EPJ Web of Conferences **251**, 04028 (2021) https://doi.org/10.1051/epjconf/202125104028
CHEP 2021

Free-running data acquisition system for the AMBER experiment

Martin Zemko[1,3], *Vladimir* Frolov[5], *Stefan* Huber[6], *Vladimir* Jary[3], *Igor* Konorov[6], *Antonin* Kveton[2], *Dmytro* Levit[6], *Josef* Novy[3], *Dominik* Steffen[6], *Benjamin Moritz* Veit[1,4], and *Miroslav* Virius[3]

[1]CERN, Geneva, Switzerland
[2]Charles University in Prague, Czech Republic
[3]Czech Technical University in Prague, Czech Republic
[4]Johannes Gutenberg University Mainz, Germany
[5]Joint Institute for Nuclear Research in Dubna, Russia
[6]Technical University of Munich, Germany

Abstract. Triggered data acquisition systems provide only limited possibilities of triggering methods. In our paper, we propose a novel approach that completely removes the hardware trigger and its logic. It introduces an innovative free-running mode instead, which provides unprecedented possibilities to physics experiments. We would like to present such system, which is being developed for the AMBER experiment at CERN. It is based on an intelligent data acquisition framework including FPGA modules and advanced software processing. The system provides a triggerless mode that allows more time for data filtering and implementation of more complex algorithms. Moreover, it utilises a custom data protocol optimized for needs of the free-running system. The filtering procedure takes place in a server farm playing the role of the high-level trigger. For this purpose, we introduce a high-performance filtering framework providing optimized algorithms and load balancing to cope with excessive data rates. Furthermore, this paper also describes the filter pipeline as well as the simulation chain that is being used for production of artificial data, for testing, and validation.

1 Introduction

The majority of physics experiments use trigger systems to initiate the data collection process and reduce the amount of data coming from the detectors. These systems usually meet the needs of experiments, but there are some use cases that cannot be covered by them. Specifically, we can point out experiments containing various detectors with diametrically different response times. Slower detectors are not able to process the same trigger signal as faster ones. This results in a huge conflict in the data acquisition system. As an example, the detector apparatus of the AMBER experiment will include a hydrogen time projection chamber (TPC) as an active target that has very long drift times. Due to this requirement, it is not possible to use a triggered data taking approach, and the need for a triggerless system has emerged.

The paper deals with such unique acquisition system that is currently being developed for the AMBER experiment at CERN. This system is based on its predecessor – the triggered iFDAQ system [1, 2], which is built from FPGA modules. However, the new DAQ takes a step further into a free-running acquisition. Such approach does not contain any triggers, and it is purely based on a continuous readout of the detectors. The filtering logic is moved to later stages of the system, where more sophisticated algorithms are applied on the full information of the detectors. Favourably, the physics programme of the AMBER experiment can benefit from extended capabilities of such filtering methods.

Phase 1 of the AMBER experiment concentrates on measurements with conventional beams provided by the M2 beam line at CERN. The high energy muon beam will be used to measure the charge proton radius on an active high pressure hydrogen TPC target down to low Q^2 values. The hadron beams will be used to investigate the pion induced Drell-Yan process and J/Psi production as well as proton-antiproton production cross sections, which allow more precise interpretation of existing data on dark matter searches. The whole experiment is supposed to run over the next 5 to 10 years [3].

2 DAQ architecture

In general, triggered acquisition systems have to make trigger decisions fast (in order of microseconds) due to small buffers on front-end electronics. This puts significant constraints on trigger systems. As a consequence, they are usually composed out of simple logical units, which only combine information of the fastest detectors. Therefore, only simple algorithms are utilized. In order to overcome these limitations, we propose a modern architecture, which moves the filtering logic to later stages of the DAQ system. At these levels, memory buffers are much bigger, so, they can handle more data, and data processing can take much longer. This gives us the possibility to design and perform more complex filtering decisions. In addition, data from all detectors including slower ones are utilized, which increases the reliability of the filtering decisions.

On the other hand, the system outlined above brings several challenges; firstly, the problem of synchronization of front-end electronics. In our case, a precise clock signal is distributed to all front-end electronics. Based on this clock signal, the data are timestamped and synchronized later. Another challenge is related to the data processing itself. The free-running readout of detectors produce an enormous amount of data that are supposed to be processed in real time. It means that one has to ensure a sufficient throughput of the DAQ system and a stable data flow. This challenge is resolved by an advanced acquisition system consisting of high-performance FPGA multiplexers, switches, and computers as it is shown in figure 1.

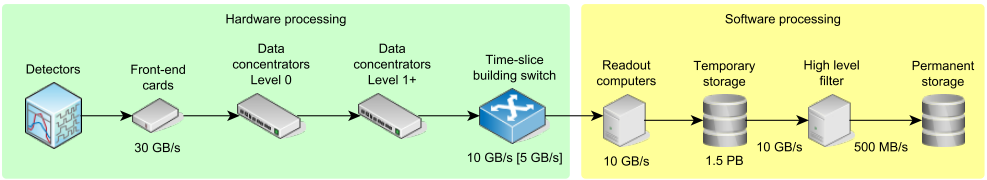

Figure 1. Data flow in the AMBER acquisition system

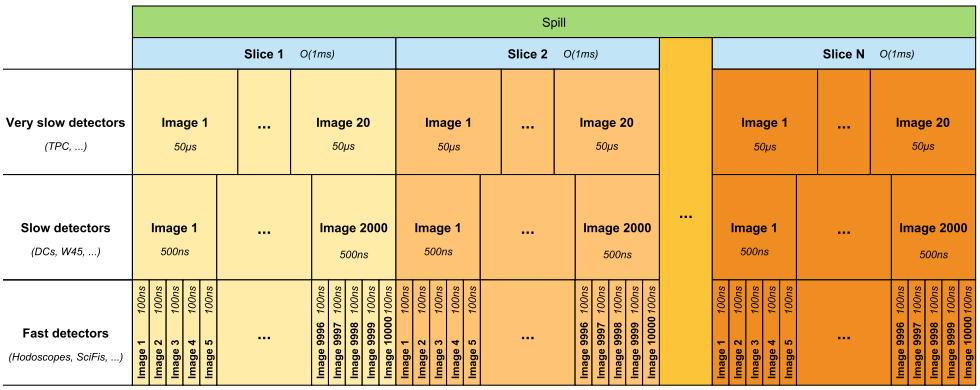

Figure 2. Data structure of the AMBER acquisition system – spills follow the beam extraction cycle, the horizontal axis represents time, whereas the vertical axis separates the data fragments from different types of detectors

2.1 Data protocol

The system uses a custom data format reflecting the needs of the AMBER experiment. It consists of several layers of encapsulated data that comply with well-defined rules. Data themselves are always aligned to 32-bit words that simplify the handling and synchronization. The protocol introduces a concept of time slices and images containing all data within a specified time interval. Due to the structure of the beam, the data taking period is divided into spills (usually 200), and each spill lasts around 4.8 s. Spills are furtherly divided into subintervals called time slices that last several milliseconds. Time slices contain all information acquired from detectors within the slice duration. They cover the full spill and create a continuous data stream. Additionally, time slices consist of images that have various lengths depending on the detectors. The full data structure can be observed in figure 2.

At the very beginning, frontend cards produce 32-bit long data words. The format and meaning of data words vary according to the type and nature of the detector. Though, some detectors may require more data words to describe their hits. If it is needed, such data words are clearly addressed and recognized by detector-specific formatting. Subsequently, data words of the same type and resolution are encapsulated into a single packet called an image. This action is performed still within front-end cards that serialize data words and wrap them using an image header and footer. The image header contains additional identifiers used for the unique indexation of detectors within the experiment. The image footer contains a cyclic redundancy check (CRC) used for data validation during their transmission.

Then, the data are processed by detector specific multiplexers (L0 multiplexers) that add a new layer of encapsulation. They merge images from different detectors, keeping chronological order of detector information, and wrap them into a time slice that contains additional information such as length, spill number, timing, etc. Consequently, time slices are transferred to data concentrators, i.e. L1+ multiplexers, where they are encapsulated again. Headers and footers at this level contain additional error words indicating on which port an error occurred. In general, several types of errors can be distinguished, e.g. inconsistent or missing data, desynchronization, port failure, etc., and all these errors are indicated directly within the data stream. There can be several levels of data concentrators; and thus, several layers of encapsulations can be found in the data stream. This approach allows us to design a multi-level DAQ structure with the use of identical devices. Multiplexers are based on Virtex-6

iFDAQ DHMx cards with 17 high-speed serial links providing a throughput at 650 MB/s [4]. These cards contains 4GB of DDR3 memory which is used to buffer data. At each level, the data are transmitted via optic fibres using the same protocol as described in the figure 2 [5].

Afterwards, the data are transmitted to the time-slice building switch containing 8 inputs and 8 outputs. The switch is based on the same FPGA module as the multiplexers, but it uses a different firmware. It combines time slices belonging to the same time period and forwards them to the readout computers. This device creates the same type of encapsulation as data concentrators; and therefore, the data protocol is homogenous also at this level. In total, the switch can handle the maximum throughput at the level of 5 GB/s [4].

The outputs of the switch are connected to readout computers that are equipped with commercial PCIe readout cards (Nereid Kintex 7 XC7K160T) called spillbuffers. Each spillbuffer can handle a maximum data rate at the level of 1.6 GB/s [4]. These cards receive, check, and validate the data. They also create the last level of encapsulation and transmit time slices to the readout software via the PCIe bus using DMA data transfers. Subsequently, the software validates the slices again and removes unnecessary protocol encapsulations except for the last slice level. Each readout engine is equipped with an external chassis that fulfils the role of the temporary storage. A single chassis provides 24 drive bays fully occupied by 16 TB HDDs and configured in the RAID 0 configuration.

At the last stage, a farm of high-performance computers is used for data reduction. The filtering computers copy the data from the temporary storage, analyse them and partially reconstruct physics events. If an event is identified, the corresponding data are preserved and sent to a permanent storage (CTA); otherwise, the data are discarded. This computer farm plays the role of the high-level trigger that filters the data by selecting only interesting events.

3 Optimizations of the readout software

Generally, servers may have several CPUs. As an example, figure 3 shows the computer architecture containing two CPUs (the same as our readout engines). In such systems, memory modules and PCIe resources are always associated with one CPU that communicates with the modules and provides the fastest access time. One CPU and its memory modules create a so-called NUMA (Non Uniform Memory Access) domain. A multi-socket computer can have several NUMA domains. If one CPU wants to access memory that does not belong to it, it has to send request via the CPU associated with this memory. Such redirection of communication causes delays in access time. Nevertheless, these inefficiencies are avoided by the proper software design [6].

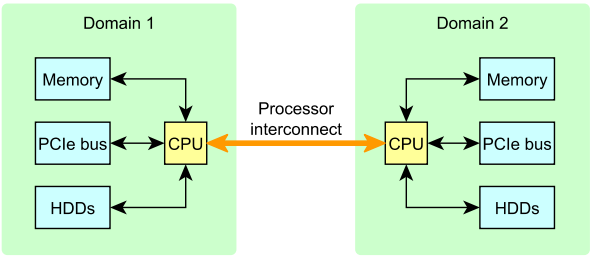

Figure 3. Communication diagram of a dual-socket server with division into two NUMA domains

A method that solves this issue is called siloing [6], and it is based on a single principle: data should be processed on the CPU closest to the source. In the terms of readout software, this means that the data coming from spillbuffer cards are distributed evenly across all NUMA domains. Then, they are processed on the same NUMA domain as they are stored. Considering identical CPUs and memory modules, the data are easily divided using a round-robin manner. This approach provides a reasonable trade-off between the complexity of algorithms and the resulting performance. Preferably, more complex balancing algorithms can be implemented according to specific use cases. If the data are spread across several CPUs, they have to be merged at some point. In our system, this synchronization happens at the very end of the processing chain, where blocks are serialized to a file. Before the data are written to a file, they are pushed to a buffer that keeps tracks of all memory blocks and serializes them in the original sequence. Therefore, the buffer produces the correct sequence of data blocks at the output of the software.

4 Data filtering

The essential part of the new AMBER DAQ system is the high-level trigger (HLT) that is used to reduce the data. It allows to use information of all detectors in the filter decisions. Such approach represents a huge advantage because the filtering is not limited by buffers of front-end electronics, and low latency does not limit the filter complexity any more. In addition, the high-level trigger provides wide variety of filtering algorithms and methods. On the other hand, the implementation of the HLT introduces several challenges. Firstly, high-performance computing clusters are needed. At this stage, parallel algorithms play significant roles and reduce the total cost of the system. The overall software architecture is also important and can markedly speed up the data processing. To achieve maximal performance, the software is highly optimized for the hardware it is running on and utilize its resources effectively. All these aspects will be discussed in the following sections.

4.1 Filter pipeline

In general, the filtering process consists of several steps that create a pipeline shown in figure 4. At the input, the HLT software expects a raw data stream formatted as described in the section 2.1. Usually, the data come from a data file, but they can originate also from network streams. Then, the HLT skims the data and searches them for time slices and their inner structures. Each slice is independent on others and can be processed individually. This allows to process time slices in parallel by distributing them to individual NUMA domains and processing threads.

In the processing threads, slices are completely decoded, and a slice object model is created. This model provides an easy access to the images. After that, images of so-called

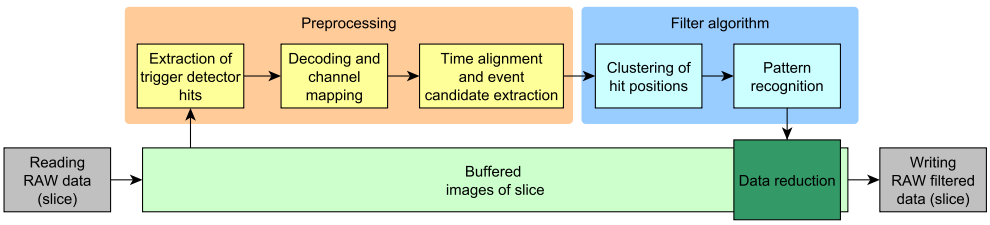

Figure 4. Filter pipeline of the high-level trigger

primary detectors (detectors participating in filter decisions) are extracted. Images of other detectors are not taken into account and are buffered waiting for the data reduction. Then, hits of primary detectors are decoded according to mapping information that describe connections between physical positions of sensitive elements and their frontend channels. For this purpose, a data container that can store several billions of items has been implemented. The container internally uses a multi-level structure of hash tables providing the constant access complexity $O(1)$ [7], which enables to perform fast lookups and channel mapping.

Consequently, hits of primary detectors are aligned in time. The next step is the identification of images containing interesting events by looking for coincidences of hits from different detectors in variable time windows. The algorithm uses a modified wavelet tree [8] as a fundamental data structure that represents a given time interval. Then, hits of primary detectors are inserted into this data structure followed by a pruning algorithm. As an output, time windows within which a physics event can possibly be contained are obtained. Afterwards, a k-means clustering algorithm is performed in order to define the event time more precisely. Mean times of the windows are used as cluster seeds for the clustering. Eventually, the resulting cluster times and their associated hits are called event candidates. They define our points of interest in the later processing.

After the time alignment, the spatial alignment and clustering of hits is applied. These steps depend on the selected filter algorithm and included detectors. The event candidates are checked for specific rules defined in the filter procedure. The output is a filtered list of event candidates.

Finally, the data filtering is performed at the level of images. Every image, that contains at least one valid event candidate, is supposed to be saved to the permanent storage: such images are marked with a save flag. In order to prevent time uncertainties, one of the neighbouring images is also marked together with the original one. It is either the previous image or the next one, depending on the position of the candidate inside of the image. When all event candidates are processed, all marked images are saved to the output file keeping their initial order and structure.

4.2 Load balancing

Paragraphs above described a single path of data. In order to ensure a sufficient performance, a massive parallelization is inevitable. At the first glance, more computers (processors) can be used for processing of the task; and load balancing mechanism has to be introduced. Then, three levels of load balancing are distinguished:

1. **Balancing between computers**
   At the first level, data are separated into data files called chunks. These chunks represent the smallest transferable blocks of data on the file system level. Therefore, these data files are processed on several computers independently. Considering identical computers, a simple round-robin approach is used. In this new system, the HLT supervisor supervises, monitors, and distributes the load evenly across all available computing nodes as it is depicted in figure 5. The communication is based on a custom DIALOG messaging library [9] and a simple text-based interface. The supervisor sends only control and configuration commands; thus, there is no need for a high throughput.

2. **Balancing between CPUs**
   Another level of load balancing happens inside of individual computers. As it was mentioned in section 3, a single computer can contain several CPUs and their corresponding NUMA domains. Utilizing the siloing principle, data are processed in the same domain

as they are stored, and data blocks are distributed evenly across all domains. This is easily achieved by splitting data files into smaller blocks of time slices. The processed chunk is split only at slice boundaries to avoid slice corruptions. Considering that slices are independent, such splitting is a valid approach.

3. **Balancing between cores**
   Subsequently, these blocks of time slices are processed within a single CPU. And since all CPUs contain more than one core, all cores are utilized in the processing. The same procedure as in the previous level is reused here. Simply, blocks of time slices are divided into individual time slices that are processed separately on individual cores. Also in this case, boundaries of the slices are respected.

When the data processing is finished, the data are merged to a single output data chunk. The synchronization takes place at the very end of the processing chain. A dedicated buffer accepts processed blocks and writes them to an output file in the time-sorted sequence.

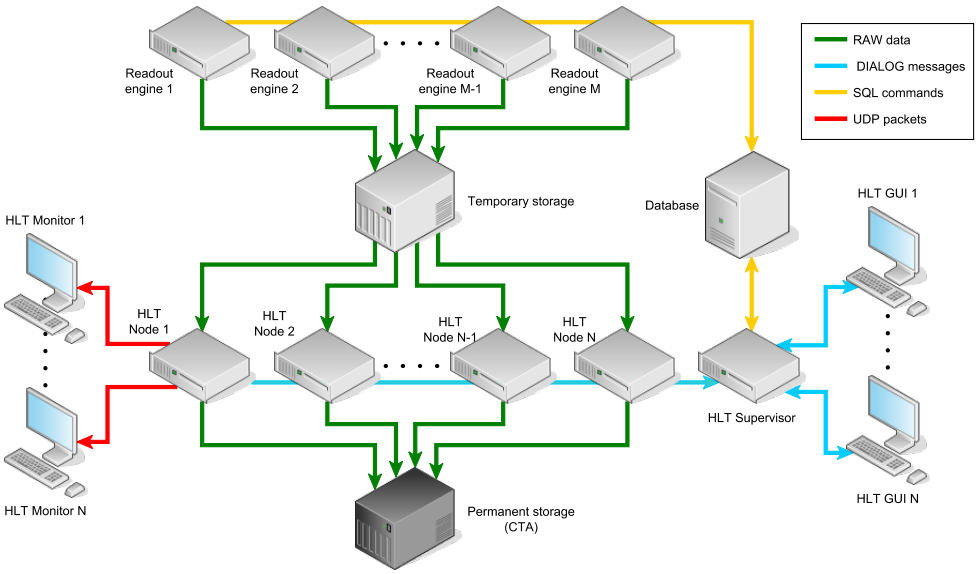

Figure 5. Communication diagram of the high-level trigger

5 Filter performance

The filtering stage uses several optimization techniques in order to achieve the maximum possible performance using the available hardware. However, the overall performance of the filter is not constant due to several aspects. First of all, it depends on the used filter algorithm. These algorithms vary in their processing complexity, and thus, also in their speeds. Therefore, developers of the filter logic should be aware of this fact, and they have to be careful in the selection of used algorithms. As an example, a simple filter logic that computes trajectories of particles can be considered. The algorithm calculates the 3-dimensional linear regression of hits and tries to estimate segments of particle trajectories called tracklets. Afterwards, full particle tracks are built out of the tracklets finding the minimum χ^2 estimate.

If the final particle track complies with the predefined filter rules, the corresponding event candidate is considered as a valid. As a result, a list of valid event candidates is obtained.

Another important attribute, which affects the filter performance, is the number of detectors included in the filter logic. The more detectors are included, the more images and hits have to be decoded and processed afterwards. Durations of slices and images as well as the beam intensity play significant roles. Figure 6 shows the dependency of the data rate on the number of detector planes, i.e. projections. The processing rate decreases with higher numbers of projections that have to be decoded. The plot also shows the connection between the data rate and the number of processing threads. As it can be seen in the figure, the linear scalability is reached in the most cases. In one scenario, a non-linear data rate is observed, which is limited approximately to 3 GB/s. This behaviour is the subject of our further investigation. The tests were performed on the Supermicro AS-1014S-WTRT server equipped with the AMD CPU 7282 (16 cores, 32 threads) and 128 GB RAM. The installed operating system was the CERN CentOS 7 including the kernel version 3.10.0.

Based on these measurements on a single computer, the full-scale filter farm would require around 20 computers to achieve the desired filtering rate of 10 GB/s. In a case of 56 projections, 10 servers will provide the sufficient performance.

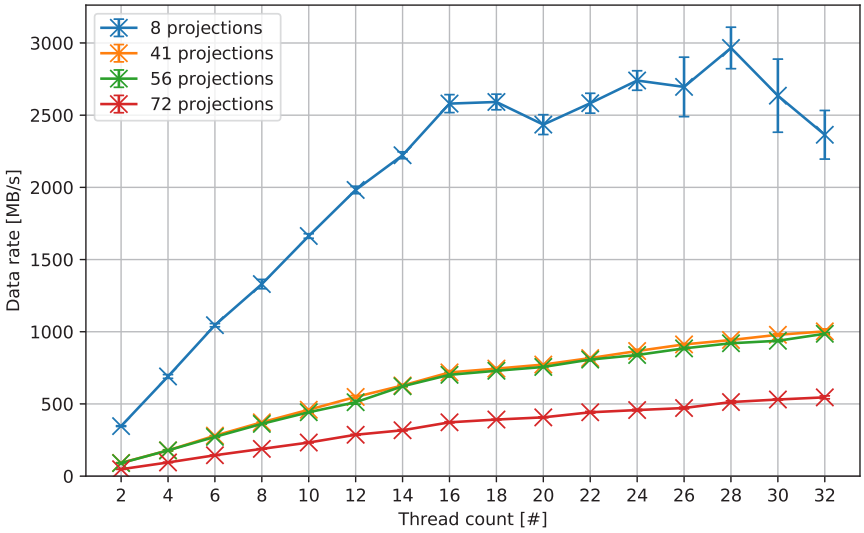

Figure 6. Dependency of data rate on the number of projections in a slice and processing threads

6 Data generator

Aforementioned system comprises complex software modules that require advanced testing methods. To develop and debug the whole DAQ chain, a data generator for the free-running data format is needed. Unfortunately, these data do not exist because the AMBER DAQ system is being developed now. Therefore, a way how to generate artificial data had to be found with the use of available software tools and Monte Carlo simulations. In fact, our goal is to perform a full simulation of the experiment.

6.1 Monte Carlo generator

The software used for Monte Carlo simulations at the AMBER experiment is the TGeant software package that is based on the Geant4 toolkit. The main goal of the TGeant package is to simulate responses of the experimental setup to physical processes one wants to study. To obtain trustworthy results, accurate geometry descriptions of the experimental apparatus are needed in advance. Subsequently, the TGeant simulates interactions of particles with matter and their transportations. Specifically, physics interactions are optimized for the properties and range of the detector setup [10].

The TGeant produces so-called Monte Carlo events containing simulated hits. Though, these data differ from the output of the real acquisition system. This is mainly due to the use of various detectors, digitization methods, frontend cards, chips, etc., which are not parts of the simulation. Moreover, data have to be formatted and encapsulated according to the specific rules of the free-running protocol. Therefore, all these aspects are taken into account. Nevertheless, Monte Carlo data are perceived as the ideal starting point for further simulations.

6.2 Generator

To obtain the data in the right format, several steps have to be taken in a sequence. This sequential procedure is called the generator pipeline, and it is shown in figure 7. Simulated Monte Carlo hits are stored in specific text files – TGeant ASCII files. At the beginning, these files are parsed and their object model is created in the memory. Then, the time-of-flight correction is applied, which modifies timestamps of individual hits and align them in time. Such correction is used to shift hits back to their original event time (interaction time). In real data taking systems, this procedure simulates properties of the detector setup such as cable delays and propagation times. Afterwards, Monte Carlo hits are perfectly time aligned with the generated event time.

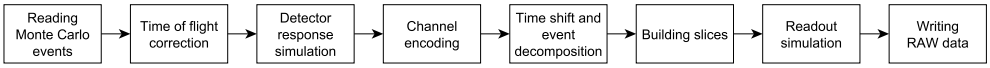

Figure 7. Simulation pipeline of the data generator

6.2.1 Detector response

Consequently, specific responses of detectors to passing particles are simulated. This procedure is called the detector response simulation. Hits are grouped by detectors and processed by detector simulation units. For each detector, a single unit describing its specific behaviour is implemented as it is illustrated in figure 8. These units are developed and maintained by detector experts who implement specific behaviour and formulas utilized in the simulation. The output of the detector response simulation is a list of events and their digits. And every Monte Carlo hit generates at least one digit in the simulation.

As the next step, the channel encoding procedure is executed. It takes identifiers of activated sensitive elements and translates them into the logical addressing scheme of the readout electronics. The encoding procedure gives us identifiers of data links carrying the data to the DAQ. Then, the event decomposition process takes place. It breaks events apart into individual hits and shifts them in time to achieve a desired rate, i.e. beam intensity. This parameter corresponds with the Poisson distribution and can be specified by user.

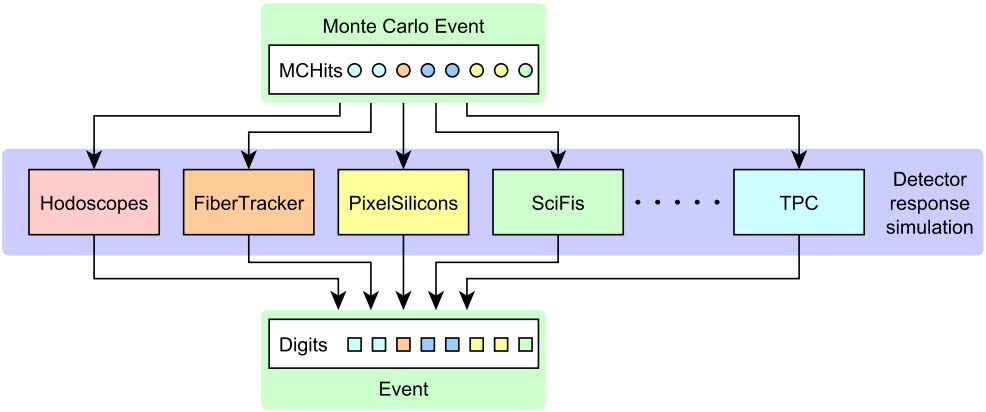

Figure 8. Inner structure of the detector response simulation

The next step of the simulation chain is a slice building process that incorporates digits into data structures containing slices and images. It relies on an iterative processing of individual digits. In short, this operation searches for the correct placement of every single digit. Its position depends on the data format itself and protocol properties. Within this step, all these characteristics are taken into account. A recursive descent algorithm is implemneted, which pushes digits from upper levels of the data format (spills or slices) to lower levels (images). This procedure is repeated for every single digit until no more digits remain. Eventually, every digit is accommodated in a certain image. At the end, a multilevel structure of containers containing correctly associated digits in their corresponding images is generated.

There is yet another functionality performed at this point. User can decide whether they want to generate either flat data (containing a single slice level) or structured data (containing more slice levels). It is because every multiplexer adds one layer of encapsulation; and thus, the final data structure depends on connections between multiplexers. This functionality allows us to test various structures of the DAQ system.

6.2.2 Readout response

After the slice building process is done, a readout response simulation takes place, which is also illustrated in figure 7. This particular task is intended to simulate the behaviour of the readout electronics and translate the digit information into the frontend specific format of data words. Such procedure is unique for every chip type. Besides, frontend cards can perform some secondary data processing such as compression or aggregation of values.

At the end of the generator pipeline, a serialization algorithm is executed. Here, slices and images are serialized into a single data stream that is directed into a file. The data are formatted and wrapped by appropriate headers and footers. The only values that are calculated at this step are sizes of slices. These sizes are then written to corresponding footers.

7 Future development

The outlined acquisition system should provide a sufficient performance at the level of 5 GB/s sustained rate and 15 GB/s on-spill rate. For the online data filtering, the goal is 1 GB/s

of a sustained throughput per node. As presented in figure 6, the real processing rate is a half of the desired data rate if many projections are included. Luckily, several possibilities of improvements exist. For example, GPU units providing higher number of cores can be introduced, and more slices can be processed in parallel. Another possibility is to implement a distributed clustering algorithm utilising numerous GPU cores. In any case, the use of GPU cards will be considered in our further development. Additionally, alternative possibilities of monitoring of filter algorithms are examined. Our goal is to collect monitoring data from all processing stages of the HLT. Their understanding is critical for the performance as well as for their validation.

Regarding the data generator framework, the simulation modules can be extended further in cooperation with detector experts. Considering the wide variety of hardware available at the AMBER experiment, all major detectors and chips can be included to the simulations and improve their reliability and precision. Eventually, a tighter integration with the TGeant software is also foreseen in the future; as it would simplify the data generation process even more.

8 Summary

We designed the free-running data acquisition system for the AMBER experiment. We depicted its structure and the custom data protocol. Several optimization methods used in our readout process have been described as well. This system also introduces the advanced high-level trigger framework that provides high-performance filtering capabilities; and its inner composition has been described in this paper. We have also measured its performance and discussed load-balancing methods used in it. Additionally, the data generator framework capable of the full simulation of the AMBER experiment was presented. The generator produces data that are used to test the functionality of the filter software. At the end, we discussed possible extensions and further developments of the high-level trigger and the data generator. We plan to test the proposed DAQ system in the upcoming pilot run of the AMBER experiment with the real data coming from detectors at the end of 2021.

References

[1] D. Steffen, M. Bodlak, V. Frolov, S. Huber, V. Jary, I. Konorov, A. Kveton, D. Levit, J. Novy, O. Subrt et al., PoS **TWEPP-17**, 127 (2018)

[2] O. Šubrt, M. Bodlák, V. Frolov, S. Huber, M. Jandek, V. Jarý, I. Konorov, A. Květoň, D. Levit, J. Nový et al., EPJ Web of Conferences **214**, 01032 (2019)

[3] B. Adams, C.A. Aidala, G.D. Alexeev, M.G. Alexeev, A. Amoroso, V. Andrieux, N.V. Anfimov, V. Anosov, A. Antoshkin, K. Augsten et al., Tech. rep., CERN, Geneva (2019), `http://cds.cern.ch/record/2676885`

[4] S. Huber, I. Konorov, A. Kveton, D. Levit, J. Novy, D. Steffen, B.M. Veit, M. Virius, M. Zemko, S. Paul, IEEE Transactions on Nuclear Science **(to be published)** (2020)

[5] Y. Bai, M. Bodlak, V. Frolov, V. Jary, S. Huber, I. Konorov, D. Levit, J. Novy, D. Steffen, M. Virius, JINST **11**, C02025 (2016)

[6] D. Gallatin, *NUMA Siloing in the FreeBSD Network Stack* (2019), `https://papers.freebsd.org/2019/EuroBSDCon/gallatin-NUMA_optimizations_network_stack.files/slides.pdf`

[7] M.L. Fredman, J. Komlós, E. Szemerédi, *Storing a Sparse Table with 0(1) Worst Case Access Time* (1984), Vol. 31, pp. 538–544, ISSN 1557735X

[8] Y. Zhu, D. Shasha, *Efficient Elastic Burst Detection in Data Streams*, in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Association for Computing Machinery, New York, NY, USA, 2003), KDD '03, p. 336–345, ISBN 1581137370, https://doi.org/10.1145/956750.956789

[9] Y. Bai, M. Bodlak, V. Frolov, S. Huber, V. Jary, I. Konorov, D. Levit, J. Novy, D. Steffen, O. Subrt, International Journal of Physical and Mathematical Sciences **11**, 401 (2017)

[10] T. Szameitat, Ph.D. thesis (2017)