Using FPGA accelerators for the calorimeter decoding in the LHCb upgrade

LHCB Technical Note

Issue:	Draft
Revision:	1
Reference:	LHCb-INT-2017-032
Created:	13th Dec 2017
Last modified:	15th March 2017

Prepared By: Alexandru Amihalachioaei/Gh. Asachi Technical University of Iasi
Christian Faerber/CERN

Abstract

The upgrade of the LHCb detector will result in a huge increase of bandwidth up to 40 Tbit/s. The data from the LHCb calorimeters along with the Muon detector data plays one of the most important roles in the current L0 hardware trigger and in the future event filtering process. Due to bandwidth limitations the calorimeter data is compressed and will need to be decoded in the event filter farm. Doing the decompression at 40 MHz is a challenging task even with all the resources available on the current computing farm. This note studies an alternative to use FPGAs to decode the data, in particular the new Intel Xeon and FPGA computing platform.

Document Status Sheet

Table 1 Document Status Sheet

1. Document Title: Using FPGA accelerators for the calorimeter decoding in the LHCb upgrade			
2. Document Reference Number: LHCb-INT-2017-032			
3. Issue	**4. Revision**	**5. Date**	**6. Reason for change**
Draft	1	13 December 2017	First version

Table of Contents

List of Figures

List of Tables

1. Introduction

This note describes the usage of two FPGA computing platforms to accelerate the decoding of the raw data generated by the LHCb calorimeter. The first is a two socket platform which contains the Intel Xeon E5-2560v2 CPU, a ten core chip, operating at a base frequency of 2.8 GHz and a Stratix V FPGA. They communicate to each other using the Intel Quick Path Interconnect interface. The FPGA has cache-coherent access to the system main memory.

The second platform is equipped with the Intel Xeon E5-v2560v4 CPU, with 14 cores running at 2.4 GHz and an Arria 10 FPGA. The FPGA is placed on the same package with the CPU, allowing for an even faster communication between the Xeon and the FPGA.

1.1. Calorimeter Data Flow

There are two types of calorimeters used at the LHCb detector, the electromagnetic calorimeter and the hadron calorimeter. In addition, the Scintillating Pad detector and Pre-Shower detector are used for trigger but they are to be removed for the upgrade. The calorimeter data is very important because it is used together with the Muon data for the existing Hardware Level 0 trigger.

The path of the calorimeter data to the event builder starts with the front end boards which are connected to the calorimeter's electronics. Currently, there are 188 of these boards used for the electromagnetic calorimeter and 50 boards for the hadron calorimeter. Each front end board will read data from 32 ADC channels. In total, there are 6016 ADC channels for the electromagnetic calorimeter and 1488 channels for the hadron calorimeter. The next step in the data flow is sending data from the front end boards to the tell boards. At the moment, there are 28 tell boards used for the ECAL and 8 for the HCAL. One Tell board will receive data from multiple front-end boards. Given that front end electronics will transmit data at full 40 MHz bunch crossing rate, the calorimeter raw data needs to be sent to the event builder with an outstanding bandwidth which can vary between 1.32 Tbit/s and 5.25 Tbit/s for the ECAL data. The bandwidth will depend on the number of particles that hit the calorimeter and on the energy of those particles.

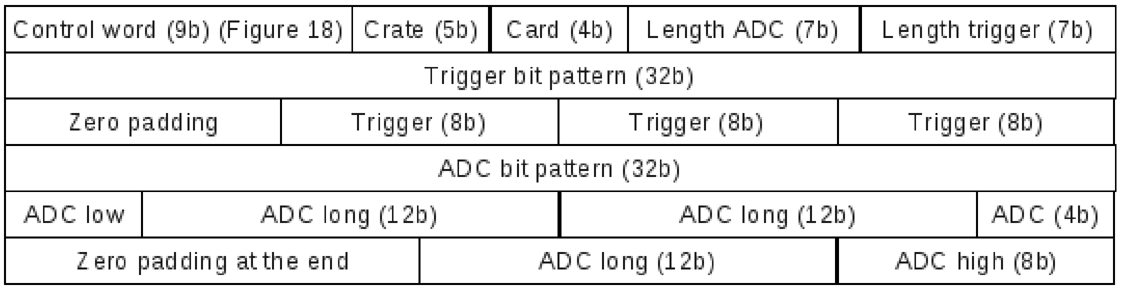


Figure 1 HCAL and ECAL data format

The raw data received from a front end board is formatted as it is seen in Figure 1. The first word is the header which provides information about the location of the front end board from the calorimeter. The front end boards are stored in crates, each crate containing multiple boards. The data from the 32 adc channels is either stored using 4 bit or 12 bit. There is a 32 bit word which will represent the coding mode for each adc value. The bit will be zero if the value is coded using 4 bits or it will be set to 1 in case the value is coded using 12 bits. In order to decode this kind of data the CPU has to do a lot of bit shifting operations. FPGA's are a very good choice for improving this kind of operations because bit shifting would mean only routing wires inside the chip.

2. Raw Data Decoding

2.1. Stratix V implementation

The Stratix V Xeon FPGA platform was the first one to be tested. The FPGA is able to make read or write requests and write or receive groups of 64 bytes in one single request from the system main memory. These groups of 64 bytes are also named Cache Lines. The Intel software framework will allocate a shared workspace which is accessible by the FPGA and by the software application. The FPGA can be programmed to either use physical or virtual addresses, which allows the FPGA to work with higher level computer science concepts such as Linked Lists, Trees and other suitable data structures that may apply to the workload.

The first step in the decoding is to send the raw data to the FPGA memory. One bank from either HCAL or ECAL front-end boards has a size that varies between 24 and 96 bytes. The first implementation was simpler and a little bit inefficient. Given that a Cache Line has 64 bytes, the data is sent using one or two cache lines, depending on the size of the bank.

After the data is sent to the FPGA, the adc values will be computed using a two clock cycle pipeline. The implementation is based by looking at memory as a collection of 4 bit values. In the first stage, 32 addresses with 4 bit offset are computed in parallel which point to the adc locations in the raw bank. In the second stage, using the previously computed addresses, all 32 adc values are read form the memory at the same time. The output for each value from the memory will be 12 bit wide and depending on the adc bit pattern it will be truncated to 4 bits. These operations are repeated for every front-end board found in the input data.

The result of decoding one event is a collection of Gaudi objects called “CaloAdc". One object contains the adc value and information about the calorimeter cell. The cell information is used to identify the position of the cell on the x-y axis of the calorimeter. The output format is presented in Figure 2. The values could be packed using only 16 bits but this would mean doing some additional bit shifting operations again on the CPU to unpack the data. To avoid this bit shifting operations, the first implementation used a 48 bit format for each calorimeter cell, using 8 bits for row, column, area and calorimeter id and another 16 bits for the adc value.

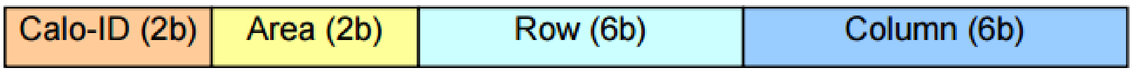
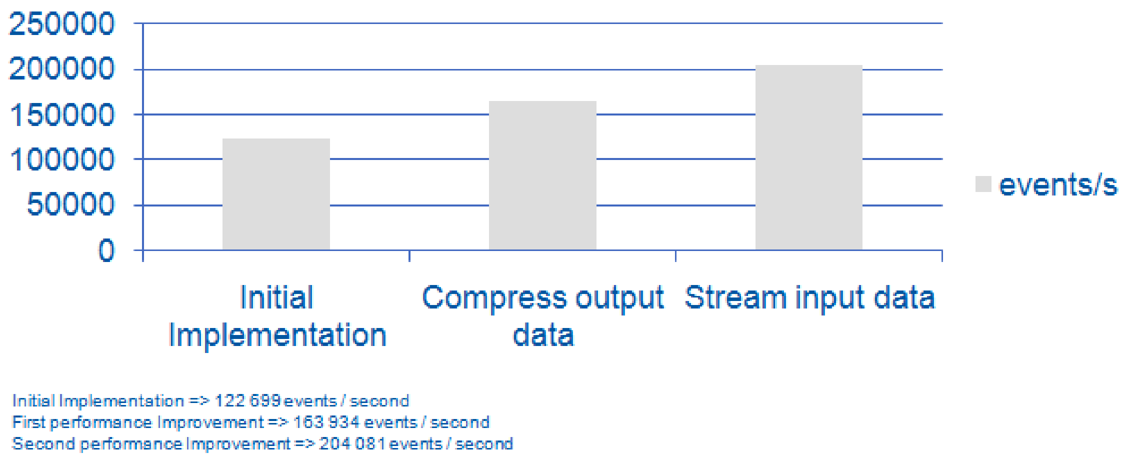


Figure 2 Calorimeter cell identification format

After a first functional version, work has been done on performance improvements. In the first performance improvement, the size of the output data has been decreased. The number of cells is not very big, only being maximum 6016 cells in case of using the electromagnetic calorimeter. These cells can be pre computed and stored in a vector on the software side and can be easily accessed using a known index. The FPGA will compute the index for every calorimeter cell and this way only 32 bits can be sent for each cell, 16 bits for the adc value and 16 bits for the cell identification number. This has improved performance by 33 % as less data is written to the main memory.

The second improvement was to eliminate the unused space when sending the event data to the FPGA. Instead of sending fragmented information, aligned so that can be easily accessible by the FPGA, all the event data was streamed to the FPGA in the original format. An additional finite state machine was implemented to process the stream and select data from each front end card which is then decoded as before. The performance improved by 66 % compared to the initial implementation and also reduced the CPU usage needed to align the data. The performance results of all three versions are illustrated in Figure 3.



Initial Implementation => 122 699 events / second
First performance Improvement => 163 934 events / second
Second performance Improvement => 204 081 events / second

Figure 3 FPGA performance using different implementations

The implementation used 40 % of the FPGA logic resources. From that percentage, 30 % was used by the modules created by Intel for the address translation and for connecting the FPGA to the QPI bus. Also the memory usage was not high, multiple events being cached in the FPGA's memory with only using 29 % of the M20K memory blocks available on the Stratix V chip.

In order to measure the running time and efficiency of the FPGA implementation, some performance counters have been implemented inside the chip to track the number of clock cycles the Stratix V needs to decode one single event. The FPGA was able to decode one single event in only 650 clock cycles. The performance was compared against the C++ software on the Intel Xeon E5-2560v2 processor. The Xeon decoded the same event in 400 000 clock cycles which is a big difference compared to the FPGA.

Also tests have been done using 20 000 events. In this case, the running time of both versions was compared. As is illustrated in Figure 4, the FPGA was faster by a factor of 23 than the E5-2560 with one thread and faster by a factor of 2 than the Xeon using all cores.

2.2. Arria 10 implementation

The second platform which has been tested contains a Xeon E5-2560v4 CPU and an Arria 10 FPGA on the same package. The Arria 10 chip has 427200 Adaptive Logic Modules, a big improvement compared to the Stratix V. Another significant improvement is the bandwidth between the main memory and the FPGA. The interface used for memory access has a clock of 400 MHz instead of the 200 MHz frequency used on the Ivy Bridge platform, providing in theory double the bandwidth.

As more bandwidth is available with the Arria 10 platform, we observe substantial performance improvements compared to the Stratix V. But even with the increased bandwidth of the new Intel UPI interface, the bottleneck is still the write bandwidth to the main memory.

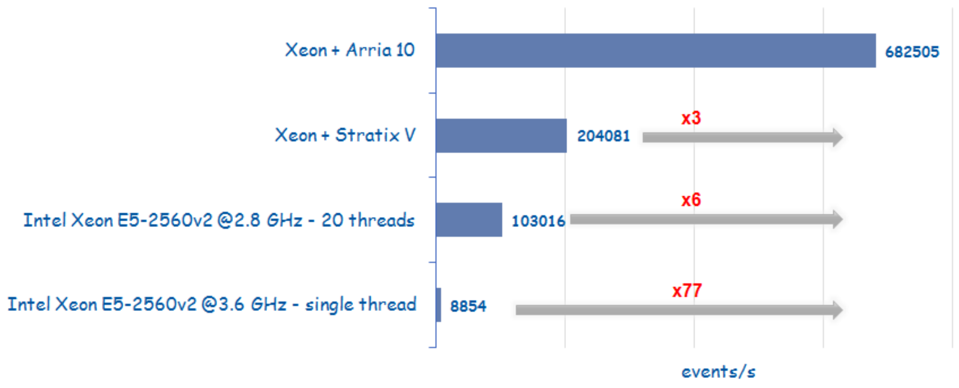


Figure 4 First results using the Xeon + FPGA computing platforms to decode ECAL raw data

3. Zero suppression

Doing the only the decoding inside the FPGA comes with a serious bottleneck. The data size after the decoding is increased by a factor between 2 and 3. This is not trivial for the decoding using the Intel Xeon and FPGA platforms. In our tests the write bandwidth was slower than the read bandwidth. In order to get the full power of the FPGA, some additional processing must be done to reduce the amount of data written.

After the decoding process, the LHCb trigger software performs a zero suppression algorithm which will eliminate the energy values that do not pass a given threshold. This is very beneficial for FPGA processing as the amount of data that needs to be written to the main memory is reduced by a significant factor. Also, for the electromagnetic calorimeter, a 2D zero suppression is performed. The 2D zero suppression includes the neighbours of the cells that passed the first step of the zero suppression. The energy from a neighbour cell will be compared to another threshold value which is lower than the main threshold value.

3.1. Stratix V implementation

The FPGA implementation was a little bit slow at first, being only a factor of 2.5 faster than the C++ algorithm running on a single Xeon thread. But after various improvements the performance increased by an impressive factor. Even since the first versions of the implementation, the QPI bandwidth was no longer a bottleneck. The limiting factor was the big number of clock cycles required to complete the zero suppression process inside the FPGA. In order to overcome this limitation, the design was modified to compute multiple events in parallel.

The data flow is presented in Figure 5. First, the FPGA makes read requests to the QPI. The QPI will send back read responses that are processed by the read finite state machine which will save the data into the FPGA memory. Then the raw data is processed by multiple decoding cores. The decoding finite state machine will extract from the stream of data the information for a single front end board. After the extraction, the data is sent to the decoding pipeline. All decoding cores share the same pipeline. The pipeline controller will decide which decoding core can use the pipeline using a simple round-robin algorithm. The pipeline performs the raw data decoding of all 32 adc values read by a front end board, in parallel. Also, the first step of the zero suppression is done on the pipeline where the adc values are compared with a threshold value.

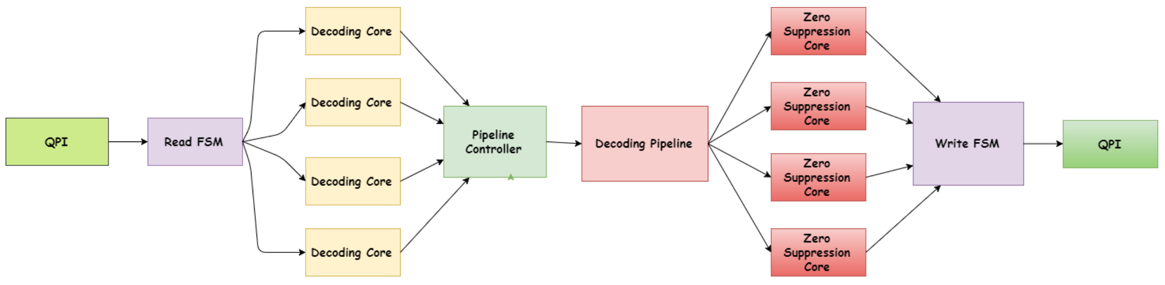


Figure 5 Stratix V FPGA 2D zero suppression data flow

It takes around 20 clock cycles for the decoded data to exit the pipeline. After the data processed in the pipeline, the next step is to perform the 2D zero suppression using multiple zero suppression cores. Because the zero suppression process is the slowest, inside each zero suppression core there are six threads processing the data for one event. If eight cores are used to process eight events in parallel then there are actually 48 threads processing the data. Finally, after the zero suppression process is finished in one of the cores, the data is written back to the main memory by the write state machine. The write state machine will check using a round-robin algorithm if there is data available to write from one of the zero suppression cores.

Because the write bandwidth bottleneck was eliminated, the FPGA is able to process more events per second compared to the decoding only version. For the benchmarks, the FPGA was configured to decode six events in parallel. A higher number can be achieved as there is still space left in the FPGA, but six cores is enough to saturate the FPGA-QPI bandwidth. The comparison results can be observed in Figure 7. The Xeon + FPGA implementation is a factor of **61.91** faster than one Xeon thread and **5.32** times faster than the Intel Xeon E5-2560v2 using all 10 cores.

In order to get the optimal performance when using the FPGA as an accelerator, it is better to process events in batches. If only one event is transferred to the FPGA, the computation is actually 0.03 % slower than using one single thread. Figure 6 illustrates the impact of the batch size to the acceleration factor that can be obtained when using the Stratix V FPGA. The number of events needed to reach optimal performance is not very high, with only about two thousand events that in total take only about 50 megabytes to store.

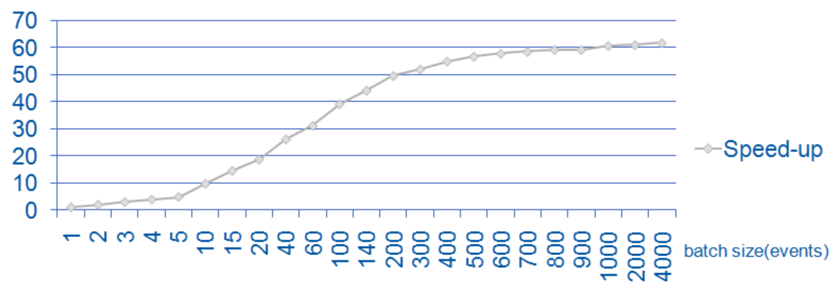


Figure 6 Batch size impact over the FPGA acceleration factor using the Stratix V FPGA

3.2. Arria 10 implementation

The first problem which was encountered when porting the design to the new FPGA was the clock speed. Because the logic was designed for a 200 MHz clock, the design was not able to meet the timming requirements for 400 MHz. In order to avoid changing the entire design to run at 400 MHz, which could have been very time consuming, only the interface has been modified to run at the higher frequency. More logic was added in order to benefit from the higher bandwidth. The updated data flow can be observed in Figure 7.

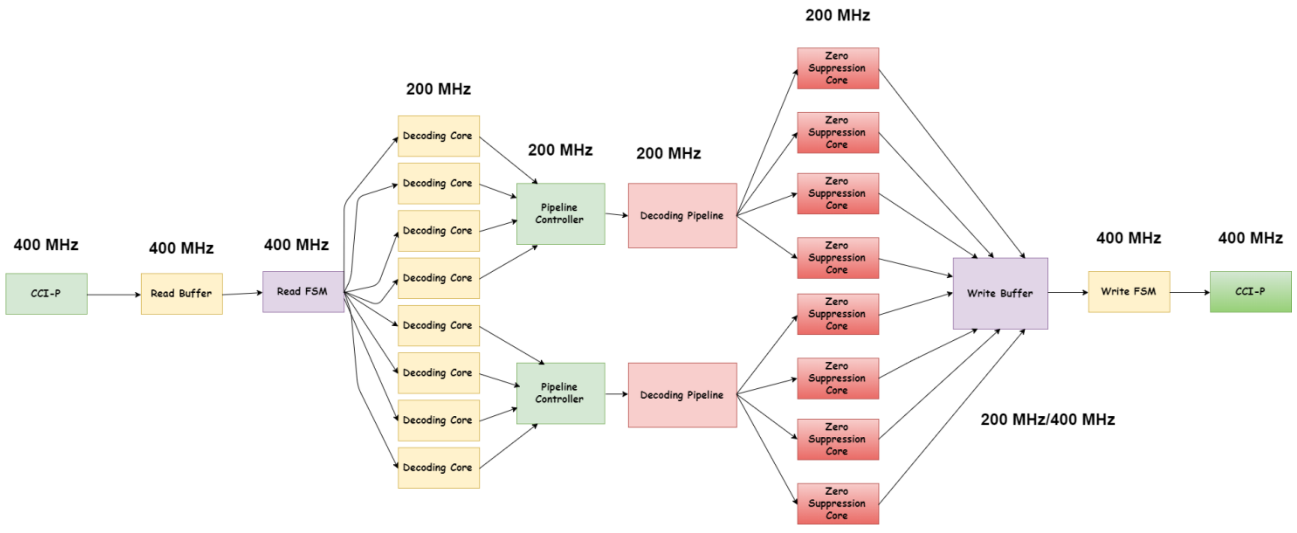


Figure 7 Arria 10 FPGA 2D zero suppression data flow

The benchmarks show a significant improvement. The bandwidth between the CPU and the FPGA is not fully used and a higher acceleration is possible by adding more cores or increasing the frequency of some particular design components.

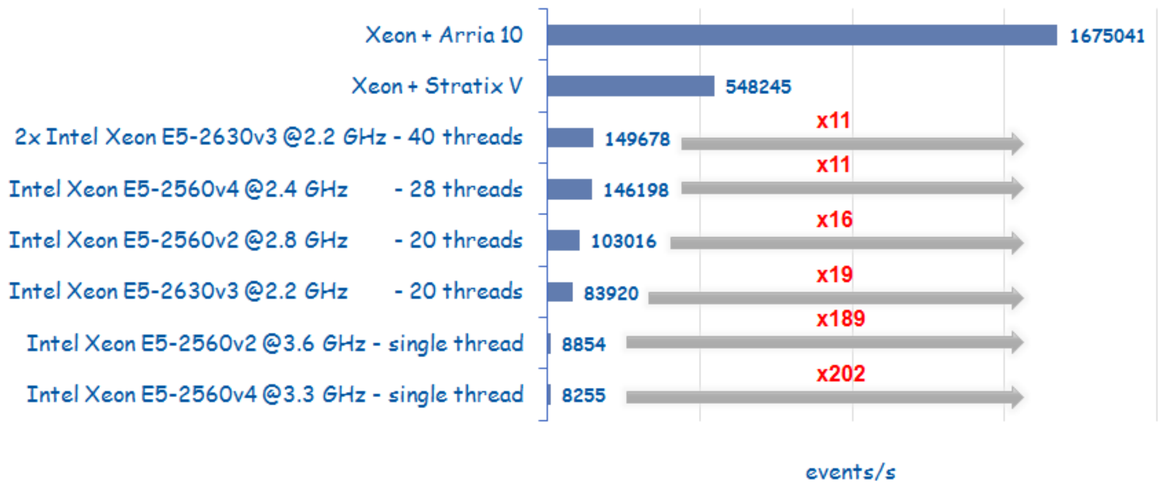


Figure 8 Performance gains using the Intel Xeon + Arria 10 platform when decoding ECAL data

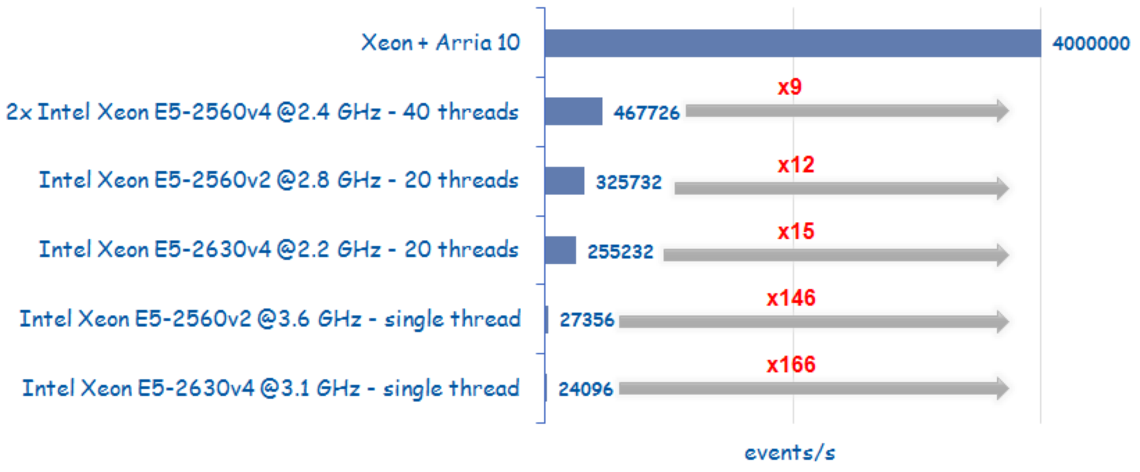


Figure 9 Performance gains using the Intel Xeon + Arria 10 platform when decoding HCAL data

The resource usage of both FPGAs is illustrated in Table 1. The Arria 10 FPGA has higher density and provides superior performance.

FPGA	ALMs used	Registers	Memory [bits]	Decoding cores/pipelines	Acceleration factor
Stratix V	136261/234720 (58 %)	145013	6,634,265 (13 %)	6/1	61
Arria 10	269120/427200 (63 %)	368843	20,518,928 (37 %)	14/2	189

Table 2 FPGA resource usage for 2D zero suppression

4. Cost comparison

FPGA devices provide significant computing speed and power efficiency when compared to CPUs but they are also more expensive. For the cost comparison, a two socket system was used for tests with Intel Xeon E5-2630v4 CPUs. An optimized version of the raw data decoding algorithm has been tested using all 40 threads of the system. Separate benchmarks have been done for both ECAL and HCAL data.

Type of data	Device	Number of events	Time[s]
ECAL	2x Xeon E5-2630v4	1,000,000,000	3659,62
HCAL	2x Xeon E5-2630v4	1,000,000,000	1000.23
ECAL	Arria 10	1,000,000,000	597,01
HCAL	Arria 10	1,000,000,000	239,20

Table 3 Intel Xeon E5-2630v4 benchmarks

With the combined speed of **214,599 events per second,** if we consider an event rate of **30 MHz** then **140** nodes are needed only for calorimeter raw data decoding.

In the FPGA case, given that the chip can process data at very high bandwidth, the price of the network equipment is also included in the total cost estimation. Also, if using PCIe cards, the bus limit has also been taken into account. In case of PCIe x8.0 cards, the number of nodes has been adjusted taking into account a limit of 38 Gbits/s per way. Considering both types of data, the combined processing rate is **1,195,871 events per second**, with only **25 FPGAs** required if there are no bandwidth bottlenecks. Note that the servers with FPGAs are only required to send and retrieve data from the FPGA so there is a possibility to further reduce the costs as the servers can be used for something else.

Data Type	Average Input Speed [Gbit/s]	Average Output Speed [Gbit/s]	Average Total speed [Gbit/s]
HCAL	48.97	35.84	84.81
ECAL	65.73	43.29	109.02
Combined	60.93	41.16	102.09

Table 4 FPGA transfer speeds

Device	Price per node [EUR] (FPGA + network + server)	Number of nodes	Total cost [EUR]
2x Xeon E5-2630v4	0 + 0 + 3500	140	490,000
Nallatech 385A PCIe 3.0 x16	6000 + 550 + 4000	25	263,750
Nallatech 385A PCIe 3.0 x8	5000 + 550 + 4000	38	362,900
Intel Skylake + Arria 10 (estimate)	0 + 550 + 8000	25	212,500

Table 5 Cost comparison between CPUs and different FPGA solutions

5. References

[1] “Raw Data Format”, **EDMS 565851**, updated 9 December 2005

[2] “The Readout of the LHCb Calorimeters”, **EDMS 527942**, 11 March 2008