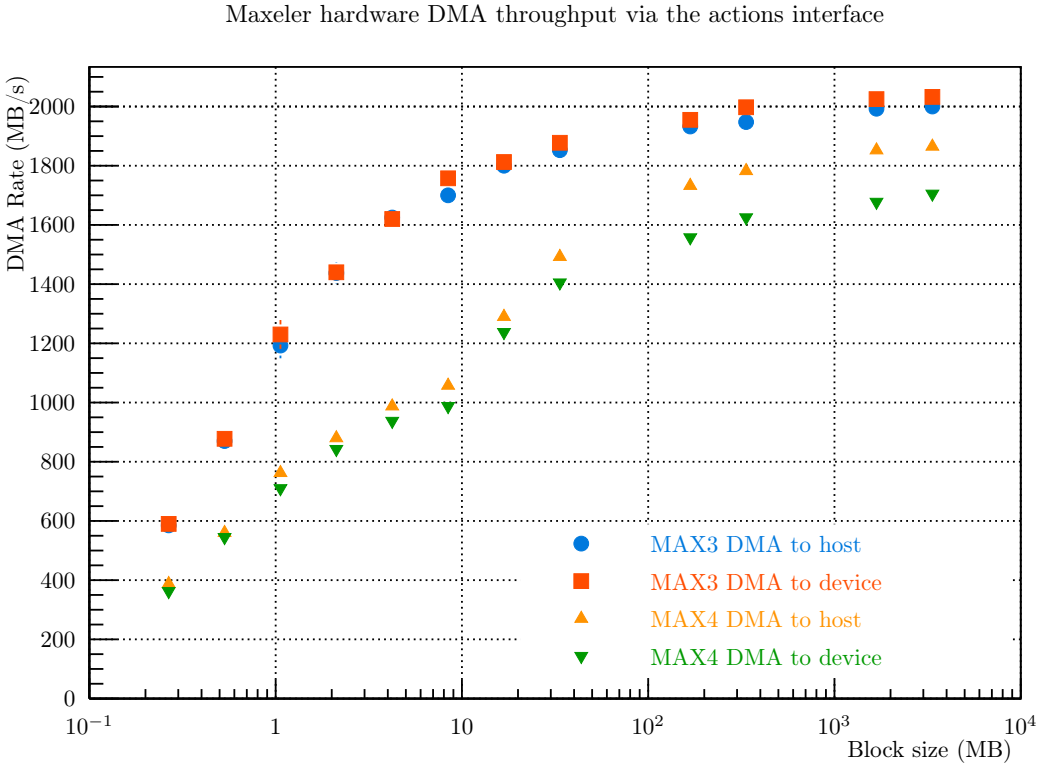

Figure 5.9: MAXELER MAX3 (40 Gbps PCI-Express) and MAX4 (MPC-X, 40 Gbps InfiniBand) DMA throughput using the `actions` interface.

Figure 5.9 shows the results of these measurements with the MAX3 board (XILINX Virtex-6) via PCI-Express and the MAX4 board (ALTERA Stratix-V) via InfiniBand. The overall throughput is strongly dependent on the block size. This dependence is much stronger than on the custom C-RORC DMA engine. The C-RORC reaches the maximum PCI-Express throughput at a block size of around 10 kB. The MAX3 and MAX4 with the `actions` interface only start to saturate at block sizes above 100 MB. The stronger dependency is partly expected. A new action can only be started after the previous one was completed. This clearly introduces a latency limitation because the data streams from and to the DFE cannot be decoupled. The maximum throughput on the MAX3 is around 2 GB/s for both directions. The MAX3 has two FPGAs, one that handles the PCI-Express interface and one that actually runs the user specified DFE code. The

maximum bandwidth between these two FPGAs is slightly above 2 GB/s, limited by the FPGA interconnect bus. So even though the MAX3 has a PCI-Express generation-2 interface with eight lanes, the usable throughput to and from the DFE is only around half this value. The same latency dependency is also there on the MAX4 via InfiniBand. While the raw interface bandwidth of the InfiniBand link is the same as the raw PCI-Express interface bandwidth, 40 Gbps, the maximum throughput is lower and the block size dependency even stronger. The DMA device-to-host throughput is limited to around 1.9 GB/s and the DMA host-to-device throughput to around 1.7 GB/s.

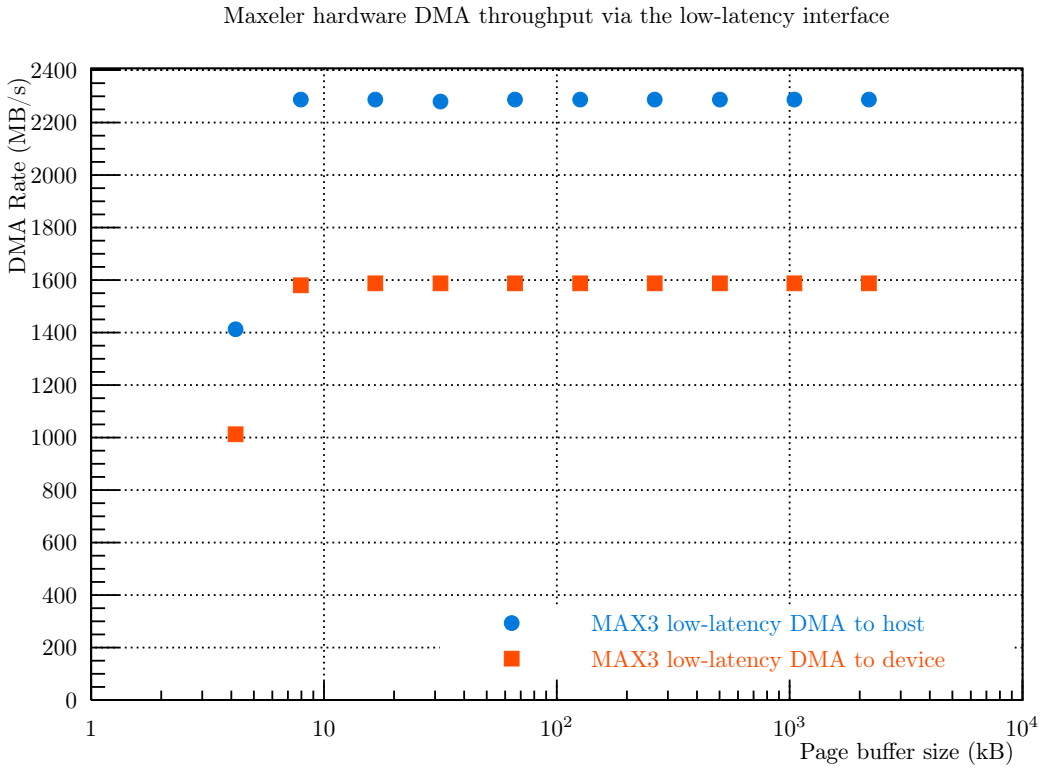

Figure 5.10: MAXELER MAX3 DMA throughput using the low-latency interface.

The throughput measurements with the low-latency interface were done with the same DFE implementation and the same input-to-output data ratios but with a different host code. The low-latency interface works with "slots" of a configurable size. The maximum slot size is one page, 4 kB. This value is used for all measurements. The number of slots define the maximum queue length of the data to and from the DFE. The total queue size is, therefore, the number of slots multiplied with the slot size. This value is referred to as "page buffer size" in the measurement results in Figure 5.10. For this interface, the data streams to and from the DFE are independent and similar to the C-RORC HLT implementation. Like on the C-RORC, the full bandwidth is available from around 10 kB buffer size. The maximum DMA device-to-host throughput is around 2.3 GB/s and the DMA host-to-device throughput is limited to around 1.6 GB/s.

The available PCI-Express or InfiniBand bandwidth on the MAX3 and the MAX4 hardware with both the `action`-interface and the low-latency interface is, even in the best case, only around 65 % of the usable PCI-Express bandwidth in the C-RORC implementation. This also confirms that this hardware would not be suitable as a replacement for the C-RORC in the ALICE HLT application. Nevertheless, the cluster finder can be evaluated with the dataflow description in the co-processor approach because the same dataflow code can then be used with any readout approach.

In order to use a cluster finder instance from a dataflow description in the MAXELER framework a bit of "glue logic" is placed around. A *DdlReaderKernel* translates the data from the 128 bit

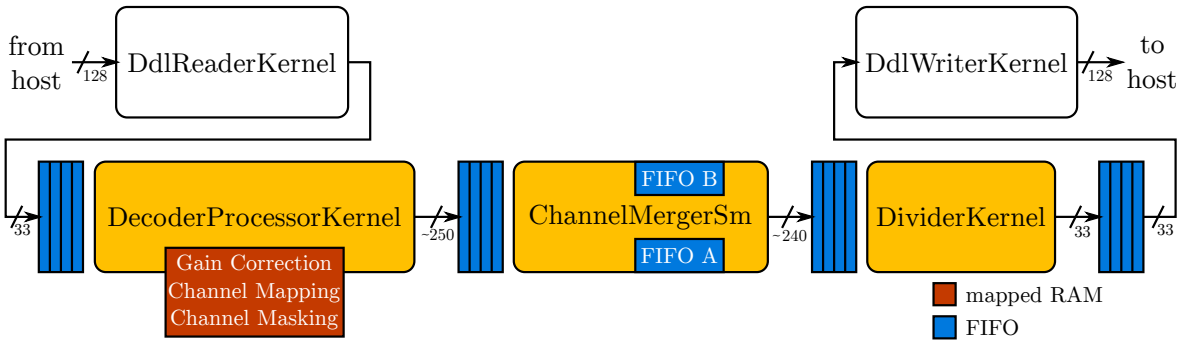

Figure 5.11: Cluster finder as co-processor implementation.

PCI-Express host interface into the 33 bit DDL input data format. The DDL data format could also come directly via PCI-Express, but using the *DdlReaderKernel* provides a more efficient PCI-Express bandwidth usage. A second option would be to change the interface of the *DecoderProcessorKernel* to 128 bit and directly connect it to the host interface. This would slightly reduce the resource usage of the overall design, but made the implementation only usable in the co-processor application. The translation logic consumes only a marginal amount of FPGA resources and does not limit the throughput of the cluster finder. A similar kernel, the *DdlWriterKernel*, is placed between the output of the cluster finder and the PCI-Express interface for the same reasons. An overview of the full chain of kernels for one cluster finder instance in the co-processor application is shown in Figure 5.11.

	Look-up tables	Flip-flops	Block RAMs	DSPs
MAX3 with 1 cluster finder instance	11211	13363	32	8
→ used by kernels	4029	6381	6	8
→ used by manager	6914	6933	27	0
→ stray resources	175	49	0	0
MAX3 with 6 cluster finder instances	42262	55861	160	48
→ used by kernels	23670	38261	33	48
→ used by manager	17721	17466	127	0
→ stray resources	655	134	0	0
C-RORC co-processor implementation with 6 cluster finder instances	81591	113753	285	48
→ used by cluster finders	26706	31014	114	48
→ PCI-Express & 12 DMA engines	30416	48725	126	0

Table 5.1: FPGA resource usage of the dataflow implementation on the MAX3 board in comparison to the C-RORC cluster finder co-processor firmware variant.

Table 5.1 shows the resource usage of one and six cluster finder instances compiled for the MAX3 board in comparison to the C-RORC cluster finder co-processor firmware. The total numbers include all framework and infrastructure parts of the full MAXELER implementation, not only the data processing kernels. The resource usage of the dataflow implementation is broken down into kernel, manager, and stray resources. The kernel resource usage contains all logic that is part of the individual kernels. These are the resources used for the cluster finder and the input/output formatting kernels, but do not contain FIFOs between the kernels. The FIFOs between the kernels, as well as the interfaces towards the host machine, are part of the manager. As expected, the number of kernel resources scales roughly with a factor of six from one to six cluster finder instances. The manager resources contain a static framework part which is independent of the number of kernels, so this number grows slower.

The lowest two lines of this table show the resource usage of the HLT cluster finder co-processor firmware variant, which has a similar functionality as the dataflow implementation. The MAX3 hardware contains a Virtex-6 FPGA that is from the same device family as the FPGA on the C-RORC. The resource usage in terms of look-up tables, flip-flops, DSPs and block RAMs is, therefore, comparable in the sense that both implementations use exactly the same building blocks at the FPGA level. Both implementations are synthesized with six cluster finder instances and identical configuration parameters. However, one important aspect on the MAX3 board is that all PCI-Express interface and DMA transfer logic is part of a second FPGA on this board. This second FPGA handles the PCI-Express interface and its configuration is not changed when a new dataflow firmware is loaded. The "manager" part in Table 5.1, therefore, contains the logic for an FPGA-to-FPGA interconnect to this interface FPGA. The C-RORC, on the other hand, implements all functionality in one FPGA. The firmware variant shown here contains a full 12-channel DMA engine via an eight-lane PCI-Express generation-2 interface where each of the 12 channels can be operated independently. The MAX3 manager parts contain a four-lane Aurora[3] FPGA-to-FPGA interconnect to the second FPGA.

The Maxeler framework in the implementation on the MAX3 hardware takes only comparably small portions of FPGA resources and leaves large parts of the FPGA available for the user-defined dataflow processing algorithms. A comparison of the full MAX3 implementation with the full C-RORC co-processor implementation is difficult. The functionality that is implemented in one FPGA on the C-RORC is distributed across two FPGAs on the MAX3. The inter-FPGA link is typically much lighter than a PCI-Express DMA engine. The resource usage of the interface-FPGA on the MAX3 is not published. Only the numbers for the primary MAX3 FPGA containing the user described dataflow implementation are available. The approaches for the data transfer to and from the FPGA are different as well. The equivalent of the resource usage of only the cluster finder instances in the C-RORC VHDL implementation are the numbers for resources used by the dataflow kernels plus a share of the manager resources. The functionality and the approaches of both implementations are different and make it hard to do a fair comparison. On this coarse scale, the resource usage of both implementations is in the same order of magnitude. The resource usage numbers are analyzed in more detail in Section 5.2.3. The standalone implementation also enables to better compare individual parts of the algorithm.

5.2.3 Integration as Standalone Dataflow Core into the C-RORC HLT Firmware

Standalone kernels are a special feature of the MaxCompiler. A single dataflow kernel is not integrated into the full Maxeler framework but instead implemented as a netlist without ties to the Maxeler firmware infrastructure. The generation of standalone kernels requires a special license that is not included by default in the Maxeler university program. Access to this feature was generously granted by Maxeler for this work.

```
1 public static void buildChannelProcessorNetlist() {
2   StandaloneKernelHWManager manager = new StandaloneKernelHWManager(
3     "DecoderProcessorKernel", DFEModel.VECTIS, GenerationMode.HARDWARE_NETLIST);
4   Kernel kernel = new DecoderProcessorKernel(manager.makeKernelParameters());
5   manager.setKernel(kernel);
6   manager.build();
7 }
```

Figure 5.12: Building a standalone kernel. A special manager type enables to synthesize a single kernel into a netlist.

Standalone kernels can be generated with a dedicated manager type that is assigned a single kernel instance. Figure 5.12 shows how a standalone kernel can be built. The result is a netlist that can be integrated into another firmware project. In order to be usable without the Maxeler

[3] Aurora is a light-weight protocol on top of multi-gigabit transceivers for Xilinx FPGAs

context, some aspects of the input and output interfaces are slightly different for standalone kernel compared to regular dataflow kernels. The streaming inputs and outputs are implemented with FIFO interfaces. The kernel pulls data from an input FIFO as long as it is not empty and pushes its results to an output FIFO as long as it is not full. Standard kernels have an interface to an on-chip slow-control bus to get and set scalars and mapped memories. In the case of the standalone kernels, this on-chip bus is not available and these parameters are routed to the kernel's top level entity as parallel input or output ports.

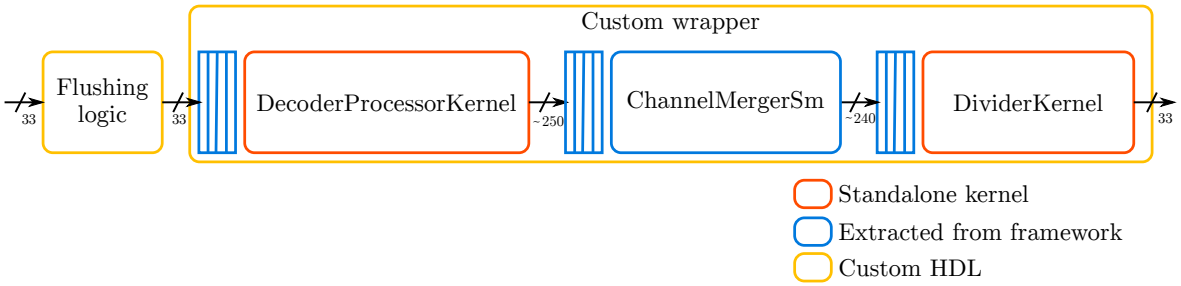

Figure 5.13: Building blocks of the cluster finder implementation using dataflow standalone kernels and framework generated parts.

Standalone kernels are only available for single kernels. One kernel is compiled to one netlist. The feature is also not available for kernel state machines and does not include the FIFOs between kernels. This means that it is currently not possible to use the MaxCompiler to build a single netlist for a chain of multiple kernels or manager state machines. A full cluster finder instance, however, consists of two kernels and a manager state machine, as well as the FIFOs in front and in between. Standalone kernels can only be built separately for the *DecoderProcessorKernel* and the *DividerKernel*. In order to integrate these into the existing C-RORC HLT firmware framework, the *ChannelMergerSm* and the FIFOs are still missing. While it would be possible to use the VHDL implementation for the *ChannelMergerSm* and the FIFOs, the MAXELER approach also provides the required files by combining the results from different dataflow build steps with a custom wrapper.

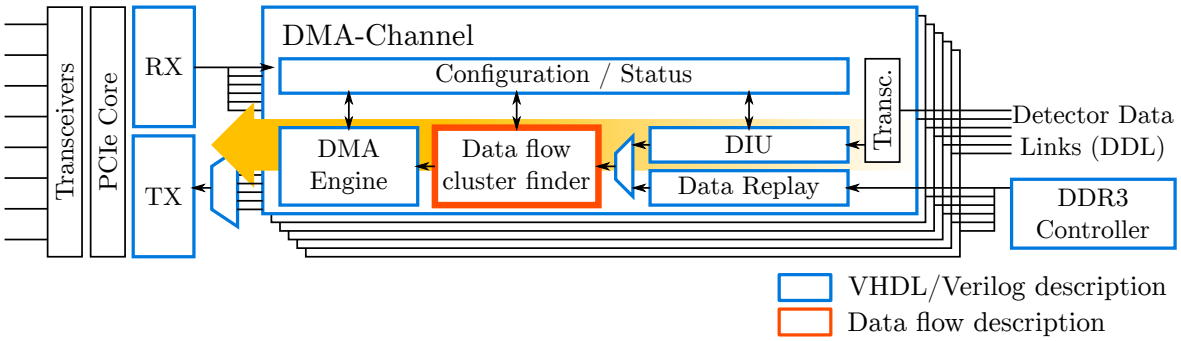

Figure 5.14: Dataflow cluster finder integrated into an RTL HDL firmware design.

The *DecoderProcessorKernel* is compiled as a standalone kernel and gives a netlist that can be used. The same is true for the *DividerKernel*. The *ChannelMergerSm* is created as a VHDL file when building the regular dataflow firmware in the MAXELER framework. The manager state machine has no framework-specific interfaces and can, therefore, be extracted as well. The FIFOs are also available from the standard implementation without ties to the framework. The big advantage of using the generated cores instead of own variants is that they are automatically adjusted to changing stream widths if the input or output interfaces of the kernels or state machines are modified in the dataflow description. A custom HDL wrapper around the two standalone kernels, the manager state machine, and the generated FIFOs together with some flushing logic enables to combine those parts into a single netlist. The components involved in this netlist are shown in Figure 5.13. A custom build script picks up the sources and netlists

from the different dataflow compiler steps for the standalone kernels and the full implementation. These are then compiled to a single combined netlist containing a full cluster finder instance including the FIFOs. This netlist can be instantiated in the C-RORC HLT firmware in exactly the same way as the VHDL implementation. A schematic representation of this integration is shown in Figure 5.14. The HLT C-RORC firmware build flow picks up the netlist generated with the dataflow tools and integrates it into the detector readout or the co-processor firmware. Both the VHDL and the dataflow implementations are integrated with the same clock frequency and both variants meet the same timing requirements. The *DecoderProcessorKernel* and the *DividerKernel* are pipelined with the same concepts as their VHDL equivalents. The functionality of the *ChannelMergerSm*, which is the throughput limiting component in the chain, is even identical. This makes the throughput of both implementations identical. The latency of the dataflow implementation is slightly higher and shown in more detail in Section 5.2.4.

5.2.4 Resource Usage Comparison

	Dataflow implementation				VHDL implementation			
	LUTs	FFs	BRAMs	DSPs	LUTs	FFs	BRAMs	DSPs
DecoderProcessorKernel	1226	2005	4	8	828	1121	4	8
ChannelmergerSm	1680	953	8	0	1444	1439	8	0
DividerKernel	1910	2866	2	0	1620	1481	0	0
Total	5045	6063	21	8	4451	5169	19	8

Table 5.2: Comparison of the resource usage between the dataflow and the VHDL implementation for the individual modules and the full instance. The "total" resource number line contains more than the sum of the individual modules since the FIFOs between the modules and configuration, status and monitoring logic are not included in the modules themselves.

Table 5.2 shows the FPGA resource usage of the cluster finder in the dataflow implementation and the HLT VHDL implementation. The dataflow numbers were extracted from a C-RORC firmware with six standalone dataflow cluster finder instances integrated into the HLT readout firmware. The numbers for the VHDL firmware were taken from the HLT readout firmware with six cluster finder instances as used in the HLT production system at the start of Run 2. Both designs were synthesized with the same target clock frequency and the same build system. The resource numbers are compared here for the individual dataflow modules as well as for a single full cluster finder instance. Since the partitioning of the algorithm is slightly different in the VHDL implementation, these numbers show the accumulated resource counts of the modules with the corresponding functionality as the dataflow kernel. The FIFO blocks between modules were explicitly excluded from the resource counts to have comparable numbers between both implementations. All resource numbers were extracted from the firmware variants after the place-and-route step.

The *DecoderProcessorKernel* as the first processing step contains all used DSP blocks in both variants. Eight DSPs are used for seven multiply-accumulate operations in the same way in both implementations. Both variants also use four block RAMs for the cluster finder mapping and configuration storage. The number of both the look-up tables and the flip-flops is around 50–80 % higher in the dataflow implementation compared to the VHDL variant. The *ChannelMergerSm* is conceptually identical between the dataflow and the VHDL implementation. However, the VHDL variant contains fewer look-up tables and more flip-flops than the dataflow implementation. The reason for this difference is that the VHDL implementation enforces the usage of flip-flops instead of look-up tables as shift registers at some positions to achieve a higher clock frequency. The *DividerKernel* is implemented in a slightly different way in both variants. The dataflow implementation uses more flip-flops, more look-up tables, and more Block RAMs compared to the VHDL implementation. The additional block RAMs in the dataflow case are used as delay units to match the latency of division results to values that are only passed through.

Latency in clock cycles	Dataflow implementation	VHDL implementation
DecoderProcessorKernel	38	19
DividerKernel	44	36

Table 5.3: Latency in clock cycles of the dataflow kernels and the VHDL implementation. The *ChannelMergerSm* is not shown here since its functionally is identical for both implementations and its latency is defined by the data.

The generally higher resource usage for the dataflow implementation already suggests that the latency of the individual kernels may not be identical. This inevitably leads to different resource requirements. The latency is the depth of the pipeline within the corresponding modules or kernels. Table 5.3 shows the latency numbers for the two dataflow kernels and their equivalents from the VHDL implementation. For both the *DecoderProcessorKernel* and the *DividerKernel*, the latency of the dataflow implementation is higher than the latency in the VHDL implementation. The dataflow implementation of the *DecoderProcessorKernel* even has double the latency of the VHDL variant. The increased latency of the dataflow implementation explains the generally higher resource usage for the kernels in comparison to the VHDL equivalents. The Channel Merger is not shown here because this state machine is implemented identically in both variants and has a data dependent processing time and latency.

5.2.5 Lines of Code

The implementation of the cluster finder from the dataflow description uses a completely different abstraction layer than the VHDL implementation at the RT level. The dataflow description happens at an algorithmic level, while the VHDL implementation is described at a cycle-accurate bit-level processing model. One of the strong benefits of describing an algorithm at a higher abstraction level is the reduced description size. A mathematical operation can typically be described in a single line of code in the dataflow context but may require several modules for format conversions, calculations and possibly dedicated IP core instances to achieve the same behavior at the RT level. Each of the individual modules of the cluster finder consist of a single file in the dataflow description. The FIFOs between kernels are not part of the kernel code itself but are automatically added by the framework. The functionality of a dataflow kernel is distributed across several source files in the modular RTL approach, including FIFOs. In order to compare both descriptions, the RTL equivalents contain the sum of the lines of code for all HDL files connected to the according functionality. It has to be noted that both descriptions are not written in a way to keep the number of lines of code particularly low or high. Both contain comments, likely in an inconsistent verbosity. Comparing these numbers is to illustrate a possible reduction in code complexity. However, the actual numbers have to be taken with a grain salt. Both descriptions can be written with more or fewer comments and complexity. This immediately affects the number of lines of code but does not necessarily improve the maintainability of the description.

Table 5.4 shows the numbers of code lines for both cluster finder implementations. The functionality of the *DecoderProcessorKernel* is around 14 times smaller in the dataflow description than in the RTL code. The *ChannelMergerSm* as a manager state machine is partly at the same abstraction level as its RTL equivalent. Its dataflow description is still only around half the size of the VHDL code. The *DividerKernel* in the dataflow description is around eleven times smaller than the equivalent functionality in the RTL code. The "manager" code lines in the dataflow case contain the instances of the kernels, the manager state machine from the table above, and communication channels between them. There is no direct equivalent for this in the RTL code, since the hierarchy of modules is arranged slightly different there. The row with the "total" number of code lines contains more than the sum of the previous rows, since it includes data type definitions and configuration parameters. Not part of the RTL lines of code is the description of the IP core properties required to generate the according cores. However, their instances in the

	Dataflow lines of code (MaxJ)	RTL lines of code (VHDL / Verilog)
DecoderProcessorKernel	219	3066
ChannelmergerSm	944	1973
DividerKernel	81	946
Manager	128	n.a.
Total	1691	7968

Table 5.4: Lines of code comparison between the cluster finder dataflow description and the RTL HDL implementation. The "total" row is more than the sum of the individual modules since it contains additional code for data type definitions and parameters.

code are included. Both descriptions use a couple of packages and wrappers to provide generic configuration parameters and data type definitions to all kernels and the state machine.

The per-kernel breakdown of the number of code lines clearly shows the benefits of the dataflow description. The code volume is reduced by more than a factor of ten for both dataflow kernels. This comes with a significantly reduced code complexity. The *ChannelMergerSm* has to be treated separately because this is not a dataflow kernel and is partly described at the same abstraction level as the RTL equivalent. Both variants contain some static overhead for parameters and data types. Even including the manager state machine, the dataflow description code is around a factor of 4.7 times smaller than the RTL code in this case.

5.3 Evaluating Cluster Finder Modifications

The reduced code volume and the higher-level description of the algorithm enables testing ideas much more easily than what would be possible at the RT level. Changing data types in the VHDL description requires regenerating IP cores with possibly changing latency numbers whenever derived signals are used in more than simple mathematical operations. The dataflow approach hides all aspects related to the latency or the IP core generation from the user and handles them automatically in the background. The previous sections confirmed that the resource usage of the dataflow implementation is at least in the same order of magnitude as the manual RTL implementation. This enables to also use the dataflow description for feasibility studies and resource estimations. This can be valuable even if the final implementation would be done manually again.

5.3.1 Adjusting the Level of Pipelining

A dataflow kernel is implemented as a pipeline. In the optimal case, a new data word can be processed on each clock cycle. The latency of the kernel, as the number of clock cycles it takes from pushing a data word into the kernel until the corresponding result is sent out, depends on the depth of this pipeline. The same kernel can be implemented with a higher or lower depth of the pipeline. A lower depth means more operations have to be performed in the same clock cycle, reducing the resource usage but also reducing the maximum operation frequency. A higher depth reduces the number of operations per clock cycle, improves the timing performance but also increases the resource usage. Selecting the appropriate pipeline depth is a trade-off between clock frequency and resource usage.

The resource usage comparison in Section 5.2.4 already showed that the default dataflow implementation has a higher pipeline depth than the VHDL implementation. The MAXELER framework enables to control the level of pipelining applied to dataflow kernels with an optimization parameter. The `optimization.pushPipeliningFactor()` method can set the desired level of pipelining as a floating-point value between 0 and 1 for each kernel. A value of 0 requests the lowest possible

latency and resource usage, while the default value of 1 uses the maximum pipeline depth. This setting is a suggestion to the compiler and it depends on the kernel to which extent this can be respected. Changing the level of pipelining in the dataflow approach is about changing a number in a single line of code. With the manual VHDL approach, changing the level of pipelining requires to carefully add or remove register stages, re-time operations, and possibly re-create IP cores with different settings.

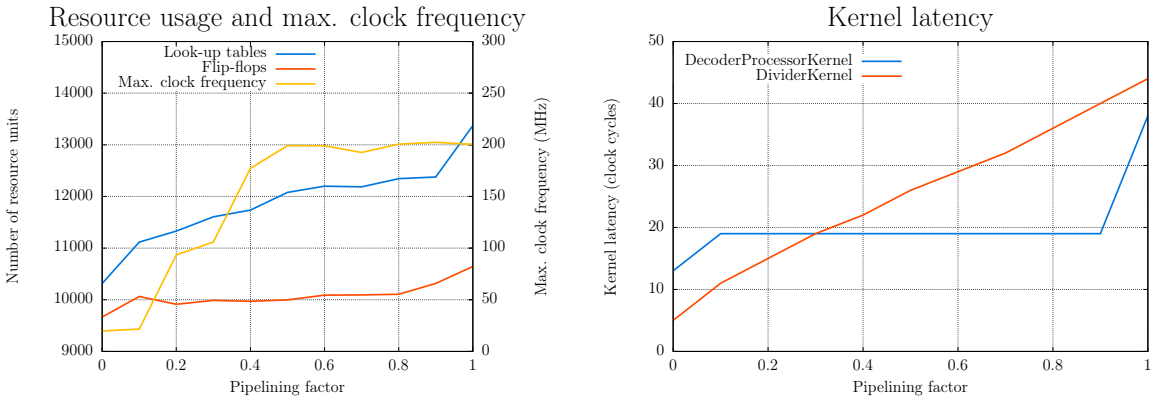

Figure 5.15: Left: FPGA resource usage for the cluster finder data flow implementation including all MAXELER framework overhead for different pipelining factors and the resulting maximum clock frequencies. Right: latency of the two kernels for the different pipelining factors.

The dataflow cluster finder was implemented with different pipelining factor settings between 0 and 1 in steps of 0.1. Figure 5.15 on the left shows the resulting resource usage and the maximum clock frequency. The resource numbers are extracted from a full MAXELER implementation containing one cluster finder instance. The numbers include the resource overhead from the dataflow framework and can be compared with the numbers of the MAX3 implementation with one cluster finder instance from Table 5.1. A pipelining factor of 0 results in the lowest resource usage of around 9700 flip-flops and 10300 look-up tables, around 3000 look-up tables fewer than the default configuration. With a total number of around 5000 look-up tables for only the cluster finder, a reduction by around 3000 of them makes a significant change in the overall resource usage. The number of flip-flops does not vary as strong as the number of look-up tables. A difference of around 1000 flip-flops is observed between the minimum and the maximum pipelining factors. The influence on the maximum achievable clock frequency is shown in yellow. All pipelining factor settings were synthesized with a target frequency of 200 MHz. The value shown here is the best value that was achieved by the place-and-route tool. The minimum pipelining factor of 0 reduces the maximum clock frequency to around 25 MHz. The maximum frequency rises quickly up to a pipelining factor of around 0.5. The first timing closure for 200 MHz was achieved with a pipelining factor of 0.8. The step from 0.9 to 1.0 raised the resource usage again considerably.

Figure 5.15 on the right shows the corresponding latency for the two dataflow kernels. The latency of the *DividerKernel* changed roughly linearly between five clock cycles with the minimum pipelining to 44 clock cycles for the maximum. The latency of the *DecoderProcessorKernel* is less dependent on the pipelining setting and is constant at 19 cycles between the factors 0.1 and 0.9. However, it doubles to 38 cycles with the maximum pipelining setting.

Comparing the pipelining factor dependency of the kernel latency numbers with the latency numbers of the VHDL implementation as shown in Section 5.2.4 suggests that the dataflow implementation with a pipelining factor of 0.8 is closest to the VHDL counterpart. With this setting, the latency numbers of both kernels are identical between the dataflow and the VHDL implementation. Table 5.5 shows a comparison of the FPGA resource usage in the same way as done in Section 5.2.4. This table compares the dataflow implementation with a pipelining factor of 0.8 with the VHDL counterpart. In this configuration, the resource usage of both implementations moves even closer. The dataflow variant uses slightly more look-up tables but

Pipelining factor 0.8	Dataflow implementation				VHDL implementation			
	LUTs	FFs	BRAMs	DSPs	LUTs	FFs	BRAMs	DSPs
DecoderProcessorKernel	1005	1782	4	8	828	1121	4	8
Channelmergersm	1659	955	8	0	1444	1439	8	0
DividerKernel	1929	2088	0	0	1620	1481	0	0
Total	4822	5059	19	8	4451	5169	19	8

Table 5.5: Comparison of the resource usage between the dataflow and the VHDL implementation for the individual modules and the full instance at the reduced pipeline depth. The pipelining factor of 0.8 leads to the same latency in both implementations.

a few flip-flops less. The overall difference is less than 10 % in both cases. The usage of block RAMs and DSPs is even identical now.

Varying the level of pipelining in the dataflow kernels allows users to influence the trade-off between clock frequency and resource usage, tuning the dataflow implementation to the particular needs. Choosing a pipelining factor that results in the same kernel latency values as the VHDL implementation makes both variants even more comparable. There are some differences in the resource usage and how things are implemented in both variants. However, the relative difference at identical latency numbers is below 10 % for each resource type. This confirms, on one hand, that the VHDL approach is not far from the pipelining tactics used in the dataflow implementation. On the other hand, it also means that using the high-level dataflow description brings approximately the same performance and resource usage as the low-level approach.

5.3.2 Varying the Fixed-Point Precision

The cluster finder uses a fixed-point number representation for any calculations up to the *DividerKernel*. The widths and the precision of the intermediate values are defined from the input vectors. The raw TPC data consists of 10 bit unsigned integer ADC values. The per-pad gain correction factor applied in the protocol decoding step is a fixed-point value with one integer and 12 fractional bits. The gain correction, therefore, covers the value range between 0 and $(2-2^{-12})$. Multiplications and additions increase the integer parts of the number representations in order to avoid overflows. The fractional precision of the gain correction factor is applied to all calculations that include charge values. The precision of the gain correction factor, therefore, directly affects the internal calculations.

The gain correction precision required for the experiment was defined from simulations and analyses before and throughout Run 1 to 12 fractional bits. This value was used during Run 1 and Run 2. However, despite being implemented with 12 fractional bits in hardware, a gain calibration value other than 1.0 was never used in the production systems throughout Run 1 and Run 2.

The VHDL implementation of the cluster finder tries to use a generic value at a central configuration package to keep the code as far as possible independent from the actual value. However, the multiplication steps, the format conversions, as well as FIFOs between the processing stages have to be manually regenerated as soon as the value changes. The IP cores cannot use the centrally defined value. Additionally, changing the precision may affect the latency of operations, requiring changes to the surrounding logic. For these reasons, it is not possible to define the fixed-point precision of this factor centrally at once place in the VHDL implementation. Changing it requires careful adjustments at several places throughout the code. In the same context, it is infeasible to evaluate the effect of different precision values to the overall resource usage in the VHDL implementation.

However, the dataflow approach hides all these tedious adjustments and IP core generation steps from the user. A truly generic definition of the fixed-point fractional precision is possible there

and is actually used. Changing the precision is possible by changing a single number in the source code. This enables to quickly evaluate the effect of using a different precision to the resource usage of the whole design.

```
1   public static int cFixFraction = 12;
2   // [...]
3   public static DFEStructType tClusterCandidate() {
4     StructFieldType Qmax = new StructFieldType("Qmax", fcfFix(cGxCint, cFixFraction));
5     StructFieldType Tmean = new StructFieldType("Tmean", dfeUInt(cSampleWidth));
6     StructFieldType Qtot = new StructFieldType("Qtot", fcfFix(cSeqCint, cFixFraction));
7     StructFieldType SeqT = new StructFieldType("SeqT", fcfFix(cSeqTint, cFixFraction));
8     StructFieldType SeqP = new StructFieldType("SeqP", fcfFix(cSeqPint, cFixFraction));
9     StructFieldType SeqT2 = new StructFieldType("SeqT2", fcfFix(cSeqT2int, cFixFraction));
10    StructFieldType SeqP2 = new StructFieldType("SeqP2", fcfFix(cSeqP2int, cFixFraction));
11    // [...]
12  }
```

Figure 5.16: The fixed-point precision is defined at one central place and is used for all intermediate values. Changing the precision is a single line code change.

Figure 5.16 shows how the fixed-point fractional precision is defined in the dataflow context and how this value is used in the interface type definitions. The compiler automatically resolves the resulting data types for the mathematical operations using any of these fixed-point input vectors. It creates all underlying IP cores accordingly.

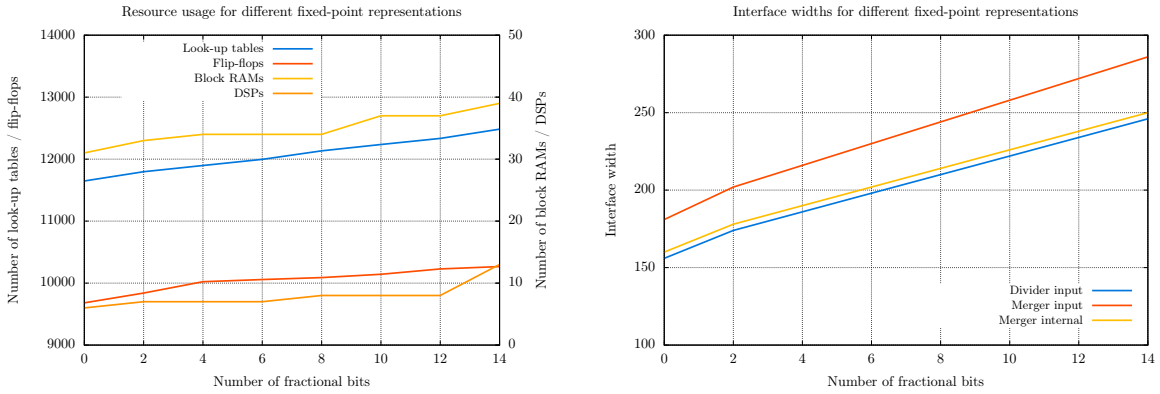

Figure 5.17: Left: Resource usage for different fixed-point precision values. Look-up tables and flip-flops show a linear dependency while DSPs and block RAMs increase step-wise according to their native internal widths. Right: Corresponding interface widths between the kernels.

Figure 5.17 shows the resulting resource usage numbers and interface widths for different fixed-point precision values. The numbers are again extracted from a fully implemented dataflow design with a single cluster finder instance including all framework logic. The design was built for all multiples of two fractional bits between 0 and 14 bit. The number of look-up tables and flip-flops scale roughly linearly for different precision values. The resource usage difference between the 0 and 12 bit precision adds up to around 680 look-up tables and 550 flip-flops, making up less than 10 % difference in relation to the resource usage of a single cluster finder instance. The numbers of block RAMs and DSPs increase whenever a multiple of their intrinsic width is exceeded. The number of DSPs jumps from seven to eight at the transition from six to eight fractional bits, because the first multiplication result exceeds the intrinsic DSP result width of 48 bit and, therefore, requires a second DSP block for the multiplication. The next jump in the number of DSPs is from 12 to 14 bit precision because two more multiplications exceed the 48 bit boundary. The block RAM usage is mainly dominated by the widths of the data vectors between the different processing stages. Figure 5.17 on the right shows the widths of the interfaces between the *DecoderProcessorKernel* and the *ChannelMergerSm*, between the *ChannelMergerSm* and the *DividerKernel*, as well as the internal connection from the *ChannelMergerSm* to the buffer FIFOs

in the same module. The bus widths also scale roughly linearly with the precision because all of them contain six fixed-point values with the given precision. The higher the precision, the wider the data words that have to be transferred between the arithmetic operations. Wide data words have a strong influence on the timing performance because both the combinatorial and the routing complexities increase. The lower the fixed-point precision, the smaller and faster the hardware implementation.

The dataflow framework provides the tools to easily and quickly evaluate the effect of different fixed-point precision values on the overall resource usage. Doing the same at the RT level comes with significantly higher efforts.

5.3.3 Moving Processing Steps into Hardware or Software

The hardware cluster finder implementation provides an output structure of 24 bytes per cluster as shown in Table 4.1. Measurements with detector data from various runs have shown that this setup reduces the size of each sub-event in average by around a factor of 1.5 compared to the original raw data size as discussed in Section 4.2.5. The maximum and the total charge of a cluster are provided as fixed-point numbers, whereas the position in the time and the pad direction, as well as the intermediate results for the widths in the time and the pad direction, are provided as floating-point numbers. The final stage of the hardware cluster finder consists of a fixed-to-float conversion step and a floating-point divider to calculate these values. A simplified schematic representation of the divider stage is shown in Figure 5.18. A single divider instance is shared for all four division operations.

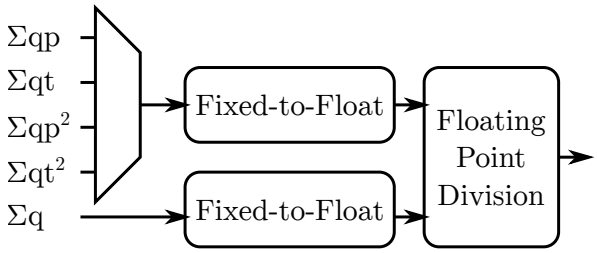

Figure 5.18: Sketch of the cluster finder divider implementation.

The first processing steps in software then complete the calculation of the cluster widths by subtracting the squared time and the squared pad position respectively as shown in Equations (4.4) and (4.5). Additionally, an offset of 0.5 is added to the pad position of each cluster to treat its position relative to the center of each pad. In a second step, the positions and widths in the time and the pad direction are multiplied with a scaling factor and converted back from floating-point to integer. This treats all values as an integer multiple of a minimal unit for each property. A third step reduces the bit width of each property to the expected value ranges. Values exceeding the target bit width are capped to the maximum value. These steps remove precision from the results that is not required for physics analysis and enable a significantly better compression of the clusters. Table 5.6 shows a comparison of the data types and the widths of the individual cluster properties after the cluster finder and then after the software-based format optimization. The overall size of a cluster is, therefore, reduced from 24 bytes to around 10 bytes, resulting in an additional data reduction of more than a factor of two without applying the actual HLT data compression algorithms. This compression step is also described in [Kol 12].

The decision on the implementation of the cluster finder as it is in use was mostly dominated by the resources available in the H-RORC FPGA during Run 1. With respect to the C-RORC and especially looking towards Run 3 where an improved hardware cluster finder compression saves critical IO bandwidth on the readout board, additional processing steps could be integrated into the hardware or moved back into the software. The dataflow framework provides an easy to use environment to try out several options.

Cluster property	Cluster finder output		Software transformation	
Row	6 bit	uint	6 bit	uint
Pad	32 bit	floating-point	14 bit	uint
Time	32 bit	floating-point	15 bit	uint
Pad width	32 bit	floating-point	8 bit	uint
Time width	32 bit	floating-point	8 bit	uint
Total charge	30 bit	fixed-point	16 bit	uint
Max. charge	23 bit	fixed-point	10 bit	unit
Flags	5 bit		n.a.	
Sum	24 byte		10 byte	

Table 5.6: Bit widths of the cluster properties after the cluster finder and after the software data format optimizations.

5.3.3.1 Moving Processing Steps into the Hardware

Figure 5.19 sketches the dataflow graph of the operations when implemented fully in hardware. The existing division operation is shown on the left followed by the additional operations necessary to realize the software cluster format optimization and the coordinate transformations already in hardware. A multiplier provides the squares of the time and pad values. These are then subtracted from the intermediate cluster width properties. Individual scaling factors are applied with another multiplication and a final step of rounding and resizing provides the appropriate bit width per cluster property.

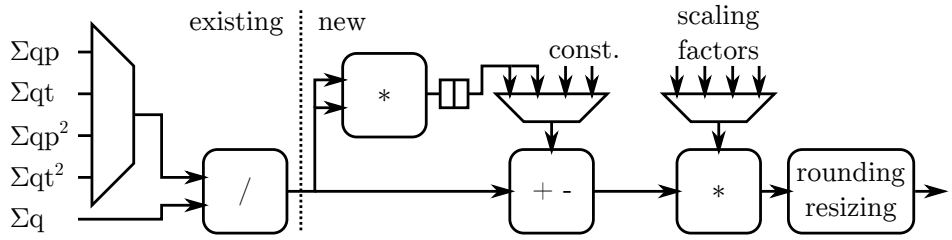

Figure 5.19: Extended cluster finder divider with additional data processing to reduce output data volume.

For the implementation in hardware, the existing divider can be extended. The new operations use the same 32 bit single-precision floating-point data representation for the multiplications and additions/subtractions. The final rounding and resizing step is applied after conversion to an integer data type. With the floating-point implementation, all intermediate results like the multiplication and the addition are floating-point values as well. This keeps the data path at 32 bit width.

With the given dataflow framework, the data type and the required precision can easily be specified, optionally for each processing step individually. All IP cores are generated accordingly in the background. This makes it very easy to simply try out different implementations.

Firmware variant	FFs	LUTs	DSPs	RAMs
Full firmware, 1 cluster finder instance, previous divider, transformation in software	10318	12410	8	37
Full firmware, 1 cluster finder instance, extended divider, transformation in hardware	11280	13253	16	39
Difference to the previous implementation	962	843	8	2

Table 5.7: FPGA Resource usage comparison of the previous divider and the extended divider.

Table 5.7 shows the resource usage of the extended divider including the data transformation steps in hardware in comparison to the resource usage of the existing divider instance applying the transformations in software. The numbers were extracted after the place-and-route step from a full dataflow firmware containing one cluster finder instance. The first line shows the total resource usage of a design with the previous divider, applying the transformation in software. The second line shows the numbers including the transformation in hardware. The bottom line highlights the difference in the number of resources between both variants. Moving the transformation step into the hardware needs around 960 additional flip-flops and around 840 additional look-up tables. This increases the resource usage of flip-flops and look-up tables of a single cluster finder instance by around 15–20 %. The number of DSPs is doubled from six to 16 due to the additional floating-point operations. The number of block RAMs increases because those are used instead of deep shift registers. Both the MAX3 board and the C-RORC provide sufficient FPGA resources to implement six instances of the cluster finder including the transformation in hardware.

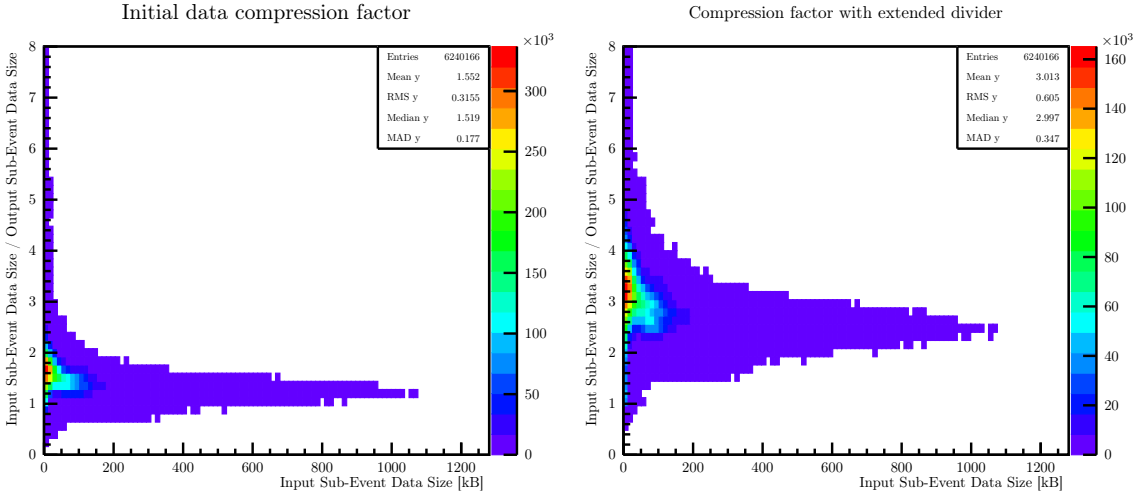

Figure 5.20: Data compression ratios with the previous and the extended divider implementation.

Table 5.6 already described the reduced cluster size by moving the transformation steps into hardware. The minimum size of a cluster is 10 bytes after the transformation. In order to ease data access and the transfer via PCI-Express, the size of transformed clusters transferred from the FPGA into the host machine is padded to 12 bytes, corresponding to three 32 bit data words. Compared with the previous 24 bytes per cluster, this reduces the size of a single cluster by a factor of two. Static header and trailer words at the beginning and the end of each sub-event reduce the gain in compression to slightly below two. Figure 5.20 confirms this with compression ratio measurements with around six million sub-events. While the mean compression ratio is at around 1.55 with the initial divider implementation on this dataset, moving the transformation steps into hardware increases the compression ratio to slightly above 3.

These evaluations show that with a moderate increase of resource usage in the order of 15–20 % for flip-flops and look-up tables, an improved compression by a factor two is possible. This effectively halves the PCI-Express bandwidth requirements. Despite being possible with the Run 2 hardware, the extended divider with the transformation steps in hardware was not implemented in the HLT production system. The reason is that it does not provide significant benefits that weight more than the efforts and adjustments to get the system production-ready. The transformation component in software is running as part of the online reconstruction framework on all HLT compute nodes. Neither the PCI-Express bandwidth of the C-RORC nor the CPU load of the transformation component oppose limits to the HLT processing capabilities in Run 2. Enabling the transformation in hardware requires a completely new cluster finder output data format in combination with software adjustments to keep the system compatible with the existing cluster

finder and operational also with previously recorded data. The transformation in hardware is, however, an important evaluation for ALICE in Run 3 where the PCI-Express bandwidth might get a crucial aspect. The dataflow framework provides the means to quickly evaluate the resource consumption of moving additional processing steps into the hardware.

5.3.3.2 Moving Processing Steps into the Software

Cluster property	Data type after Channel Merger
row	6 bit uint
q_{max}	fixed<11, 12>
$\sum q$	fixed<18, 12>
$\sum qp$	fixed<26, 12>
$\sum qt$	fixed<28, 12>
$\sum qp^2$	fixed<34, 12>
$\sum qt^2$	fixed<38, 12>
flags	5 bit uint
Sum	238 bit / 30 bytes

Table 5.8: Bit widths of the intermediate cluster properties transferred from the Channel Merger to the Divider.

If the FPGA resource usage has to be reduced and the PCI-Express bandwidth is not an issue, there is also the option to move processing steps from the hardware to the software. The full divider functionality could be implemented in software with little effort by sending out the cluster information after the merger stage. Table 5.8 shows the bit widths of the signals between the merger and the divider component. The total number of bits required for these signal, including the full fixed-point precision, is 238 bit. Transferring clusters of this size via PCI-Express would likely be done in multiples of 32 bytes or eight 32 bit words. This would reduce the mean compression from around 1.5 to 1.1 and would require to do any type conversions and divisions in software. It would be the cheapest option in terms of resource usage but also comes with a reduced compression and, therefore, an increased bandwidth requirement towards the host. The increased CPU requirements also had to be checked carefully. Another option is to do only the conversion from fixed-point to floating-point in hardware but move the actual division step to the software. This way, the size of a cluster would remain at 24 bytes as with the divider before, but the method resulted in a reduced FPGA resource consumption.

Firmware variant	FFs	LUTs	DSPs	RAMs
Full firmware, 1 cluster finder instance, previous divider, transformation in software	10318	12410	8	37
Full firmware, 1 cluster finder instance, conversion only, division in software	9319	11473	8	37
Difference to the previous implementation	-999	-937	0	0

Table 5.9: FPGA resource usage comparison of the previous divider and an implementation without division in hardware.

Table 5.9 shows the resource usage of that variant in comparison with the previous implementation. Doing only the data type conversion for the weighted sums in hardware saves around a thousand look-up tables and flip-flops, making up approximately a 18–20 % reduction in the total amount of resources required for a single cluster finder instance. The number of DSPs and block RAMs is not affected by these changes.

This option could be a viable solution for Run 3 if the PCI-Express bandwidth can handle 24 bytes per cluster but the FPGA resources are limited. This would, however, come with increased CPU power requirements for the additional calculations previously done in hardware.

5.3.3.3 Replacing Floating-Point Operations in Hardware

All mathematical operations in the cluster finder up to the divider stage are done with integer or fixed-point data types. The division and the first steps of the transformation are calculated with floating-point data types. The resulting cluster properties after the transformation are again integer data types. This fact raises the question if the transition to floating-point for the intermediate steps is really necessary or if all calculations could be done with integer and fixed-point data types, possibly saving FPGA resources. A fixed-point implementation could be possible in three variants. The first option with no division in hardware was already shown in Section 5.3.3.2. This variant would increase the current size of the clusters if the 12 bit fractional precision is kept. A second option would be to implement only the division step in fixed-point data types and leave the transformation steps in the software. A third option includes the division and the transformation in hardware. The transition from the floating-point to the fixed-point arithmetic typically analyses the value ranges of the operands to find appropriate fixed-point representations that provide the same relative error as the floating-point counterparts. One aspect that makes this transition more complicated is the fact that the same divider instance is used for four different divisions. A common representation had to be found that matches the conditions for all four divisions. The analysis of fixed-point data types for the division and the transformation and its effect on the resource usage was not done as part of this work but is left for future investigations. The dataflow framework provides the tools to do evaluations like these with comparably little efforts.

Chapter 6

Results, Conclusions, and Outlook

6.1 Results

The goals of this work were initially defined in Section 1.5 as the following:

- Development of a hardware platform to be used as the RORC for ALICE in Run 2. The hardware has to support the Run 2 upgrades of the detectors and processing algorithms while keeping compatibility with existing and reused infrastructure, hardware, and software from Run 1.
- Firmware and software developments to integrate the RORC into the HLT as well as its operation throughout Run 2. This includes the integration, maintenance, and extension of the hardware cluster finder.
- Evaluation of the hardware description at an algorithmic level using the dataflow approach. The hardware cluster finder serves as a reference for a high energy physics readout preprocessing algorithm with detailed performance numbers available from low-level implementations.

All goals initially defined for this work were successfully achieved. The Common Read-Out Receiver Card (C-RORC) was developed as a custom hardware platform for ALICE in Run 2 and finally found its place in the production systems of two major LHC experiments. It is, in addition, used in various other development systems and is partly foreseen for use in Run 3. The C-RORCs are currently a central part of the ALICE online architecture. All detector data from the TPC and the TRD, as well as all data to and from the HLT, are handled and processed by these devices. The boards were integrated into the Run 2 HLT system as a combination of hardware, firmware, and software developments in the course of this work. The hardware cluster finder was adapted to the Run 2 detector and data-taking conditions and continuously improved, contributing a considerable proportion to the overall HLT processing power. A high-level hardware description of the cluster finder could be realized with a dataflow framework, reaching a performance and a resource usage comparable to the low-level approach, yet with significantly reduced code volume and complexity. The integration of a dataflow-described firmware block into the HLT firmware environment could be shown as a proof of concept.

6.1.1 Hardware Platform

The C-RORC hardware platform, which was developed and maintained in the course of this work, has been used in ALICE production systems throughout Run 2. While the hardware was originally targeted towards a common use in the ALICE Run 2 DAQ and HLT systems, it additionally found its place in the Run 2 readout system of the ATLAS experiment. The

board can handle the increased data rates and processing requirements for the upgraded ALICE detectors in Run 2 and is compatible with the hardware and software infrastructure reused or extended from Run 1. The hardware-based data pre-processing in the C-RORCs is an essential part of the event reconstruction and compression scheme in the HLT. The C-RORC meets all performance and reliability requirements and was available for the installation into the ALICE and ATLAS production systems in time. The boards have been used to read out and process detector data in almost any ALICE physics data-taking run since the start of Run 2 in 2015.

The H-RORCs and parts of the D-RORC boards from Run 1 could not be reused in Run 2 due to obsolete interface standards towards the host machine and the lack of support for higher link rates or extended data processing capabilities. At the time the hardware decisions for Run 2 had to be made, there was no commercial hardware available that could fulfill the ALICE requirements. Changes to the readout architecture enabling the use of commercial readout hardware were not possible. Even later commercial boards were still not even close to the needs of the experiment. Therefore, the development of a custom FPGA-based readout board, the C-RORC, was required. The hardware requirements and the detailed reasons for this custom development are described in Section 2.1. The board contains a Xilinx Virtex-6 FPGA with an eight-lane PCI-Express generation-2 interface to the host machine and 12 serial optical links. With the increased link density and bandwidth to the host machine in combination with sufficient FPGA resources, one C-RORC replaces up to six of the previous H-RORC/D-RORC boards. Benchmarks of the PCI-Express interface described in Section 2.2.2 have shown that the hardware can sustain transfer rates of around 3.6 GB/s of user data. This is close to the physical limit of the link and more than sufficient for the needs of the experiment in Run 2. The 12 serial optical links are implemented with parallel optical solutions using three Quad-SFP transceiver modules to fit the optical connections into the IO-bracket of the PCI-Express card. The serial links can be operated at any rate between a few hundred Mbps and around 6.6 Gbps, covering the range of all ALICE use cases for Run 2. Two pluggable DDR3 memory modules can additionally provide large on-board storage if needed. Both modules can be operated independently at 1066 Mbps, providing a measured sequential read data rate of up to 8.3 GB/s per module. Synchronous flash memory chips in combination with a configuration controller using the PCI-Express SMBus sideband signals form a fast, flexible and reliable FPGA configuration solution, enabling a fail-safe firmware upgrade procedure. The board is kept as generic as possible while fulfilling all ALICE requirements to enable the application of the hardware for other additional purposes.

The conceptual and the schematic design of the board was carried out as part of this work. The PCB layout and the production of prototypes was provided by a company. The first boards produced worked as expected and only minor modifications were made from the prototypes to the final hardware revision. The ATLAS experiment joined the project for the upgrade of their readout system for Run 2 after the first C-RORC prototypes were produced and successfully tested. Both experiments benefited substantially from the collaboration on both the technical aspects and developer exchange, as well as the reduced hardware price per board due to the larger overall production volume. Preparation and coordination of the large-scale production were achieved with the significant help of colleagues from ALICE and ATLAS, as well as from the CERN procurement service. A detailed power-on and test plan to be conducted on a provided test bench of hardware, firmware, and software to confirm the functionality of each of the produced boards already on the contractor site was prepared as part of this work. This procedure ensured that each board worked as intended before it was shipped to CERN. 378 of the 380 boards from the main production batch passed the test immediately, two boards initially failed the test but passed after replacing the FPGA. Additional hardware tests at CERN did not reveal any hardware problems. The full set of boards was available at CERN from mid-2014, shortly before the HLT server nodes were delivered. The C-RORCs were installed into the ALICE and ATLAS systems in autumn 2014. After around four years of operation in the production systems, there have been no known C-RORC hardware failures. The time it has taken from the schematic design until the final hardware was produced was considerably long, approximately three years. However, none of the delays were due to technical difficulties, but mostly due to contractual issues with external partners.

Besides its main use case in the computer clusters of the ALICE DAQ and HLT systems and its secondary application in the ATLAS experiment, the hardware has also found its place in various test and development systems. ALICE foresees to use the C-RORCs during Run 3 for detectors that adhere to the current DDL protocol. ATLAS is even planning to increase the number of C-RORCs for Run 3. Other experiments plan to make use of the spare ALICE C-RORCs after Run 2 for their readout systems. Additionally, the board is used as a development platform with different firmware variants for the readout of the next generation readout link in ALICE, the GBT, until the final ALICE Run 3 readout board is widely available. These applications are described in Section 2.4. Both firmware developments and hardware concepts have also been transferred to other projects and experiments.

6.1.2 Firmware and Software Developments for the C-RORC in the HLT

A total of 74 C-RORCs were installed into the production system of the ALICE HLT as its central experimental data input and output interface. These are connected to more than 500 detector links. The C-RORCs handle two different dataflow directions with three different firmware variants. The detector data input to the HLT is realized by two different firmware variants providing either raw data readout from the optical links into the RAM of the host machine or data readout in combination with online data pre-processing already in the FPGA for the TPC links. The output interface, to provide the HLT decisions and the compressed event data to the DAQ system, utilizes the C-RORCs in the opposite dataflow direction. The C-RORCs were installed into the HLT cluster in autumn 2014 and were commissioned together with the full HLT system until the start of Run 2 in mid-2015. Reliable and fully functional firmware variants for the operation of all three HLT use cases were timely available and integrated into the HLT for the first ALICE data-taking operations. All HLT C-RORC firmware variants, as well as the integration into the HLT data transport framework and parts of the device driver library, were developed during this work. The firmware and software were maintained and continuously improved throughout Run 2 by implementing new features, adjusting to changing experimental conditions, and improving error detection mechanisms and automation. The resource usage estimations made during the hardware design phase proved to match the final requirements of the HLT application. All HLT firmware variants were compatible with the available resources and have reasonable expansion space for future extensions or improvements.

The C-RORC connects to the existing ALICE fiber installation reused from Run 1 via breakout fibers. The selected optical technology of fibers and transceiver modules, as well as the firmware implementation of the optical link, proved to be reliable and stable throughout the Run 2 operation. The reconfiguration of the link speed at runtime is realized with the cluster provisioning framework and a service on the host machine. All machines were reliably configured to the correct link speed at boot time throughout Run 2 and there were hardly any link level errors observed.

The firmware description was organized in a strictly modular way with well-defined interfaces to enable reuse of building blocks present in more than one firmware variant. All firmware variants were maintained and version tracked in the same source code repository. This code base was forked multiple times for other applications. Compile time switches and parameters enabled any variant to be built from the source files with a single command. Firmware builds were made with an automated system, ensuring builds from clean repository images without any local modifications, therefore, guaranteeing reproducible results for each build.

The PCI-Express interface to the host machine was implemented using a custom DMA engine for 12 independent DMA channels in firmware. The number of channels required was determined from the maximum number of detector links connected to a C-RORC in the HLT. The custom implementation of the DMA engine was necessary because none of the commercial cores available could provide the required amount of independent channels. This custom development thus provides the maximum degree of flexibility and the highest throughput for the HLT applications. It supports both vectored IO and IOMMUs which makes the firmware implementation completely

independent from the DMA buffer allocation scheme on the host machine. All 12 DMA channels can be operated independently and in parallel to slow-control access to FPGA internal configuration and status registers. Although the DMA firmware approach reuses the concepts from the Run 1 HLT, the implementation had to be made from scratch due to the new interface and the generic buffer allocation support. Each DMA channel interacts with two DMA buffers in the host RAM, an *EventBuffer* for the detector data and a *ReportBuffer* for descriptors and metadata. Both buffers are used as ring buffers which reduces the memory management logic required to only compare a read and a write pointer. The concept is described in detail in Section 3.5.1. The measured PCI-Express payload throughput was close to the physical limit of the link and proved to be sufficient for all HLT data-taking conditions in Run 2. The implemented PCI-Express DMA data transfer scheme from the host to the C-RORC was capable of exceeding the optical link bandwidth in the HLT configuration and so did not impose bandwidth limits during the operation. The DMA transfer measurements are described in detail in Sections 3.4.1 and 3.4.2. While earlier host architectures showed an impact of the DMA transfer rate from non-optimal buffer placement, the machines used in the HLT were found to be able to handle the maximum C-RORC throughput without load or buffer placement dependencies.

The previous kernel module and device driver from the Run 1 HLT could not be reused in Run 2 as the support for this kind of memory allocation scheme was dropped from the mainline kernel. The new kernel module in combination with a userspace device driver provides different DMA memory allocation options and support for vectored-IO as well as IOMMUs. As a result, all device access can now be controlled from userspace. A unique feature of the new memory allocation scheme is the option to map the same DMA kernel buffer twice and consecutively into userspace. This enables to hide fully the ring buffer concept from the data processing framework and handle the buffer wrap-around transparently without additional logic. By pre-loading the DMA memory location descriptor table into each DMA engine in the C-RORC before starting the data transfer, the maximum PCI-Express bandwidth is made available for the detector payload. Only a minimum of bandwidth is required for flow-control tasks with this approach. Descriptor storage space in the firmware was not an issue in the HLT application and the amount of storage could be increased at any time if needed. The userspace device driver, *librorc*, was not available in Run 1. Both the data transport framework and any standalone tools were interfacing the kernel module directly. With the new *librorc* driver layer, the same software stack can be used for the integration into the HLT data transport framework as well as for standalone tools. This simplifies the development, provides better testing capabilities and avoids repeating code across tools and files. The *librorc* proved to be a reliable choice for rapid development of C-RORC interface tools, it is working stably and is used for a number of other applications based on the HLT C-RORC DMA firmware implementation. The same library is also used in the HLT data transport framework. The C-RORC is integrated as a data source component to receive detector data from the optical links as well as a data sink component to send the HLT decision and compressed detector data to the DAQ system. Using the *librorc* library reduced the code size of the previous source and sink component implementations considerably. It was, furthermore, possible to use the same device handler code for both the source and the sink components, combining previously independent and highly redundant software parts into a common code base.

The new firmware-based link status monitor proved to be highly valuable to detect inconsistent device states or faulty detector links immediately while running and before detector data consistency was affected. The conceptual changes to the handling of corrupt detector data in the HLT source components improved the data processing in the merger components and reduced the chances for delayed processing due to timeouts waiting for missing fragments. This improved the resilience of the system on corrupt input data. The C-RORC readout software, as well as the integration into the HLT data transport framework, worked reliably from the beginning. A few changes to the balancing of detector links across the boards were made with the help of the HLT colleagues in order to optimize the overall throughput of the HLT chain. An important result here is that using less than 12 links for some boards provided better results for the HLT system as a whole, even though the individual boards and the readout applications can handle the operation of 12 links at the corresponding link bandwidth. The C-RORC integration into the HLT data

transport framework was stable since the beginning of the Run 2 data-taking operation. This integration was improved and extended continuously throughout Run 2.

In parallel, a hardware-software co-simulation environment was developed to efficiently debug and analyze the combination of PCI-Express DMA firmware and low-level device access software. This environment addresses the problem of hardware simulation mismatches with respect to the actual use case. By connecting the same software application with a simulation model of the hardware instead of the real board, the same sequence of operations that would go to the actual hardware can now be monitored and simulated. The development of this framework proved to be highly valuable to catch implementation bugs on both the software and the firmware side. Despite some effort to set this up, this investment was well worthwhile with the number of issues that were caught and fixed early during the firmware and software development.

The DDR3 memory interface of the C-RORC is actively used in the HLT to replay detector data as if they were coming via the optical links. One or two 2 GB DDR3 modules were installed per HLT C-RORC, depending on the number of detector links per boards. One module provides data replay for up to six DDLs. A DDR3 controller configuration was found that provided sufficient bandwidth for delivering data to emulate six fully saturated DDLs while keeping the FPGA resource usage and clock frequency requirements at a minimum. The data replay functionality was also integrated into the HLT data transport framework. By replaying generated, previously recorded, or faulty detector data, the readout process could be tested for performance and resilience. Furthermore, by injecting detector data into the full HLT system with a configurable data volume, data rate, and event rate, the functionality of the HLT could be confirmed and HLT configurations could be evaluated for different data-taking scenarios. The data replay operation of the HLT is possible, independent of the state of the LHC, the detectors or the DAQ system. The cluster-wide data replay functionality of the C-RORCs was frequently used when commissioning the RCU2 readout at the increased link rates and before demanding data-taking operations to verify the suitability of an HLT configuration. It is still applied after each maintenance period, hardware intervention, or change of the HLT cluster operation mode before the cluster is re-enabled for data-taking operations.

The PCI-Express bus between the host machine and the C-RORC does not guarantee a certain bandwidth or latency. All transactions are packet-based with transfer negotiations using a credit-based token system. A lack of tokens may temporarily stall the transfer. The maximum consecutive PCI-Express dead times are monitored in the firmware and can be analyzed during or after the operation in the production system. This monitoring made it possible to choose a trade-off for the amount of FIFO resources required to achieve a sufficient buffering with a minimal resource usage. The maximum consecutive PCI-Express dead times were measured to be around 50 µs for the C-RORC in the HLT server machines as shown in Section 3.9. This value could be compensated with appropriate buffer FIFOs in the firmware. With this system in place, readout interruptions due to PCI-Express dead time were not observed in the HLT during Run 2. If necessary, there are sufficient FPGA resources available to increase the buffering capabilities of the firmware in the future.

The C-RORCs were integrated into the HLT cluster management and monitoring system. Three different firmware variants were deployed to a total of 74 boards in the cluster. These are used with different link speeds and varying configurations, depending on data-taking conditions. An automated system ensures the correct operation of the hardware and the firmware at various levels. Key hardware metrics, such as the FPGA die temperature and PCI-Express link parameters, are monitored and a temperature regulation via an automated control of the fan speed is integrated into each firmware variant. All boards in the cluster remained within reasonable temperature ranges for all firmware variants. An excessive FPGA temperature was not observed during operation in the HLT machines throughout Run 2. In addition, the readout processes monitor their dataflow in the firmware from the corresponding optical link to the PCI-Express interface and report unexpected states or conditions immediately into the HLT data transport framework logging system. A firmware upgrade procedure is integrated into the cluster manage-

ment tools, making an automated cluster-wide upgrade as easy as changing the target firmware revision in a central configuration file.

6.1.3 Integration, Maintenance and Extension of the Hardware Cluster Finder

The hardware cluster finder is a central part of the HLT data compression and reconstruction scheme. Considerable amounts of CPU computing resources are saved by processing the raw TPC data already inside the C-RORC FPGA. The hardware cluster finder searches for characteristic signatures in the space charge distribution as read out from the TPC. It replaces the raw data with a list of cluster properties. In combination with software-based data transformation and compression techniques, the hardware cluster finding is part of the overall HLT data compression scheme. This data compression is required in order to be able to store experimental data for Pb–Pb collisions where the raw data rate exceeds the bandwidth to the permanent storage systems. Additionally, it enables to use the experiment granted storage quota efficiently. The higher the HLT compression, the more experimental data can be recorded, and the higher the statistical significance of physical analysis results.

At the start of Run 2 in 2015, the same front-end electronics and readout link speeds as in Run 1 were used for the TPC readout. For this reason, the existing cluster finder implementation from the H-RORC in Run 1 could be ported to the C-RORC for Run 2 without algorithmic changes. Six hardware cluster finder instances per C-RORC process the data from all 216 TPC DDLs with 36 boards. This increased the link density for the TPC readout in the HLT by a factor of three compared to the Run 1 setup. The integration of the cluster finder core into the C-RORC firmware required a few implementation changes from the previous hardware to the new hardware. The increased parallelism could be achieved by instantiating the same core six times. This provided a ready to use hardware pre-processing core with a proven algorithm for the TPC readout with the C-RORC in the HLT for the first year of ALICE data-taking in Run 2. Technological advancements and architectural changes from the FPGA on the H-RORC to that on the C-RORC slightly reduced the resource usage of the core. This variant of the cluster finder was available, integrated and fully functional in the HLT production system in time for the commissioning phase before the start of Run 2. The operation of this variant in the HLT production system throughout 2015 provided additional time to prepare the hardware cluster finder for the next generation of TPC readout hardware as well as improvements to the algorithm.

The correct operation of the cluster finder implementation is crucial for the ALICE data-taking operation since the raw detector data is replaced with the compressed cluster finder results. Data processing errors in the cluster finder cannot be recovered and can render the recorded data useless. In order to ensure the correct operation across firmware changes, multiple steps of hardware verification were added to the project. Unit tests check the individual firmware modules for correctness on common tasks using an HDL simulation environment. A full cluster finder hardware simulation verifies the chain of processing modules for individual sub-events. A main drawback of the hardware simulation is its processing speed which does not scale to reasonably-sized verification datasets. Formal verification methods are not possible for the full cluster finder implementation. Running the final implementation in hardware is the fastest way to process large verification datasets but this lacks insight in the case of discrepancies to the reference model. For this reason, a combined approach was developed. Large datasets were processed in the hardware and compared with the output of the reference model. A modular firmware approach made it possible to connect existing parts of the already available firmware variants to form a co-processor implementation. Raw detector data is read from the host RAM into the board, processed with a cluster finder instance, and the results are written back to the host RAM. A single C-RORC with the co-processor firmware containing six cluster finder instances can process any TPC event. This enabled to extend the hardware verification to the order of hundreds of Gigabytes of raw data to be compared with an approved software reference implementation. This co-processor firmware

variant was used for the verification of all cluster finder developments, algorithmic changes and performance measurements in this work. It provided a fast and reliable data processing and verification platform with a data processing volume and throughput that is orders of magnitude greater than the previous low-level verification approaches.

The TPC front-end electronics were upgraded to the Readout Control Unit 2 (RCU2) in early 2016 in order to increase the readout bandwidth of the detector. The new hardware provided changes in the data order and an increased link rate, roughly doubling the readout bandwidth compared to the previous setup. Architectural changes in the readout scheme required major firmware reworks on the cluster finder. The doubling of the readout bandwidth from the RCU2 required rewrites of individual processing steps and additional register stages at various places in order to meet the timing requirements of the target clock frequency. The clock frequency had to be scaled over-proportionally compared to the changes in the link speed, due to a more efficient usage of the available link bandwidth in the TPC electronics. The cluster finder in the H-RORC during Run 1 could be operated with 3.3 times the native link clock frequency without stalling the readout. Evaluations with recorded detector data have shown that this was no longer sufficient for the RCU2. A cluster finder clock frequency of four times the native link clock frequency was necessary in order to guarantee the processing of the TPC data from the RCU2 without risking stalling of the readout.

A limitation, at the time when the RCU2 was deployed, was the lack of testing data recorded with the new protocol. For this reason, a firmware that supported both the old RCU1 data format for data replay purposes and the RCU2 format for data-taking was the preferred solution. The biggest challenge of this approach was to meet the timing requirements. A cluster finder clock frequency with a factor of four greater than the native link clock frequency (78.125 MHz) resulted in 312.5 MHz. In relation to the FPGA capabilities, this imposed strict limitations on the number of logical operations per clock cycle and routing delays between synchronous components. Several parts of the cluster finder were adjusted and partly rewritten to meet these requirements. A common firmware that supported both protocol versions at the increased link rate could be realized and was deployed in the HLT for both the data replay operation and the data-taking in the first quarter of 2016. The cluster finder algorithm itself was not changed for this firmware variant.

Besides the crucial tweaks and adjustments to support the RCU2, a couple of changes were made to the hardware cluster finder throughout Run 2 to improve the data handling and the processing algorithm. These changes contained a number of protocol error detection, error handling and error reporting mechanisms. Protocol errors that led to parts of the data being ignored went unnoticed during Run 1. By detecting, counting, and reporting these to the HLT logging system, excessive error counts could be relayed to the detector team while running. Several TPC channels were observed to contain high error rates with this monitoring in place and the channels could quickly be masked by the detector team. The previous hardware implementation and the software emulator showed inconsistent results for corrupt input data. Therefore, the error handling in the hardware implementation was adjusted to make it identical to that of the emulation software. This ensured consistent results, even for faulty data, and simplified the verification of the processing results to an unconditional binary equality check.

The firmware variant supporting both the old and the new RCU protocol and link speed, as well as the improved error handling, was deployed in the HLT before the start of the data-taking operations with the RCU2 in mid-2016. It provided a stable readout of the RCU2 for the data-taking operations while enabling the replay of RCU1 data for HLT testing and development purposes. The cluster finder, on average, slightly reduces the data volume by replacing the raw data with the cluster properties. This compression depends on the number of clusters contained per raw data unit and varies across sub-events. The cluster finder in Run 1 provided a mean data reduction factor of around 1.2. This mean value increased to around 1.4 for the same algorithm in the first half of Run 2. This increase was not directly related to HLT firmware improvements but resulted from detector changes. Nevertheless, the higher mean cluster finder compression also increased the overall HLT compression ratio in Run 2.

A major change in the hardware cluster finder algorithm and its implementation is the improved noise rejection in the peak finding step and the tagging of clusters that are split in the time or the pad direction. These changes are described in detail in Section 4.3. In collaboration with the HLT colleagues and the TPC Offline group, an improved peak finding algorithm was determined that rejected large portions of noise-induced clusters without affecting the detection of valid clusters. A big challenge in the implementation of the new algorithm in the hardware was the significantly higher combinatorial complexity. The new algorithm takes a window of up to five charge samples in combination with local minima and maxima into account, compared to only two samples in the previous implementation. Additionally, the implementation is configurable with several parameters at runtime and still supports the previous approach for backward compatibility. The cluster finder clock frequency remained at four times the native link clock frequency which required to push the limits further in order to meet the timing requirements. The peak finding algorithm has a strong impact on the number of clusters found in the data. Using the new algorithm reduced the number of clusters by around 21% for detector data from 2017 and by around 32% on data from 2016. Detailed studies on the number of clusters and candidates are shown in Section 4.3.2. The mean data compression ratio of the cluster finder could be increased from 1.4 to around 1.8 with these changes. In combination with software-based optimizations of the cluster property representation and tweaks to the compression algorithm by HLT colleagues, these efforts could improve the overall HLT compression from around a factor of five to a factor of above eight. This new hardware cluster finder, with the improved peak finding and the tagging of split clusters, was deployed in the HLT in August 2017. The final version of the new hardware cluster finder was analyzed by the TPC Offline group with dedicated physics runs where both the raw data and the HLT results were stored. These studies confirmed equivalent and partly even improved physics performance results compared to the previous hardware cluster finder.

The hardware implementation of the cluster finder is designed in such a way that it can handle the maximum data rate provided by the optical link. This also means that its throughput is matched to the link bandwidth and not necessarily pushed to the limit of the FPGA. Therefore, measuring the equivalent computing performance of this approach needs a reference. The HLT cluster finder emulator software is a bit-wise identical model of the hardware. It is, however, not designed for efficiency in terms of CPU resources and processing time for the same reasons. In terms of physics performance, the HLT output provides equivalent results as processing the raw detector data in the Offline cluster finder software implementation. However, the Offline implementation uses a different algorithm and performs more calculations than those done in the hardware processing steps in the HLT. Additionally, each hardware cluster finder instance processes the data received on a single detector link, while the Offline software processes full events containing data from all links at once. The union of all 216 cluster finder instances, distributed across 36 C-RORCs and in combination with software-based data transformation steps on all HLT compute nodes, is, from the physics point of view, equivalent to what is done with the software cluster finder instances in the Offline framework. The performance measurement setup chosen in this work compares the cluster finding throughput at a per-node level. One C-RORC with six cluster finder instances can process any TPC sub-event and is, therefore, capable of processing full TPC events. Its throughput could be confirmed to be equivalent to the aggregated input bandwidth of the six optical links using the C-RORC on-board memory with data replay. The same detector data was processed with the Offline cluster finder on an HLT node with the maximum parallelism possible on this node. The HLT nodes and the C-RORC represent a comparable technology level, with the node being slightly newer than the FPGA. Newer FPGAs will be able to run higher link rates and clock frequencies, whilst newer CPUs have shown to provide a higher processing throughput in the software implementation. In this setup, with one HLT node containing one C-RORC with six cluster finder instances, the hardware implementation together with four CPU cores is around ten times faster than processing the same data with 48 parallel instances fully in software. Details on the performance measurements are shown in Section 4.4.2.

6.1.4 Evaluation of Dataflow Hardware Description

Another aspect of this work was to investigate higher-level hardware description methods to mitigate the limitations of the common FPGA hardware description approaches in terms of development time, complexity, flexibility, and maintenance effort. In this context, the dataflow approach was evaluated as a promising candidate. The hardware cluster finder made a perfect case study because it is a typical application in the readout of high energy physics experiments and a highly optimized RTL implementation is available for comparison. The dataflow approach is different from the most prominent high-level synthesis solutions because it opposes strict rules on how the algorithm can be described. By enforcing these rules, the compiler guarantees that the algorithm can be implemented as an efficient processing pipeline and does not have to fall back to iterative processing steps with state machines or von Neumann-like processing techniques. This study was carried out with a dataflow framework by Maxeler, including a hardware platform with an FPGA that is similar to that on the C-RORC. The similarities in the hardware platform enabled the direct comparison of both approaches. The dataflow hardware is described at an algorithmic level with a Java-like syntax and processed with a compiler that handles all steps to map this description onto an FPGA. The dataflow description is independent of the FPGA vendor on the underlying hardware. Selecting a target architecture and device is just a compiler option. This highlights a strong advantage of the dataflow approach with respect to code portability and is a major difference to RTL hardware description. However, the choice of targets is at the same time limited to a fixed set of supported devices. The dataflow developments in this work were started with a dataflow engine based on a Xilinx Virtex-5 FPGA. After the release of the dataflow engines with Xilinx Virtex-6 FPGAs, the dataflow API changed and required partial rewrites of the code. In addition, the Virtex-5 support was discontinued with the new API. The advantage of the dataflow description's target device independence is, therefore, clearly connected to the update and compatibility policy, as well as the choice of supported devices, given from the tool vendor.

Describing hardware with the dataflow language at an algorithmic level proved to be magnitudes faster than partitioning and implementing the same algorithm manually at the RT level, and, obviously, more compact. As a result, changes to an algorithm turned out to be much simpler than performing the same in the manual RTL approach. An implementation of the hardware cluster finder in the dataflow framework was possible with little effort compared to an RTL implementation. It was evaluated with different options in the context of an integration into the HLT. A co-processor approach, implementing only the hardware cluster finder in the full dataflow framework, would have been possible. However, this would have required dataflow hardware in addition to the C-RORCs for the optical link readout. This concept was used to evaluate the dataflow description, yet, its implementation in the HLT would have been more expensive and more complex without significant gains. A realization of the detector link readout solely with dataflow hardware, instead of the C-RORCs, could have been possible with direct optical connections to the dataflow engines. However, limitations of the dataflow hardware with respect to parallel access, PCI-Express bandwidth, and optical connectivity made this option infeasible. A third approach that could enable high-level hardware description, especially in the high energy physics community, would be to integrate a dataflow-described algorithm into a custom hardware, firmware, and software environment. Detector readout often requires special hardware with custom protocols and interfaces due to rigid or constrained environments or interfaces, as well as large-scale hardware synchronization. Commercial readout hardware is hardly feasible for these applications. HLS tool vendors, on the other hand, primarily support commercial hardware. Bringing a dataflow-described processing core into a custom hardware platform could close this gap. Evaluating this approach was possible with the help of Maxeler providing a dedicated build target for netlists that can be instantiated elsewhere.

The core implementation of the cluster finder in the dataflow framework is the same for all implementation options. Since the RTL implementation of the cluster finder is already a processing pipeline for most of its parts, the structure of the corresponding dataflow implementation remained very similar. A big advantage of the dataflow approach is that arithmetic operations,

such as multiplications and divisions, can be written in a single line of code for any data type. In combination with stream offsets, this greatly reduced the code volume and improved its readability, flexibility, and maintainability. Requirements that were given from the detector readout architecture, however, made several convenient dataflow framework features unavailable and led to a more complicated host software interface. A strong limitation in the context of the HLT application is that the dataflow host software can only communicate with the dataflow engine as a whole, while multiple processes in the HLT software operate all channels and cluster finder instances of the C-RORC independently. The same dataflow implementation of the hardware cluster finder could be compiled for two different dataflow engines with FPGAs from two different vendors. Two Maxeler devices were evaluated with respect to their bandwidth to the host machine. Both achieved only up to 65 % of the throughput measured with the C-RORC, despite having the same raw bandwidth to the host machine. This fact, in combination with the lack of optical connectivity, made both cards unsuitable as a direct C-RORC replacement.

Standalone kernels are a special feature of the Maxeler framework, where the compiler does not integrate the kernel into the dataflow framework, but creates a netlist that can also be used in a different scope. With a custom VHDL wrapper, it was possible to combine the chain of two standalone kernel netlists and a manager state machine from the dataflow framework to create a single and reusable netlist. This netlist could be instantiated in place of the VHDL cluster finder in the HLT C-RORC firmware. This firmware variant contains the full HLT firmware infrastructure with six optical links and the custom DMA engine, in combination with six instances of a cluster finder core generated from a dataflow description. The netlist could be integrated into the C-RORC firmware with the existing RTL build flow and enabled a detailed comparison of both variants. Both implementations have the same throughput by design. The functionality before and after the merging stage is implemented as a pipeline in both cases. The merging stage itself is identical even at the clock cycle level. The dataflow kernels, with their default settings, contain more register stages which lead to a higher latency for the dataflow variant. The resource usage of both variants is in the same order of magnitude. The dataflow implementation uses slightly more resources. However, this difference is marginal and can be explained by the higher level of pipelining. By controlling the level of pipelining via parameters to the dataflow description, an implementation variant could be found that brings an identical latency and a similar timing performance as the VHDL implementation. With these adjustments applied, the resource usage of both implementations is nearly identical. Detailed numbers are shown in Sections 5.2.4 and 5.3.1. This confirms that the generated code from the dataflow description provides the same performance as the handwritten code in this case. In terms of code complexity and volume, the dataflow approach significantly undercuts the manual VHDL approach. The dataflow description hides numerous details that have to be handled manually when using the VHDL approach. In addition, the higher-level description improves the maintainability of the code base. A comparison regarding the number of lines of code required for both approaches gave a reduction factor of above ten for both kernels in the dataflow approach. However, neither of the descriptions made explicit efforts towards compactness or a consistent level of comments across the descriptions. The code volume is clearly not an objective metric and is listed here to emphasize the benefits of the higher abstraction level.

This higher abstraction level of the dataflow description enabled quick evaluations of algorithmic changes to the hardware implementation. Two different changes were investigated: firstly, a changed fixed-point precision of the cluster finder properties, and, secondly, the option to move processing steps from software to hardware or from hardware to software with different trade-offs. Studies such as these are use cases where the dataflow framework can show its full potential. Whilst not necessarily required for the HLT in Run 2, these evaluations provide important design considerations for the next ALICE hardware cluster finder in Run 3. The efforts to conduct these studies in the dataflow framework were marginal compared to the work required to achieve the same results with an RTL description.

Unfortunately, the dataflow framework used in this work cannot be used for the ALICE hardware cluster finder in Run 3. The next ALICE readout hardware is being built based on an FPGA that is not present on any of the current vendor-supported dataflow engines. The dataflow framework

is, therefore, not available for this platform and it is not possible to create standalone dataflow kernels for it.

6.2 Conclusions and Outlook

The C-RORC hardware platform was very successful. The hardware became available in time, found its place in two major LHC experiments in Run 2, is used for present development systems and is partly foreseen for use in Run 3. The boards are running stable and are well integrated into the production systems since autumn 2014. Within the constraints of the experiment, all requirements of the hardware were met and it provides sufficient resources for upgrades or extensions. From a broader perspective, the C-RORC project highlights some important developments within the last few years that will have a significant impact on the next generations of physics experiment readout chains.

The complexity of custom FPGA boards has reached a point where it can exceed the capabilities of individual designers or small teams, especially in the academic environment. Continuously growing device sizes, pin counts, interface speeds, number of supply voltages, device documentation and user guide volumes make a custom board design a very tedious and time-consuming process. The increasing interface speeds require a careful PCB design with expensive tools for signal and power integrity simulations beyond the rule-of-thumb PCB design approaches from the past years. Using these tools requires experienced and trained designers. Debugging potential hardware problems may require expensive equipment for in-circuit measurements or protocol analysis. The time taken to develop a concept to a working custom FPGA board prototype, as an individual or a small team, can easily take years, whilst the next generation of FPGAs may become available before the present project is deployed. In the case of the C-RORC, the Xilinx 7-series FPGAs became available during the PCB layout phase. Using these FPGAs would have eased the firmware development efforts considerably. However, changing the FPGA at this point would have required to restart large portions of the board design from scratch. Another highly time-consuming process was the transition from a working prototype to a large-scale manufactured hardware platform. The time required for this step was much more than the sum of the quoted PCB production, assembly and component lead times. This was caused by coordination, technically complex negotiations, contractual difficulties, as well as inter-country purchasing and financing regulations and processes. The evaluation of readout solutions for ALICE in Run 2 was started in 2010, whilst the C-RORCs were installed into the production systems in 2014. There were no technical issues with the hardware.

Much work and many delays could have been avoided with commercial hardware if that had been possible. However, the need for custom readout hardware was caused by a general discrepancy between the requirements from the detector readout architecture and the features of commercial hardware. One reason is that the detector hardware often comes with custom interfaces and protocols which are simply not available on commercial boards. This is typically due to rigid environmental conditions, large-scale device synchronization or complexity reduction in constrained areas. In some cases this can be handled with adapters or custom add-on boards to the commercial hardware, making the situation at least technically solvable. Avoiding custom interfaces and protocols where possible is an important step towards enabling detector readout with commercial hardware. Another reason is that the custom detector hardware is typically a few technology generations behind the FPGA boards. This is simply due to the extensive planning and development times of the detectors. The detector technology freeze happens years before the technology decision of the readout hardware. Commercial FPGA boards that are available at the time the detector technology is decided upoan, are no longer available when they should get into operation. In the meantime, both the maximum optical link speed and the bandwidth for host interfaces are increasing for commercial hardware, whilst the capabilities of the host machines are typically even higher. An efficient detector readout solution would, therefore, need a state-of-the-art host interface with comparably slow links for the given FPGA technology with a high link multiplicity to make the best use of the host interface. However, in most cases the commercial

hardware matches the total maximum optical bandwidth with the host interface, leading to fewer but faster links. Assuming compatibility with all detector interfaces and protocols, a commercial readout solution would then require more boards with large portions of the host interface and the serial link bandwidth unused compared to a solution with custom readout boards.

If a custom hardware development is unavoidable, common efforts can significantly reduce the risk and cost factors. The ALICE RORCs in Run 1 were independent projects for the DAQ and HLT groups with respect to hardware, firmware, and software. Both groups had their own teams of hardware, firmware, and software developers. Additionally, hardware re-usage across LHC experiments was not a common practice. This was certainly intentional to some extent to eliminate single points of failure across experiments and groups. Nevertheless, this increases the costs of development, maintenance, and operation. With the C-RORC in Run 2, a common hardware platform was developed across the ALICE groups that was even adopted by the ATLAS experiment. Developers from all target applications worked together on technical and administrative tasks to realize the hardware design and to test the boards more efficiently than what would be achievable individually. Sharing available knowledge and manpower is a crucial step if custom hardware is necessary. The C-RORC project greatly benefited from the collaboration of the ALICE and ATLAS groups and, consequently, the next generation of ALICE readout hardware has been initiated as a combined effort between ALICE and LHCb.

The firmware and software development was a process that never ceased throughout the project phase. Proof-of-concept and test implementations dominated the technology evaluation phase of the project, while hardware tests and getting things working made up large parts of the work around the first prototypes. The development of a test stand for mass hardware verification performed by non-expert users was a big challenge during the hardware production phase. In parallel, the production-ready firmware and software for the integration of the C-RORC into the HLT application, including the first iteration of the hardware cluster finder, had to be developed. While the physics program of Run 2 was ongoing, the firmware and software were prepared for the upgrade of the TPC readout electronics to the RCU2. In parallel, general firmware and software improvements, as well as adjustments, had to be made to maintain the HLT installation during the different data-taking periods. Following the RCU2 upgrade, the cluster finder algorithm was extended to improve the noise resilience and the overall data compression ratio. Whilst highly beneficial for the Run 2 data-taking, these works were partly already evaluations for Run 3 where a significantly higher compression ratio will be needed. Several ideas exist to improve further the hardware cluster finding.

FPGAs play a crucial role for high energy physics detector readout applications and have their place between ASICs, DSPs, GPUs, and CPUs. The hardware cluster finder is a typical example of an FPGA-based data pre-processing algorithm, saving considerable amounts of CPU processing power in the HLT. A similar architecture with a hardware-based cluster finder is also foreseen for Run 3. FPGA device sizes have been growing almost exponentially in the last few years, enabling complex and highly parallel data processing on FPGAs. However, the firmware development still mostly takes place at the RT level and describing algorithms at this level is a tedious and error-prone task. The firmware development, and especially the verification, took extensive amounts of time in this work. With the changing readout architecture of the TPC in Run 3, hardly any piece of the firmware code will be reusable for the next cluster finder, requiring many developments from scratch. High-level synthesis is a concept that could potentially ease the hardware development with a description at an algorithmic level. The great number of tools available is both a blessing and a curse because there are various options to choose from. Yet, up to now, no real standard has evolved that is foreseeably compatible with the current and upcoming device and tool generations for the next couple of years. Nevertheless, several promising candidates are available.

The dataflow approach was evaluated in this work, enabling highly efficient hardware implementations with a performance at the same level as the manual RTL approach. The description at the algorithmic level reduced both code volume and description complexity while improving the maintainability. Evaluating the effect of different design decisions on the hardware implementation was easily possible and provided reasonable resource usage and timing performance

estimations. Testing ideas in hardware was possible at a pace that was out of the scope of an RTL approach. It was also possible to extract a dataflow-described algorithm from the tool vendor framework and to integrate it into a custom hardware and firmware environment.

These evaluations, however, also revealed some shortcomings of the HLS approach. These were partly specific to the framework used here but can, to some extent, also apply to other HLS solutions. Device support strongly depends on the tool vendor. An HLS vendor deciding to stop support for a given device renders the HLS code useless. Porting RTL code to another device requires adjusting IP core implementations, whilst porting HLS code from one tool to another may require a rewrite. In additon, a target device which is not supported by a given HLS tool cannot be described with this language. An HLS solution supplied with a framework of firmware and software as an integrated solution requires explicit vendor support to extract HLS modules into a custom hardware or firmware environment. Even thouhg it was technically possible to integrate a dataflow module into the C-RORC, this was due to the fact that both a supported dataflow engine and the C-RORC shared a very similar FPGA. A dataflow cluster finder for the next generation of ALICE readout boards, for example, is not possible because the new FPGA type and vendor is currently not supported by this dataflow framework. Better support for custom hardware tends to be available from the FPGA vendor-provided HLS tools. However, code developed with these tools is then obviously also limited to FPGAs from that vendor only. Therefore, code developed with a Xilinx-provided HLS solution for the C-RORC would not have been reusable with the next generation of ALICE readout boards based on Intel/Altera FPGAs.

The experience with the dataflow implementation of the hardware cluster finder has shown that the firmware development and maintenance could be greatly simplified with the availability and adaption of suitable HLS tools. A central aspect for this field of applications is, however, the support for custom hardware as it was used in ALICE during Run 2, and will again be used during Run 3 and in various other physics experiments.

List of Figures

List of Tables

Glossary

ADC	Analogue to digital converter
ALICE	A Large Ion Collider Experiment
ALTRO	ALICE TPC Readout Chip
AMD	Advanced Micro Devices, chip vendor
API	Application programming interface
ARM	Advanced RISC Machine, processor family
ASIC	Application specific integrated circuit
ATLAS	A Toroidal LHC Apparatus, experiment at CERN
ATX	Advanced Technology eXtended, a mainboard specification
AXI	Advanced eXtensible Interface, on-chip interconnect bus
BAR	Base address register
BGA	Ball grid array
BPI	Byte peripheral interface
BSP	Board support package
BSV	Bluespec SystemVerilog
C-RORC	Common Read-Out Receiver Card
CBM	Compressed Baryon Matter, experiment at FAIR
CDH	Common data header
CERN	European Center for Nuclear Research, Geneva
CMC	Compact mezzanine connector
CMS	Compact Muon Solenoid, experiment at CERN
CN	Compute node, server type in the HLT
COTS	Commercial off-the-shelf
CPLD	Complex programmable logic device
CPU	Central processing unit
CRC	Cyclic redundancy check
CRU	Common Readout Unit
CTP	Central Trigger Processor
CXP	A network interconnect technology
D-RORC	DAQ Read-Out Receiver Card
DAQ	Data acquisition
DCS	Detector control system
DDG	DDL data generator
DDL	Detector data link
DDR	Double data rate, digital signal sampling method
DFE	Dataflow engine
DFG	Dataflow graph
DIMM	Dual in-line memory module, memory module form factor
DIU	DDL destination interface unit
DMA	Direct memory access
DRAM	Dynamic random access memory
DRP	Dynamic reconfiguration port
DSP	Digital signal processor
DW	Double word, 32 bit word
E2000	An industrial fiber connection type
ECS	Experiment control system

EEPROM Electrically erasable PROM

EMCAL Electromagnetic calorimeter

EMI Electromagnetic interference

EPN Event processing node

ESD Electrostatic discharge

FAIR Facility for Antiproton and Ion Research

FC Fibre Channel, network technology

FCF Fast cluster finder

FDR Forteen data rate, signal rate specifier

FEC TPC front-end card

FEE Front-end electronics

FELIX Front-End LInx eXchange, ATLAS readout board

FEP Front-end processor, server type in the HLT

FF Flip-flop

FIFO First-in first-out queue

FLES First Level Event Selector, the CBM event builder

FLI Foreign language interface

FLIB FLES Input Board, CBM readout board

FLP First level processor

FMC FPGA mezzanine connector

FPGA Field programmable gate array

G-RORC GBT Read-Out Receiver Card

GBT Gigabit Transceiver Project

GBT-X Radation tolerant GBT ASIC chip

GCC GNU compiler collection

GDB GNU debugger

GDC Global Data Collector, ALICE DAQ server type

GNU A free operating system and an extensive collection of computer software

GPIO General purpose input/output

GPU Graphics processing unit

GTX Xilinx gigabit transceiver variant

GUI Graphical user interface

H-RORC HLT Read-Out Receiver Card

HDL Hardware description language

HLS High level synthesis

HLT High Level Trigger

HMPID High-Multiplicity Particle Identification Detector

HPC High performance computing

IB InfiniBand, network technology

IBERT Integrated bit error rate tester

IC Integrated circuit

IDE Integrated development environment; or: Integrated drive electronics, a computer bus interface

II Initiation interval

IO Input/output

IOMMU Input/output memory management unit

IP Intellectual property

IPMI Intelligent Platform Management Interface

IROC TPC inner readout chamber

IRQ Interrupt

ITS Inner Tracking System

JTAG Joined Test Action Group

LC An optical fiber connection type

LDC Local Data Concentrator, ALICE DAQ server type

LED Light emitting diode

LHC Large Hadron Collider

LHCb LHC-beauty, experiment at CERN

LLVM Collection of modular and reusable compiler and toolchain technologies

LS Long shutdown

LUT Look-up table

LVDS Low voltage differential signaling

MAC Media Access Control, a network protocol layer

MAD Median absolute deviation

MIG Memory interface generator

MPC Maximum performance computing

MPO Multi-fiber Push-On, an optical fiber connection type

MSI Message signaled interrupt

NAPI New API, a kernel network API

OM Optical multi-mode, an optical fiber classification

PANDA Proton Antiproton Annihilations at Darmstadt, experiment at FAIR

PASA Pre-Amplifier Shaper, ALICE TPC front-end chip

PC Personal computer

PCB Printed circuit board

PCI Peripheral component interconnect, server bus

PCI-X PCI eXtended

PDA Portable driver architecture

PFXL Xilinx PlatformFlash XL

PHOS Photon Spectrometer detector

PID Particle identification

PLD Programmable logic device

PLL Phase locked loop

PMD Photo-Multiplier Detector

PROM Programmable read-only memory

QCD Quantum chromo dynamics

QDR Quad data rate, digital signal sampling method

QGP Quark gluon plasma

QPI QuickPath Interconnect, processor interconnect

QSFP Quad SFP

RAM Random access memory

RCU TPC Readout Control Unit

RICH Ring Imaging Cherenkov detector

RISC Reduced instruction set computer

RMS Root mean square

ROBIN Read Out Buffer INterface, ATLAS readout board

ROD Read Out Driver, ATLAS readout board

ROI Region of interest

ROM Read-only memory

RORC Read-Out Receiver Card

ROS ATLAS Read Out System

RTL Register transfer level

SAMPA ALICE front-end ASIC

SATA Serial ATA, a computer bus interface

SCA Slow Control Adapter, extension to the GBT-X

SDD Silicon Drift Detector

SDR Software defined radio

SFP Small Formfactor Pluggable, transceiver form factor

SIU DDL source interface unit

SMA SubMiniature version A, a coaxial connector

SMBus System management bus

SO-DIMM Small outline DIMM

SoC System on chip

SPD Silicon Pixel Detector

SPI Serial programming interface

SPL Spacial programming language

SRAM Static random access memory

SSD Silicon Strip Detector

T-RORC TPC Read-Out Receiver Card

TCP Transmission control protocol

TDAQ Trigger and Data Acquisition

TLM Transaction level model

TOF Time Of Flight detector

TPC Time Projection Chamber

TRD Transition Radiation Detector

TWI	Two-wire interface
UVM	Universal Verification Methodology
VHDL	VHSIC hardware description language
VHSIC	Very-high-speed integrated circuits
VME	Versa Module Europa, a computer bus standard
XML	Extended markup language

Bibliography

[AA+ 14] O. Arcas-Abella, G. Ndu, N. Sonmez, M. Ghasempour, A. Armejach, J. Navaridas, W. Song, J. Mawer, A. Cristal, et al.: "An empirical evaluation of High Level Synthesis languages and tools for database acceleration", *in 2014 24th International Conference on Field Programmable Logic and Applications (FPL), p. 1–8* [IEEE, 2014], ISBN 978-3-00-044645-0, ISSN 1946-147X, doi:10.1109/FPL.2014.6927484.
URL: https://doi.org/10.1109/FPL.2014.6927484

[AA+ 16] O. Arcas-Abella, N. Sonmez: "Bluespec SystemVerilog", *in FPGAs for Software Programmers*, (edited by D. Koch, F. Hannig, D. Ziener), chap. 9, *pp. 165–172* [Springer, 2016], ISBN 978-3-319-26408-0, doi:10.1007/978-3-319-26408-0_9.
URL: https://doi.org/10.1007/978-3-319-26408-0_9

[Aad+ 08] G. Aad, et al.: "The ATLAS Experiment at the CERN Large Hadron Collider", *Journal of Instrumentation, vol. 3: p. S08003* [2008], doi:10.1088/1748-0221/3/08/S08003.
URL: https://doi.org/10.1088/1748-0221/3/08/S08003

[Aam+ 08] K. Aamodt, et al.: "The ALICE experiment at the CERN LHC", *Journal of Instrumentation, vol. 3: p. S08002* [2008], doi:10.1088/1748-0221/3/08/S08002.
URL: https://doi.org/10.1088/1748-0221/3/08/S08002

[Abb+ 16] B. Abbott, R. Blair, G. Crone, B. Green, J. Love, J. Proudfoot, O. Rifki, W. Vazquez, W. Vandelli, et al.: "The evolution of the region of interest builder for the ATLAS experiment at CERN", *Journal of Instrumentation, vol. 11, no. 02: p. C02080* [2016], doi:10.1088/1748-0221/11/02/C02080.
URL: https://doi.org/10.1088/1748-0221/11/02/C02080

[Abe+ 12] B. Abelev, J. Adam, D. Adamová, M. M. Aggarwal, G. Aglieri Rinella, M. Agnello, A. Agostinelli, N. Agrawal, Z. Ahammed, et al.: "Upgrade of the ALICE Experiment: Letter of Intent", *Tech. Rep. CERN-LHCC-2012-012. LHCC-I-022. ALICE-UG-002*, CERN [2012].
URL: https://cds.cern.ch/record/1475243

[Abg+ 14] N. Abgrall, O. Andreeva, A. Aduszkiewicz, Y. Ali, T. Anticic, N. Antoniou, B. Baatar, F. Bay, A. Blondel, et al.: "NA61/SHINE facility at the CERN SPS: beams and detector system", *Journal of Instrumentation, vol. 9, no. 06: p. P06005* [2014], doi:10.1088/1748-0221/9/06/P06005.
URL: https://doi.org/10.1088/1748-0221/9/06/P06005

[Abl+ 17] T. Ablyazimov, A. Abuhoza, R. P. Adak, M. Adamczyk, K. Agarwal, M. M. Aggarwal, Z. Ahammed, F. Ahmad, N. Ahmad, et al.: "Challenges in qcd matter physics –the scientific programme of the compressed baryonic matter experiment at fair", *The European Physical Journal A, vol. 53, no. 3: p. 60* [Mar 2017], ISSN 1434-601X, doi:10.1140/epja/i2017-12248-y.
URL: https://doi.org/10.1140/epja/i2017-12248-y

[Acc 16] Accellera Systems Initiative Inc.: "SystemC Synthesizable Subset" [2016], version 1.4.7.

[Ado+ 08] R. Adolphi, et al.: "The CMS experiment at the CERN LHC", *Journal of Instrumentation*, *vol. 0803*: *p. S08004* [2008], doi:10.1088/1748-0221/3/08/S08004.
URL: https://doi.org/10.1088/1748-0221/3/08/S08004

[ALI] ALICE Collaboration: "Alice Experiment Offline Project".
URL: https://alice-offline.web.cern.ch/

[ALI 04] ALICE Collaboration: "ALICE Technical Design Report of the Trigger, Data Acquisition, High-Level Trigger, and Control System", *Tech. Rep. CERN–LHCC–2003–062*, CERN [2004].
URL: https://edms.cern.ch/document/456354/2

[ALI 11] ALICE TPC Group: "RCU Firmware V2.1 User Manual" [2011].

[ALI 13] ALICE Collaboration: "Upgrade of the ALICE Time Projection Chamber", *Tech. Rep. CERN-LHCC-2013-020. ALICE-TDR-016*, CERN [Oct 2013].
URL: https://cds.cern.ch/record/1622286

[Alm+ 13] J. Alme, T. Alt, L. Bratrud, P. Christiansen, F. Costa, E. David, T. Gunji, T. Kiss, R. Langøy, et al.: "RCU2 — The ALICE TPC readout electronics consolidation for Run2", *Journal of Instrumentation*, *vol. 8, no. 12*: *p. C12032* [2013].
URL: https://doi.org/10.1088/1748-0221/8/12/C12032

[Alt+ 06] T. Alt, H. Appelshäuser, S. Bablok, B. Becker, S. Chattopadhyay, C. Cheshkov, C. Cicalo, J. Cleymans, R. W. Fearick, et al.: "Benchmarks and Implementation of the ALICE High Level Trigger", *IEEE Transactions on Nuclear Science*, *vol. 53, no. 3*: *pp. 854–858* [2006], ISSN 00189499, doi:10.1109/TNS.2006.873770.
URL: https://doi.org/10.1109/TNS.2006.873770

[Alt 17] T. Alt: "An FPGA based pre-processor for the ALICE High Level-Trigger", *Ph.D. thesis*, Goethe-University Frankfurt [2017].

[Alv+ 08] A. A. Alves, et al.: "The LHCb Detector at the LHC", *Journal of Instrumentation*, *vol. 3, no. CERN-LHCb-DP-2008-001*: *p. S08005* [2008], doi:10.1088/1748-0221/3/08/S08005.
URL: https://dx.doi.org/10.1088/1748-0221/3/08/S08005

[And 06] A. Andronic: "The ALICE Transition Radiation Detector and Time Projection Chamber", [Oct 2006], presentation, accessed in Mar. 2018.
URL: https://web-docs.gsi.de/~andronic/physics/gas/trd_alice.pdf

[And+ 16] J. Anderson, K. Bauer, A. Borga, H. Boterenbrood, H. Chen, K. Chen, G. Drake, M. Dönszelmann, D. Francis, et al.: "FELIX: a PCIe based high-throughput approach for interfacing front-end and trigger electronics in the ATLAS Upgrade framework", *Journal of Instrumentation*, *vol. 11, no. 12*: *p. C12023* [2016], doi:10.1088/1748-0221/11/12/C12023.
URL: https://doi.org/10.1088/1748-0221/11/12/C12023

[Bar+ 16] S. Barboza, M. Bregant, V. Chambert, B. Espagnon, H. H. Herrera, S. Mahmood, D. Moraes, M. Munhoz, G. Noël, et al.: "SAMPA chip: a new ASIC for the ALICE TPC and MCH upgrades", *Journal of Instrumentation*, *vol. 11, no. 02*: *p. C02088* [2016], doi:10.1088/1748-0221/11/02/C02088.
URL: https://doi.org/10.1088/1748-0221/11/02/C02088

[Bau+ 13] G. Bauer, T. Bawej, U. Behrens, J. Branson, O. Chaze, S. Cittolin, J. A. Coarasa, G. L. Darlea, C. Deldicque, et al.: "10 Gbps TCP/IP streams from the FPGA for the CMS DAQ eventbuilder network", *Journal of Instrumentation, vol. 8, no. 12: p. C12039* [2013], doi:10.1088/1748-0221/8/12/C12039.
URL: https://doi.org/10.1088/1748-0221/8/12/C12039

[Bec+ 16] T. Becker, O. Mencer, G. Gaydadjiev: "Spatial Programming with OpenSPL", *in FPGAs for Software Programmers*, (edited by D. Koch, F. Hannig, D. Ziener), chap. 5, *pp. 81–95* [Springer, 2016], ISBN 978-3-319-26408-0, doi:10.1007/978-3-319-26408-0_5.
URL: https://doi.org/10.1007/978-3-319-26408-0_5

[Bos+ 03] R. E. Bosch, A. J. de Parga, B. Mota, L. Musa: "The ALTRO chip: a 16-channel A/D converter and digital processor for gas detectors", *IEEE Transactions on Nuclear Science, vol. 50, no. 6: p. 2460–2469* [2003], ISSN 0018-9499, doi:10.1109/TNS.2003.820629.
URL: https://doi.org/10.1109/TNS.2003.820629

[Bri+ 06] K.-T. Brinkmann, P. Gianotti, I. Lehmann: "Exploring the mysteries of strong interactions – the panda experiment", *Nuclear Physics News, vol. 16, no. 1: pp. 15–18* [2006], doi:10.1080/10506890600579868.
URL: http://dx.doi.org/10.1080/10506890600579868

[Buc 16] J. Buchholz: "Evaluation of single photon avalanche diode arrays for imaging fluorescence correlation spectroscopy : FPGA-based data readout and fast correlation analysis on CPUs, GPUs and FPGAs", *Ph.D. thesis*, Heidelberg University [2016], doi:10.11588/heidok.00020212.
URL: https://doi.org/10.11588/heidok.00020212

[Bun+ 15] P. Buncic, M. Krzewicki, P. Vande Vyvre: "Technical Design Report for the Upgrade of the Online-Offline Computing System", *Tech. Rep. CERN-LHCC-2015-006. ALICE-TDR-019*, CERN [2015].
URL: https://cds.cern.ch/record/2011297

[Can+ 11] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, T. Czajkowski: "LegUp: High-level Synthesis for FPGA-based Processor/Accelerator Systems", *in Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11, *p. 33–36* [ACM, 2011], ISBN 978-1-4503-0554-9, doi:10.1145/1950413.1950423.
URL: https://doi.org/10.1145/1950413.1950423

[Can+ 16] A. Canis, J. Choi, B. Fort, B. Syrowik, R. L. Lian, Y. T. Chen, H. Hsiao, J. Goeders, S. Brown, et al.: "LegUp High-Level Synthesis", *in FPGAs for Software Programmers*, (edited by D. Koch, F. Hannig, D. Ziener), chap. 10, *pp. 175–190* [Springer, 2016], ISBN 978-3-319-26408-0, doi:10.1007/978-3-319-26408-0_10.
URL: https://doi.org/10.1007/978-3-319-26408-0_10

[Car+ 14] F. Carena, W. Carena, S. Chapeland, V. C. Barroso, F. Costa, E. Dénes, R. Divià, U. Fuchs, A. Grigore, et al.: "The ALICE data acquisition system", *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 741: pp. 130 – 162* [2014], ISSN 0168-9002, doi:10.1016/j.nima.2013.12.015.
URL: https://doi.org/10.1016/j.nima.2013.12.015

[Car+ 16] J. M. P. Cardoso, M. Weinhardt: "High-Level Synthesis", *in FPGAs for Software Programmers*, (edited by D. Koch, F. Hannig, D. Ziener), chap. 2, *pp. 23–47* [Springer, 2016], ISBN 978-3-319-26408-0, doi:10.1007/978-3-319-26408-

0_2.
URL: https://doi.org/10.1007/978-3-319-26408-0_2

[CER 14] CERN: "Restarting the LHC: Why 13 Tev?", [2014].
URL: https://cds.cern.ch/record/1998739

[Cos+ 16] F. Costa, V. C. Barroso, C. Soos, R. Divia, A. Wegrzynek, T. Kiss, H. Engel, G. Simonetti, J. Niedziela, et al.: "The ALICE C-RORC GBT card, a prototype readout solution for the ALICE upgrade", *in 2016 IEEE-NPSS Real Time Conference (RT), pp. 1–5* [2016], doi:10.1109/RTC.2016.7543109.
URL: https://doi.org/10.1109/RTC.2016.7543109

[Cou+ 08] P. Coussy, A. Morawiec: "High-Level Synthesis: From Algorithm to Digital Circuit" [Springer, 2008], first edn., ISBN 978-1-4020-8587-1, doi:10.1007/978-1-4020-8588-8.
URL: https://doi.org/10.1007/978-1-4020-8588-8

[Cou+ 09] P. Coussy, D. D. Gajski, M. Meredith, A. Takach: "An Introduction to High-Level Synthesis", *IEEE Design & Test of Computers, vol. 26, no. 4: pp. 8–17* [2009], ISSN 0740-7475, doi:10.1109/MDT.2009.69.
URL: https://doi.org/10.1109/MDT.2009.69

[Cyt+ 91] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, F. K. Zadeck: "Efficiently Computing Static Single Assignment Form and the Control Dependence Graph", *ACM Transactions of Programmable Language Systems, vol. 13, no. 4: pp. 451–490* [1991], ISSN 0164-0925, doi:10.1145/115372.115320.
URL: https://doi.org/10.1145/115372.115320

[dC+ 11] J. de Cuveland, V. Lindenstruth: "A First-level Event Selector for the CBM Experiment at FAIR", *Journal of Physics: Conference Series, vol. 331, no. 2: p. 022006* [2011], doi:10.1088/1742-6596/331/2/022006.
URL: https://doi.org/10.1088/1742-6596/331/2/022006

[Del+ 00] G. Dellacasa, L. Ramello, E. Scalas, M. Sitta, N. Ahmad, S. Ahmad, T. Ahmad, W. Bari, M. Irfan, et al.: "ALICE time projection chamber: Technical Design Report", Technical Design Report ALICE [CERN, 2000].
URL: https://cds.cern.ch/record/451098

[Den 74] J. B. Dennis: "First version of a data flow procedure language", *in Programming Symposium*, (edited by B. Robinet), chap. 6, *pp. 362–376* [Springer, 1974], ISBN 978-3-540-37819-8, doi:10.1007/3-540-06859-7_145.
URL: https://doi.org/10.1007/3-540-06859-7_145

[Div+ 07] R. Divià, P. Jovanovic, P. Van de Vyvre: "Data Format over the ALICE DDL; version 11", *Tech. Rep. ALICE-INT-2002-10-V11*, CERN [2007].
URL: https://cds.cern.ch/record/1027339

[Div 14] R. Divià: "ALICE Common Data Header Specifications" [2014], ALICE Internal Note.
URL: https://aliceinfo.cern.ch/Notes/node/216

[Div 16] R. Divià: "Run 2 DAQ Systems: ALICE", [2016], ALICE, ATLAS, CMS & LHCb Second Joint Workshop on DAQ@LHC.
URL: https://indico.cern.ch/event/471309/contributions/1981069/attachments/1256304/1854691/2016.04.12.ALICE.Run2.DAQ.pdf

[Edw 06] S. A. Edwards: "The Challenges of Synthesizing Hardware from C-Like Languages", *IEEE Design & Test of Computers, vol. 23, no. 5: pp. 375–386* [2006], ISSN 0740-7475, doi:10.1109/MDT.2006.134.
URL: https://doi.org/10.1109/MDT.2006.134

[Esc+ 14] D. Eschweiler, V. Lindenstruth: "The Portable Driver Architecture", *in Proceedings of the 16th Real-Time Linux Workshop*, [Open Source Automation Development Lab (OSADL), 2014].

[Esc 16] D. Eschweiler: "Efficient device drivers for supercomputers", *Ph.D. thesis*, Goethe-University Frankfurt am Main [2016].

[Eva+ 08] L. Evans, P. Bryant: "LHC Machine", *Journal of Instrumentation, vol. 3, no. 08: p. S08001* [2008], doi:10.1088/1748-0221/3/08/S08001.
URL: https://doi.org/10.1088/1748-0221/3/08/S08001

[Fab+ 04] C. W. Fabjan, L. Jirdén, V. Lindestruth, L. Riccati, D. Röhrich, P. Van de Vyvre, O. Villalobos Baillie, H. de Groot: "ALICE Trigger, Data-Acquisition, High-Level Trigger and Control System: Technical Design Report" [CERN, 2004].
URL: https://cds.cern.ch/record/684651

[Fär+ 17] C. Färber, R. Schwemmer, J. Machen, N. Neufeld: "Particle Identification on an FPGA Accelerated Compute Platform for the LHCb Upgrade", *IEEE Transactions on Nuclear Science, vol. 64, no. 7: pp. 1994–1999* [2017], ISSN 0018-9499, doi:10.1109/TNS.2017.2715900.
URL: https://doi.org/10.1109/TNS.2017.2715900

[Gaj+ 83] D. D. Gajski, R. H. Kuhn: "Guest Editors' Introduction: New VLSI Tools", *Computer, vol. 16, no. 12: pp. 11–14* [1983], ISSN 0018-9162, doi:10.1109/MC.1983.1654264.
URL: https://doi.org/10.1109/MC.1983.1654264

[Gaj+ 94] D. D. Gajski, L. Ramachandran: "Introduction to High-Level Synthesis", *IEEE Design & Test of Computers, vol. 11, no. 4: p. 44–54* [1994], doi:10.1109/54.329454.
URL: https://doi.org/10.1109/54.329454

[Geo+ 11] S. Georgakakis, J. Evans: "Overview of High Level Synthesis Tools", *Journal of Instrumentation, vol. 6, no. 02: p. C02005* [2011], doi:10.1088/1748-0221/6/02/C02005.
URL: https://doi.org/10.1088/1748-0221/6/02/C02005

[Gut+ 05] C. G. Gutierrez, R. Campagnolo, A. Junique, L. Musa, J. Alme, J. Lien, B. Pommersche, M. Richter, K. Roed, et al.: "The ALICE TPC Readout Control Unit", *in IEEE Nuclear Science Symposium Conference Record, 2005, vol. 1, p. 575–579* [IEEE, 2005], ISBN 0-7803-9221-3, ISSN 1082-3654, doi:10.1109/NSSMIC.2005.1596317.
URL: https://doi.org/10.1109/NSSMIC.2005.1596317

[Hae+ 06] G. Haefeli, A. Bay, A. Gong, H. Gong, M. Muecke, N. Neufeld, O. Schneider: "The LHCb DAQ interface board TELL1", *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 560, no. 2: pp. 494 – 502* [2006], ISSN 0168-9002, doi:10.1016/j.nima.2005.12.212.
URL: https://doi.org/10.1016/j.nima.2005.12.212

[Haf 13] J. Haffner: "The CERN accelerator complex. Complexe des accélérateurs du CERN", [2013], general Photo.
URL: https://cds.cern.ch/record/1621894

[Hal+ 10] R. Halstead, J. Villarreal, R. Moussalli, W. Najjar: "Is there a tradeoff between programmability and performance?", *in 2010 Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers, p. 836–840*

[2010], ISSN 1058-6393, doi:10.1109/ACSSC.2010.5757683.
URL: https://doi.org/10.1109/ACSSC.2010.5757683

[Heu 15] R. D. Heuer: "LHC Schedule according to MTP2015", [2015], Presentation for CERN Counsil, 176th session.

[Hur+ 07] A. R. Hurson, K. M. Kavi: "Dataflow Computers: Their History and Future", *Wiley Encyclopedia of Computer Science and Engineering* [2007], doi:10.1002/9780470050118.ecse102.
URL: https://doi.org/10.1002/9780470050118.ecse102

[Hut+ 17] D. Hutter, J. de Cuveland, V. Lindenstruth: "CBM First-level Event Selector Input Interface Demonstrator", *Journal of Physics: Conference Series*, *vol. 898, no. 3*: *p. 032047* [2017], doi:10.1088/1742-6596/898/3/032047.
URL: https://doi.org/10.1088/1742-6596/898/3/032047

[Ian 90] R. A. Iannucci: "Parallel Machines: Parallel Machine Languages: The Emergence of Hybrid Dataflow Computer Architectures" [Springer, 1990], ISBN 978-1-4613-1543-8, doi:10.1007/978-1-4613-1543-8.
URL: https://doi.org/10.1007/978-1-4613-1543-8

[IEE 12] IEEE Computer Society: "IEEE Standard for Standard SystemC Language Reference Manual", *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, *pp. 1–638* [2012], doi:10.1109/IEEESTD.2012.6134619.
URL: https://doi.org/10.1109/IEEESTD.2012.6134619

[IEE 13 I] IEEE Computer Society: "IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language", *IEEE Std 1800-2012 (Revision of IEEE Std 1800-2009)*, *pp. 1–1315* [2013], doi:10.1109/IEEESTD.2013.6469140.
URL: https://doi.org/10.1109/IEEESTD.2013.6469140

[IEE 13 II] IEEE Computer Society: "IEEE Standard for Test Access Port and Boundary-Scan Architecture", *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, *pp. 1–444* [2013], doi:10.1109/IEEESTD.2013.6515989.
URL: https://doi.org/10.1109/IEEESTD.2013.6515989

[Int 15] Intel Corporation: "Intel Completes Acquisition of Altera", http://newsroom.intel.com/community/intel_newsroom/blog/2015/12/28/intel-completes-acquisition-of-altera [2015], accessed in Jan. 2016.

[Kav+ 86] K. M. Kavi, B. P. Buckles, U. N. Bhat: "A Formal Definition of Data Flow Graph Models", *IEEE Transactions on Computers*, *vol. C-35, no. 11*: *pp. 940–948* [1986], ISSN 0018-9340, doi:10.1109/TC.1986.1676696.
URL: https://doi.org/10.1109/TC.1986.1676696

[Keb+ 13] U. Kebschull, H. Engel: "ALICE C-RORC as CBM FLES Interface Board Prototype", *vol. 2013-1* of *GSI Report*, chap. 532 - Nuclear and Quark Gluon Matter (NQM) (POF2-532) / SUC-GSI-Frankfurt - Strategic university cooperation GSI-U Frankfurt/M (SUC-GSI-FR) / HGF-IVF-VH-GS-201 - HGS-HIRe : (HGF-IVF-VH-GS-201), *p. 75 p.* [GSI Helmholtzzentrum für Schwerionenforschung, Darmstadt, 2013].
URL: http://repository.gsi.de/record/51979

[Khr 09] Khronos Group: "OpenCL Specifications" [2009].
URL: http://www.khronos.org/registry/OpenCL/specs/opencl-1.0.pdf

[Kol 12] T. Kollegger: "The ALICE High Level Trigger: The 2011 Run Experience", *in 2012 18th IEEE-NPSS Real Time Conference*, *pp. 1–4* [2012], doi:10.1109/RTC.2012.6418366.
URL: https://doi.org/10.1109/RTC.2012.6418366

[Kug 09] A. Kugel: "The ATLAS ROBIN – A High-Performance Data-Acquisition Module", *Ph.D. thesis*, Universität Mannheim [2009].
URL: http://ub-madoc.bib.uni-mannheim.de/2433/

[Leh+ 17] J. Lehrbach, M. Krzewicki, D. Rohr, H. Engel, A. G. Ramirez, V. Lindenstruth, D. Berzano, A. Collaboration: "Alice hlt cluster operation during alice run 2", *Journal of Physics: Conference Series, vol. 898, no. 8: p. 082027* [2017], doi:10.1088/1742-6596/898/8/082027.
URL: http://dx.doi.org/10.1088/1742-6596/898/8/082027

[LHC+ 16] LHC Team, S. Pandolfi: "LHC Report: end of 2016 proton-proton operation", [2016].
URL: https://cds.cern.ch/record/2229040

[Lá+ 15] A. László, E. Dénes, Z. Fodor, T. Kiss, S. Kleinfelder, C. Soós, D. Tefelski, T. Tölyhi, G. Vesztergombi, et al.: "Design and performance of the data acquisition system for the NA61/SHINE experiment at CERN", *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 798: pp. 1 – 11* [2015], ISSN 0168-9002, doi:10.1016/j.nima.2015.07.011.
URL: https://doi.org/10.1016/j.nima.2015.07.011

[Mak 13] T. Makimoto: "Implications of Makimoto's Wave", *Computer, vol. 46, no. 12: pp. 32–37* [2013], ISSN 0018-9162, doi:10.1109/MC.2013.294.
URL: https://doi.org/10.1109/MC.2013.294

[Mar+ 09] G. Martin, G. Smith: "High-Level Synthesis: Past, Present, and Future", *IEEE Design & Test of Computers, vol. 26, no. 4: pp. 18–25* [2009], ISSN 0740-7475, doi:10.1109/MDT.2009.83.
URL: https://doi.org/10.1109/MDT.2009.83

[Max 13] Maxeler Technologies: "Programming MPC Systems" [2013].
URL: https://www.maxeler.com/media/documents/MaxelerWhitePaperProgramming.pdf

[Mee+ 12] W. Meeus, K. Van Beeck, T. Goedemé, J. Meel, D. Stroobandt: "An overview of today's high-level synthesis tools", *Design Automation for Embedded Systems, vol. 16, no. 3: pp. 31–51* [2012], ISSN 1572-8080, doi:10.1007/s10617-012-9096-8.
URL: https://doi.org/10.1007/s10617-012-9096-8

[Men] Mentor Graphics Corporation: "ModelSim ASIC and FPGA Design Simulator", Accessed in Sept. 2017.
URL: https://www.mentor.com/products/fpga/verification-simulation/modelsim/

[Men 12] Mentor Graphics Corporation: "ModelSim ® SE Foreign Language Interface Manual" [2012].

[Mid 03] J. Middelink: "Bigphysarea", https://www.polyware.nl/~middelink/En/hob-v4l.html#bigphysarea [2003], accessed in Jun. 2017.

[Mil+ 15] V. Milutinović, J. Salom, N. Trifunovic, R. Giorgi: "Guide to DataFlow Supercomputing: Basic Concepts, Case Studies, and a Detailed Example" [Springer, 2015], ISBN 978-3-319-16229-4, doi:10.1007/978-3-319-16229-4.
URL: https://doi.org/10.1007/978-3-319-16229-4

[Moc 05] P. Mochel: "The sysfs Filesystem", *in Proceedings of the Linux Symposium, pp. 313–326* [2005].

[Mor+ 07] P. Moreira., A. Marchioro, K. Kloukinas: "The GBT, a proposed architecture for multi-Gbps data transmission in high energy physics", *in Proceedings of TWEPP 2007, p. 71* [2007], doi:10.5170/CERN-2007-007.332.
URL: https://doi.org/10.5170/CERN-2007-007.332

[Muk 07] M. Mukejord: "Development of the ALICE Busy Box", Master's thesis, University of Bergen [2007].

[Naj+ 16] W. A. Najjar, J. Villarreal, R. J. Halstead: "ROCCC 2.0", *in FPGAs for Software Programmers*, (edited by D. Koch, F. Hannig, D. Ziener), chap. 11, *pp. 191–204* [Springer, 2016], ISBN 978-3-319-26408-0, doi:10.1007/978-3-319-26408-0_11.
URL: https://doi.org/10.1007/978-3-319-26408-0_11

[Nan+ 16] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, et al.: "A Survey and Evaluation of FPGA High-Level Synthesis Tools", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 10: pp. 1591–1604* [2016], ISSN 0278-0070, doi:10.1109/TCAD.2015.2513673.
URL: https://doi.org/10.1109/TCAD.2015.2513673

[Ner+ 17] P. Neroutsos, U. Kebschull: "Readout and stimulus of a microwave multiplexed thermal sensor", *Verhandlungen der Deutschen Physikalischen Gesellschaft, Münster, vol. HK 15.2* [2017].

[Nik 04] R. S. Nikhil: "Bluespec System Verilog: Efficient, Correct RTL from High Level Specifications.", *in MEMOCODE, p. 69–70* [IEEE, 2004], ISBN 0-7803-8509-8, doi:10.1109/MEMCOD.2004.1459818.
URL: https://doi.org/10.1109/MEMCOD.2004.1459818

[Nik 08] R. S. Nikhil: "Bluespec: A General-Purpose Approach to High-Level Synthesis Based on Parallel Atomic Transactions", *in High-Level Synthesis: From Algorithm to Digital Circuit*, (edited by P. Coussy, A. Morawiec), chap. 8, *p. 129–146* [Springer, 2008], ISBN 978-1-4020-8588-8, doi:10.1007/978-1-4020-8588-8_8.
URL: https://doi.org/10.1007/978-1-4020-8588-8_8

[Oan+ 15] A. D. Oancea, C. Stuellein, J. Gebelein, U. Kebschull: "A resilient, flash-free soft error mitigation concept for the CBM-ToF read-out chain via GBT-SCA", *in 2015 25th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–4* [2015], ISSN 1946-147X, doi:10.1109/FPL.2015.7293999.
URL: https://doi.org/10.1109/FPL.2015.7293999

[Pai 07] F. Painke: "Hardware-Software Interface and Data Flow of the ALICE HLT", Master's thesis, KIP, Heidelberg University [2007].
URL: http://www.kip.uni-heidelberg.de/Veroeffentlichungen/download.php/4632/ps/1790.pdf

[PCI 17] PCI-SIG: "Frequently asked questions", [2017], accessed in Oct. 2017.
URL: https://pcisig.com/faq

[Pel+ 13] O. Pell, O. Mencer, K. H. Tsoi, W. Luk: "Maximum performance computing with dataflow engines", *in High-Performance Computing Using FPGAs*, (edited by W. Vanderbauwhede, K. Benkrid), *pp. 747–774* [Springer, 2013], ISBN 978-1-4614-1791-0, doi:10.1007/978-1-4614-1791-0_25.
URL: https://doi.org/10.1007/978-1-4614-1791-0_25

[Phi 01] Philips Semiconductors: "The I2C Bus Specification" [2001], rev. 2.1.

[PS 07] PCI-SIG: "PCI Express Card Electromechanical Specification Revision 2.0" [PCI-SIG, 2007].
URL: https://cds.cern.ch/record/1247943

[PS 09] PCI-SIG: "PCI Express: Base Specification; Revision 2.1" [PCI-SIG, 2009].
URL: https://cds.cern.ch/record/1247940

[PV 14] J. G. Panduro Vazquez: "The ATLAS Data Acquisition System: from Run 1 to Run 2", *Tech. Rep. ATL-DAQ-PROC-2014-035*, CERN [2014], doi:10.1016/j.nuclphysbps.2015.09.146.
URL: https://doi.org/10.1016/j.nuclphysbps.2015.09.146

[PV 17] J. G. Panduro Vazquez: "Functional diagram of the ATLAS Trigger and Data Acquisition system in Run 2 showing expected peak rates and bandwidths through each component.", [2017], accessed in Sept. 2017.
URL: https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ApprovedPlotsDAQ

[Rei+ 17] S. Reiter, H. Engel, S. Lange, W. Kühn: "Test of a PCIe based readout option for PANDA", *Verhandlungen der Deutschen Physikalischen Gesellschaft, Münster, vol. HK 27.74* [2017].

[Ric+ 07] M. Richter, K. Aamodt, T. Alt, S. Bablok, C. Cheshkov, P. T. Hille, V. Lindenstruth, G. Ovrebekk, M. Ploskon, et al.: "High Level Trigger Applications for the ALICE Experiment", *IEEE Transactions on Nuclear Science, vol. 55, no. 1: pp. 133–138* [2007], ISSN 00189499, doi:10.1109/TNS.2007.913469.
URL: https://doi.org/10.1109/TNS.2007.913469

[Roh+ 12] D. Rohr, S. Gorbunov, A. Szostak, M. Kretz, T. Kollegger, T. Breitner, T. Alt: "ALICE HLT TPC tracking of Pb–Pb Events on GPUs", *Journal of Physics: Conference Series, vol. 396: p. 012044* [2012], doi:10.1088/1742-6596/396/1/012044.
URL: https://doi.org/10.1088/1742-6596/396/1/012044

[Roh 17 I] D. Rohr: "Comparison of the chi2 of TPC tracks created with clusters found by offline clusterer, new, and old hardware cluster finder", [2017], ALICE Performance Figure ALI-PERF-140004.
URL: https://aliceinfo.cern.ch/Figure/node/12225

[Roh 17 II] D. Rohr: "Comparison of TPC dE/dx separation power using clusters from offline cluster finder, old hardware cluster finder, and new cluster finder with normal and with reduced number of bits for the cluster charge", [2017], ALICE Performance Figure ALI-PERF-140017.
URL: https://aliceinfo.cern.ch/Figure/node/12227

[Roh 17 III] D. Rohr: "Comparison of TPC/ITS matching efficiency using TPC clusters found by offline cluster finder, new, and old hardware cluster finder", [2017], ALICE Performance Figure ALI-PERF-140013.
URL: https://aliceinfo.cern.ch/Figure/node/12226

[Roh 17 IV] D. Rohr: "HLT TPC Total Compression Factor by Hardware Cluster Finder and Huffman Compression in 2016 on 13 TeV pp Data", [2017], ALICE Performance Figure ALI-PERF-129070.
URL: https://aliceinfo.cern.ch/Figure/node/11226

[Roh 17 V] D. Rohr: "HLT TPC Total Compression Factor by Hardware Cluster Finder and Huffman Compression in 2018 on 13 TeV pp Data", [2017], ALICE Performance Figure ALI-PERF-160499.
URL: https://aliceinfo.cern.ch/Figure/node/14083

[Roh 17 VI] D. Rohr: "Reduction of Numbers of TPC Clusters using 2017 data and new versions of HLT hardware cluster finder", [2017], ALICE Performance Figure ALI-PERF-141274.
URL: https://aliceinfo.cern.ch/Figure/node/12356

[Roh 17 VII] D. Rohr: "Reduction of Numbers of TPC Clusters using new HLT Cluster Finder", [2017], ALICE Performance Figure ALI-PERF-129074.
URL: https://aliceinfo.cern.ch/Figure/node/11227

[Roh 17 VIII] D. Rohr: "Tracking performance in high multiplicities environment at ALICE", *in 5th Large Hadron Collider Physics Conference (LHCP 2017) Shanghai, China, May 15-20, 2017*, [2017].

[Ron 15] F. Ronchetti: "ALICE: from LS1 to readiness for Run 2", [2015], accessed in Jun. 2017.
URL: http://cerncourier.com/cws/article/cern/60878

[Ros 12] A. D. Rosso: "Higgs: the beginning of the exploration. Étude du Higgs : ce n'est que le début", *CERN Bulletin, vol. 47/2012, no. BUL-NA-2012-357: p. 3* [2012].
URL: https://cds.cern.ch/record/1494477

[Rub+ 99] G. Rubin, P. Van de Vyvre, P. Csató, E. Dénes, T. Kiss, Z. Meggyesi, J. Sulyán, G. Vesztergombi, B. Eged, et al.: "The ALICE Detector Data Link", *Tech. Rep. CERN-OPEN-2000-107*, CERN [1999], doi:10.5170/CERN-1999-009.493.
URL: https://doi.org/10.5170/CERN-1999-009.493

[SBS 00] SBS Implementers Forum: "The System Management Bus Specification" [2000], rev. 2.0.

[Sch+ 16] M. Schmid, C. Schmitt, F. Hannig, G. A. Malazgirt, N. Sonmez, A. Yurdakul, A. Cristal: "Big Data and HPC Acceleration with Vivado HLS", *in FPGAs for Software Programmers*, (edited by D. Koch, F. Hannig, D. Ziener), chap. 7, *pp. 115–136* [Springer, 2016], ISBN 978-3-319-26408-0, doi:10.1007/978-3-319-26408-0_7.
URL: https://doi.org/10.1007/978-3-319-26408-0_7

[Sil+ 99] J. Silc, B. Robic, T. Ungerer: "Processor Architecture - From Dataflow to Superscalar and beyond." [Springer, 1999], ISBN 978-3-540-64798-0, doi:10.1007/978-3-642-58589-0.
URL: https://doi.org/10.1007/978-3-642-58589-0

[Sin+ 16] D. Singh, P. Yiannacouras: "OpenCL", *in FPGAs for Software Programmers*, (edited by D. Koch, F. Hannig, D. Ziener), chap. 6, *pp. 97–114* [Springer, 2016], ISBN 978-3-319-26408-0, doi:10.1007/978-3-319-26408-0_6.
URL: https://doi.org/10.1007/978-3-319-26408-0_6

[Sri 15] S. Sridharan: "Evaluation of 'OpenCL for FPGA' for Data Acquisition and Acceleration in High Energy Physics", *Journal of Physics: Conference Series, vol. 664, no. 9: p. 092023* [2015], doi:10.1088/1742-6596/664/9/092023.
URL: https://doi.org/10.1088/1742-6596/664/9/092023

[Ste 04] T. Steinbeck: "A Modular and Fault-Tolerant Data Transport Framework", *Dissertation thesis*, University of Heidelberg [2004], doi:10.11588/heidok.00004575.
URL: https://doi.org/10.11588/heidok.00004575

[Tau 17] A. Tauro: "ALICE Schematics", [2017], general Photo.
URL: https://cds.cern.ch/record/2263642

[TE 14] TE Connectivity: "Data Center Applications Reference Guide - Networking & Storage" [2014], Rev. 2.

[The 13] The OpenSPL Consortium: "OpenSPL: Revealing the Power of Spatial Computing", [2013], white paper, accessed Jun. 2018.
URL: http://www.openspl.org/wp-content/uploads/OpenSPL-WP1.pdf

[The 16] The Linux Foundation: "NAPI", https://wiki.linuxfoundation.org/networking/napi [2016], accessed in Jun. 2017.

[Tie 15] R. Tieulent: "ALICE Upgrades: Plans and Potentials", *in LHCP 2015*, [2015].
URL: https://cds.cern.ch/record/2111883

[US 15] US Conec Ltd.: "Frequently Asked Questions", http://www.usconec.com/resources/faq.htm [2015], accessed in Apr. 2016.

[vdB+ 97] H. C. van der Bij, R. A. McLaren, O. Boyle, G. Rubin: "S-LINK, a data link interface specification for the LHC era", *IEEE Transactions on Nuclear Science, vol. 44, no. 3: pp. 398–402* [1997], ISSN 0018-9499, doi:10.1109/23.603679.
URL: https://doi.org/10.1109/23.603679

[Web+ 16] S. G. Weber, A. Andronic: "ALICE event display of a Pb–Pb collision at 5.02A TeV", [2016], general Photo.
URL: https://cds.cern.ch/record/2202730

[Xil 10] Xilinx: "Xilinx Virtex-6 Family FPGAs - Product Table" [2010].
URL: http://www.xilinx.com/support/documentation/selection-guides/virtex6-product-table.pdf

[Xil 11] Xilinx: "Virtex-6 Memory Interface Solutions User Guide" [2011], Rev. 1.7.
URL: http://www.xilinx.com/support/documentation/ip_documentation/ug406.pdf

[Xil 12] Xilinx: "Xilinx Virtex-6 Configurable Logic Block User Guide, UG364" [2012], Rev. 1.2.
URL: http://www.xilinx.com/support/documentation/user_guides/ug364.pdf

[Xil 14] Xilinx: "Virtex-6 FPGA Data Sheet: DC and Switching Characteristics, DS152" [2014], Rev. 3.6.
URL: http://www.xilinx.com/support/documentation/data_sheets/ds152.pdf

[Xil 17] Xilinx : "UltraFast High-Level Productivity Design Methodology Guide, UG1179" [2017], v2017.2.
URL: https://www.xilinx.com/support/documentation/sw_manuals/ug1197-vivado-high-level-productivity.pdf

[Zha+ 08] Z. Zhang, Y. Fan, W. Jiang, G. Han, C. Yang, J. Cong: "AutoPilot: A Platform-Based ESL Synthesis System", *in High-Level Synthesis: From Algorithm to Digital Circuit*, (edited by P. Coussy, A. Morawiec), chap. 6, *pp. 99–112* [Springer, 2008], ISBN 978-1-4020-8588-8, doi:10.1007/978-1-4020-8588-8_6.
URL: https://doi.org/10.1007/978-1-4020-8588-8_6

[Züg 17] T. Zügel: "Development and Evaluation of a Clusterfinder based on OpenCL", Master's thesis, Frankfurt University [2017].

Zusammenfassung

Einleitung

Die Grundlagenforschung versucht mit einer Vielzahl von Experimenten zu unterschiedlichsten physikalischen Aspekten ein tieferes Verständnis unserer Welt zu fördern. Die Teilchenkollisionsexperimente der Hochenergiephysik tragen hierbei zum zentralen Verständnis der Struktur der Materie bei. All diese Experimente sind hochkomplexe Systeme mit einer Vielzahl von Komponenten, Detektoren und Technologien. Sie nutzen unterschiedliche physikalische Eigenschaften der Kollisionsprodukte aus, um diese zu detektieren, deren Spuren, Energien oder Impulse zu messen und daraus die physikalischen Stoßprozesse zu rekonstruieren. Die Detektoren übersetzen die Signaturen der Kollisionsprodukte in elektrische Signale. Diese werden digitalisiert, gesammelt und ausgelesen, um die physikalischen Vorgänge aus den aufgezeichneten Daten rekonstruieren zu können. Eine gemeinsame Herausforderung dieser Experimente ist die Koordination, Synchronisation und Auslese dieser massiv-parallelen Systeme bei Ereignisraten von mehreren Kilohertz. Eine typische Auslesekette beginnt mit spezialisierter Front-End Elektronik nahe am Detektor selbst, oft mit ASICs umgesetzt, um die Detektoreigenschaften sowie die räumlichen und strahlungsbedingten Anforderungen bestmöglich abdecken zu können. Abhängig vom Detektor folgt eine Aggregations-Stufe, die Daten von mehreren Front-End Einheiten auf eine kleinere Anzahl schnellerer Übertragungskanäle zum Data Acquisition (DAQ) System hin zusammenfasst. Diese Kanäle sind oft als serielle optische Verbindungen ausgelegt, um eine räumliche und elektrische Trennung zwischen den Detektoren und dem Auslesesystem zu erreichen. Das DAQ System ist in den meisten Fällen ein Rechnercluster, mit dem die Auslese koordiniert wird sowie Daten teilweise bereits rekonstruiert und zur dauerhaften Speicherung vorbereitet werden.

Eine Technologie, die bei vielen Ausleseketten an verschiedenen Stellen eingesetzt wird, sind programmierbare Hardware-Bausteine in der Form von FPGAs. Der Einsatz dieser Komponenten in der Front-End Elektronik hängt stark von den Umgebungsbedingungen ab. In den Aggregations-Stufen, als Schnittstelle ins DAQ System sowie als erste Datenverarbeitungsstufe sind diese jedoch stark vertreten. FPGAs bestehen aus einem Array aus Logikelementen. Deren Funktion und ihre Verbindung untereinander ist vom Nutzer konfigurierbar. Integrierte Funktionseinheiten für Speicher, mathematische Operationen und Schnittstellen nach außen erweitern den Funktionsumfang. Mit FPGAs kann benutzerdefinierte Logik umgesetzt werden, die für kleine bis mittlere Stückzahlen deutlich günstiger als ASICs ist und jederzeit an geänderte Bedingungen angepasst werden kann. Außerdem können FPGAs durch ihre Parallelität eine Rechenleitung bieten, die in vielen Anwendungsfällen auch von CPU, GPU oder DSP Systemen nicht erreichbar ist. FPGAs haben in den letzten Jahren ihren festen Platz in den Ausleseketten von Physikexperimenten gefunden. Mit immer größeren und schnelleren Chips werden außerdem immer komplexere Verarbeitungsschritte in Hardware möglich.

Der Large Hadron Collider (LHC) an der Europäischen Organisation für Kernforschung (CERN) ist derzeit der leistungsstärkste Teilchenbeschleuniger der Welt. Dutzende verschiedene Experimenten sind daran angebunden. ALICE ist eines der vier größten Experimente am LHC und auf die Erforschung von Schwerionenkollisionen sowie des dabei entstehenden Quark-Gluon-Plasma spezialisiert. ALICE besteht aus 19 Sub-Detektoren, die mit Hilfe eines komplexen Synchronisierungs-, Trigger- und Auslesesystems zeitgleich betrieben werden. Der größte De-

tektor in ALICE in Bezug auf die physikalische Größe, das Datenvolumen und die Anzahl der Ausleselinks ist die Time Projection Chamber (TPC). Die TPC ist ein zylindrisches Gasvolumen um den Kollisionspunkt. Es enthält ein starkes elektrischen Feld entlang der Strahlachse. Gasmoleküle, die von Kollisionsprodukten ionisiert werden, driften entlang dieses elektrischen Felds zu einem Auslesesystem, das die Ladungsverteilung im gesamten Gasvolumen über die Zeit aufzeichnet. Daraus sind Rückschlüsse auf Teilchentrajektorien und Impulse möglich. Über 500 serielle optische Links stellen den Datentransport zwischen den Detektoren und dem DAQ System sicher. Parallel wird eine Kopie der Detektordaten an den High Level Trigger (HLT) geschickt, der die Daten rekonstruiert, kalibriert und komprimiert. Durch die Kompression und Speicherung vorverarbeiteter Daten können die ursprünglichen Rohdaten verworfen und damit die Datenmenge zur dauerhaften Speicherung signifikant reduziert werden. Nur diese Kompression macht es möglich, die bei Schwerionenkollisionen anfallenden Datenmengen überhaupt speichern zu können. Sie trägt außerdem zu einer effizienten Nutzung des für das Experiment verfügbaren Speicherkontingents bei. Die Datenrekonstruktion, -Kompression und -Aggregation erfolgt mit kommerziell erhältlichen Rechner- und Netzwerktechnologien. Die Rechnercluster der DAQ und HLT Systeme bestehen jeweils aus rund 200 Knoten.

Eine zentrale Komponente dieser Auslesearchitektur ist die Schnittstelle zwischen den optischen Detektorlinks und den Rechnerclustern der DAQ und HLT Systeme. In der ersten LHC Runperiode (Run 1) zwischen 2009 und 2012 wurden hierfür speziell entwickelte FPGA-basierte Auslesekarten eingesetzt, die Read-Out Receiver Cards (RORCs). Die RORCs in den DAQ und HLT Systemen wurden unabhängig voneinander entwickelt und stellten jeweils zwei serielle optische Links sowie eine Schnittstelle zum Host-Rechner zur Verfügung. In der HLT-Variante war neben der reinen Datenauslese zudem eine Datenvorverarbeitung in Form eines Hardware Cluster Finder Algorithmus integriert. Dieser Algorithmus extrahiert bereits bei der Auslese charakteristische Signaturen aus den TPC Rohdaten. Diese hoch-parallele FPGA-basierte Vorverarbeitung ersparte beträchtliche Mengen an CPU Rechenleistung im Vergleich zu äquivalenten Verarbeitungsschritten in Software.

Motivation und Ziele der Arbeit

Nach dem Ende der ersten LHC Runperiode wurden mehrere ALICE Detektorsysteme weiter ausgebaut und für höhere Daten- und Ereignisraten in der zweiten LHC Runperiode (Run 2) ab 2015 vorbereitet. Die DAQ und HLT Systeme wurden zum größten Teil mit neuer Hardware ersetzt, um den Anforderungen von Run 2 gerecht werden zu können. Die bisherigen FPGA Auslesekarten konnten auf Grund der höheren Linkraten sowie einer veralteten Host-Schnittstelle nicht mehr mit der verbesserten Detektorhardware und den neuen Server-Maschinen verwendet werden. Ein Ziel dieser Arbeit war daher die Entwicklung und der Betrieb einer Ausleselösung für ALICE in Run 2, die den erhöhten Anforderungen gerecht wird, mit den bestehenden Schnittstellen kompatibel ist, eine zeitgemäße und zukunftssichere Technologie bietet und für Erweiterungen der Verarbeitungsschritte in Hardware gerüstet ist. Die Auswahl der Hardware war von Anfang an darauf ausgelegt, dass dieselbe Plattform in den DAQ und HLT Systemen in Run 2 zum Einsatz kommt. Firmware und Software Implementierungen erfolgten für beide Gruppen separat, da die Anwendungen und die aus Run 1 wiederverwendeten Technologien deutlich verschieden sind. Die Firmware-Entwicklungen für den HLT Produktivebetrieb sowie die Integration ins HLT Datenverarbeitungssystem waren ebenfalls Ziele dieser Arbeit.

In einem ersten Schritt musste eine Hardware gefunden werden, die die erhöhten Anforderungen erfüllen kann, in entsprechender Stückzahl rechtzeitig verfügbar ist und in die bestehende Experiment-Infrastruktur integriert werden kann. Hierzu mussten Hardware-Technologien evaluiert, Prototypen erstellt und getestet sowie eine experimentweite Beschaffung organisiert und koordiniert werden. Ein zweiter Aspekt war die Integration der neuen Auslesekarten in die Hardware- und Softwareumgebung des HLT. Dies umfasste die Entwicklung der Auslesefirmware sowie die Einbindung in das bestehende verteilte HLT Datenverarbeitungssystem. Weiterhin war die Integration, Wartung und Weiterentwicklung der FPGA-basierten Datenvorverabeitung in

Form des Hardware Cluster Finders erforderlich, der auch in Run 2 einen zentralen Bestandteil der HLT Datenverarbeitungskette bildet. Diese beiden Arbeitspakete waren streng vom Zeitplan des Experiments abhängig. Rechtzeitig vor Beginn der zweiten LHC Runperiode musste eine zuverlässige Hardwarelösung in den Produktivsystemen von DAQ und HLT installiert sein. Diese musste in die Auslese- und Datenverarbeitungssoftware integriert sein und gründlich getestet für die Aufzeichnung von Experimentdaten bereitstehen.

Ein dritter Aspekt dieser Arbeit betrifft die Beschreibung von Datenverarbeitungsschritten in Hardware. Die gängigen Hardwarebeschreibungsmethoden arbeiten üblicherweise auf der Register-Transfer-Ebene, die Datenbewegungen und -Verarbeitungsschritte zwischen Registern Taktsynchron beschreibt. Die Implementierung von Algorithmen auf dieser Ebene ist möglich und wird seit Jahren so betrieben. Auch der Hardware Cluster Finder ist auf dieser Ebene beschrieben. Jedoch bringt die Beschreibung eines Algorithmus auf dieser Ebene einen hohen Aufwand an Entwicklung, Verifikation und Wartung mit sich. Bereits kleine Änderungen am Algorithmus können große Arbeiten an der Implementierung bedeuten. In den letzten Jahren wurden einige Werkzeuge zur Hardwarebeschreibung auf algorithmischer Ebene in Form von High-Level Synthese (HLS) veröffentlicht, die insbesondere den Aufwand von Entwicklung, Verifikation und Wartung vereinfachen sollen. Die meisten dieser Ansätze verwenden Teilmengen imperativer Programmiersprachen, die mit Anweisungen erweitert wurden, um die zeitliche und räumliche Parallelität einer Hardware-Implementierung auszudrücken. Ein vielversprechender Ansatz, der sich konzeptionell von den populärsten Ansätzen etwas abgrenzt, ist die Datenfluss-Programmierung. Datenfluss-Architekturen hatten es zwar nie in kommerziell erfolgreiche Rechnersysteme geschafft, die Konzepte lassen sich jedoch vergleichsweise einfach auf FPGAs abbilden, um damit hocheffektive Datenverarbeitungssysteme zu entwickeln. Am Beispiel des Hardware Cluster Finder sollte im Rahmen dieser Arbeit der Datenfluss-Ansatz zur High-Level Synthese untersucht werden. Die bestehende Implementierung des Hardware Cluster Finders auf Register-Transfer-Ebene diente hierbei als direkte Referenz, um den Datenfluss-Ansatz zu evaluieren und die Realisierbarkeit einer algorithmischen Hardwarebeschreibung für die Datenverarbeitung einer Experimentauslese zu untersuchen.

Die C-RORC Hardware Plattform

Im Rahmen dieser Arbeit wurde die Common Read-Out Receiver Card (C-RORC) als eine gemeinsame Auslesekarte für die ALICE DAQ und HLT Systeme in Run 2 entwickelt. Diese Karte musste in der Lage sein, die optischen Detektorlinks sowohl mit der in Run 1 verwendeten Signalrate, als auch mit erhöhten Raten für die in Run 2 erweiterten Detektoren auslesen zu können. Die Schnittstelle zum Host-Rechner musste hierbei genügend Bandbreite zur Verfügung stellen, um die eingehenden Detektordaten weiterzuleiten. Im Zuge des technischen Fortschritts war eine Lösung mit erhöhter Link-Dichte gewünscht, da dies die Anzahl der benötigten Host-Rechner zur Installation der Karten reduziert. Weiterhin mussten ausreichend FPGA Ressourcen verfügbar sein, um die aus Run 1 im HLT bereits eingesetzte Datenvorverabeitung in Form des Hardware Cluster Finders in Run 2 mit den erhöhten Anforderungen weiterführen und erweitern zu können. Für die HLT-Anwendung wurde Speicher auf der Karte benötigt, um Detektordaten zu Testzwecken abspielen zu können. Eine spezielle elektrische Schnittstelle wurde von der DAQ-Anwendung benötigt, um die bestehende Flusskontrolle mit dem Triggersystem weiterverwenden zu können. Neben diesen unbedingt notwendigen technischen Voraussetzungen waren eine kompakte Bauform, eine dynamische Aus-

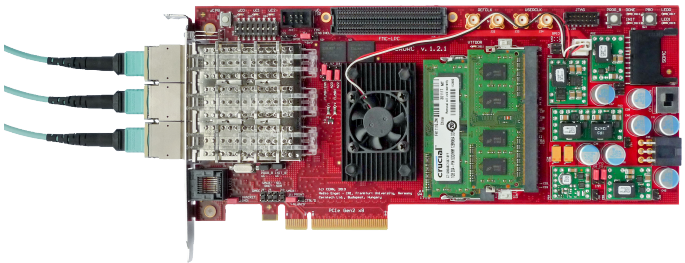

Abbildung 1: Foto der C-RORC Karte.

wahloption der seriellen Signalraten sowie ein zuverlässiges und flexibles Konfigurationssystem für die effiziente Integration und den Betrieb in den Produktivsystemen vorgesehen.

Weder zum Zeitpunkt der Hardware-Auswahl, noch zu einem späteren Zeitpunkt war eine kommerzielle Hardware-Plattform verfügbar, die diese Anforderungen erfüllen konnte. Hardware-Entwicklungen anderer Experimente waren entweder noch nicht weit genug fortgeschritten, oder nicht kompatibel mit den ALICE Anforderungen. Aus diesem Grund wurde die C-RORC als eine eigene FPGA-basierte Auslesekarte konzipiert und entwickelt. Ein Foto der Karte ist in Abbildung 1 gezeigt.

Eine PCI-Express Schnittstelle zum Host-Rechner bietet ausreichend Bandbreite zur Übertragung der Detektordaten. Messungen der Übertragungsraten ergaben Durchsatzwerte von etwa 3.6 GB/s, was für alle Ausleseszenarien der Karte in Run 2 mehr als ausreichend ist. Parallel-optische Schnittstellen ermöglichen die Auslese von bis zu 12 Detektorlinks pro Karte. Eine C-RORC kann damit bis zu sechs der vorherigen Karten ersetzen. Der Einsatz von Quad-SFP (QSFP) Transceiver Modulen ermöglicht es, alle 12 Links in der Frontblende unterzubringen und die Höhe der Karte somit auf eine Höheneinheit zu beschränken. Optische Aufteilkabel zwischen den Steckverbindungen für einzelne Fasern, wie sie im Experiment verbaut sind, sowie den parallel-optischen Anschlüssen an der C-RORC machen diese Lösung kompatibel mit der bestehenden Glasfaser-Installation. Zwei Steckplätze für Speichermodule bieten die Möglichkeit, mehrere Gigabyte Speicher in die Firmware einzubinden. Die modulare Lösung bedeutet keine Hardware-Mehrkosten, falls dieser Speicher für eine Anwendung nicht gebraucht wird. Weiterhin kann die Größe des benötigten Speichers damit individuell pro Board festgelegt und jederzeit geändert werden. Die beiden Speichermodule können unabhängig voneinander angesprochen werden und bieten eine sequenzielle Lesebandbreite von jeweils ca 8.5 GB/s. Ein zentraler Aspekt, der so auch in keinen anderen kommerziellen Karten verfügbar ist, ist das C-RORC Konfigurationssystem. Das FPGA muss bei Inbetriebnahme für die gewünschte Funktionalität konfiguriert werden. Diese Konfigurationsdaten werden üblicherweise aus einem nicht-flüchtigen Speicher auf der Karte ins FPGA geladen. Sobald diese Konfiguration geladen ist, ist das FPGA auch über die PCI-Express Schnittstelle ansprechbar. Damit können Daten ausgelesen, aber auch die gespeicherte Konfiguration erneuert werden. Um über PCI-Express ansprechbar zu sein, muss die FPGA-Konfiguration innerhalb von einigen Millisekunden nach Einschalten des Host-Rechners erfolgen, andernfalls wird das Gerät vom System ignoriert. Die initial korrekte und schnelle Konfiguration ist demnach unabdingbar, um überhaupt mit der Karte kommunizieren zu können. Falls dieser Prozess fehlschlägt, müssen die meisten FPGA-Karten an ein Programmierkabel angeschlossen werden. Dies ist eine Situation, die insbesondere bei einer Cluster-weiten Installation von dutzenden Karten in Server-Maschinen unbedingt vermieden werden muss. Um die zeitlichen Anforderungen zu erfüllen, ist die C-RORC mit synchronem Flash-Speicher ausgestattet, der die notwendige Bandbreite zur schnellen Konfiguration bietet. Zwei parallele Einheiten dieses Flash-Speichers ermöglichen mehr als eine Konfigurationsdatei auf der Karte zu halten. Dies bietet die Möglichkeit, eine Backup-Konfiguration dauerhaft auf der Karte zu halten, falls der Betrieb mit einem neuen Firmware-Abbild fehlschlägt. Weiterhin ist ein Mikroprozessor verbaut, der den FPGA-Konfigurationsprozess überwachen und steuern kann. Dieser ist über PCI-Express Seitenkanäle mit dem Host-Rechner verbunden und kann auch angesprochen werden, falls die PCI-Express Schnittstelle nicht verfügbar ist. Die Kombination von parallelen, schnellen Flash-Speichern mit dem Kommunikationskanal zum Mikroprozessor stellt eine zuverlässige Konfigurationslösung zur Verfügung, die auch mit einer fehlgeschlagenen Konfiguration umgehen kann. Alle weiteren Hardwaremerkmale sind weitgehend generisch gehalten, um einen Einsatz der Karte auch für andere Anwendungen zu ermöglichen. Die seriell-optischen Links können mit einem konfigurierbaren Taktgenerator bei beliebigen Signalraten zwischen einigen hundert Kbps und etwa 6.6 Gbps betrieben werden. Eine generische digitale Eingangs-/Ausgangsschnittstelle ermöglicht es, die Funktionalität der Karte mit eigenen oder kommerziell erhältlichen Aufsteckkarten zu erweitern. Eine LVDS Schnittstelle über Ethernet-Buchsen stellt die für die DAQ Anwendung gebrauchte Verbindung zum Triggersystem zur Verfügung, kann aber auch für andere Zwecke benutzt werden. Mehrere I^2C Ketten sowie LEDs auf der Karte bieten Status- und Kontrollinformationen von Peripheriekomponenten.

Die C-RORC wurde im Rahmen dieser Arbeit als FPGA Auslesekarte für ALICE konzipiert, umgesetzt und integriert. Eine Entwicklung von eigenen Karten bringt immer mehr Herausforderungen insbesondere für kleine Entwicklerteams. Die Komplexität steigt durch immer mehr Funktionsumfang und schnellere Schnittstellen der Komponenten. Über tausend Pins allein am FPGA und Firmwareentwicklung bereits für die Stromversorgung einer Karte sind keine Seltenheit. Im selben Maße wächst der Umfang der Dokumentation sowie die Kosten und das erforderliche Wissen für passende Entwicklungs-, Simulations- und Messwerkzeuge. Nicht selten is die nächte Generation FPGAs verfügbar, bevor eine Eigenentwicklung wirklich in Produktion gehen kann.

Die Konzeption der ALICE Auslesehardware für Run 2 wurde 2010 begonnen. Anforderungsanalyse, Technologieauswahl und Stromlaufplan wurden im Rahmen dieser Arbeit durchgeführt. Das PCB Layout sowie die Produktion erster Prototypen wurde über eine Firma eingekauft. Parallel wurde die Firmware und Software mit Hilfe von kommerzieller Hardware mit reduziertem Funktionsumfang vorbereitet, um die eigene Hardware damit testen zu können. Die ersten beiden Prototypen-Serien waren Anfang und Mitte 2013 verfügbar. Zu diesem Zeitpunkt hat sich das ATLAS Experiment dazu entschieden, dieselbe Hardware für ihr Auslesesystem in Run 2 einzusetzen. Die Serienproduktion der Karten erfolgte daraufhin in Zusammenarbeit zwischen den ALICE und ATLAS Gruppen. Dadurch konnte der Preis pro Karte durch das insgesamt höhere Produktionsvolumen reduziert werden. Weiterhin hat der Zusammenschluss der Hardware- und Firmware-Experten beider Experimente zu einem regen Austausch und zur Zusammenarbeit zwischen den Gruppen geführt. Knapp 400 C-RORCs wurden bis Mitte 2014 gefertigt, wobei Teile der Koordination und ein großer Teil der Entwicklung einer Testumgebung zur Verifikation aller Karten bereits beim Hersteller im Rahmen dieser Arbeit geleistet wurden. Die Karten wurden im Herbst 2014 in die ALICE und ATLAS Produktivsysteme eingebaut und sind seitdem zentrale Bestandteile der jeweiligen Experiment-Auslese.

Neben den Hauptanwendungen in den Run 2 Auslesesystemen von ALICE und ATLAS hat die Karte eine Anwendung in verschiedenen Test- und Evaluationssystemen gefunden. Die Karte wird als Auslese-Prototyp für Run 3 eingesetzt, solange die finale Hardware nicht in ausreichender Stückzahl verfügbar ist. Teilweise sollen die Karten auch in Run 3 weiterverwendet werden. Die nach Run 2 in ALICE nicht mehr benötigten Karten sind bereits für andere Experimente vorgesehen. TPC Strahltests und Front-End Elektronik Massentests werden mit C-RORCs mit einer modifizierten HLT-Firmware durchgeführt. Andere Experimente testen mit der Karte einen möglichen Wechsel zu einer PCI-Express basierten Experimentauslese. Die generische Implementierung der Karte ermöglicht hierbei eine Vielzahl von Anwendungen. Einige Firmware- und Hardwarekonzepte aus dieser Arbeit wurden in andere Hardware- und Firmwareprojekte übernommen, die zum Teil sogar völlig unabhängig von der Hochenergiephysik sind.

Implementierung im HLT

Abbildung 2: Links: Foto des ALICE HLT Clusters zu Beginn von Run 2. Mitte: Offener HLT Rechnerknoten mit C-RORC in rot. Rechts: Optische Aufteilkabel zur Glasfaster-Installation.

Nach Ende der ersten LHC Runperiode wurde die gesamte HLT-Hardware ersetzt, um die Daten- und Ereignisraten in Run 2 verarbeiten zu können. Der HLT besteht nun aus 180 identischen

Rechner-Knoten die jeweils eine GPU enthalten und mit InfiniBand und Ethernet miteinander verbunden sind. 74 dieser Knoten enthalten außerdem eine C-RORC als zentrale Ein- und Ausgangsschnittstelle für Detektordaten in und aus dem HLT. Die 74 C-RORCs bilden hierbei die Verbindung zwischen über 500 Glasfaserleitungen und den HLT Rechnerknoten. Bis zu 12 Detektorlinks werden pro C-RORC ausgelesen. Glasfaser-Aufteilkabel verbinden die parallel-optischen Schnittstellen der C-RORCs mit der bestehenden Experiment-Infrastruktur. Fotos des Clusters, eines HLT Rechners sowie der Glasfaserinstallation sind in Abbildung 2 gezeigt. Die C-RORCs sind in zwei Datenflussrichtungen im HLT integriert. Sie bilden die Eingangsschnittstelle des HLT, um die Detektordaten von den optischen Links entgegenzunehmen, gegebenenfalls bereits im FPGA mit dem Hardware Cluster Finder vorzuverarbeiten, und an den Host-Rechner weiterzugeben. Weiterhin stellen die C-RORCs die Ausgangsschnittstelle des HLT, indem sie die vom HLT komprimierten und rekonstruierten Daten vom Host-Rechner lesen und über die optischen Links an das DAQ System schicken. Neben den Firmwarevarianten, die im HLT Produktivsystem eingesetzt sind, wurden verschiedene Varianten für Hardware- und Firmwaretests sowie zur Evaluation der Datenvorverabeitungs-Algorithmen entwickelt. Eine modulare Firmware-Architektur ermöglicht die Verwaltung aller Varianten in einem zentralen Quellcode-Repository und die Instanziierung von denselben Modulen in verschiedenen Firmware-Varianten. Die Signalrate der optischen Links ist zur Laufzeit konfigurierbar. Damit kann dieselbe Firmware-Variante für unterschiedliche Detektoren eingesetzt werden. Die Schnittstelle zum Host-Rechner ist mit einer eigenen Direct Memory Access (DMA) Implementierung zum direkten Zugriff auf den Host-Speicher realisiert. Dies war erforderlich, da es mit keiner der verfügbaren kommerziellen FPGA DMA-Lösungen möglich war, 12 unabhängige Kanäle zur Verfügung zu stellen. Hierfür konnten HLT Konzepte aus Run 1 wiederverwendet werden, die Implementierung musste jedoch durch die geänderte Schnittstelle von Grund auf neu erfolgen. Der Datentransfer zwischen dem FPGA und dem Host-Rechner erfolgt mit zwei Puffern pro Kanal, einer für die Detektordaten und einer für Deskriptoren. Durch periodisches Abfragen des Deskriptor-Puffers werden neue Daten erkannt. Die Implementierung als Ringpuffer ermöglicht es, die Flusskontrolle auf eine einfache Zwei-Zeiger-Arithmetik zu reduzieren. Sowohl für den Datentransport vom FPGA zum Host-Rechner, als auch in die Gegenrichtung, stellt dieser Ansatz ausreichend Bandbreite zur Verfügung. Im Host-Rechner bildet ein Kernelmodul die Grundlage für die Kommunikation mit der Karte per Software sowie die Reservierung von DMA Speicher. Sämtliche Interaktionen mit der Karte erfolgen aus dem Benutzer-Kontext mit einer Treiber-Bibliothek. Die C-RORCs sind ins HLT Datenverarbeitungssystem als Datenquellen und -Senken unter Verwendung dieser Bibliothek integriert. Jeder Detektorlink ist von einem eigenständigen Prozess verwaltet. Dies ermöglicht es, beliebige Link-Konfigurationen unabhängig voneinander zu betreiben. Sämtliche Firmware-Varianten, die gesamte Integration ins HLT-System sowie große Teile der Treiber-Bibliothek wurden im Rahmen dieser Arbeit entwickelt.

Ein generelles Problem bei der Firmware-Entwicklung, insbesondere im Zusammenspiel mit Software, ist die Verifikation. Auf Modul-Ebene können Unit-Tests die gängigsten Fehler mit vertretbarem Aufwand abfangen. Eine formelle Verifikation ist in der Regel auf System-Ebene nicht möglich. Eine Verifikation auf System-Ebene arbeitet deshalb oft mit einem Satz von typischen Anfragen und Bedingungen, um Fehler automatisiert zu erkennen. Wird das System selbst durch Software-Interaktion gesteuert, ist oft nicht klar, ob die Anwendungsfälle der Verifikationsmethoden wirklich die Realität widerspiegeln. Aus diesem Grund wurde hier ein System zur Hardware/Software Co-Simulation entwickelt. Die Hardware-Simulationsumgebung wurde hierbei so erweitert, dass sie per Software dynamisch gesteuert werden kann. Eine Option in der Treiber-Bibliothek ermöglicht es, dass Anfragen nicht direkt an die Hardware, sondern an die Hardware-Simulationsumgebung geschickt werden. Auf diese Weise kann die gesamte C-RORC Firmware im Zusammenspiel mit der Host-Software emuliert werden. Beide Prozesse können mit den üblichen Debugging-Werkzeugen jederzeit pausiert und analysiert werden. Auf diese Weise konnte die gesamte HLT Quell-Komponente mit einer Simulation der Firmware betrieben werden und Fehler sowohl auf Firmware-, als auch auf Softwareseite identifiziert und behoben werden.

Die HLT Anwendung verwendet die steckbaren C-RORC Speichermodule zum Abspielen von generierten, zuvor aufgezeichneten oder bewusst verfälschten Detektordaten zu Systemtests. De-

tektordaten aus dem Speicher werden so ins System eingespeist, als würden sie über die optischen Links ankommen. Dabei sind sowohl die Datenmenge, als auch die -Rate frei konfigurierbar. Auf diese Weise können sowohl einzelne Karten, als auch das HLT Gesamtsystem, auf Durchsatz und Fehlertoleranz getestet werden. Diese Funktionalität ist direkt in den HLT-Komponenten integriert und kann unabhängig vom Status des LHC oder der Detektoren jederzeit für Tests verwendet werden. Dieses System wird regelmäßig eingesetzt, um die Funktionalität des HLT nach Unterbrechungen sicherzustellen oder die Konfiguration für anspruchsvolle Datenaufzeichnungsbedingungen zu testen.

Die C-RORCs sind weiterhin in das HLT Cluster Management- und Monitoring-Framework integriert. Die FPGA-Temperatur sowie Schnittstellen-Parameter werden kontinuierlich überwacht und zusammen mit den Monitoring-Parametern der Rechner-Knoten in einer Monitoring-Datenbank gespeichert. Die Zuordnung von FPGA Firmware-Variante und Link-Konfiguration zu den einzelnen Karten im Cluster ist über das Cluster-Management System gesteuert. Dieses stellt sicher, dass jede Karte im Cluster korrekt für den Detektor konfiguriert ist, zu dem die angeschlossenen optischen Links gehören. Ein Firmware-Update oder ein Austausch eines Knotens kann damit einfach und zuverlässig automatisiert erfolgen.

Hardware Cluster Finding

Der Hardware Cluster Finder ist ein Algorithmus, der in den TPC Rohdaten nach charakteristischen Signaturen, sogenannten “Clustern”, sucht und deren Parameter berechnet und ausgibt. Die Rohdaten werden nach der Auslese verworfen und nur die komprimierten Cluster werden gespeichert. Die Implementierung dieses Algorithmus als parallele Verarbeitungspipeline direkt im Auslese-FPGA war bereits während der ersten LHC Runperiode in den HLT Auslesekarten integriert. Die Hardware-Vorverarbeitung der Daten war hierbei deutlich effizienter als eine vergleichbare Software-Implementierung und konnte die benötigte CPU Rechenleistung im HLT signifikant reduzieren. Der Algorithmus kann mit laufenden gewichteten Summen beschrieben werden, die sich effizient mit FPGAs berechnen lassen. Die Hardware-Implementierung besteht aus vier aufeinander folgenden Berechnungsschritten. In einem ersten Schritt wird der Datenstrom dekodiert und Ladungswerte sequentiell aufgereiht. Daraufhin wird die zeitliche Abfolge der Ladungswerte auf lokale Hochpunkte untersucht, ein Bereich um die Hochpunkte ausgeschnitten und deren Positionsparameter in Zeitrichtung berechnet. In einem dritten Schritt werden die zeitlichen Parameter benachbarter Detektorbereiche zusammengefasst, um die zugehörigen Eigenschaften in Ortsrichtung zu bestimmen. Im letzten Schritt erfolgt eine Division beider Parametersätze durch die Gesamtladung. Zu diesem Algorithmus existiert ein Bit-genaues Software-Emulationsmodell zur Verifikation der Hardware. Der Offline Cluster Finder, der einen etwas anderen Algorithmus verwendet, bildet die Referenz in Bezug auf die physikalischen Ergebnisse.

Die bestehende FPGA-Implementierung aus den Run 1 Auslesekarte wurde in einem ersten Schritt auf die C-RORC portiert. Dank ähnlicher Hardware-Architektur und anfangs derselben TPC Front-End Elektronik wie in Run 1, war dies mit nur kleineren Firmware-Anpassungen möglich. Sechs Cluster Finder Instanzen sind hierbei pro C-RORC integriert. Die Daten von 216 TPC Ausleselinks werden von 36 C-RORCs im HLT vorverarbeitet. Durch die neuere Architektur des C-RORC-FPGAs ging der Ressourcenverbrauch pro Cluster Finder Instanz geringfügig zurück. Diese Firmware-Variante war zu Beginn der zweiten LHC Runperiode im HLT integriert.

Ein wichtiger Aspekt der Arbeiten an der Datenvorverabeitung ist die Verifikation der Hardware-Implementierung. Da die ursprünglichen Rohdaten verworfen und nur die Cluster gespeichert werden, kann ein Fehler in der Hardware-Implementierung große Mengen an aufgezeichneten Daten unbrauchbar machen. Ein verifiziertes Software-Modell des Cluster Finders existiert, die Übereinstimmung der Hardware-Implementierung mit dem Software-Modell muss jedoch sichergestellt sein. Eine Hardware-Simulation ist vergleichsweise langsam, sodass damit keine ausreichenden Datenmengen für eine zuverlässige Aussage zur Korrektheit der Implementierung getroffen werden können. Eine Verifikation der Hardware durch Abspielen von Rohdaten aus dem Kartenspeicher und einem Vergleich der Resultate mit der Software-Referenz ist möglich, aber umständlich. Aus

diesem Grund wurde eine Coprozessor-Implementierung des Cluster Finders entwickelt, der kontinuierlich Rohdaten vom Host-Rechner lesen, diese mit dem Cluster Finder bearbeiten und an den Host zum direkten Vergleich mit der Software-Referenz zurück schicken kann. Auf Grund der modularen Firmware-Architektur war diese Implementierung aus Bausteinen der bestehenden Firmware-Varianten möglich. Mit sechs parallelen Cluster Finder Instanzen können vollständige TPC Ereignisdaten von einer einzigen Karte verarbeitet werden. Mit der Coprozessor-Implementierung konnten große Mengen Rohdaten mit der Software-Implementierung vergleichen und Änderungen am Hardware Cluster Finder zuverlässig getestet werden.

Eine zentrale Erweiterung für ALICE in Run 2 war das Upgrade der TPC Ausleseelektronik mit der Readout Control Unit 2 (RCU2). Mit einer höher parallelisierten Auslesearchitektur kann etwa die doppelte Datenrate vom Detektor zu den RCUs ausgelesen werden. Um diese Datenrate an die DAQ und HLT Onlinesysteme weiterreichen zu können, wurde die Signalrate der optischen Links entsprechend erhöht. Die veränderte Auslesearchitektur brachte außerdem Anpassungen im Datenformat mit sich. Beide Aspekte hatten deutliche Auswirkungen auf die Datenvorverarbeitung mit dem Cluster Finder in den HLT C-RORCs. Zum einen musste der Cluster Finder mit einer überproportional erhöhten Taktrate betrieben werden, da die verfügbare Bandbreite mit der neuen Hardware deutlich effektiver genutzt wird. Zum anderen war eine Abwärtskompatibilität mit dem vorherigen Datenformat gewünscht, um weiterhin beliebige Daten zu Testzwecken vom Kartenspeicher abspielen zu können. Beide Aspekte stellten hohe Ansprüche an die Firmwareentwicklung, da die erforderlichen Taktraten zusammen mit der kombinatorischen Komplexität des Algorithmus bereits nahe an die technischen Möglichkeiten des FPGAs kamen. Um diese Funktionalität zu erreichen, wurden einige Teile des Cluster Finders neu implementiert und an vielen Stellen zusätzliche Registerstufen eingezogen. Die korrekte Funktion wurde mit Hilfe der Coprozessor-Implementierung sichergestellt. Im Rahmen der Anpassungen des Cluster Finders an die neue Ausleseelektronik wurde zudem die Daten- und Protokollfehlererkennung verbessert und gefundene Fehler an das HLT Monitoring-System weitergeleitet. Ungewöhnlich hohe Fehlerzahlen einzelner Detektorkanäle während einer Auslese konnten somit unmittelbar an das Detektor-Team weitergegeben und zeitnah behoben werden. Die Software-Referenz und die Hardware-Implementierung ergaben bisher bei fehlerhaften Eingangsdaten unter Umständen unterschiedliche Resultate. Durch Anpassung der Hardware-Implementierung an die Software-Referenz auch für korrupte Eingangsdaten konnte der Abgleich zwischen den Resultaten beider Methoden auf einen direkten binären Vergleich reduziert werden. Diese Firmware-Variante wurde mit der Installation der RCU2 Karten ins Experiment Anfang 2016 im HLT Produktivsystem eingesetzt.

Im Rahmen fortschreitender Datenanalysen wurde vom TPC-Team der Wunsch geäußert, Cluster zu markieren, die auf Grund überlappender Signale vom Algorithmus separiert wurden. Eine erste Implementierung hat gezeigt, dass eine unerwartet hohe Anzahl der Cluster dadurch markiert wurden. Genauere Untersuchungen haben ergeben, dass die betreffenden Cluster zu einem beträchtlichen Anteil rauschinduziert sind. Die in 2016 in der TPC eingesetzte veränderte Gasmischung hat deren relative Anzahl nochmals gegenüber der vorherigen Mischung erhöht. In Zusammenarbeit zwischen den HLT und TPC Gruppen konnte ein verbesserter Algorithmus gefunden werden, der diese rauschinduzierten Cluster unterdrücken kann. Die Hardware-Implementierung dieses Algorithmus wurde im Rahmen dieser Arbeit umgesetzt. Messungen haben gezeigt, dass sich die Anzahl gefundener Cluster dadurch um etwa 21 % (Daten mit der Gasmischung von 2017) bzw. 31 % (Daten mit der Gasmischung von 2016) im Vergleich zur vorherigen Implementierung reduziert. Die Qualität der Physikresultate ist von der geringeren Anzahl Cluster nicht betroffen und liefert teilweise sogar bessere Resultate als zuvor. Die reduzierte Anzahl an Clustern bedeutet direkt eine erhöhte Datenkompression im HLT und damit ein reduziertes Datenvolumen für die dauerhafte Datenspeicherung. Die erhöhte kombinatorische Komplexität in Kombination mit den Anforderungen an die Flusskontrolle haben es erfordert, zentrale Teile des bestehenden Cluster Finders neu zu implementieren. Sowohl der alte, als auch der neue Algorithmus sind damit zur Laufzeit auswählbar und über Parameter konfigurierbar. Um diese Variante innerhalb der von der Taktfrequenz gegebenen Randbedingungen im C-RORC FPGA implementieren zu können, wurde die Unterstützung für das alte Run 1 Datenformat entfernt. Diese Firmwarevariante

war von Mitte 2017 an im HLT aktiv und hat zusammen mit softwareseitigen Anpassungen die Datenkompressionsrate des HLT auf einen Faktor von über 8 erhöht.

Die Bestimmung der Rechenleistung des Hardware Cluster Finders im Vergleich zu einer reinen Softwarelösung benötigt eine definierte Testumgebung. Das zur Verifikation verwendete Emulationsmodell der Hardware ist funktionell identisch, als solches aber nicht auf Durchsatz optimiert. Ein Vergleich damit ist demnach nicht aussagekräftig. Die Referenz-Implementierung in Bezug auf die Physik-Resultate ist der Offline Cluster Finder. Diese Implementierung verwendet jedoch einen anderen Algorithmus, enthält mehr Verarbeitungsschritte als die Hardware-Implementierung und arbeitet auf vollen Ereignisdaten. Der Hardware Cluster Finder ist im Gegensatz dazu auf 216 Instanzen in 36 C-RORCs verteilt, die parallel die Daten von jeweils nur einem der 216 optischen Links pro Instanz verarbeiten. Weiterhin arbeiten im HLT nachfolgend Komponenten, die abschließende Transformationsschritte in Software durchführen. Das Ergebnis dieser gesamten Kette von Hardware Cluster Finder mit nachfolgenden Software-Verarbeitungsschritten ist vergleichbar mit dem des Offline Cluster Finders. Ein Durchsatz-Vergleich von Hardware und Software-Lösung ist auf Knoten-Ebene möglich. Eine C-RORC mit sechs Cluster Finder Instanzen kann volle TPC-Ereignisse bearbeiten. Der Durchsatz dieses Setups kann mit dem maximalen Durchsatz von parallelen Offline Cluster Finder Instanzen auf demselben Knoten verglichen werden. Die Messungen des Hardwaredurchsatzes haben gezeigt, dass die Hardware-Implementierung die maximal vorgesehene Linkrate und damit etwa 1875 MB/s Rohdaten verarbeiten kann. Eine einzelne Instanz der Transformationskomponente verarbeitet etwa 560 MB/s und kann daher mit etwas weniger als 4 Kernen mit der Hardware mithalten. Die maximal parallelisierten Offline Cluster Finder Instanzen mit 48 Einheiten auf den 48 virtuellen CPU-Kernen eines Knoten schaffen zusammen 185,9 MB/s. Die Hardware Implementierung und vier CPU Kerne sind damit etwa einen Faktor 10 schneller als die reine Softwareimplementierung, oder äquivalent zu 480 CPU Kernen. Diese Zahlen beziehen sich auf das Servermodell im HLT und die C-RORC mit einer Eingangdatenrate, die auf die tatsächliche Linkbandbreite reduziert ist. Eine unlimitierte Eingangsdatenrate auf dem FPGA ermöglicht einen höheren mittleren Durchsatz. Neuere Rechner schaffen einen höheren Durchsatz in der Softwareimplementierung, entsprechend sind aber auch mit neueren FPGAs höhere Taktraten möglich.

Datenfluss-Implementierung des Cluster Finders

Die Detektoren und das Experiment wurden über Run 2 hinweg kontinuierlich weiterentwickelt. Jede Iteration bildete wiederum die Grundlage für nachfolgende Evaluationen, zum Teil als notwendige Erkenntnisse zur Planung der nächten Ausbaustufe des Experiments für Run 3. Die Firmware-Entwicklungen, insbesondere am Hardware Cluster Finder, haben auch im Rahmen dieser Arbeit aufgezeigt, wie aufwendig bereits kleinere algorithmische Änderungen umzusetzen sein können. Nicht selten müssen dabei Firmware-Teile von Grund auf neu implementiert und verifiziert werden. Ein Grund hierfür ist, dass die Hardwarebeschreibung in der Regel auf Register-Transfer-Ebene stattfindet und damit die Partitionierung eines Algorithmus in Raum und Zeitrichtung komplett manuell erfolgen muss. Kleine algorithmische Änderungen können daher große strukturelle Auswirkungen auf diese Partitionierung haben. Ansätze zur High-Level Synthese versprechen dieses Problem zu lösen, indem sie die Hardwarebeschreibung auf die algorithmische Ebene heben. Im Rahmen dieser Arbeit wurde hierzu der Datenfluss-Ansatz untersucht. Ein kommerzielles Framework aus FPGA-Karte, einem Compiler sowie Softwarebibliotheken erlaubt es, Algorithmen aus einer Datenflussbeschreibung heraus unkompliziert auf der Hardware zu betreiben. Implementierungsdetails wie der Datentransfer zu und vom FPGA, Speicherreservierung, Flusskontrolle in der Firmware usw. werden funktionsfertig bereitgestellt und deren Interna bleiben vor dem Nutzer verborgen. Die Datenflussbeschreibung des Algorithmus wird vom Compiler in einen Datenfluss-Graphen übersetzt und mit Hilfe einer internen Bibliothek von Firmwarebausteinen auf das FPGA abgebildet. Die Datenfluss-Beschreibung selbst erfolgt in dem hier verwendeten Framework in einer Sprache mit Java-ähnlichem Syntax, die einerseits auf eine in Hardware abbildbare Teilmenge reduziert und andererseits um Aspekte der Steuerung von räumlicher und zeitlicher Parallelität auf der Hardware erweitert wurde. Der Compiler stellt

mit Regeln sicher, dass die somit beschriebene Funktionalität als effiziente Verarbeitungs-Pipeline im FPGA implementiert werden kann. Die Verarbeitungsschritte werden in "Kernels" eingebunden. "Manager" beschreiben die Struktur auf Systemebene und verbinden Kernels untereinander, mit Schnittstellen zum Host-Rechner, zum Speicher auf der Karte oder zu Netzwerkkomponenten. Das Framework bringt eine Simulations- und Entwicklungsumgebung mit, in der sowohl der Datenfluss-, als auch der Host-Code entwickelt, verifiziert und auf der Hardware ausgeführt werden kann. Eine im Rahmen dieser Arbeit besonders interessante Funktionalität dieses Frameworks ist die Möglichkeit, einzelne Kernels zu eigenständigen Netzlisten zu kompilieren. Diese Netzlisten können damit auch außerhalb des Datenfluss-Frameworks in eigenen Hardware- oder Firmware-Umgebungen verwendet werden.

Der Hardware Cluster Finder Algorithmus wurde in dieser Arbeit mit dem Datenfluss-Framework implementiert. Die Struktur dieser Implementierung hat sich dabei im Vergleich zur bereits bestehenden hochoptimierten low-level Implementierung kaum verändert. Der Algorithmus konnte unkompliziert in der Datenfluss-Sprache beschrieben werden. Die Beschreibung von mathematischen Operationen unabhängig von den Datentypen, Latenzen und mit den gängigen Operatoren aus der Softwareprogrammierung vereinfachte und beschleunigte die Entwicklung beträchtlich. Zur Integration der Datenfluss-Implementierung des Cluster Finders in die HLT-Umgebung wurden verschiedene Möglichkeiten untersucht. Eine Ersetzung der C-RORC mit einer reinen Datenfluss-Implementierung der gesamten Ausleselösung war auf Grund von Beschränkungen der Datenfluss-Hardware sowie des Frameworks nicht möglich. Die Auslagerung des Cluster Findings in dedizierte Datenfluss-Hardware wäre nur mit deutlich erhöhten Kosten und Komplexität möglich gewesen. Ein vielversprechender Kandidat war jedoch die Integration des Datenfluss Cluster Finders in die bestehende HLT C-RORC Hardware-, Firmware- und Softwareumgebung über die eigenständigen Netzlisten. Dies konnte exemplarisch gezeigt werden. Die Datenfluss-Implementierung des Cluster Finders konnte über die eigenständige Netzlisten für die Kernels sowie etwas eigener Logik zusammengefasst werden. Dieser Baustein konnte an Stelle des bisherigen VHDL Cluster Finders in die HLT C-RORC Firmware integriert und mit den bestehenden HLT Auslesewerkzeugen erfolgreich getestet werden. Der Durchsatz beider Implementierungen ist konstruktionsbedingt identisch. Auch die maximal möglichen Taktraten sind ähnlich. Die Datenfluss-Implementierung verwendet mit den Standardeinstellungen etwas mehr FPGA-Ressourcen als die händische Implementierung, nutzt aber auch mehr Pipeline-Stufen und hat damit eine etwas höhere Latenz. Durch die Anpassungen von Datenfluss-Optimierungsparametern kann dieselbe Latenz wie bei der händische Implementierung erreicht werden. Diese Anpassung macht auch den Ressourcenverbrauch beider Varianten beinahe identisch. Der Quellecode-Umfang der Datenfluss-Implementierung ist, auf Grund der höheren Abstrahierung, deutlich geringer und damit leichter zu entwickeln, zu erweitern und zu warten.

Ein großer Vorteil des Datenfluss-Ansatzes ist die Möglichkeit Erweiterungen und Änderungen am Algorithmus schnell und einfach zu evaluieren. In dieser Arbeit wurden beispielhaft die Auswirkungen auf den Ressourcenverbrauch durch die Variation der Fixed-Point Genauigkeit sowie das Verschieben von Verarbeitungsschritten zwischen Software und Hardware untersucht. Beides sind essentielle Schritte für die Planung einer Hardware-Implementierung, die mit einer Register-Transfer Beschreibung der Hardware extrem zeitaufwändig sind. Auf der Datenfluss- / Algorithmusebene waren diese Evaluationen in kurzer Zeit möglich.

Ergebnisse, Schlüsse und Ausblick

Die Ziele der Arbeit waren die Entwicklung einer gemeinsamen Ausleselösung für die ALICE DAQ und HLT System in Run 2, die Integration und Betreuung dieser in der HLT Anwendung sowie die Untersuchung von Ansätzen zur Hardwarebeschreibung auf algorithmischer Ebene. Diese Ziele wurden erreicht. Auf Grund fehlender kommerzieller Lösungen wurde eine eigene Hardwareplattform entwickelt. Es konnte dabei eine Hardwarekonfiguration gefunden werden, die neben den eigentlichen Zielanwendungen in den ALICE DAQ und HLT Systemen außerdem im ATLAS Experiment, sowie zahlreichen weiteren Testsystemen und Experimenten ihre Anwendung gefunden

hat. Weiterhin ist die Karte teilweise für den Einsatz in Run 3 vorgesehen. Die Karte wurde im Rahmen dieser Arbeit über den gesamten Produktlebenszyklus hinweg von der Konzeption an über Entwicklung, Koordination, Produktion, Test, Firmware- und Softwareentwicklung sowie Integration und Produktivbetrieb im HLT betreut. Sie war rechtzeitig für den Betrieb in den Run 2 Produktivsystemen verfügbar, erfüllt alle Anforderungen und hat bisher keine technischen Ausfälle zu verzeichnen. Trotzdem hat sich auch bei diesem Projekt gezeigt, dass eigene Hardwareentwicklungen, auch ohne technische Schwierigkeiten, insbesondere in kleinen Teams sehr zeitaufwändig sind. Die nächte Generation FPGAs, die den Firmware-Entwicklungsaufwand deutlich reduziert hätte, war auf dem Markt bevor die ersten eigenen Prototypen hergestellt waren. Die Arbeitszeit zu Hardwareentwicklung, -Produktion und -Test hat zudem einen beachtlichen Teil des Projektes ausgemacht, die bei einer kommerziellen Lösung deutlich geringer ins Gewicht gefallen wäre. Die Anforderungen und Rahmenbedingungen waren in diesem Fall jedoch nicht mit kommerzieller Hardware erfüllbar.

Die Firmware und Software der HLT Anwendung wurde für Run 2 vorbereitet und über Run 2 hinweg kontinuierlich weiterentwickelt. Hierfür mussten große Teile von Grund auf neu entwickelt werden. Ein zentraler Aspekt in der HLT Anwendung ist die Datenvorverarbeitung mit dem Hardware Cluster Finder im Auslese-FPGA. Dieser Algorithmus konnte auf die neue Hardware portiert werden, wurde im Rahmen des TPC-Elektronikupdates auf den doppelten Durchsatz optimiert und hat mit einem verbesserten Algorithmus zu einer deutlich erhöhten HLT Datenkompressionsrate bei teilweise sogar verbesserten Physik-Parametern beigetragen. Im Vergleich zu äquivalenten Verarbeitungsschritten in Software, ist eine HLT FPGA-Auslesekarte mit Hardware Cluster Finder etwa einen Faktor 10 schneller als eine parallelisierte Softwareverarbeitung auf einem HLT Knoten.

Die Firmware-Entwicklungen um den Hardware Cluster Finder waren durchweg erfolgreich und mit sehr positiven Resultaten für das Gesamtexperiment verbunden, jedoch auch sehr aufwändig. Änderungen an der Auslesearchitektur sowie dem Algorithmus hatten die Neuentwicklung mehrerer zentraler Komponenten mit hohem Verifikationsaufwand erfordert. Die Beschreibung des Algorithmus mit dem Datenfluss-Ansatz hat sich hierbei als effiziente, kompakte und leicht wartbare Alternative gezeigt. Sowohl der Ressourcenverbrauch, als auch der Durchsatz waren vergleichbar mit einer händischen Implementierung. Beides konnte mit einem Bruchteil der Entwicklungszeit und des Code-Umfangs erreicht werden. Evaluationen von Verarbeitungsschritten in Hardware waren damit sehr einfach möglich. Jedoch war die hier untersuchte Lösung sehr stark an ein Anbieter-Framework mit fester Hardware und Software gebunden. Mit diesem lies sich die Auslesearchitektur um den Cluster Finder nur schwer abbilden. Eine Extraktion der Datenfluss-Beschreibung in eine eigenes Firmware- und Softwareumgebung konnte erfolgreich gezeigt werden, ist jedoch Herstellerseitig so nicht vorgesehen und ist damit in einer Produktivumgebung schlecht wartbar. Insgesamt zeigte sich damit jedoch auch ein generelles Problem von High-Level Synthese Werkzeugen: Eigenständige HLS-Lösungen sind oft auf einen Satz fester Hardwareplattformen ausgelegt, sodass eine Unterstützung dieser, sowie eigener Hardware, stark vom HLS-Anbieter abhängt. HLS-Lösungen der FPGA-Hersteller bieten in der Regel eine bessere Unterstützung von eigener Hardware. Ein gemeinsamer Standard hat sich hierzu jedoch bisher nicht entwickelt, sodass diese HLS-Entwicklungen nur selten zwischen FPGAs verschiedener Hersteller übertragen werden können. Insgesamt vereinfacht eine algorithmische Hardwarebeschreibung den Entwicklungsaufwand beträchtlich, sie reduziert den Umfang der Beschreibung und erleichtert die Wartung und Weiterentwicklung. Dies konnte im Rahmen der Arbeit auch am Beispiel des Hardware Cluster Finders gezeigt werden. Die Möglichkeit eine solche Beschreibung dann auch einfach in eine eigene Hardware-, Firmware- und Softwareumgebung zu integrieren kann die Firmwareentwicklung für Datenverarbeitungsschritte in der Hochenergiephysik deutlich vereinfachen.