



CERN Summer Student Report
franz.danylo.machado.perez@cern.ch
franz.machado.p@uni.pe

Data transfer between LHCb data acquisition system and FPGA accelerators

Franz Machado

Supervised by Riccardo Fantechi

Abstract

In this work we review the data acquisition system used by the LHCb experiment and we perform a test on data transfer between PCIe40, EB and the RETINA buffer. We use a method that consists in merging two data streams that use a specific format into one data stream. This method could be more efficient and takes into account certain characteristics of the firmwares used in previous runs.

Geneva, Switzerland
September 29, 2021

Contents

1	Introduction	3
1.1	CERN	3
1.2	LHCb	4
1.3	High Luminosity upgrade	4
2	LHCb Data Acquisition	5
3	Accelerators	7
3.1	Accelerators on the RETINA project	7
4	Current Work	8
4.1	Proposal	8
4.2	Work detail	9
4.2.1	Proper synchronization	10
4.2.2	Step 0:	10
4.2.3	Step 1:	11
4.2.4	Step 2:	11
4.2.5	Step 3:	12
5	Conclusion	13
6	Recommendations	13
	Acknowledgements	14
	Bibliography	15

1 Introduction

1.1 CERN

Conseil européen pour la recherche nucléaire or The European Organization for Nuclear Research, known as CERN, is a European research organization that operates the largest particle physics laboratory in the world. Since CERN began in 1954, there have been many significant breakthroughs, both in particle physics and technologies.

CERN houses the largest particle accelerator complex in the world. Through a very complex multiple system, CERN can accelerate protons up to 6.5 Tev in the Large Hadron Collider (LHC). Inside the accelerator, two high energy particle beams travel at close to the speed of light before they are made to collide. The beams travel in opposite directions in separate beam pipes (two tubes kept at ultra-high vacuum). They are guided around the accelerator ring by a strong magnetic field maintained by superconducting electromagnets until they are bent to collide which results in the interaction between the partons of the protons at up to 13 Tev.

Along the ring of the LHC, 8 experiments can be found. While 4 of them are massive experiments (ALICE, ATLAS, CMS and LHCb), the others are much smaller (TOTEM, MoEDAL, FASES and LHCf) [1].

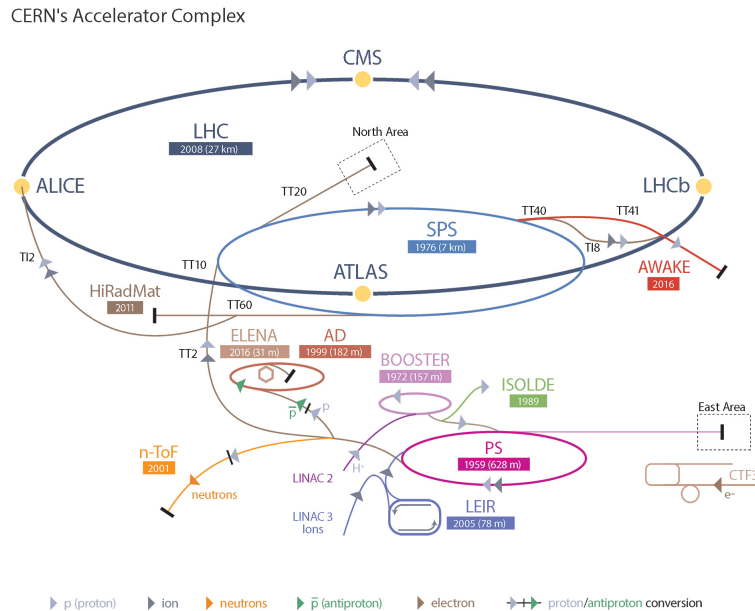


Figure 1: CERN Accelerator Complex (operating and approved projects).

1.2 LHCb

The Large Hadron Collider beauty (LHCb) experiment specializes in investigating the slight differences between matter and antimatter by performing measurements on B-quarks to look for CP-violation.

The construction of the LHCb experiment was done in a different way than the experiments like ATLAS and CMS, leaving this experiment asymmetric. This is primarily because LHCb focuses on studying mainly forward particles, those thrown forwards by the collision in one direction (low angle but very high precision).

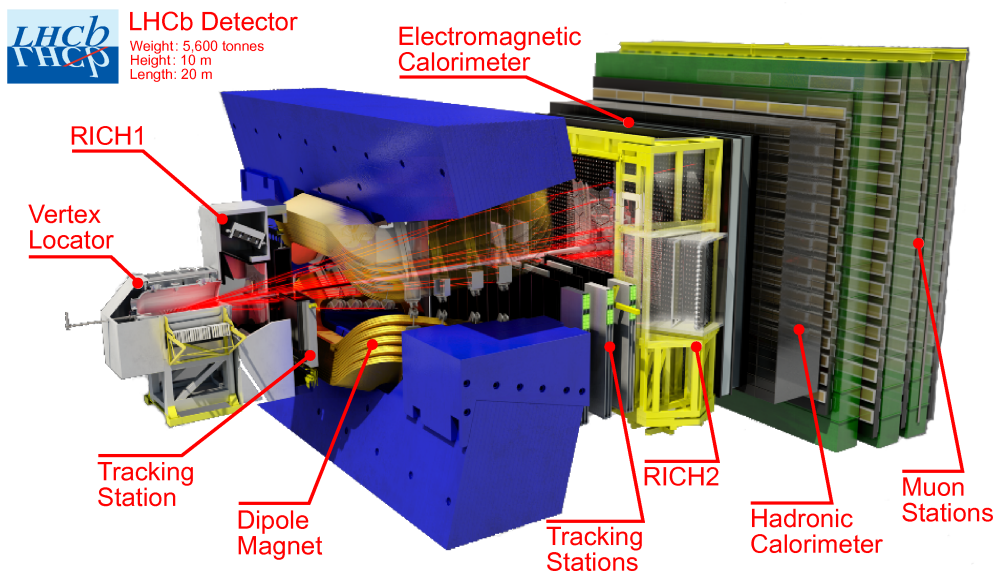


Figure 2: Overview of the asymmetric LHCb detector.

1.3 High Luminosity upgrade

In order to find events of very low probability (events that are interesting for physics) a very large amount of data is needed to detect them. A solution for that is to increase luminosity because it will produce more data, allowing physicists to study known mechanisms in greater detail and observe rare new phenomena that might reveal themselves. For example, the High-Luminosity LHC will produce at least 5 times more Higgs bosons per year, compared to LHC in 2017.

Increasing the luminosity will not be the only challenge to overcome, the capacities of the data acquisition systems of the detectors and sub-detectors that are currently used must also be increased, since a greater amount of data will have to

	Runs 1&2	Run 3
Event size [kB]	50-60	100
event rate [MHz]	1	40
# data sources	313	500
# data sinks	up to 200	up to 5000

Table 1: Key parameters of the LHCb DAQ [2].

be analyzed every second. For this, multiple ideas are being studied and developed while you read this document.

In the next section we describe how the data acquisition system that has been used in the LHCb experiment works. After this section we will focus on describing the work that has been done in this research internship.

2 LHCb Data Acquisition

This section will provide us with a brief introduction to the data acquisition system used by the LHCb experiment. For this we will use a schematic diagram, as shown in figure 3. As is known, in the previous run the proton-proton collisions occurred at a rate of 40 MHz, that is, collisions occur every 25 ns.

The data produced by the sub-detectors in each event will be handled by about 500 read-out boards with PCIe interfaces. All the interfaces are installed in about 160 servers which form the Event Builder (EB) system whose job is to combine fragments from all the sub-detectors in a single event block. The stream of event blocks is sent to a processing system where they are reconstructed to extract physics quantities such as energy or transverse loss momentum.

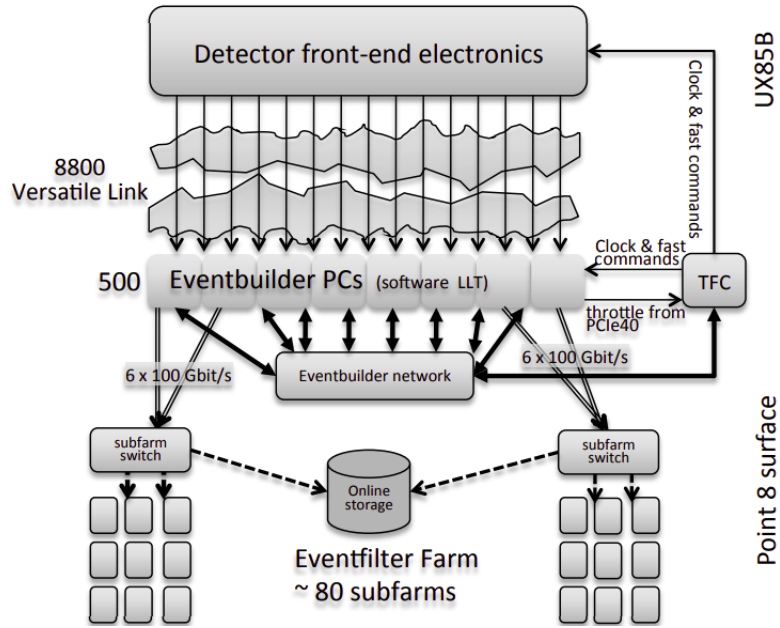


Figure 3: Overview of the LHCb DAQ system.

After we have had discussed about the purpose of the EB we can represent the EB using another schematic diagram where two processing systems are shown, each one with an independent memory. Using figure 4 we can understand how data streams move within EBs. The data taken by each detector enters into the event builders via optical interfaces within the PCIe40 cards. This PCI-express card will push the data from the optical interfaces directly into Memory 2 via a direct memory access (DMA) transaction (bold blue arrow). Meanwhile, the EBs elect, in a round-robin fashion, one of the other EBs which will be responsible for building a certain set of events. All EBs will send the data they have related to this set to the elected EB via a LAN network (bold green arrow) [3].

The elected EB will receive this data via its EB Network Interface. It will then store this data in Memory 1 (bold orange arrow) and will move the data it already acknowledged from Memory 2 to Memory 1 (thin green arrow). If all the data is available, it will be combined (i.e. the event is built) and the data package will be forwarded to the Event Filter Farm via LAN (bold grey arrows).

The main challenge of this process is the large amounts of data being transferred. For example, the incoming detector data (bold blue arrow) can be up to 100 Gb/s such that the memory load in CPU2 is quite heavy while the computational workload of both CPU's is moderate.

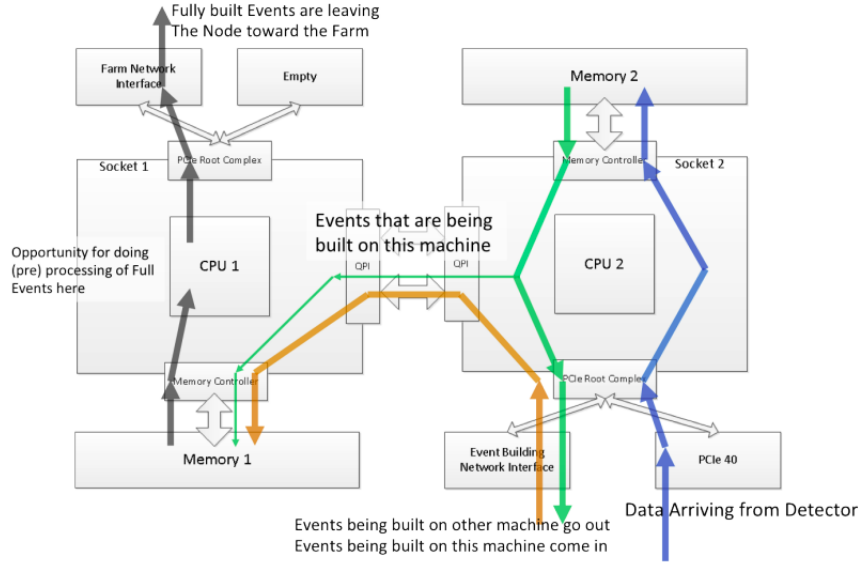


Figure 4: Overview of one Event Builder and the data streams within the system.

3 Accelerators

Due to the high luminosity upgrade we will generate data quicker than our ability to analyze, understand, transmit, and reconstruct it. That's why we need something to offload work from general-purpose processors.

One possibility is to use an accelerator which performs a set of operations with higher performance or greater energy efficiency using a piece of specialized hardware where we can implement certain types of functions, for example filtering out not interesting events or computing quantities to offload the following processing stage. The main use of accelerators is to speed up processes such as we mentioned, their use significantly improves the performance of this type of workloads. Typically, accelerators are boards with one or more FPGA and optical connectivity which are connected via a PCIe slot and are seen as standard PCIe devices.

3.1 Accelerators on the RETINA project

It is important to know about current and developing projects within the LHCb experiment such as the RETINA project.

Within this project, the use of FPGA boards and PCI40 cards have a vital role since they have to transmit and analyze data efficiently. In particular this project aims to do a fast tracking of physics events in the most efficient way using the same principles of the visual image elaboration in the human retina. For this, it

was found that the algorithm implemented within the FPGA boards must work with sub-microseconds latencies. In addition, these accelerators must be high speed and high bandwidth.

The FPGA cards have a PCIe x16 interface, and will plug into the EB nodes. They need to receive data from the PCIe40 cards which works as read-out units, so they should be sitting somewhere near the card, processing the data and producing in output some reconstructed data. This FPGA output should then be dealt with simply as an additional piece of raw data, as if it came from an additional detector section. This piece of data could be merged with the original one either within the FPGA card itself, adding the result to the data received from the PCIe40 board or by the EB which will receive separately the original data and the result of the FPGA processing. As a final result we will have that the EB will see some extra "raw data" in the event that is in reality partly reconstructed data, and incorporate them in the event record as normal (more information about it in [4]).

4 Current Work

In the first instance, it is important to know what is the current state of the data acquisition system that is being tested by the experiment. As we know, the detector is divided into sub detectors, which are basically built with different technologies to get data, divide it into suitable parts and send those parts as inputs for the read-out boards which do processing like clustering or packing data.

Each read-out board sends to the EB the data fragments from different events and their length and type in three different DMA streams. The EB collects them in a larger array which contains the packet sizes at the beginning and the corresponding packet data at the end. It is possible to access the data of each packet by evaluating the sizes of the previous packets since it would only be necessary to add the previous sizes.

For the optimization of the EB mechanism, it is needed to have a buffer of around 30000 events to send data to the accelerator, which has low latency and small out-board memory, this method is not the best and a more efficient data transfer mechanism should be used.

4.1 Proposal

Another method to perform that considering efficiency is to send the data in the following format: first the size and then the corresponding packet. This is more efficient because we don't have to decode the array to find data as the previous method. This is a backward compatibility feature of the PCIe40 firmware.

Also taking into account that the next update of the detector will bring with it

data_stream_1	data_stream_2
0xF0000007	0xF1000004
0x00000001	0x00000001
0x00000002	0x00000002
0x00000003	0x00000003
0x00000004	0x00000004
0x00000005	0xF1000004
0x00000006	0x00000005
0x00000007	0x00000006
...	...

Table 2: Example of the used format.

an increase in the volume of data to be acquired [table 1], reading data through different channels (data streams) using the aforementioned data format is essential to avoid the overload on each read-out card

We have used two streams because, as I mentioned, the PCIe40 card has PCIe x16 interface, but for firmware reasons can only transfer at x8. So they have implemented the firmware such that the data is sent on two x8 channels to fill the x16 completely. Our accelerator board instead cannot have two x8 channels, known in jargon as "bifurcation". So this is the reason for the merging.

In the developed code in this work, two data streams have been considered as inputs, data_stream_1 and data_stream_2, which carry the information of each sub-detector. The format being used is the following, the size of each packet is detailed in the header of each packet and has the following form 0xFa000bcd, where "a" is the identifier of each stream and "bcd" is the length of each packet. After the header, the information contained in each package is detailed. This is a very simplified format to work with due the complexity of the real one, we can find a more extensive discussion on it in [5]. We can visualize an example of the mentioned format below.

4.2 Work detail

After describing the data format that will be used in this simulation, we will proceed to develop a code in C where we emulate the behavior of the various software and firmware components of the data acquisition process. This code implements the essential parts for the data streams within the system such as the generators, the consumers, EB's and the RETINA buffer, but unlike the previous work, not only one input port for data will be simulated, but two ports will be implemented.

4.2.1 PROPER SYNCHRONIZATION

At this point it's extremely important to introduce the concept of proper synchronization owing to the fact that we will use it in many steps we will describe later. As there are many processes related to firmware and software which must run in parallel, we must be extremely careful to synchronize all of them in order to have a safe transfer of data without losing any packets along the path.

First, we need to distribute the processes using different threads, this is a common method where two or more functions can run concurrently. Then, to avoid intern conflicts we need to lock any shared resource, an efficient method to do this is mutex locks. Also many flags are implemented to help in the synchronization process.

4.2.2 STEP 0:

In order to achieve our goal, an extensive revision of the base code¹ was done. We found that the base code had as a purpose to simulate the data read and write synchronization mechanism between the emulated PCIe40 card, FPGA accelerator, Buffer, and the EB.

In this case the code works in the following way: The PCIe40 card sent data continuously to the FPGA accelerator through a Buffer, and the EB would read the same data through the Buffer only when the FPGA has finished reading the data. To achieve this, the flow had to be done concurrently, where the reading and writing was done continuously and in parallel. Hence, three threads were created, representing the aforementioned hardware peripherals, along with a circular array representing the Buffer. Every thread had two properties; a thread number and a circular array. The thread number is useful when checking which thread is allowed to read at any given moment. The circular array represents the hardware peripherals' data storage which could accept data continuously, that's why it is circular. To keep track of the slots being written to and read from, two pointers were created: read and write. The read pointer of the Buffer is only updated once the data is read by the EB and not the FPGA. This is done to avoid any discrepancies in the data being read by the FPGA and the EB, this is known as safe transfer [6].

¹https://github.com/danylomachado1/DT-DAQ-FPGA/blob/main/base_code.c

4.2.3 STEP 1:

In this first step, our task was to develop a new code² that works as follows: first we must receive two data streams as input, then these data must be read to finally be stored in two independent outputs.

For this, two new threads were created (`write_func_stream_1` `write_func_stream_2`) which generate data arrays by emulating in a very simple way the output signal of the sub-detectors, at this stage the input data were just consecutive numbers from 1 to 750 which is the allocated size for the input array. These new threads are called generators. For proper functionality we need to create new write and read pointers too. Additionally, similar to the base code, a couple of circular arrays were created to simulate the main buffers (`buffer_data_stream_1` `buffer_data_stream_2`). Finally, two additional threads (`read_func_stream_1` / `read_func_stream_2`) were necessary to copy the data stored into the main buffers to two independent output arrays. We can simplify the aforementioned by using the scheme shown in figure 5.

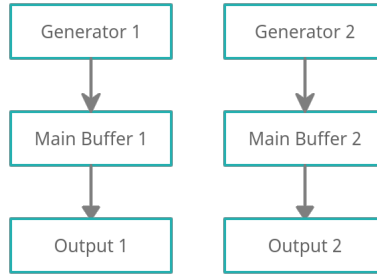


Figure 5: Schematic representation of step 1.

4.2.4 STEP 2:

In the second update of the code³, our work consisted of implementing an additional function in order to replicate the data format described in table 2. Then we had to implement a new method which allowed us to join the independent outputs of the previous step into a single common output. This common output is twice larger than the independent ones.

To replicate the format of Table 2, a basic knowledge about the hexadecimal format was necessary, we call to this new function as "array_input". To achieve the second task we had to create global variables (`r_stream_1` / `r_stream_2`) in

²https://github.com/danylomachado1/DT-DAQ-FPGA/blob/main/DAQ_FPGA_v1.c

³https://github.com/danylomachado1/DT-DAQ-FPGA/blob/main/DAQ_FPGA_v2.c

order to save the relative positions of the data within the main buffers and thus make a safe copy on the common output. This common output is saved into "joint.out" file, also the independent components of this common output are saved into "stream1.out" and "stream2.out". We can use as a guide the scheme shown in figure 6.

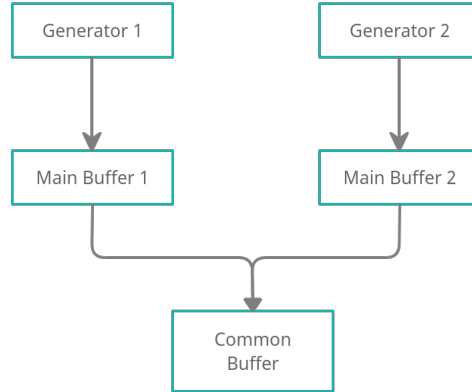


Figure 6: Schematic representation of the operation of the developed code.

4.2.5 STEP 3:

So far we have implemented data transfer between the PCIe 40 card and the main buffers. In the final code⁴ we must ensure that the two data streams are copied independently to the EB, where a consumer mechanism must also be implemented, and, in parallel, to a common output.

Then, thanks to an extensive modification within the threads (`read_func_stream_1` / `read_func_stream_2`) we were able to implement an independently transfer of data from the main buffers to the EB's (`empty_sacc_1` / `empty_sacc_2`) and in parallel, the transfer of the data from the main buffers to a common output which we call the RETINA buffer (`joint_data_empty`). We use the same output files we used in step 2 to save the common output and the independent components of it. In addition to this, various global and local variables were implemented in order to respect the safe copy mechanism.

⁴https://github.com/danylomachado1/DT-DAQ-FPGA/blob/main/DAQ_FPGA_v3.c

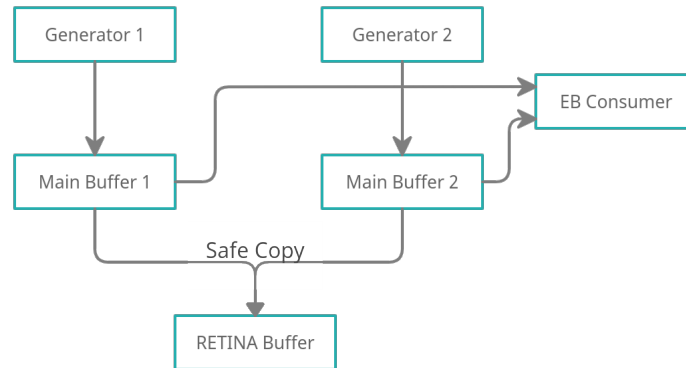


Figure 7: Schematic representation of the operation of the developed code.

5 Conclusion

- An extensive review of the data acquisition system used by LHCb was done in order to have a complete view of the principal components and learn how they work.
- The proposed test was performed successfully, a proper data transfer was tested using a code written in C language. As a result we obtain a file where the common output was saved, this common output is RETINA buffer.

6 Recommendations

- As a recommendation for performing a similar project (Summer Student project) within a large project, it would be great if some high level documentation would be available. The data acquisition system, the PCIe40 cards or accelerators are extensive projects and it takes time to get familiar with each one. Having some organized high-level documentation where we could find basic concepts and relations between them would make it far easier and foremost less time consuming to get familiar with the project and the code base.
- My final recommendation is directed to those who, like me, were assigned a project in a new research area. Let's not be afraid to ask and get involved with the project assigned to you, your supervisor will not hesitate to help. Let's have fun.

Acknowledgements

Starting a new project in a completely different area from the one you are used to is always challenging because you can find new tools and methods to work with. However, this internship has been one of the best experiences I have had in my life. Just knowing that I can contribute to the tests for a future design of one of the most complex data acquisition systems for the LHCb experiment is fascinating.

All the work achieved in this internship I owe to Riccardo Fantechi, who knew how to help me and guide me throughout the process by providing me with documents and answering all my questions.

Bibliography

- [1] CERN. CERN Annual report 2019. Technical report, CERN, Geneva, 2020. URL <https://cds.cern.ch/record/2723123>.
- [2] LHCb Trigger and Online Upgrade Technical Design Report. Technical report, May 2014. URL <https://cds.cern.ch/record/1701361>.
- [3] Wendo Beuker. Design of a Tracking Processor Unit for LHCb Data Acquisition system. Aug 2018. URL <https://cds.cern.ch/record/2634790>. CERN Summer Student short final report.
- [4] Federico Lazzari. An optical network for accelerating real-time tracking with FPGAs. Apr 2020. URL <https://cds.cern.ch/record/2716393>.
- [5] Niko Neufeld. Raw-data Transport Format and ODIN Fragment Format. Jun 2021. URL <https://edms.cern.ch/document/2100937/4>.
- [6] Yara Al-Quorashy. Interfacing FPGA Track Processor Units with the LHCb Event Builder. Aug 2019. URL <https://cds.cern.ch/record/2685341>. CERN Summer Student short final report.