Data Acquisition System for the COMPASS++/ AMBER Experiment

Vladimir Frolov, Stefan Huber, Igor Konorov, Antonin Kveton, Dmytro Levit, Josef Novy, Dominik Steffen, Benjamin Moritz Veit, Miroslav Virius, Martin Zemko, and Stephan Paul

***Abstract*— We present a new data acquisition system for the COMPASS++/AMBER experiment designed as a further development of the intelligent FPGA-based data acquisition framework. The system is designed to have a maximum throughput of 5 GB/s. We designed the system to provide free-running continuous readout, which allows us to implement a sophisticated data filtering by delaying the decision until the hardware filter and high-level trigger stage which processes data. The system includes front-end cards, fully digital hardware filter, data multiplexers, a timeslice builder, and a high-level trigger farm. The data selection and data assembly require a time structure of the data streams with different granularity for different detectors. We define a unit of detector data as image and combine images from different detectors within a time window to timeslices. By routing data based on the timeslices, we can average data rates and easily achieve scalability. The main component that allows us to achieve these goals is a high-performance and cost-effective hardware timeslice builder. The timeslice builder combines streaming data by their time and consists of the data switch and the spillbuffer build. The scalable architecture allows us to increase the throughput of the system and achieve a true triggerless mode of operation.**

***Index Terms*— Data acquisition, data handling, high-energy physics instrumentation computing.**

I. Introduction

THIS article presents a fully FPGA-based data acquisition system for the upcoming experiment COMPASS++/ AMBER[1] at CERN [1]. The experiment is dedicated to study strong interaction effects at a wide range of four-momentum transfer. This experiment aims to measure the proton charge radius by elastic muon–proton scattering at low four-momentum transfer, the spectroscopy of mesons and baryons at intermediate four-momentum transfer, and to study the structure of mesons and baryons at high four-momentum transfer.

Section II compares our system with the upcoming free-running systems of other experiments. Section III explains the free-running concepts followed by the description of COMPASS++/AMBER data acquisition system and hardware. Section V presents the performance measurements of the system on a test setup. Finally, Section VI describes the future extension of the system.

II. State-of-the-Art Systems

In the last decade due to progress in the microelectronics, computing, and FPGA technology, it became possible to read and process detector data without classical data reduction scheme using a level-1 trigger. This stimulates the development of the free-running concepts for data acquisition in the high-energy physics experiments to allow us to implement more efficient data selection. Several experiments, for example, LHCb[2] [2], sPHENIX[3] [3], CBM[4] [4], DUNE[5] [5], develop free-running systems. The common approach of these systems is to get data into a server and merge them in software. The systems require at least two layers of servers and network infrastructure.

We introduce the fully FPGA-based data merger, which we call the timeslice builder, in our data acquisition system. The timeslice builder processes data streams in pipeline entirely in FPGA fabric which results in a high-performing and a compact design. The compactness of the system also translates into lower hardware costs.

Another advantage of the fully hardware-based design is the high reliability and fast recovery after a crash. The FPGA-based system requires only a fast reset signal to recover in contrast to the software-based systems which have to reload and reinitialize the full software stack. The reliability translates into the uptime of the system: the data acquisition systems

Manuscript received October 30, 2020; revised January 13, 2021; accepted May 23, 2021. Date of publication June 30, 2021; date of current version August 16, 2021. This work was supported by the German Ministry of Education and Research and Excellence Cluster Origins.

Vladimir Frolov is with JINR, 141980 Dubna, Russia, and also with CERN, 1211 Geneva, Switzerland.

Stefan Huber, Igor Konorov, Dmytro Levit, Dominik Steffen, and Stephan Paul are with the Physikdepartment E18, Technische Universität München, 85748 Garching, Germany (e-mail: dmytro.levit@tum.de).

Antonin Kveton is with the Department of Low Temperature Physics, Charles University, 180 00 Prague, Czech Republic.

Josef Novy and Miroslav Virius are with the Department of Software Engineering, Czech Technical University, 120 01 Prague, Czech Republic.

Benjamin Moritz Veit is with Johannes Gutenberg-Universität Mainz, 55099 Mainz, Germany, and also with CERN, 1211 Geneva, Switzerland.

Martin Zemko is with Czech Technical University, 120 01 Prague, Czech Republic, and also with CERN, 1211 Geneva, Switzerland.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNS.2021.3093701.

Digital Object Identifier 10.1109/TNS.2021.3093701

[1]COmmon Muon Proton Apparatus for Structure and Spectroscopy/ Apparatus for Meson and Baryon Experimental Research.

[2]The Large Hadron Collider beauty experiment at CERN.

[3]Pioneering High Energy Nuclear Interaction eXperiment with emphasis on strongly interacting particles at the Relativistic Heavy Ion Collider.

[4]Compressed Baryonic Matter experiment at the Facility for Antiproton and Ion Research.

[5]Deep Underground Neutrino Experiment at the Sanford Underground Research Facility.

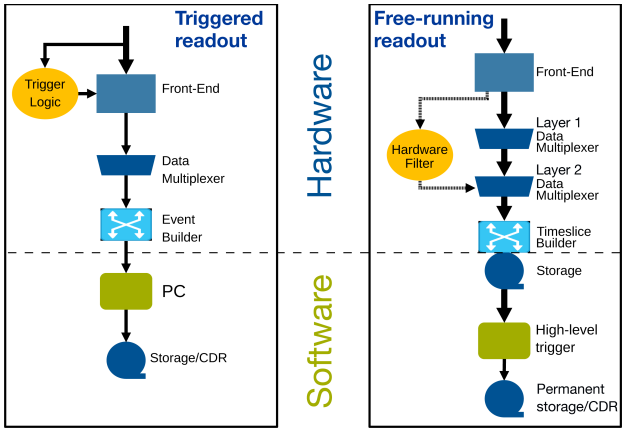

Fig. 1. Comparison between the triggered and the free-running readout.

with software-based event builder showed an uptime of 91.7% (ATLAS 2016 [6]) and 95.87% (CMS 2018 [7]), while the COMPASS data acquisition system with fully FPGA-based event builder showed an uptime of 99.6% [8].

III. Continuous Read-Out Concept

We designed the data acquisition system in the intelligent FPGA-based data acquisition framework, iFDAQ [9], for continuous triggerless readout to fulfill the wide range of physics cases covered by the experiment. The word "intelligent" in the name of the framework describes the ability of the system to self-recover from the loss of synchronization caused by missing or corrupted data. The system delivers a continuous data stream with timing information that can be processed for feature extraction and high-level triggering in real time.

Fig. 1 shows the evolution of the COMPASS iFDAQ architecture from the triggered to the free-running system. The system with the triggered readout generates triggers using data from a subset of the detectors. The front-end electronics buffers data while waiting for the trigger decision. The trigger logic must have a latency smaller than 2 μs because the buffers in the front-end electronics are usually small [10]. This requirement limits the complexity of the trigger decision. The free-running system reads and buffers the data in the much larger buffers of the data multiplexers for the entire accelerator cycle for timeslice building. After the timeslice building the system buffers data on hard disks for data reduction in the high-level trigger.

While we designed the system to be able to deliver all data to the high-level trigger for final data reduction, we also foresee an optional hardware filter for additional data reduction. The hardware filter receives the same data as the data acquisition and has significantly more time to meet the filter decision than in the conventional triggered scheme. Therefore, the hardware filter can incorporate more sophisticated algorithms which will improve the quality of the filter decision.

We aim to operate the system in the free-running mode. To merge data from different detectors that are generated by the same physical event, we match data by the time when data are generated in the detector. Therefore, we use a readout mode based on timeslices with a typical period in the order of hundreds of microseconds to a few milliseconds. Fig. 2 shows the time structure of the data streams in the iFDAQ framework. A timeslice is a basic time period used to combine data from different sources. The time control system, TCS, generates the timeslice information centrally and distributes it to the whole experiment. Because every subdetector has different time precision, we subdivide the timeslice into finer detector images. The width of an image is different for every detector and reflects the detector's timing precision.

We base the data reduction in the free-running system on the selection of images. The images contain a timestamp which we use to associate images for data reduction. We keep only those images for which there is a matching trigger decision.

We can also apply a hardware filter for the free-running readout without changing the data structure. In this mode, we keep the images which are selected by the hardware filter or read all images without data reduction but store the filter decision in the second case. We use the filter decision for final data reduction in the high-level trigger.

The time structure brings other advantages as well. We use an FPGA-based switch together with an FPGA-based spill-buffer card as a timeslice builder which combines timeslice data from the detectors. Fig. 3 shows the functionality of the switch. Before data processing, data are stored in the large external memory of the data multiplexers. The switch steers data flow, receives data, and routes the timeslice data from all detectors to the same spillbuffer. Long timeslices averages data rates on all input streams of the timeslice builder and increases efficiency of the timeslice builder algorithm. The algorithm consists of the 4×4 switching units arranged in two layers. The switching units change their configuration so that the switch can change the routing of the input streams to the output streams. We can extend the topology of the switch by adding switching units or even FPGA cards to scale the system. The performance of the switch depends only on the throughput of the data links because the algorithm is implemented using only FPGA fabric resources.

IV. COMPASS++/AMBER Data Acquisition System

We combine the previously described components to build the data acquisition system of the COMPASS++/AMBER experiment. The experiment contains approximately 300 000 data channels without silicon pixel detector and approximately 100 million data channels of the pixel detector. The expected sustained data rate of the spectrometer without data reduction is 5 GB/s.

Fig. 4 shows the layout of the data acquisition system. For example, the experimental setup for the proton radius measurement consists of an active-target time projection chamber, a silicon tracker based on the ALICE pixel detector (ALPIDE) [11] sensors, large-area gas electron multiplier (GEM) detectors, scintillation fiber tracking stations, beam momentum stations, hodoscopes, and electromagnetic calorimeters.

The experiment follows the spill cycle of the super proton synchrotron accelerator [12]. During the so-called "ON-spill" period that lasts 4.8 s, the beam hits the target, and the detector generates up to 72 GB of data during proton radius measurement. The "ON-spill" period is followed by a longer

Fig. 2. Example of the time structure with the timeslice length of 100 μs in the iFDAQ framework.

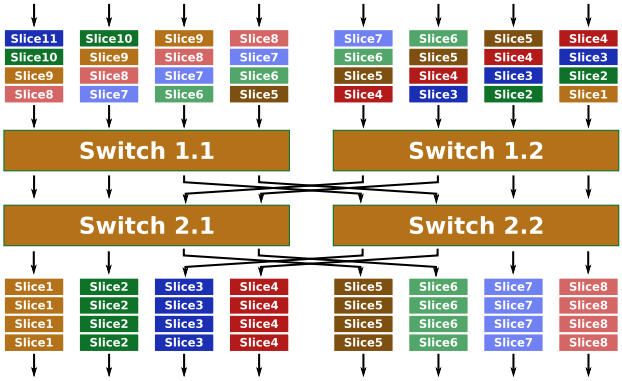

Fig. 3. Data processing in the 8 × 8 switch. Input data are stored in the memory of the data multiplexers.

"OFF-spill" period when there is no beam. The data rate during the "ON-spill" period is 15 GB/s which is higher than the system's throughput of 5 GB. Therefore, we use the "OFF-spill" period to finish processing data generated during the "ON-spill" period. The average data size of data for a timeslice with the width of 1 ms is about 14.4 MB.

The multiplexers combine raw data from different sources, buffer them in the memory, and send them over a single link, thus reducing the number of the links in the timeslice builder and the trigger processor. The switch prepares data for timeslice building by routing data of the corresponding timeslices from multiplexers to the same spillbuffer input. The trigger processor searches for events in data streams and sends the decision to the TCS controller. The spillbuffer combines data that belong to the same timeslice and sends data to the read-out server over PCIe interface. The read-out servers store data on hard disks for the high-level trigger farm, thus decoupling the read-out system and the high-level trigger. The high-level trigger farm reads data over 25-Gb/s Ethernet for data reduction and sends reduced data stream to the CERN tape archive for permanent storage.

The DAQ[6] uses a Virtex-6 iFDAQ DHMx card [9] with 17 high-speed serial links operated at 650 MB/s as multiplexers and as a switch. These cards provide 4 GB of external DDR3[7] memory which is used to buffer data in multiplexers and timing information in the switch. The 8 × 8 switch implemented in this card has a maximum throughput of 5 GB/s which allows us to process all data during the "ON-spill" period. Due to the way the timeslice builder processes data, as shown in Fig. 3, the multiplexers must buffer data for up to eight timeslices. If we use the 8 × 8 switch configuration and the full timeslice size of 14.4 MB for a 1-ms-long timeslice as an upper limit, this would require 115.2 MB of memory in the multiplexers.

We use a commercially available Nereid Kintex 7 XC7K160T PCIe board [13] with four lanes PCIe gen. 2 in the read-out server as a spillbuffer. The theoretical bandwidth of the PCIe interface is 2 GB/s, but due to the limit of the maximum payload size of 128 B in the DMA engine[8] of the host system, the achievable throughput of the system is 1.6 GB/s due to protocol overhead [14]. The throughput of the PCIe board fits well with the throughput of the attached RAID0[9] storage of 1.5 GB/s built with the RAID controller AOC-SAS3-9380-8E.

Section IV-A describes the implementation of the new DAQ algorithms.

A. 8 × 8 Switch

Fig. 5 shows the layout of the switch firmware. The switch contains a link to the TCS, eight incoming and eight outgoing links, a data consistency checker on every input stream, and an 8 × 8 switch matrix. The TCS link receives the timing information which is used in the switch matrix to determine the destination of the timeslice data. The incoming and the outgoing links use the 8-/10-b Aurora [15] protocol with native flow control for signaling backpressure. The outgoing links also provision a user flow control interface to distribute the timing information from the TCS and the current switch configuration to the spillbuffers.

We store the timing information in a deep FIFO[10] implemented using the external DDR3 memory to compensate

[6]Data acquisition.

[7]Double data rate.

[8]Direct memory access engine.

[9]Redundant Array of Inexpensive Disks level 0.

[10]First-in first-out.

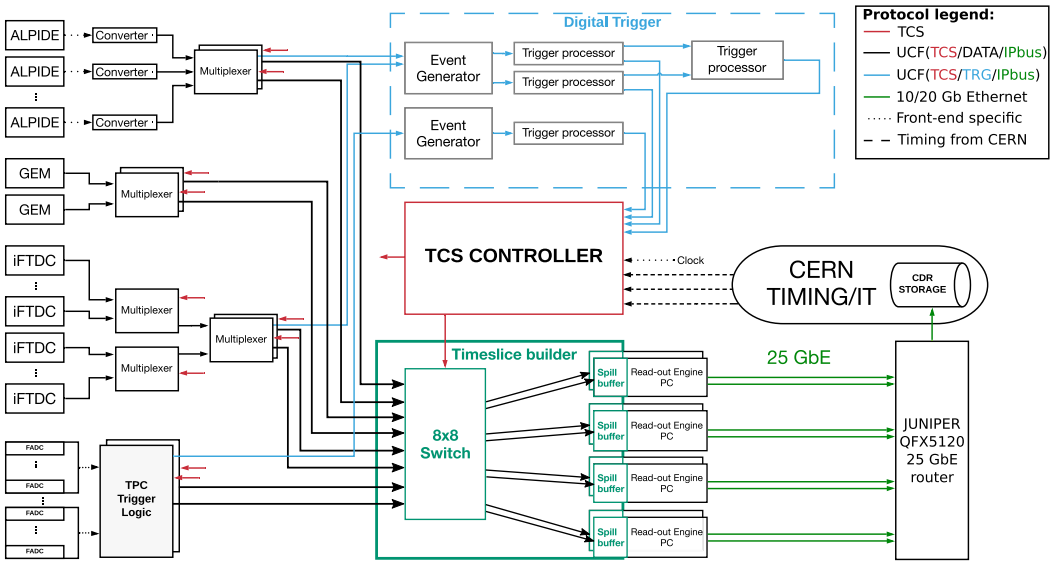

Fig. 4. Data acquisition system of the COMPASS++/AMBER experiment.

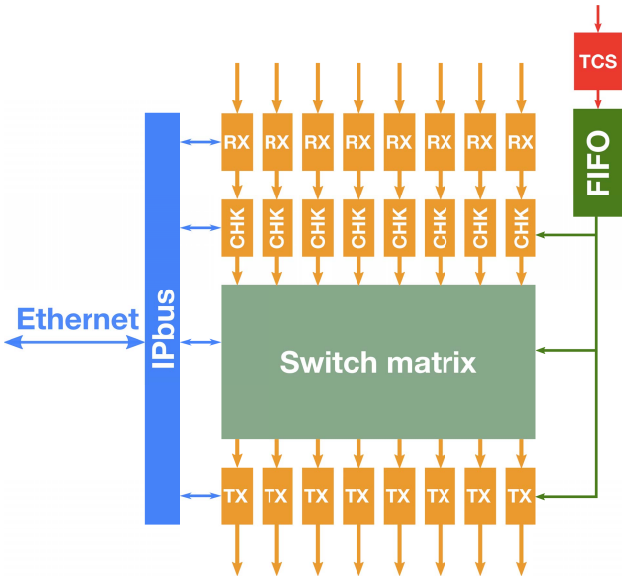

Fig. 5. Switch firmware layout.

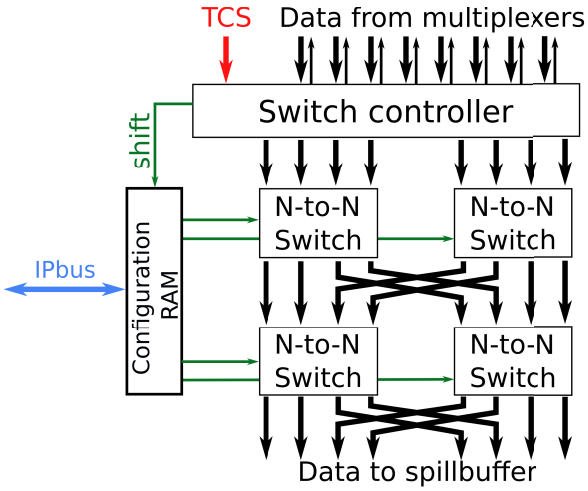

Fig. 6. 8×8 switching matrix.

for the data latency in the previous DAQ stages. The FIFO distributes timing information to the data consistency checkers, the switch matrix, and sends it to the spillbuffer over the user flow control interface.

The data consistency checkers use the timing information to check the format of the incoming data for consistency. If the data format is corrupt, the checker restores correct data format and marks the frame as broken.

The most important role of the checker is the restoration of data synchronization which is important for the event building process. If the checker detects inconsistencies between data and timing information, these inconsistencies are resolved either by inserting dummy data or discarding data from unexpected timeslices. This is done to assure uninterrupted data flow even in the case of lost links or corrupted data transmission which can happen especially during detector commissioning when detectors are included or excluded while the system is running.

The switch matrix receives eight perfectly synchronized data streams and may route them to any of the output streams. The routing is limited to the barrel-shifter configurations. Fig. 6 shows the layout of the 8×8 switch matrix. The matrix consists of the switch controller, the configuration RAM, and four 4×4 reconfigurable switching blocks which are arranged in two layers.

The switch controller monitors and controls data flow on all links. The controller is also responsible for changing the configuration of the switching blocks. When all data in one timeslice in a stream pass the switch, the controller blocks data flow on this stream until all streams finish sending data of their respective timeslices.

In parallel to controlling the data flow, the switch reads timing information from the FIFO and determines the next

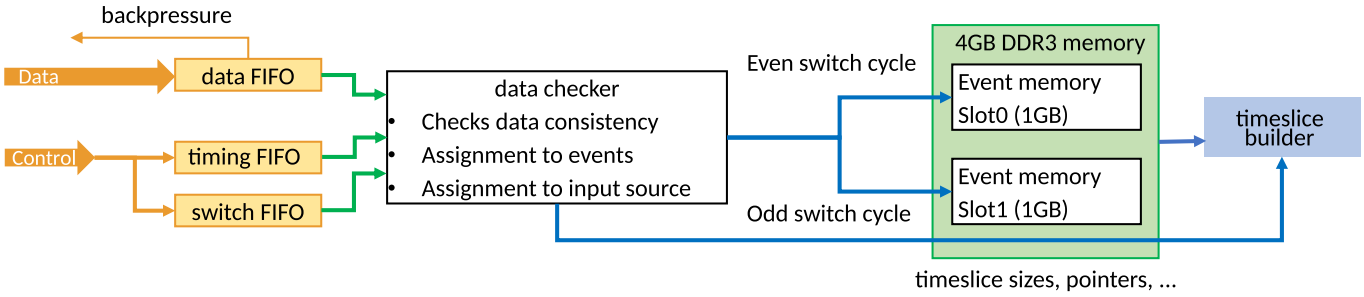

Fig. 7. Layout of the data processing pipeline in the spillbuffer card.

destination. Once all streams complete to process data of their timeslices, the switch changes topology of the switching block by selecting the next topology in the configuration RAM. After the switching blocks have applied the new topology, the controller activates data flow on all incoming streams.

The configuration RAM is implemented in the block RAM of FPGA [16]. It holds eight routing tables for all switching blocks. The firmware initializes the memory at the booting time for the full 8×8 operation, but we can change the routing table for other configurations later over the IPbus protocol [17] over Ethernet.

The switching blocks perform the actual data routing. We interconnect the switching units as shown in Fig. 6. We connect output streams of a switching unit to the inputs of the next layer unit or the output streams of the switch. Two output streams are connected to the unit or the switch streams below the unit, and two streams are connected to the diagonal units or switch streams. This interconnection guarantees that we connect any input stream to any output stream in eight distinct configurations.

To sort data by timeslices as shown in Fig. 3, we have to skew data on the input to the switch. This is done with the following algorithm. At the start of the run, the switch deactivates data flow on all input channels. The switch transmits data of the first timeslice from the input link 1 to the output link 1. When the transmission of the data of timeslice 1 is completed, the switch changes configuration and activates the input link 2. The switch transmits data of the first timeslice from the input link 2 to the output link 1. The input link 1 transmits data of the timeslice 2 to the output link 2. Switch continues to activate the input links one by one and changes the topology every timeslice transmission cycle until all links are activated. The read-out servers receive data sorted by timeslices as the result.

We configure, control, and monitor the switch as well as other DAQ modules using IPbus protocol.

B. Spillbuffer Card

Fig. 7 shows the layout of the data processing pipeline of the spillbuffer card. Each node contains up to two pipelines.

The pipeline starts with an Aurora link which receives data, timing information, and switch configuration and sends backpressure to the switch when the pipeline is busy. The data checker checks data for consistency with the timing information and switch configuration, recovers data format and synchronization in the case of corrupted data, assigns data to events, and writes data to one of the two slots assigned to this pipeline in the external memory. Data of the same timeslice is stored in one of the two memory slots. Once the processing of the timeslice data is complete, the data checker passes information such as slot ID, size of all data chunks which belong to the same timeslice, and pointers to individual timeslice data chunks from different multiplexers to the event builder. The timeslice builder reads data from memory slots allocated to different pipelines, combines these data, and sends them over the PCIe interface to the read-out server.

V. System Performance

To measure the system performance, we built the prototype of the system which consists of two DHMx cards and a Nereid Kintex-7 PCIe card installed in a server with the Intel Xeon CPU E5405 at 2.00 GHz. Fig. 8 shows the photograph of the test setup and Fig. 9 shows the layout of the test setup.

We configured one of the DHMx cards as a data generator. The data generator can generate events with fixed event size or with a randomly distributed event size based on a real run of the COMPASS experiment as shown in Fig. 10. In the latter case, the total event size is divided by 8 to account for eight links, and a fluctuation is randomly added or subtracted. The fluctuation is the square root of the total event size divided by eight.

The data generator and the switch receive triggers from the TCS, the data generator generates events in the COMPASS data format, and sends them to the switch over eight independent 312.5-MB/s Aurora links. The second DHMx card is configured as a switch. The switch receives data from the generator, sorts data by timeslices, and sends sorted data to the spillbuffer card over two 625-MB/s Aurora links. The spillbuffer card receives data and timing information on two input links and sends data from one link to the readout server.

We generate triggers with a spill cycle which emulates the spill structure of the Super Proton Synchrotron at CERN. The first phase of the cycle is the ON-spill state, which lasts for 5 s. During this state, triggers are distributed to the generator and the switch. The second phase is the OFF-spill state which lasts for 15 s. During this state, no triggers are distributed.

Fig. 11 shows data rate measured at the input to the spillbuffer during the 12-h-long test of the system under full load. The system shows consistent data flow and stable data rate of 290 MB/s. The regular drops of the data rate are

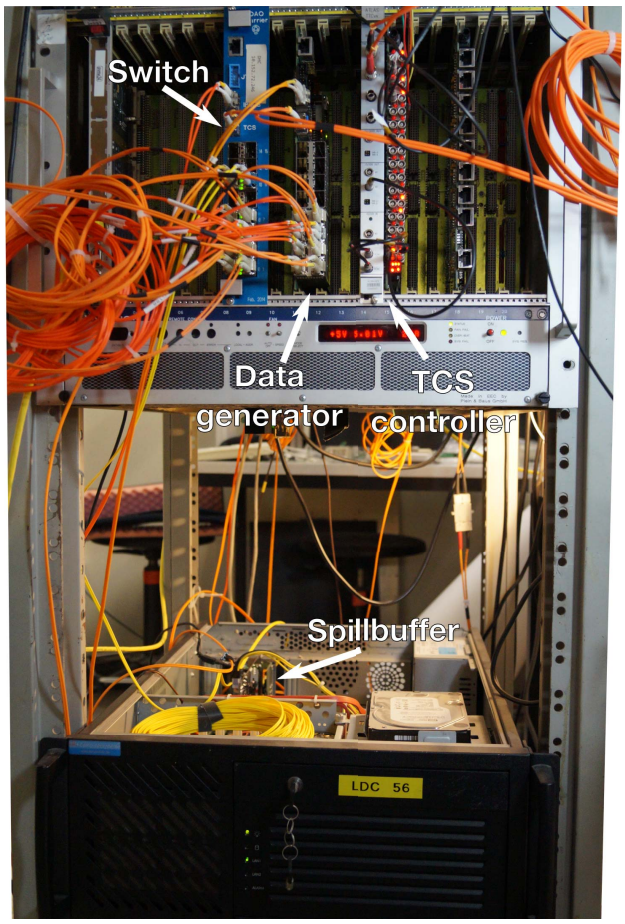

Fig. 8. Test setup.

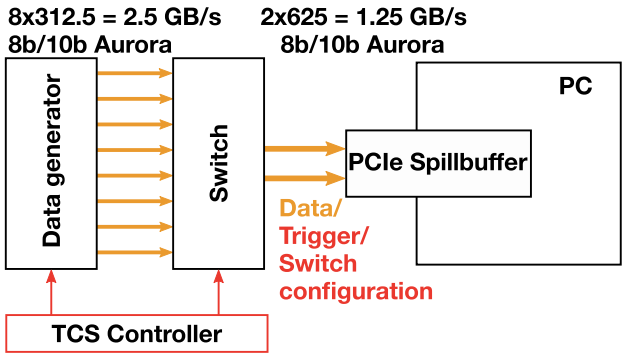

Fig. 9. Layout of the test setup.

due to the spill cycle. The measured data rate is 10% below the bandwidth limit of the input links. This reduction of the maximum throughput is the result of the variation of the event sizes. Following paragraphs describe this effect in more detail.

Fig. 12 shows the timeslice processing time distribution recorded with the test system for the timeslice width of 1 ms, realistic event size distribution, and the trigger rate 40 kHz distributed after the Poisson distribution. The distribution shows that in most of the cases, the switch is capable to process data of the timeslice for a single link within the timeslice period. The mean value of the distribution corresponds to the time

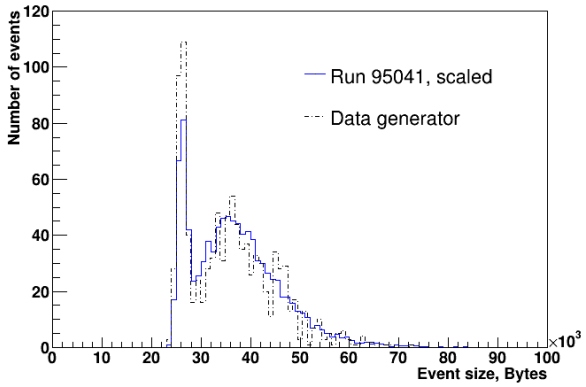

Fig. 10. Event size distribution used during the tests.

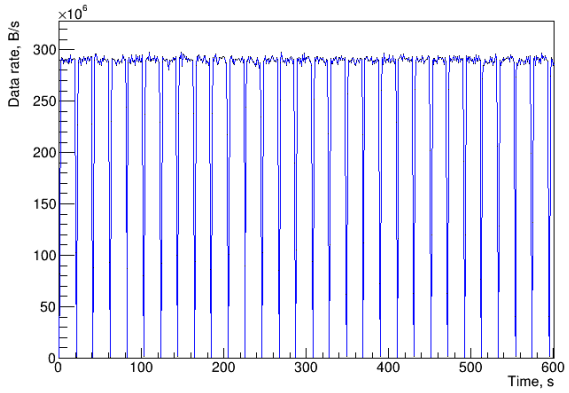

Fig. 11. Output data rate of the switch for timeslice width of 1 ms, realistic event size, and trigger rate 40 kHz distributed after the Poisson distribution.

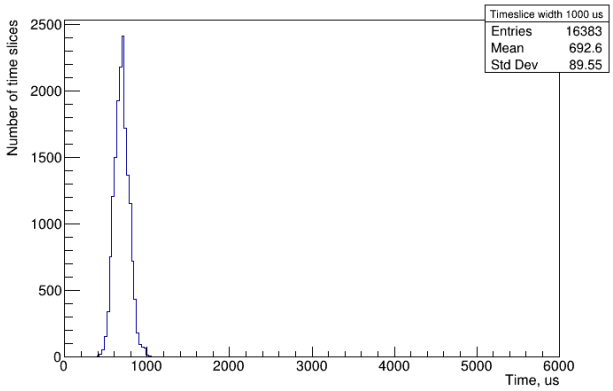

Fig. 12. Timeslice processing time distribution for timeslice width of 1 ms, realistic event size, and trigger rate 40 kHz distributed after the Poisson distribution.

needed to transmit data to the switch

$$T_{\text{proc}} = \frac{\bar{S}_{\text{event}} \cdot \bar{N}_{\text{triggers}}}{\text{MIN}(B_{\text{input}}, B_{\text{output}})} = \frac{\bar{S}_{\text{event}} \cdot R}{\text{MIN}(B_{\text{input}}, B_{\text{output}})} \cdot T_{\text{timeslice}}$$

where $\bar{S}_{\text{event}}$ is the average event size, $\bar{N}_{\text{triggers}}$ is the average number of triggers within the timeslice, R is the trigger rate, B_{input} and B_{output} are the bandwidths of the input and output links, and $T_{\text{timeslice}}$ is width of the timeslice. With the

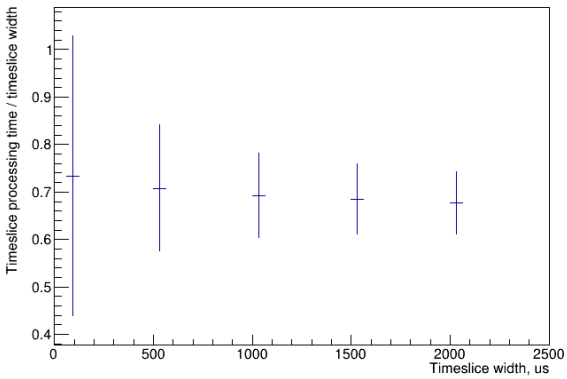

Fig. 13. Relative processing time for different timeslice widths.

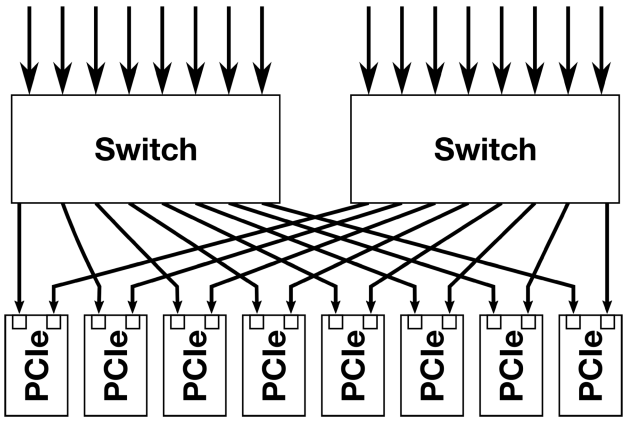

Fig. 14. Extension of the system by adding the second switch.

parameters of the test system, we obtain

$$T_{\text{proc}} = \frac{(40\ \text{kB/8 links}) \cdot 40\ \text{kHz}}{320\ \text{MB/s}} \cdot 10^{-3}\ \text{s} = 0.625\ \text{ms}.$$

This measurement shows that the performance of the system is limited by the bandwidth of the links. If the data transmission time is greater than the timeslice width, the system will have to "borrow" time from the next timeslice. In this case, one has to allocate regular trigger-free time for the system to compensate for the borrowed time. In COMPASS++/AMBER, we use the DDR3 memory in the multiplexers to buffer data and process them during the OFF-spill periods to compensate for the borrowed time.

The timeslice processing time distribution also shows the averaging effect of the timeslice-based readout. The switch averages event sizes of all events in the timeslice which gives us another metric for prediction of the necessary bandwidth.

Fig. 13 shows the measurement of the relative timeslice processing time T_{rel}:

$$T_{\text{rel}} = \frac{T_{\text{proc}}}{T_{\text{timeslice}}}$$

where T_{proc} is the timeslice processing time and $T_{\text{timeslice}}$ is the timeslice width, for different timeslice widths. The measurement shows that with increasing timeslice width, the relative processing time asymptotically approaches the bandwidth limit and its variation decreases. The effect of decreasing variation with increasing timeslice width is caused by averaging over more data.

VI. Future System Extension

The current system consists of one switch and eight spillbuffers and has a maximum throughput of 5 GB/s if we increase the bandwidth of the input links to the switch to 625 MB/s. But with the bandwidth of a switch-spillbuffer link of 625 MB/s, we are using only 40% of the bandwidth of the PCIe interface of 1.6 GB/s. We can use this resource to scale the system for higher throughput.

Fig. 14 shows the possible low-cost extension of the system which will double the system's throughput to 10 GB/s. We add a second switch to the system which sends data to the existing spillbuffers.

We can compare the cost of the current system with the upgraded system. The 5-GB/s system needs nine DHMx cards (eight as multiplexers and one as a switch) at 1200 € and eight spillbuffer cards at 1500 €. The 10-GB/s system needs 17 DHMx (16 as multiplexers and 1 as a switch) cards and eight spillbuffer cards. Therefore, the cost of the system per GB/s, C, decreases with increasing bandwidth as the following calculation shows:

$$C_{5\,\text{GB/s}} = \frac{9 \cdot 1200\ € + 8 \cdot 1500\ €}{5} = 4560\ €/\text{GB/s}$$

$$C_{10\,\text{GB/s}} = \frac{17 \cdot 1200\ € + 8 \cdot 1500\ €}{10} = 3240\ €/\text{GB/s}.$$

To integrate this change, we must modify the algorithms of the existing system. Because both switches must send corresponding timeslice data to the same spillbuffer, the destination selection must be deterministic and synchronous. The spillbuffers will be extended with a second data pipeline. Therefore, we have to change the timeslice builder algorithm to merge data from two data sources.

The upgrade is technically feasible and requires only a small investment compared to the system cost.

VII. Summary

We designed the free-running data acquisition system for the COMPASS++/AMBER experiment. The system uses a novel fully FPGA-based timeslice builder. The timeslice builder combines data from eight data streams by time and distributes full timeslices for the following data reduction in the high-level trigger.

We measured and characterized the performance of the test system that consists of the data generator, and the reduced timeslice builder with only two output streams instead of eight in the full system. The test showed that the throughput of the system is limited by the throughput of the input–output links of the timeslice builder and the fluctuation of the data size of individual timeslices. We also demonstrated that we can reduce the fluctuation of the timeslice data sizes by extending the timeslice period.

We presented a possible future extension of the system that will double the throughput of the system for a tiny fraction of

the system's cost as an example of the flexibility of this DAQ architecture.

We plan to further test the system during the upcoming dry run with the real detector data at COMPASS++/AMBER.

REFERENCES

[1] B. Adams *et al.*, "COMPASS++/AMBER: Proposal for measurements at the M2 beam line of the CERN SPS phase-1: 2022–2024," CERN, Geneva, Switzerland, Tech. Rep. CERN-SPSC-2019-022, SPSC-P-360, May 2019.

[2] R. Aaij *et al.*, "Allen: A high level trigger on GPUs for LHCb. Allen: A high level trigger on GPUs for LHCb," *Comput. Softw. Big Sci.*, vol. 4, p. 12, Dec. 2019. [Online]. Available: http://cds.cern.ch/record/2704717

[3] A. Adare *et al.*, "An upgrade proposal from the PHENIX collaboration," Brookhaven Nat. Lab, Upton, NY, USA, Tech. Rep., Jan. 2015. [Online]. Available: https://arxiv.org/abs/1501.06197

[4] P. Staszel, "CBM experiment at FAIR," *Acta Physica Polonica B*, vol. 41, no. 2, pp. 341–350, 2010.

[5] R. Acciarri *et al.*, "Long-baseline neutrino facility (LBNF) and deep underground neutrino experiment (DUNE) conceptual design report, volume 4 The DUNE detectors at LBNF," Jan. 2016, *arXiv:1601.02984*. [Online]. Available: https://arxiv.org/abs/1601.02984

[6] W. P. Vazquez and A. Collaboration, "The ATLAS data acquisition system in LHC run 2," CERN, Geneva, Switzerland, Tech. Rep. ATL-DAQ-PROC-2017-007, Feb. 2017. [Online]. Available: https://cds.cern.ch/record/2244345

[7] G. Badaro, "Daqexpert the service to increase CMS data-taking efficiency," *EPJ Web Conf.*, vol. 245, p. 01028, Nov. 2020. [Online]. Available: https://doi.org/10.1051/epjconf/202024501028

[8] S. Huber *et al.*, "Intelligence elements and performance of the FPGA-based DAQ of the COMPASS experiment," *PoS*, vol. TWEPP-17, p. 127, Mar. 2018. [Online]. Available: http://cds.cern.ch/record/2314733

[9] Y. Bai *et al.*, "Overview and future developments of the FPGA-based DAQ of COMPASS," *J. Instrum.*, vol. 11, no. 2, p. C02025, Feb. 2016.

[10] H. Fischer *et al.*, "Implementation of the dead-time free F1 TDC in the COMPASS detector readout," *Nucl. Instrum. Methods Phys. Res. A, Accel. Spectrom. Detect. Assoc. Equip.*, vol. 461, pp. 507–510, 2001.[Online]. Available: https://cds.cern.ch/record/472556

[11] M. Šuljič, "ALPIDE: The monolithic active pixel sensor for the ALICE ITS upgrade," *J. Instrum.*, vol. 11, no. 11, p. C11025, Nov. 2016.

[12] F. Velotti *et al.*, "Characterisation of SPS slow extraction spill quality degradation," in *Proc. 10th Int. Part. Accel. Conf. (IPAC)*, Melbourne, VIC, Australia, no. 10. Geneva, Switzerland: JACoW Publishing, Jun. 2019, pp. 2403–2405, Paper WEPMP034. [Online]. Available: http://jacow.org/ipac2019/papers/wepmp034.pdf, doi: 10.18429/JACoW-IPAC2019-WEPMP034.

[13] NUMATO LAB. *Nereid Kintex 7 PCI Express Development Board*. [Online]. Available: https://numato.com/docs/nereid-kintex-7-pci-express-development-board/

[14] J. Lawley, "WP350: Understanding performance of PCI express systems," Xilinx, San Jose, CA, USA, White Paper 350, 2014.

[15] *LogiCORE IP Aurora 8B/10B v6.2, User Guide 766*, Xilinx, San Jose, CA, USA, 2011.

[16] *Virtex-6 FPGA Memory Resources, User Guide 363*, Xilinx, San Jose, CA, USA, 2014.

[17] R. Frazier, G. Iles, D. Newbold, and A. Rose, "Software and firmware for controlling CMS trigger and readout hardware via gigabit Ethernet," *Phys. Procedia*, vol. 37, pp. 1892–1899, Jan. 2012.