SCUOLA
NORMALE
SUPERIORE

Classe di Scienze
Corso di perfezionamento in Fisica
XXXIV ciclo

A FPGA-based architecture for real-time cluster finding in the LHCb silicon pixel detector

Settore Scientifico Disciplinare **FIS/01**

Candidate:
Giovanni Bassi

Advisor:
Prof. Michael J. Morello

Academic year 2022–2023

Abstract

The data acquisition system of the LHCb experiment has been substantially upgraded for the LHC Run 3, with the unprecedented capability of reading out and fully reconstructing all proton–proton collisions in real time, occurring with an average rate of 30 MHz, for a total data flow of approximately 32 Tb/s. The high demand of computing power required by this task has motivated a transition to a hybrid heterogeneous computing architecture, where a farm of graphics cores, GPUs, is used in addition to general–purpose processors, CPUs, to speed up the execution of reconstruction algorithms. In a continuing effort to improve real–time processing capabilities of this new DAQ system, also with a view to further luminosity increases in the future, low–level, highly–parallelizable tasks are increasingly being addressed at the earliest stages of the data acquisition chain, using special–purpose computing accelerators. A promising solution is offered by custom–programmable FPGA devices, that are well suited to perform high–volume computations with high throughput and degree of parallelism, limited power consumption and latency. In this context, a two–dimensional FPGA–friendly cluster–finder algorithm has been developed to reconstruct hit positions in the new vertex pixel detector (VELO) of the LHCb Upgrade experiment. The associated firmware architecture, implemented in VHDL language, has been integrated within the VELO readout, without the need for extra cards, as a further enhancement of the DAQ system. This pre–processing allows the first level of the software trigger to accept a 11% higher rate of events, as the ready–made hit coordinates accelerate the track reconstruction, while leading to a drop in electrical power consumption, as the FPGA implementation requires $\mathcal{O}(50\text{x})$ less power than the GPU one. The tracking performance of this novel system, being indistinguishable from a full–fledged software implementation, allows the raw pixel data to be dropped immediately at the readout level, yielding the additional benefit of a 14% reduction in data flow. The clustering architecture has been commissioned during the start of LHCb Run 3 and it currently runs in real time during physics data taking, reconstructing VELO hit coordinates on–the–fly at the LHC collision rate.

Acknowledgements

The work presented in this thesis would not have been possible without the help and support of many colleagues, both at INFN and at CERN. One of the most valuable lessons that I learned during my entire PhD is the actual meaning of the word "collaboration", usually hidden or overlooked. Despite being a student participating in the commissioning of a HEP experiment, one of the most intensive and stressful periods for the entire physics community during the whole experiment lifetime, I have never felt alone and I had the unique opportunity to learn a lot while taking part in this effort. I am grateful to have met many brilliant physicists and engineers with whom I could share successes and failures.

Sincere gratitude goes to my advisor, Prof. Michael J. Morello, for his day–to–day supervision, guidance, encouragement, constructive feedback, and kind support throughout my PhD, well beyond his duties. Special thanks goes to Federico Lazzari for his time, patience, and guidance during the first months of my PhD when he taught me how to design, simulate, and test FPGA firmware. I am also very grateful to Prof. Giovanni Punzi for his insightful suggestions and straight–to–the–point discussions, also beyond strictly technical details. I also had the pleasure of working together with Luca Giambastiani and Daniele Passaro on the VELO clustering project, who gave proof of self–sacrifice while having me as "technical advisor" during their MSc thesis work. I would also like to thank the LHCb–Pisa research group in its entirety for the very nice moments spent both in Pisa and at CERN, which allowed me to expand my scientific horizon beyond my own initial research interests. I gratefully acknowledge the generous financial support of the Scuola Normale Superiore and of the Istituto Nazionale di Fisica Nucleare, which allowed me to spend the last year and a half of my PhD at the CERN laboratories. I would like to express my deepest gratitude to Dr. Karol Hennessy, Dr. Antonio Fernandez Prieto, and Dr. Guillaume Vouters who helped and guided me during the integration process of the clustering firmware within the VELO data acquisition chain, especially during the detector commissioning period. It was truly a privilege to work with them. A sincere thanks goes to the VELO team, the Real–Time Analysis team, and the whole LHCb commissioning team for the tight collaboration, support, and integration coordination, without which this work would not have been possible.

I could not have undertaken this journey without my family: my dad who always showed interest in what I do and like, my mum who was present and showed her love and support on all occasions, and my sister who helped me keep my feet on the ground. Words cannot express my gratitude to Silvia who has always been able to understand my own problems better and well in advance than me, showing patience while helping me go through difficult times.

Contents

List of Figures

List of Tables

Abbreviations

ALICE A Large Ion Collider Experiment

ALM Adaptive Logic Module

ALUT Adaptive Look Up Table

ASIC Application Specific Integrated Circuit

ATLAS A Toroidal LHC ApparatuS

BE Back–end Electronics

BXID Bunch Crossing(X) IDentifier

CMS Compact Muon Solenoid

DAQ Data Acquisition

DSP Digital Signal Processing

EB Event Builder

ECAL Electromagnetic CALorimeter

ECS Experiment Control System

EE EndEvent

EOP End Of Package

FE Front–end Electronics

FEB Front–End Board

FiFo First in First out memory

FPGA Field Programmable Gate Array

FSM Finite State Machine

GBT GigaBit Transceivers

GEC Global Event Cut

GWT Gigabit Wireline Transmitter

HCAL Hadronic CALorimeter

HDL Hardware Description Language

HEP High Energy Physics

HLT High Level Trigger

ICF Isolation Cluster Flagging

IP Impact Parameter

IPU Intelligence Processing Unit

L0 Level–0 trigger

LAB Logic Array Block

LE Latch Enable

LHC Large Hadron Collider

LHCb Large Hadron Collider beauty experiment

LUT Look Up Table

MaPMT Multi–anode PhotoMultiplier Tubes

MB Minimum Bias

MC Monte Carlo

MLAB Memory Logic Array Block

MUX MUltipleXer

NP New Physics

OPB Opto and Power Board

PCB Printed Circuit Board

PID Particle IDentification

PLL Phase–Lock Loop

PON Passive Optical Network

PV Primary Vertex

RAM Random Access Memory

RICH Ring Imaging CHerenkov

ROM Read Only Memory

SciFi Scintillating Fiber

SERDES SERializer–DESerializer

SiPM Silicon PhotoMultiplier

SM Standard Model

SOP Start Of Package

SP SuperPixels

TFC Timing and Fast Control

UT Upstream Tracker

VELO VErtex LOcator

Introduction

> *Cacambo, qui donnait toujours d'aussi bons conseils que la vieille, dit à Candide: Nous n'en pouvons plus, nous avons assez marché; j'aperçois un canot vide sur le rivage, emplissons–le de cocos, jetons–nous dans cette petite barque, laissons–nous aller au courant; une rivière mène toujours à quelque endroit habité. Si nous ne trouvons pas des choses agréables, nous trouverons du moins des choses nouvelles.*
>
> — Voltaire, *Candide*

The Standard Model (SM) of particle physics has proven to be one of the most successful scientific theories to date. Being completed in the seventies, it has been tested with high precision since, establishing itself as a great success of the reductionist paradigm in describing particles and fundamental interactions. Despite having passed all experimental tests, the SM is known to be an incomplete theory, lacking the description of gravity, dark matter, and dark energy. In this context, High Energy Physics (HEP) experiments aim at probing the SM to higher and higher precision, exploiting the full predictive power of the theory. Two main research paths are currently being followed, the direct search and the precision search. The direct search aims at producing new particles or unveiling new types of interactions that can lead to direct signs of physics beyond the SM. This approach led to the discovery of the Higgs boson in 2012 [1, 2], produced in $\mathcal{O}(10)$ TeV proton-proton collisions at the Large Hadron Collider (LHC) accelerator and observed by the ATLAS and CMS collaborations. However, the direct approach is limited by the collision energy in a scenario where more powerful accelerators are not foreseen in the near future and direct searches have yielded null results so far. For these reasons the precision frontier might represent the best opportunity to reveal non-SM dynamics in the next decade. On one hand this approach relies on comparing precise measurements with similarly precise SM predictions to highlight deviations from the theory. On the other hand, even if observables are characterized by relatively large theoretical uncertainties, performing measurements with ever-increasing experimental precisions allows the parameters of the underlying theoretical model to be over-constrained. The LHCb experiment installed at the LHC accelerator is one of the leading players in precision searches. Since its operations started in 2010, LHCb has been capable of collecting huge samples of charm– and bottom–hadron decays, providing key contributions to the field such as the first single–experiment observation of D^0 mixing in 2012 [3], the first observation of CP violation in the decay of D^0 mesons in 2019 [4], the measurement of $B_s^0 \to \mu^+\mu^-$ decay properties [5], precise determination of the β angle [6] and of the B_s^0 mixing frequency [7], first observations of penta– and tetra–quark states [8, 9], the detailed study of anomalies in $b \to sll$ transitions [10] and the world's most precise measurements of the γ angle [11].

Introduction

In order to improve its performance and keep up with the ambitious physics program, the LHCb experiment is currently planning a series of incremental upgrades that will allow it to collect a data sample consisting of 300 fb^{-1} by the end of Run 6, in 2038, sustaining an unprecedented instantaneous luminosity of 1.5×10^{34} $cm^{-2}s^{-1}$. The first step of this ambitious upgrade path has been recently completed. In fact, LHCb has just undergone a fundamental upgrade that involved both the detector and the trigger and data acquisition (DAQ) chain. The new DAQ system is based on trigger–less back–end electronics, reading events at the full 30 MHz LHC inelastic event rate, with an instantaneous luminosity of 2×10^{33} $cm^{-2}s^{-1}$ at Run 3, corresponding to a bandwidth of about 32 Tb/s. The subsequent event–building stage and software high–level–trigger (HLT) processing lead to a final data storage bandwidth of 80 Gb/s. The LHCb triggering process is divided into two main stages, named HLT1 and HLT2, where the first performs simpler and basic selections to reduce data rate by a factor of about 30, and it is executed by means of a farm of approximately 170 graphics cores (GPUs)[1] installed in the event builder servers. The latter executes more sophisticated and complex algorithms on a large farm of general–purpose CPU processors. The large amount of data that has to be processed upfront in real time challenges the entire acquisition system, and data handling is anticipated to be one of the key challenges for future runs with higher instantaneous luminosities. Therefore, a common effort is being made to address low–level and high–parallelizable tasks at early stages of the DAQ system, leveraging various highly specialized accelerator architectures. This approach goes by the name of heterogeneous computing and it allows the bandwidth and computing resource needs to be reduced, leaving more room for the higher–level data processing and selections, while significantly reducing the overall power consumption. LHCb is at the forefront of heterogeneous applications, having already deployed a grid of GPUs for triggering purposes in Run 3.

One of the most time consuming tasks that is still addressable at early DAQ stages is track reconstruction, where the information of multiple subdetector layers is processed to identify the trajectories of particles within the detector acceptance. Within LHCb, the "Artificial Retina" project is one of the most advanced R&D projects that aims at solving track reconstruction at the pre–event–building stage. In order to accomplish this ambitious goal, Retina employs a parallel, high–throughput, low–latency tracking architecture to be deployed on FPGA–based accelerators, capable of sustaining a 30 MHz input event rate. The first step to enable track reconstruction of the LHCb VErtex LOcator (VELO) pixel subdetector with the Retina algorithm consists in grouping contiguous active pixels inside the silicon vertex detector. The clustering process is the first key task to be performed before the actual tracking algorithms can go through the detector clusters and reconstruct tracks. Clustering is both time consuming, due to the two–dimensional geometry of the pixel detector, and highly parallelizable. In this context, this thesis reports the development, testing, and commissioning of a real–time two–dimensional clustering algorithm and the corresponding VHDL firmware that runs on the back–end FPGA–based boards of the LHCb silicon pixel detector. The algorithm has been fully integrated with the LHCb DAQ chain, allowing a significant improvement of the time needed to perform the online reconstruction for trigger purposes, while reducing the bandwidth required to output the vertex detector data. This is the first–ever implementation of a FPGA–based two–dimensional clustering algorithm that runs in real time at the unprecedented speed of 30 MHz.

[1] The current plan is to double the number of graphics cores during 2023 to maximize the experiment capabilities.

The thesis is structured as follows. Chapter 1 introduces the concept of heterogeneous computing, explaining why it is one of the most promising solutions for data–taking architectures of future envisioned HEP experiments. The key components of the LHCb upgrade are described in Chap. 2 on both the detector and the data acquisition side. Chapter 3 details the clustering algorithm, describing its functioning principles, parameters, and limitations, while its firmware implementation is accurately illustrated in Chap. 4. Chapter 5 goes through the firmware and software integration of the new algorithm within the LHCb infrastructure. Physics performances of the algorithm are instead detailed and compared to a full–fledged CPU–based software implementation of the clustering algorithm in Chap. 6. The key commissioning steps that led to a properly functional firmware within the LHCb DAQ chain are described in Chap. 7. The chapter also concludes the thesis by summarizing the achievements of this work, which is meant to be a fist step towards building a FPGA–based tracking unit running in real time, in the context of Future LHCb Upgrades.

Chapter 1

Heterogeneous computing in High Energy Physics

Starting from the Heavy Flavor Physics research case, this chapter introduces the concept of trigger and data acquisition (DAQ) in modern High Energy Physics experiments. In particular, as the High–Luminosity LHC (HL–LHC) era approaches, heterogeneous computing seems to be one of the most promising and viable options to handle ever–increasing data complexity, bandwidths, and power consumption. Example applications of the heterogeneous computing paradigm are discussed with a specific focus on the LHCb Retina project, which led, as a first stage of a more complex tracking unit, to the development and implementation of the FPGA–based two–dimensional clustering algorithm, which is the main topic of this thesis.

1.1 Probing the Standard Model: Heavy Flavor Physics

Despite its numerous achievements in explaining a wide variety of phenomena at their most fundamental level [1, 2, 12–15], the Standard Model (SM) of particle physics is far from being considered a complete theory. Instead, it is usually described as a low–energy limit of a more comprehensive model. Arguments typically used to motivate the fact that the SM might be an effective theory with a limited validity range are the baryon asymmetry of the universe, the lack of dark matter and dark energy descriptions and a coherent merging with General Relativity. These and other observations motivate the search for New Physics (NP), aiming at a high–energy completion of the SM. NP models, such as Supersymmetry and extra dimensions models, can introduce additional forces, particles or sources of symmetry but they all need to view the SM as a low–energy effective theory, given its success in explaining electromagnetic, weak, and strong interactions. Two main paths are currently being explored when it comes to NP research: the direct search that aims at revealing new particles by directly producing them in high energy collisions and the precision search that looks into indirect evidence of NP phenomena by measuring with extreme precision fundamental processes and comparing the observations with SM predictions.

One of the most promising ways of probing the SM via precision searches consists of exploiting its flavor structure. The term flavor is used to identify fields sharing the same spin and quantum numbers. Interactions being able to distinguish between them are the main focus of flavor physics. Within current understanding, quarks, leptons, and neutrinos are observed to be organized into

three flavors each (families or generations). However, their flavor structure, which, unlike other forces, is not determined by an invariance of the theory under symmetry transformations, might be a clue to NP at higher energies. The existence of the three generations, together with the unexplained pattern of masses of quarks and leptons, spanning several orders of magnitude, is referred to as the SM flavor puzzle and is one of the central mysteries of particle physics. Both quarks and leptons have flavor and mix between families, however the matrices describing the couplings between different flavors, the Cabibbo–Kobayashi–Maskawa (CKM) [16,17] matrix for quarks and the Pontecorvo–Maki–Nakagawa–Sakata (PMNS) [18,19] for neutrinos, have different structures. On one hand the CKM matrix is highly hierarchical, with diagonal elements having magnitude close to unity, and progressively smaller off–diagonal elements. On the other hand, the PMNS matrix has much larger off–diagonal elements. Moreover, flavor physics shows a wide and diverse variety of phenomena, currently obeying the SM predictions, that can be measured to test these predictions. Examples are mixing and *CP* violation in both the charm and bottom sectors and b quark decays that allow transitions between all three quark families to be probed, while being suppressed within the SM and hence representing potential sensitive channels for NP. This plethora of observables allows very high mass scales to be probed, well beyond the center–of–mass energy of any current or planned future collider [20]. This high mass–scale sensitivity is directly related to the SM flavor structure, which strongly suppresses flavor–changing neutral–current (FCNC) processes. In this respect flavor physics has two main objectives: understanding the family structure and its properties and studying the wide variety of decays of different flavor species to probe deviations from SM expectations with a multiplicity of approaches. The richness of decays of different flavor species available to experimental observations allows deviations from SM expectations to be probed with a multitude of approaches. Precise theoretical computations and predictions are fundamental to determine the key parameters of the SM to be probed at the experimental level. However, also observables characterized by bigger theoretical uncertainties are key to over–constrain the couplings of possible new degrees of freedom within new models.

The previous considerations should suggest that flavor physics is a very powerful tool to search for NP, also at energy scales much higher than those accessible through direct–type searches, and it should motivate the design and operations of dedicated experiments such as Belle II at SuperKEKB and the Large Hadron Collider beauty experiment (LHCb) at LHC. Belle II and LHCb have the unique potential to unveil new physics by confirming hints of deviations from the SM that have been recently observed in $b \to s\ell^+\ell^-$ and $b \to c\tau\nu$ transitions or finding new unexpected outcomes in the study of rare and forbidden decays such as $b \to s\nu\nu$ or $B_{(s)} \to \ell^+\ell^-$. While being described in Chap. 2, it is worth saying that LHCb has proven to be an ideal laboratory for flavor physics in the past ten years of operations. LHCb main achievements include the discovery of the ultra–rare decay $B_s^0 \to \mu^+\mu^-$ [21–24], the first single–experiment observation of charm mixing [3], the world's most precise measurements of the CKM angle γ [11] and the first evidence of *CP* violation in c–hadron decays [4]. Over the past years, several anomalies have emerged from flavor experiments that could potentially point to NP. The most prominent among the anomalies are the hints for lepton flavor universality violation (LFUV) in the charged current $b \to c\ell\nu$ transition, $R_{D^{(*)}} = \mathcal{B}(B \to D^{(*)}\tau\nu)/\mathcal{B}(B \to D^{(*)}\ell\nu)$, where $\ell = e, \mu$. The $R_{D^{(*)}}$ measurement, together with updated observations in neutral current $b \to s\ell^+\ell^-$ transitions, $R_{K^{(*)}} = \mathcal{B}(B \to K^{(*)}\mu^+\mu^-)/\mathcal{B}(B \to K^{(*)}e^+e^-)$, which recently showed agreement with the SM [25,26], clearly indicates the need for more data to better understand the origin of anomalies. For this purpose, LHCb has just completed a major upgrade (Upgrade I) that allows the experiment to operate at a luminosity of 2×10^{33} cm^{-2}s^{-1} during Run 3 and 4 data taking periods (2022–2030), five times higher with respect to Run 1 and 2 operations (2010–2018). By

the end of Run 4, a data sample of about 50 fb^{-1} is expected to be collected. Furthermore, an Upgrade II is currently being planned and will allow LHCb to take data at a luminosity of 1.5×10^{34} $\text{cm}^{-2}\text{s}^{-1}$, aiming at collecting 300 fb^{-1}, by the end of Run 6. Figure 1.1 shows the instantaneous and integrated luminosity projections for LHCb between Run 1 and Run 6. Jumps in luminosity can be observed corresponding to the start of Run 3 and Run 5, when Upgrade I and Upgrade II are planned to be installed, respectively. This upgrade path will greatly improve

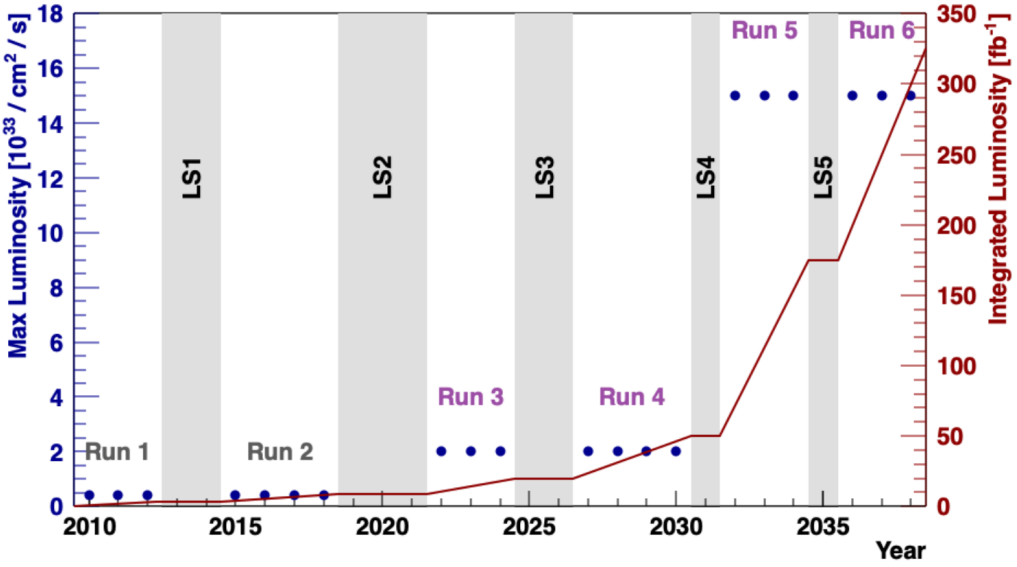

Figure 1.1: Instantaneous (blue dots) and integrated (red line) luminosity projections for LHCb between Run 1 and Run 6, referring to left and right vertical scales, respectively [27].

the sensitivity of many flavor studies, many of which are still statistically limited, and will allow us to increase our knowledge on highly suppressed processes, while being able to distinguish between different NP scenarios. Table 1.1 shows current and extrapolated uncertainties on some key flavor observables in the current and future upgrades of LHCb, motivating its ambitious physics program. On one hand weak decays of all the quarks and the CKM matrix are key probes of deviations of SM expectations and already constrain reasonably well the number of generations. On the other hand, rare decays seek to uncover new physics manifesting itself through interference with SM diagrams. Searches for charged lepton flavor violation examine the nature of lepton flavor. Whether in the B, D or K systems, comparisons of decay rates into different lepton species challenge the notion of lepton universality, strongly implied by the SM.

1.2 Trigger and Data Acquisition towards the HL–LHC era

Fulfilling the ambitious LHCb flavor–physics program at a 1.5×10^{34} $\text{cm}^{-2}\text{s}^{-1}$ peak luminosity poses several challenges, both on the detector and on the data handling and triggering sides. A direct consequence of the increase in luminosity is the higher number of interactions per event, which is expected to be around 40 during Run 5 operations[1]. In order to maintain comparable detector performance with respect to Run 3 at higher luminosities, the new detector should have a higher granularity, while reducing the amount of material. With such a high pile–up,

[1]The average number of interactions during Run 3 is around 7.6.

Observable	Current LHCb (up to 9 fb^{-1})	Upgrade I (23 fb^{-1})	Upgrade I (50 fb^{-1})	Upgrade II (300 fb^{-1})
CKM tests				
$\gamma\,(B \to DK, etc.)$	4°	1.5°	1°	0.35°
$\phi_s\,(B_s^0 \to J/\psi\phi)$	32 mrad	14 mrad	10 mrad	4 mrad
$\lvert V_{ub}\rvert/\lvert V_{cb}\rvert\,(\Lambda_b^0 \to p\mu^-\bar{\nu}_\mu, etc.)$	6%	3%	2%	1%
Rare Decays				
$\mathcal{B}(B^0 \to \mu^+\mu^-)/\mathcal{B}(B_s^0 \to \mu^+\mu^-)$	69%	41%	27%	11%
$S_{\mu\mu}\,(B_s^0 \to \mu^+\mu^-)$	—	—	—	0.2
$A_{\rm T}^{(2)}\,(B^0 \to K^{*0}e^+e^-)$	0.10	0.060	0.043	0.016
Lepton Universality Tests				
$R_K\,(B^+ \to K^+\ell^+\ell^-)$	0.044	0.025	0.017	0.007
$R_{K^*}\,(B^0 \to K^{*0}\ell^+\ell^-)$	0.12	0.034	0.022	0.009
$R(D^*)\,(B^0 \to D^{*-}\ell^+\nu_\ell)$	0.026	0.007	0.005	0.002
Charm				
$\Delta A_{CP}\,(D^0 \to K^+K^-, \pi^+\pi^-)$	29×10^{-5}	13×10^{-5}	8×10^{-5}	3.3×10^{-5}
$A_\Gamma\,(D^0 \to K^+K^-, \pi^+\pi^-)$	11×10^{-5}	5×10^{-5}	3.2×10^{-5}	1.2×10^{-5}
$\Delta x\,(D^0 \to K_S^0\pi^+\pi^-)$	18×10^{-5}	6.3×10^{-5}	4.1×10^{-5}	1.6×10^{-5}

Table 1.1: Run 1/2 and extrapolated uncertainties on some key flavor observables at the current and future LHCb upgrades [27].

the use of precision fast–timing information becomes essential to cope with the combinatorial background. Furthermore, the detector and its front–end readout electronics should cope with higher radiation damage throughout the entire data–taking period.

In addition to detector and front–end challenges, data handling and online processing represent one of the main technical difficulties in successfully operating LHCb at ever increasing luminosities, where the trend toward increasing bandwidth is a common theme among all major HEP experiments, as shown in Fig. 1.2. Therefore careful R&D and appropriate choices on Trigger and Data Acquisition (DAQ) systems are pivotal for the success of the LHCb physics program. DAQ systems have always been part of the HEP field since the beginning, and their importance is particularly clear nowadays, where detectors produce large amounts of data that generally exceed the capabilities of any practical permanent storage system. As beam crossings at the LHC occur at 30 MHz and for each beam crossing many pp collisions occur, thousands of tracks traverse the detector. However, not all the collisions in a crossing and not all the tracks in a collision event are interesting for physics studies. For these reasons, several methodologies are applied to regulate and reduce the data flow to a manageable level, by deciding which parts of the detector information should be acquired and saved. The trigger needs to be highly efficient in selecting processes for physics analyses, while providing a large rate reduction from unwanted high–rate processes. It also needs to be robust and highly flexible to react to changing conditions.

Modern DAQ systems make decisions to permanently record a particular collision event in a sequence of stages, with progressively decreasing data flow, and increasing computational cost and complexity. Usually, the lowest–level trigger decision is based on low–complexity algorithms using a limited amount of event data, whereas the last level before permanent storage is usually performed by CPU systems running high–level code, the performance and complexity of which are close to the offline analysis. During the first LHC era, the overall DAQ bandwidth

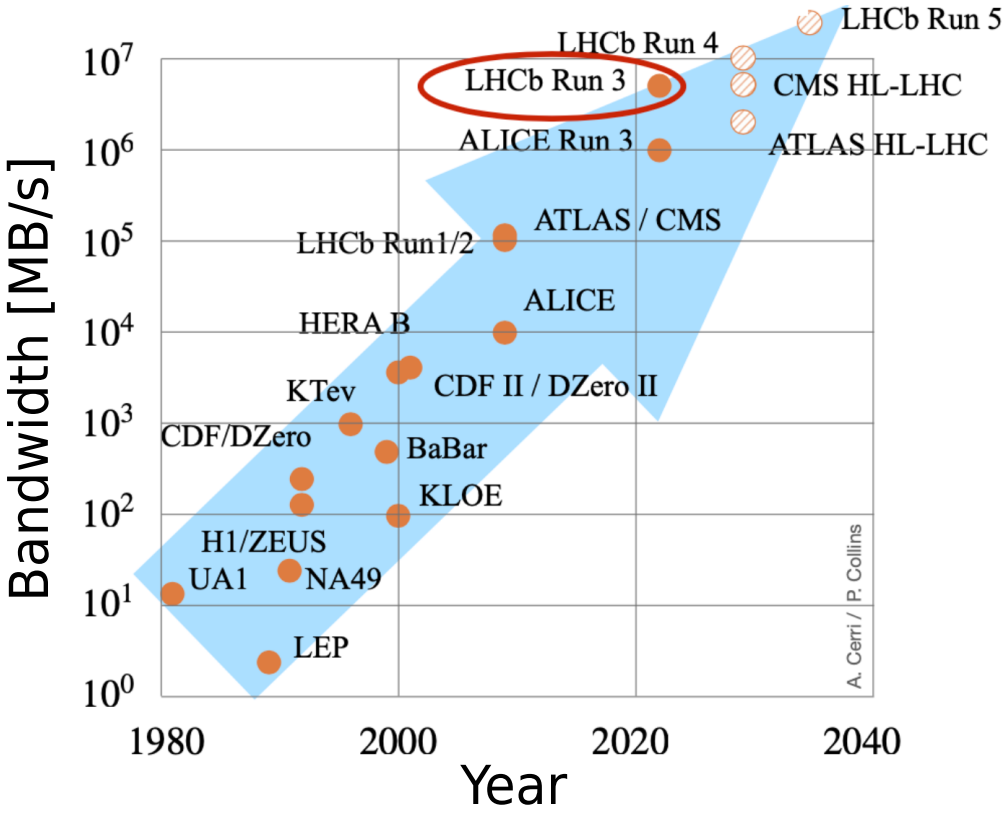

Figure 1.2: Evolution of the data bandwidth requirements over time of HEP experiments. The current LHCb Run 3 conditions are highlighted [28].

has increased, taking advantage of telecommunication technology rapid developments driven by fast internet growth, which allowed large data handling systems to be built at reasonable prices, using commercial off–the–shelf technologies (COTS). As COTS were adopted, the use of custom–developed electronics became limited to on–detector and front–end electronics, whereas intermediate trigger levels have been absorbed in the High Level Trigger implemented on commercial CPUs. However, as we move towards the High–Lumi LHC era, data handling and availability of cost–effective computing resources are getting increasingly challenging. Moreover, Moore's law slowing down poses limits on the role played by consumer CPUs, which have provided flexible computing power at low cost up to now. This is compounded by the fact that data handling demands in HEP have been growing at an even faster rate than consumer electronic technology. As a consequence, DAQ systems might represent a major cost item and technical limitation to future HEP experiments performances [29].

Taking LHCb Run 5 as an example, at an instantaneous luminosity of 1.5×10^{34} cm^{-2}s^{-1}, the detector is expected to produce up to 400 Tb/s of data, which need to be processed in real time and reduced by roughly four orders of magnitude before being saved to permanent storage. As a first challenge, the resulting number of serial data links needed to read out the whole detector would be prohibitively huge. Taking as a reference the GigaBit Transceivers (GBTs), designed and built within CERN, which are capable of sustaining a 4.8 Gb/s transmission rate,

the number of GBT links that would be required[2] to read out the whole detector if running at 400 Tb/s would be of the order of 8×10^4. Moreover, with heavy–flavor hadrons being present in all bunch crossings, with around 40 interactions each, one of the challenges to face within DAQ will be to move from a traditional trigger strategy of selecting bunch crossings based on generally interesting signatures, like displaced vertices, which will be no more effective to a more complex online event selection based on the complete information of the event.

In this context, there is the increasing need for more effective and specialized solutions that can perform real–time reconstruction at early data acquisition stages. This will allow the amount of data to be sent to general–purpose computing farms to be reduced more effectively than simple hit zero–suppression, while being able to reconstruct the charged particle trajectories as early as possible in the processing chain, so that triggering on specific heavy–flavor signatures can be performed in an optimal way. Moreover, this will lead to a significant reduction in the time needed to perform the reconstruction in subsequent processing stages, since part of the reconstruction duties is offloaded to earlier stages, specialized in parallel and efficient processing.

1.3 Heterogeneous computing: concept and applications

Heterogeneous computing is one of the most promising solutions to tackle high–data–volume–related issues, described in the previous section. Large farms of versatile and general–purpose CPUs are coupled with specialized hardware like FPGAs and GPUs, usually referred to as accelerators. Accelerators are very fast at executing heavily parallelizable tasks, leaving to CPUs only the more complex ones. Architectural heterogeneity is currently one of the main topics being developed and deployed in industry, from high–performance to cloud computing. Deploying different types of accelerators suited for different types of workload improves both performance and energy efficiency. High–throughput, low–latency interconnections between CPU and accelerators allow the different elements to operate as a single unit, while maintaining a high degree of scalability and flexibility. Accelerators are especially well suited for the development of artificial intelligence through machine vision and deep learning, where vast amounts of data need to be processed. When deploying a heterogeneous system, one of the main challenges to face in order to exploit its full potential is to be able to leverage each compute unit with user–friendly programming models that can be applied across different platforms. Another challenge is related to the effective sharing of data between accelerators and an accelerator and the host system. During recent years, several solutions have been proposed to tackle these challenges when leveraging extremely heterogeneous systems [30–33].

The heterogeneous computing model is currently being applied in HEP experiments, following the ongoing industry trend. Some implementation examples at the LHC include the following:

- ALICE deployed heterogeneous computing infrastructures already during Run 1 and 2 performing the tracking of the Time Projection Chamber (TPC) on GPUs at around 1 kHz of Pb–Pb collisions. During Run 3, ALICE performs the full raw data processing in the online computing farm in software at 50 kHz Pb–Pb interaction rate, where the dominant part of the real–time processing, the TPC tracking, is run fully on GPUs [34];

- ATLAS has been investigating possible solutions to cope with the foreseen increase in the collision rate by approximately a factor of 5 during HL-LHC operations, with respect to the current LHC operation. The earliest trigger stage needs to provide a higher rejection factor

[2] LHCb Run 3 readout system requires about 9×10^3 optical data links.

in a more difficult environment. Studies have been performed based on the architectures developed within the Fast TracKer and L1Track projects [35, 36]: track reconstruction is based on associative memories, where linear fitting algorithms are run within FPGAs;

- CMS has explored several R&D paths in terms of heterogeneous computing, one of the most relevant projects aims at reconstructing charged particle trajectories within the hardware–based trigger system, implementing a novel approach using an all–FPGA solution, as the HL–LHC era approaches [37–39]. Another advanced CMS project in this context is Patatrack [40] that aims at reconstructing charged particle trajectories in CMS pixel detector exploiting GPU power. Within this project, it has been demonstrated that low–power GPUs can be used for track reconstruction purposes, at a fraction of the cost of a single CPU node, previously deployed;

- LHCb Upgrade represents a unique opportunity to test new technologies for high luminosity, during LHC Run 3, even before ATLAS and CMS High Luminosity Phase II. The upgraded LHCb uses a triggerless readout chain that sends raw detector hits to a custom data processing center at the full 30 MHz LHC bunch crossing rate, enabling an unprecedented flexibility for trigger selections. However, the corresponding data rate (32 Tb/s) is so high that the LHCb collaboration moved from a more traditional CPU–based farm to a heterogeneous computing system, based on GPUs, to perform the High Level Trigger first stage (HLT1). Reconstruction algorithms were optimized for many–core architectures and integrated into a compact, modular, and scalable framework, called Allen [41, 42].

As demonstrated by these examples, heterogeneous trigger and computing farms are now becoming a reality in the HEP field. HEP can profit from industry advancements to deploy accelerator solutions that are becoming increasingly powerful and inexpensive. However, it is worth noting that this will make HEP dependent on a market of which it is not a driver, and future market choices may not be HEP–friendly. HEP will therefore need to adapt to future market trends or, in case its needs cannot be met by such trends, take early action to develop its own solutions.

1.4 The LHCb Retina project

Given the ongoing trend towards heterogeneous computing architectures, the LHCb collaboration has identified the accelerator–related R&D program as a critical element to ensure that trigger and reconstruction algorithms are optimally designed to take full advantage of the most effective and affordable architectures, as we move toward the HL–LHC era. To this end, LHCb has, already in Run 3, implemented a Coprocessor R&D Testbed [43], already in Run 3, to investigate new architectures, based on accelerators, that will be developed, optimized and commissioned in the context of future high–luminosity runs. The Testbed allows different R&D projects to cooperate in a common effort of processing as much data as possible at low data acquisition levels to limit the data flow to be sent to the High Level Trigger[3] stage, while reducing computing resources and power consumption needs. This is done exploiting different technologies like GPUs, FPGAs, and IPUs, while dealing with different types of algorithms and data handling, including neural networks, machine learning, and pattern matching.

[3]The LHCb–Upgrade trigger system receives data at the full 30 MHz LHC bunch crossing rate and it is divided into two main stages, named High Level Trigger 1 (HLT1), performed on GPUs, and High Level Trigger 2 (HLT2), performed on CPUs. More details about the LHCb High Level Trigger can be found in Sect. 2.10.

One of the main actors in the Testbed R&D activity is the "Artificial Retina" project [44], a parallel, high–throughput, low–latency tracking algorithm inspired by the natural vision of mammals' brains. It has been specifically designed for track reconstruction by a massive and intelligent pattern matching, exploiting the FPGA architecture capabilities (see Appendix A). While GPUs have been the de facto choice for accelerating compute–intensive operations, FPGAs have been gaining ground because their reconfigurable fabric can be tailored to a specific application or domain. FPGAs bridge the gap between CPU processors that are flexible and relatively easy to program, and ASICS that offer optimal performance at a high power efficiency, while being less flexible. FPGAs can potentially address I/O and memory access issues that have been one of the traditional bottlenecks of GPU acceleration, since FPGAs are characterized by a versatile and diverse connectivity, covering several widely–spread protocols, such as PCIe, Ethernet and Serial Lite, while allowing for highly–configurable connections both to other devices and to the host.

Figure 1.3 shows the track–reconstruction processing steps performed by the Retina architecture. Retina implements a set of cells, each sensitive to hits belonging to a reference

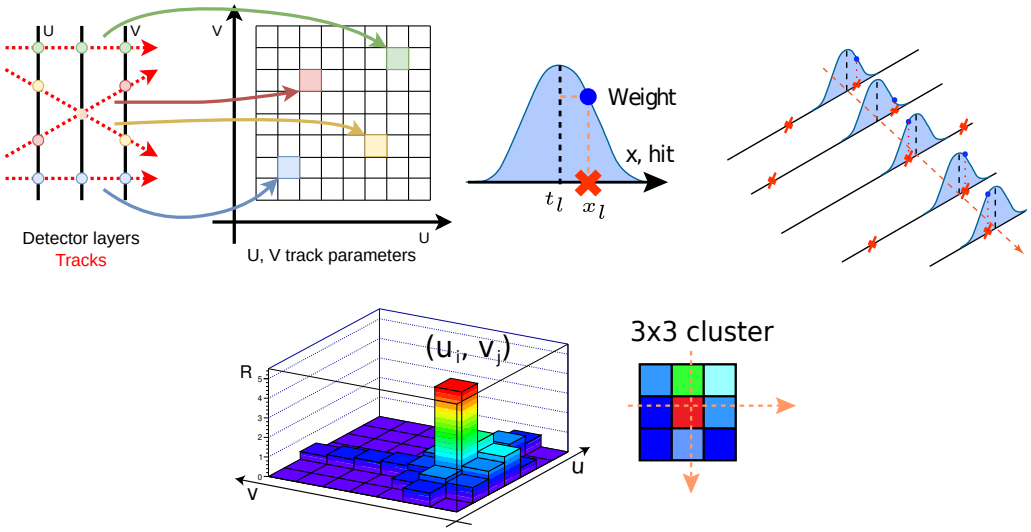

Figure 1.3: Track–reconstruction processing steps performed by the "Artificial Retina". Reference tracks and their interceptions with the detector layers are mapped into a matrix of cells (top left). Cells compute and sum up the Gaussian weight of the hits x_l (top–right). Tracks are reconstructed finding local maxima in the cell space. Track parameters are computed interpolating the activation level of the 3×3 clusters around the maxima (bottom).

track, where the response of a cell is proportional to how close the hits are to the reference track. Cells compute and sum up the Gaussian weights of the hits and tracks are reconstructed finding local maxima in the cell space, accordingly. Tracks corresponding to cells with a higher activation level are the tracks that were most likely present in the event. Unlike traditional reconstruction systems where bandwidth is progressively reduced during processing, Retina requires the bandwidth to increase significantly at some point, because multiple copies of the same data are produced and reach different cells, shrinking down only at a later stage where few tracks are reconstructed from many hits. These features of the Retina architecture make FPGAs the natural devices where to implement it, given their large internal bandwidth and high degree of parallelization. Two main components are needed to implement the Retina architecture: the

engine and the distribution network. A set of engines implements the weight and sum mechanism, where each engine corresponds to a cell of the track parameter space, and works in parallel with respect to the other engines. Since cells are spread over several FPGA chips to overcome FPGA size limitations and given that detector hits need to be collected from different sources, a distribution network that exchanges hits between different boards is needed. In order to take full advantage of the algorithm high–throughput, while significantly reducing the data bandwidth, the Retina architecture needs to be deployed at pre–build stage, before the proper build stage, where data are collected by readout boards and exchanged between servers to build complete events. In addition to data exchange through the event builder network, data are also sent from the server memory to tracking boards that implement the Retina architecture. Detector hits are then exchanged between boards by means of an optical network built using optical transceivers, fibers, and an optical patch panel. Reconstructed tracks are then sent from the tracking boards back to server memory and added to the data to be built. Figure 1.4 shows a high–level view of the Retina architecture being implemented within the event builder servers. Further details

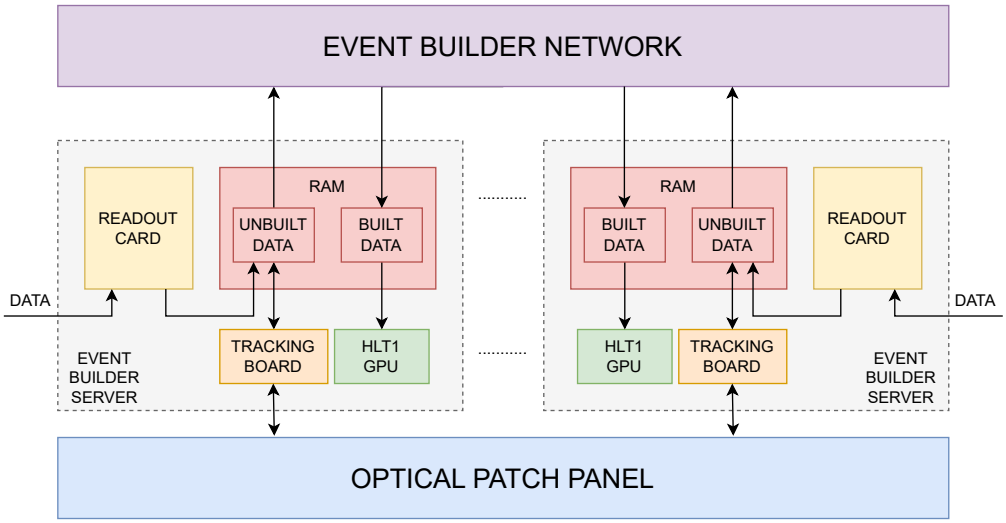

Figure 1.4: Scheme of the data flow between readout cards, Retina tracking boards, the event builder network, and HLT1 GPUs.

about the Retina architecture can be found in Ref. [45]. Retina is an architecture of rather general applicability and can be used to reconstruct tracks from hits of detectors characterized by different technologies and geometries. Over the years, several detailed studies have been performed to characterize and tune the Retina algorithm for different applications within the LHCb detector [45–49]. Within the LHCb Coprocessor R&D Testbed, Retina aims at building and successfully operating a full–size demonstrator capable of reconstructing tracks from the LHCb silicon pixel vertex locator (VELO), at the full 30 MHz input event rate. Figure 1.5 shows a study on VELO tracking efficiency (see Sect. 6) comparing the Retina response with a CPU–based tracking algorithm. This and other in-depth studies have demonstrated that High Level Trigger performances are only negligibly affected when running on tracks produced by FPGAs. In addition to the VELO application and in the context of future Runs at higher luminosities, as presented in both the Expression Of Interest for a Phase–II LHCb Upgrade [50] and the Framework TDR for the LHCb Upgrade II [27], the "Artificial Retina" is considered a viable option to reconstruct tracks downstream of the magnet at the earliest trigger level

(see Sect. 6.3). Reconstructing these tracks will allow long–lived s–quark final states (K^0_s, Λ) decaying outside the VELO to be identified. This is crucial to increase the acceptance for many important bottom and charm decay modes, such as $D^0 \to K^0_s K^0_s$, $B^0 \to K^0_s K^0_s$ and $\Lambda_b \to 3\Lambda$. Moreover, performing downstream reconstruction at early data acquisition stages will allow the time needed for long–track reconstruction and for the overall HLT1 processing to be significantly reduced.

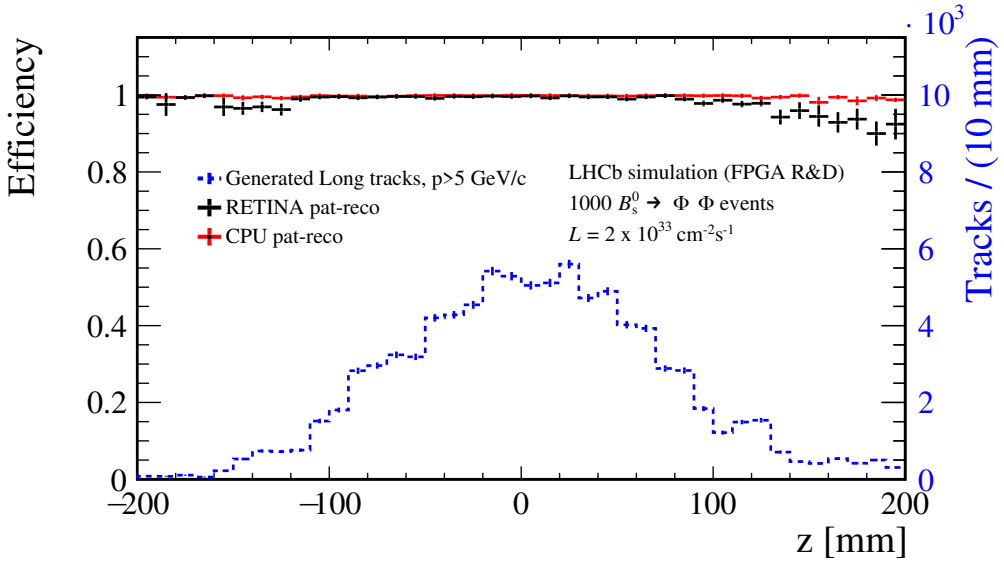

Figure 1.5: Comparison between VELO tracking efficiencies obtained with the CPU algorithm and the Retina pattern recognition algorithm. The efficiency is shown as a function of the z coordinate of the track origin vertex. 1000 $B^0_S \to \phi\phi$ simulated events at LHCb Upgrade conditions are used [49].

1.5 LHCb VELO clustering

For the purpose of applying the Retina architecture to the reconstruction of VELO detector tracks, I have developed, implemented, optimized, and commissioned a new low–latency, high–throughput two–dimensional cluster–finding algorithm and the corresponding firmware. Cluster reconstruction, which consists in grouping contiguous active pixels, is the first step in the chain of operations to perform an effective VELO tracking, regardless of the architecture in which tracking itself is carried out. Inheriting the Retina approach, the clustering algorithm, described in the subsequent chapters of the thesis, is meant to be run at the very early stage of the data acquisition system on FPGA–based VELO back–end readout cards, processing pixel hit coordinates at an unprecedented speed, exceeding the bunch crossing frequency of 30 MHz, for the first time. This approach is beneficial in terms of computing resources, as a significant task is removed from the HLT sequence, bandwidth reduction, as preprocessed data are sent out from readout cards to the event building stages, and electrical power consumption, as FPGAs are power efficient devices.

Chapter 2

The LHCb Upgrade experiment

This chapter describes the LHCb experiment in the current Upgrade I configuration, focusing on the systems closely related to the design and deployment of the clustering architecture, namely the VELO subdetector and the data acquisition and trigger system. The underlying motivations for the LHCb Upgrade I and an overview of the other subdetectors are also given.

2.1 Why upgrade LHCb?

The LHCb experiment [51] is one of the four large detectors at the Large Hadron Collider (LHC) accelerator facility at CERN, and it was designed to search for new physics through *CP*–violation studies and rare decays of heavy–flavor hadrons. LHCb has been successfully operated from 2010 to 2018 during the LHC Run 1 (2010–2012) and Run 2 (2015–2018)era, and has demonstrated excellent performance [52], collecting 9 fb^{-1} of data. Between 2018 and 2022, LHCb underwent a major update, called Upgrade I, to cope with an instantaneous luminosity of 2×10^{33} $\text{cm}^{-2}\text{s}^{-1}$, five times higher than that of Run 1 and 2, and thus with a corresponding increase in the number of interactions[1]. In addition to replacing most of the subdetectors, the front–end electronics and the data–acquisition system were completely renewed to read out and process the complete information of all subdetectors at the full LHC inelastic event rate of 30 MHz.

Despite the considerable 9 fb^{-1} data set collected during Run 1 and 2, as discussed in Chap. 1 and in Ref. [53], the precision on many of the key flavor physics observables measured by LHCb remains statistically limited. In order to probe the Standard Model even more precisely and thus get the required sensitivity to observe possible new–physics effects, significantly larger data sets are needed, and this can be obtained only by increasing the instantaneous luminosity of physics collisions. The LHCb Run 1–2 detector and data acquisition chain were designed and built with the aim of affording luminosities of about 2×10^{32} $\text{cm}^{-2}\text{s}^{-1}$ and did not allow the experiment to significantly increase the size and purity of the collected physics samples in case of a large increase in instantaneous luminosity, as foreseen during the operations of Run 3 (and beyond). The maximum allowed readout rate of the tracking subdetectors was about 1 MHz and, consequently, the High Level Trigger (HLT) was designed and sized to process not more than this amount of data. In order to increase efficiency on the most interesting physics channels, a hardware Level–0 trigger (L0) stage [54] was used, during Run 1–2, to reduce the event rate to

[1]The corresponding average number of *pp* interactions per bunch crossing increases from 1.1 to 7.6, moving from Run 2 to Run 3.

1 MHz, by exploiting the information from either the Calorimeters or the Muon System, which were, together with the small upstream VELO PileUp system [55], and later the HeRSChel subsystem [56], the only subdetectors which could be read at the beam crossing rate. However, as luminosity increases, simple selections based on transverse energy deposits will no longer allow hadronic signals to be efficiently disentangled from background and reduce the rate to 1 MHz, as previously done at lower luminosities, as shown in Fig. 2.1.

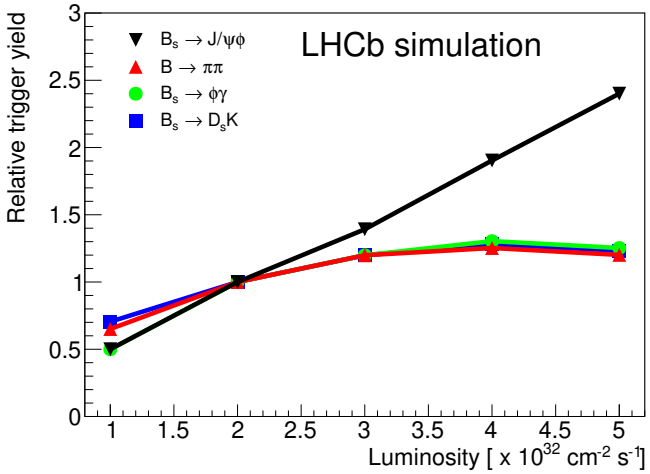

Figure 2.1: Relative trigger yields as a function of instantaneous luminosity, normalised to 2×10^{32} cm^{-2}s^{-1} [57].

In order to take full advantage of a higher luminosity and to collect significantly more physics data per fb^{-1} of integrated luminosity, the L0–hadron trigger stage has been removed in favor of a more powerful HLT, capable of applying selections that are more discriminating than simple inclusive criteria exploiting only calorimeter information. Such selections require a detector that can be read at the full 30 MHz LHC inelastic event rate and a High Level Trigger capable of processing events at the full rate and discriminating signal channels over background, based on the full event reconstruction in real time. The increase in instantaneous luminosity (by a factor of five) and the improved trigger efficiency for most decay modes (by a factor of two) will lead to an order of magnitude increase in the annual yields in most channels with respect to the previous LHCb experiment. A total integrated luminosity of around 23 fb^{-1} is expected by the end of Run 3 and 50 fb^{-1} by the end of Run 4.

For this purpose, the LHCb subdetectors and readout electronics have been redesigned and are now able to read events at 30 MHz rate and cope with the larger event multiplicity [58]. The entire software and data processing stack also needed to be upgraded to handle the expected large increase in data volume.

2.2 Overview of the LHCb Upgrade detector

LHCb is a single–arm forward spectrometer covering the pseudorapidity range $2 < \eta < 5$. It is located at interaction point 8 on the LHC ring, approximately 100 m underground. Figure 2.2 shows the layout of the upgraded detector. LHCb subdetectors are devoted to two main tasks: tracking [60, 61] and particle identification [62]. Particle tracking is performed by means of the vertex locator (VELO), an array of silicon pixel subdetectors surrounding the pp interaction region,

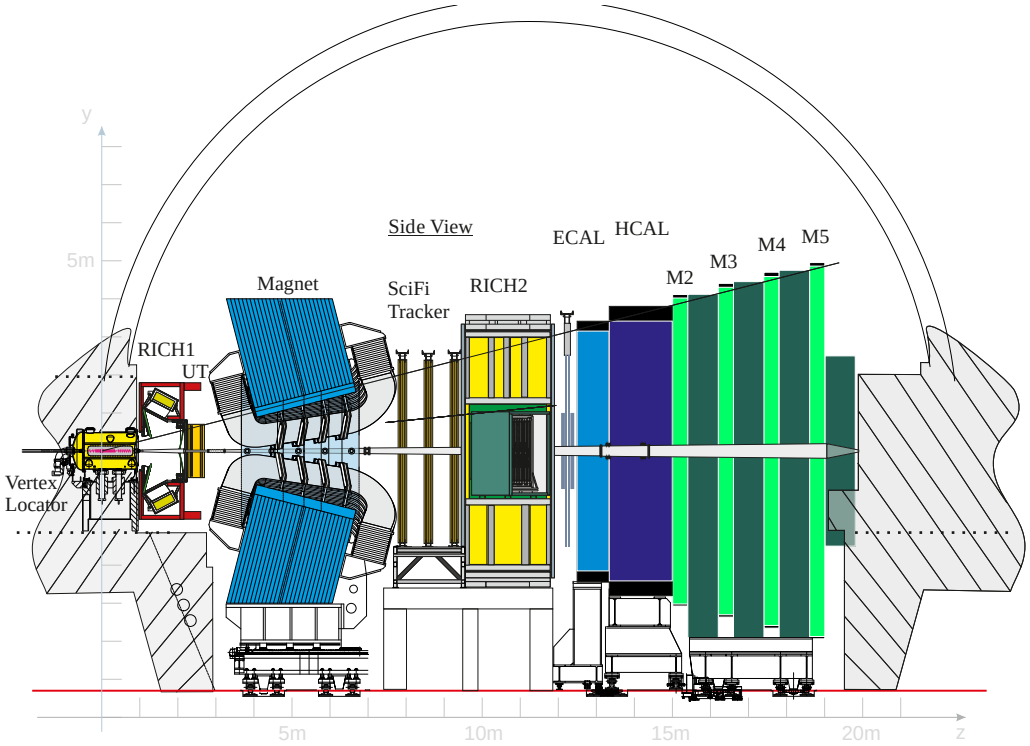

Figure 2.2: Layout of the LHCb Upgrade I detector. The Vertex Locator (VELO), the Upstream tracker (UT) and the Scintillating Fiber tracker (SciFi) are the subdetectors responsible for tracking, whereas first and second Ring Imaging Cherenkov subdetectors (RICH1 and RICH2), the electromagnetic (ECAL) and Hadronic (HCAL) calorimeters and the Muon system (M2–M5) perform the particle identification [59].

the silicon–strip upstream tracker (UT) in front of the dipole magnet, and three scintillating fiber tracker (SciFi Tracker) stations downstream of the magnet. VELO, UT and SciFi subsystems are detailed in Sects. 2.3, 2.4 and 2.5, respectively. Particle identification is performed by two ring imaging Cherenkov subdetectors (RICH1 and RICH2), a shashlik–type electromagnetic calorimeter (ECAL), an iron–scintillator tile sampling hadronic calorimeter (HCAL), and four stations of muon chambers (M2–5) interleaved with iron shielding. RICH1 and 2, ECAL, HCAL, and the muon subsystems are described in Sects. 2.6, 2.7 and 2.8, respectively. Between the UT and the SciFi the spectrometer's dipole magnet is placed, providing a vertical magnetic field with a bending power of about 4 Tm. It is a warm magnet designed to match the detector acceptance, with a vertically and horizontally increasing pole gap moving towards the downstream tracking stations.

The data acquisition chain (DAQ) starts at the detector level with front–end (FE) electronics boards that amplify and shape the signals generated within the subdetectors. The chain continues towards the back–end (BE) electronics located in a data center on the surface and connected to the FE in the cavern by 250 m–long optical fibers. Data are then handled by the event–builder and the event–filter farms. Sects. 2.9 and 2.10 detail the DAQ and trigger system, respectively. A fast signal control system (TFC) distributes clocks and fast commands to FE and BE electronics, which are configured and monitored using the experiment control system (ECS).

2.3 Vertex locator

The main goal of the vertex locator (VELO) [60] is to precisely measure tracks produced by ionizing particles, providing the positions of primary and secondary vertices. Its main physics design requirements are a track reconstruction efficiency greater than 99% and an impact parameter resolution[2] of the order of 10 μm. Physics requirements, together with mechanics, vacuum, and cooling constraints, led to the design of the VELO subdetector shown in Fig. 2.3. The 26 VELO layers are placed perpendicularly to the z axis between −287.50 mm and 750 mm, where the origin of the LHCb coordinate system is represented by the nominal interaction point. Each layer consists of two modules, each made of four sensors. A sensor is bump–bonded to three VeloPix [63] ASICs (chips), as shown in the right part of Fig. 2.3. The two sides of the

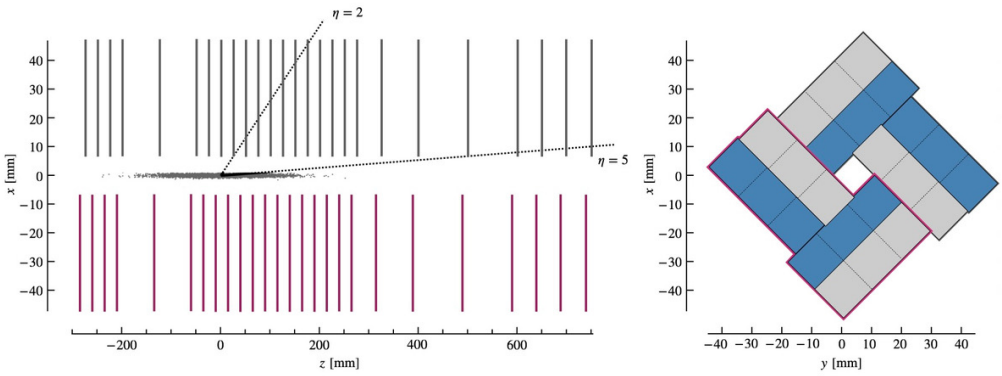

Figure 2.3: (left) $z - x$ schematic view of the VELO layers. The luminous region and the $2 < \eta < 5$ LHCb acceptance are also displayed. (right) $y - x$ schematic view of a VELO layer with the ASICs layout. Grey sensors are placed on the module upstream face whereas blue sensors on the downstream face. Purple–highlighted modules are on the C side [59].

VELO, 26 modules each, are movable in order to be retracted from the beamline under all conditions other than stable beams. Each half moves independently in the horizontal direction from a −29 mm retraction position from the beamline to +4 mm over–closure. VELO is based on a hybrid silicon pixel detector technology, where the radius of the closest active pixel edge is 5.1 mm from the beamline. The core component of the VELO is the pixel tile that is composed of a pixellated, planar silicon sensor and three pixellated ASIC chips. Each VeloPix ASIC is bounded to the sensor and provides analog signal processing and digitization. Each ASIC has an active matrix of 256×256 pixels, each 55 μm×55 μm in size, fabricated in TSMC^{TM} 130 nm CMOS process. When a pixel–hit is recorded, it is time–stamped, labeled with the pixel address, and sent out from the ASIC, without the need for a trigger signal. In order to reduce the data bandwidth out of the subdetector by a factor of approximately 30%, 2×4 neighboring pixels are grouped into SuperPixels (SP), avoiding duplication of the time stamp and address field. Four serial links carry up to 20.48 Gbit/s of data out of the ASIC by means of a low power custom serializer, named gigabit wireline transmitter (GWT) [64], that has been designed for the VELO use case. The silicon sensors, together with cooling, powering, readout, and mechanical support, are brought into a single unit, the VELO module. Figure 2.4 shows the main components of a VELO module. The module has a double–sided structure, where two tiles are glued on each face. Each VeloPix ASIC triplet is connected to the FE electronics that transmits control signals

[2]Track reconstruction efficiency and impact parameter resolution are defined in Sect. 6.

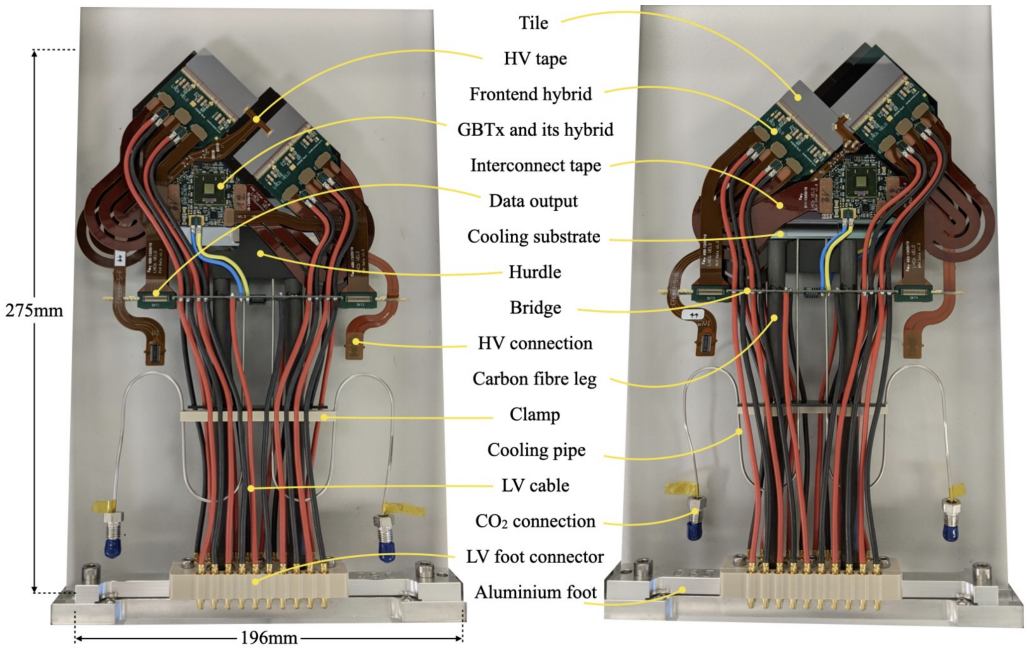

Figure 2.4: Faces of a VELO module located (left) upstream and (right) downstream. The main components are indicated [59].

and routes their data out. The VELO electronics system transports data from the ASICs to the off–detector processing units, while delivering clock and control signals to the modules, as well as low and high voltages to power the electronics and bias the sensors, respectively. As shown in Fig. 2.5, VELO electronics is located in three places: on the subdetector module, immediately outside the VELO vacuum vessel, and off the subdetector, with dedicated cabling running between them[3]. Starting from the on–module electronics, the ASIC reads the analog signals from the sensor and sends binary SP data over serial links. Data routing, together with clock distribution, control, and power, is managed on the FE by hybrid circuits. There are two types of hybrids: the first type provides the FE electronic interface to each VeloPix, connecting to the ASIC periphery. The second type houses the GBTx chip and distributes timing and fast control signals to the ASICs via the FE hybrids. The serial data from the hybrids are transmitted out of the vacuum on high–speed serial links, through a vacuum feedthrough board to the opto and power board (OPB). The vacuum feedthrough board also carries control signals to and from the FE, low, and high voltages. The OPB has DC/DC converters that provide the power needed by the module. It also performs the electrical–to–optical conversion of the 20 high–speed data links from each module and the bidirectional electrical–to–optical conversion for the three control links, one for each side of the subdetector module and one for the OPB itself. The OPB hosts a GBTx ASIC that decodes the high–speed control links, interfacing with two gigabit transceiver slow control adapter (GBT–SCA) ASICs. The final components in the VELO readout chain are the PCIe40 cards, hosted in servers in the data center. PCIe40 cards can be configured with different firmwares to fulfill different tasks (see Sect. 2.9). Two types of PCIe40 cards are directly involved in the VELO DAQ, the TELL40 and the SOL40.

[3]Cables connecting the detector module with the OPB are less than one meter long, whereas optical fibers connecting SOL40s and TELL40s cards to the OPB are roughly 100 m long.

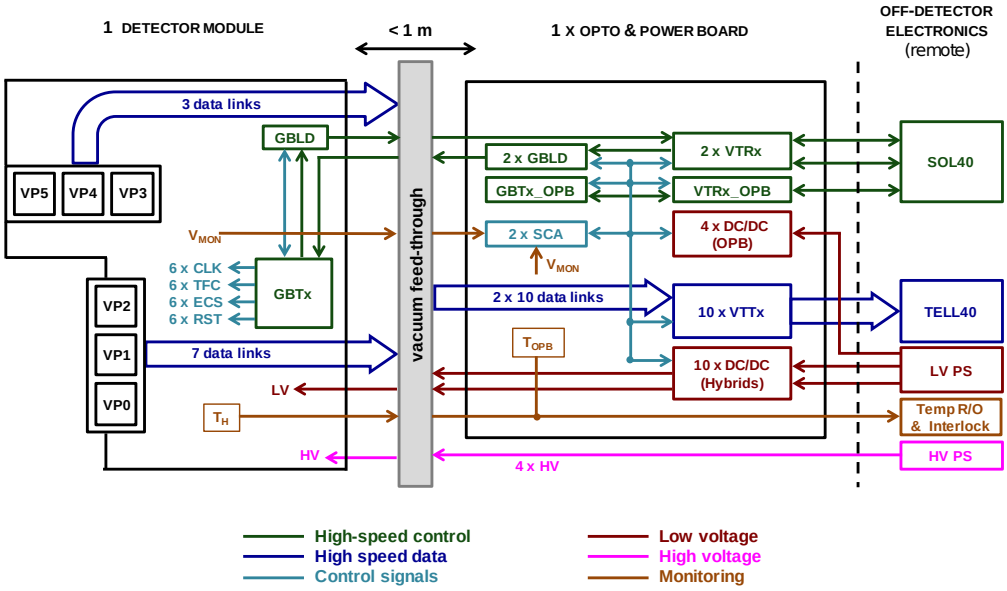

Figure 2.5: Block diagram of the VELO electronics [65].

The OPB sends the data to the TELL40 data acquisition cards whilst receiving control signals from the SOL40 readout supervisor. PCIe40 cards with the SOL40 firmware are used to control and monitor 13 VELO subdetector modules each, over a total of 39 bidirectional optical fiber links. This firmware provides the trigger and fast control commands and the protocols for communication with and via the GBTx and GBT–SCA chips. Four SOL40–flavored PCIe40 cards and 52 TELL40–flavored cards, one per module, are used to control and collect data from the entire VELO subdetector. TELL40 cards process the data from the 20 high–speed data serial links of each module. A detailed description of the VELO DAQ firmware is given in Sect. 5. For further details on the LHCb data acquisition and control chain, see Sect. 2.9.

The remaining main component of the VELO subdetector is cooling. Given that silicon sensors operate in vacuum, the heat generated (up to 40 W per module) must be dissipated by conduction. Furthermore, the silicon must be permanently cooled to below $-20°C$, to mitigate the effect of radiation damage. An evaporative cooling system has been implemented, with bi–phase CO_2 flowing through microchannels within the silicon substrate of the module.

2.4 Upstream tracker

The Upstream Tracker (UT) subdetector [61], based on a silicon microstrip subdetector technology, performs charged–particle tracking, allowing an efficient and fast track matching between VELO and SciFi. This is achieved by exploiting the fringe magnetic field between VELO and UT, so that a first estimate of momentum and charge can be performed. The UT information allows also the number of reconstructed fake tracks due to VELO–SciFi track segments mismatching to be reduced by limiting the search window in which those segments should be looked for. Furthermore, UT measures long–lived particles such as K_S^0 and Λ that decay after the VELO. Given these objectives, the UT has been designed and built so that it covers the entire acceptance of LHCb

without gaps, while providing high hit efficiencies and purities[4]. Moreover, the occupancy over the entire subdetector surface needs to be below a few percent to effectively reconstruct tracks both at small radii, where particle density is at its highest, and on the outer parts of the subdetector. The previous objectives and constraints lead to a subdetector geometry characterized by four silicon subdetector planes organized in two stations, as shown in Fig. 2.6.

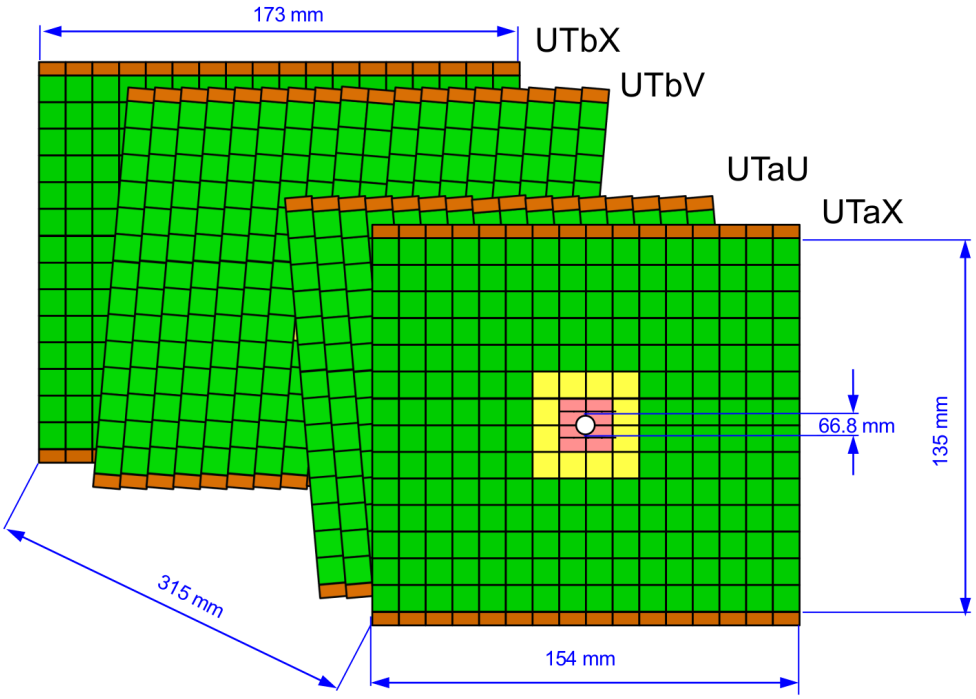

Figure 2.6: Schematic view of the four UT planes [59].

The building blocks of the subdetector are the vertical units, called staves, that carry the silicon sensors. The first station has a straight layer (UTaX) with vertical strips and a stereo layer (UTaU) with strips inclined by 5°. Both layers are made of 16 staves each. The second station is similar, with first a stereo layer (UTbV) with opposite inclination, and a straight layer (UTbX). Both layers contain 18 staves each. To ensure full coverage in the vertical direction, the sensors are arranged on both sides of the staves to obtain a vertical overlap. Similarly, a z–staggered arrangement of the staves with horizontal overlaps facilitates full horizontal coverage. Staves provide the mechanical support for the sensors and FE electronics, as well as active cooling. Since occupancies and radiation fluences span different orders of magnitude, four different types of sensor are used, with different strip pitches and lengths. The core component of the UT readout is a 128–channel ASIC, called Silicon ASIC for LHCb Tracking (SALT), built on 130 nm CMOS process. The main components of the SALT ASIC are an analog processor and a low–power, fast 6–bit ADC per channel, followed by digital signal processing, data formatting, and serializer blocks. The 6–bit SAR ADC converts the analog signal to the digital domain, and its output is processed by a DSP block, which performs operations such as pedestal subtraction and zero suppression. The UT hybrids connect the microstrip silicon sensors to the readout ASICs, distributing control signals, and routing data from the ASICs to the periphery electronics

[4]A high hit efficiency ensures that more than 99% of charged particles leave hits in at least three UT planes. High hit purity is achieved minimizing spurious hits, for example due to noise.

processing interface units, which are responsible for readout and control of the subdetector. UT hits are collected by TELL40 cards that organize them into packets which are then decoded by HLT. Similarly to VELO, charged particles can leave hits in more than one strip, therefore a one–dimensional clustering algorithm is run to consolidate neighboring hits into single–hit clusters.

2.5 Scintillating fiber tracker

The Scintillating Fiber (SciFi) [61] tracker located downstream the LHCb dipole magnet is responsible for charged particle tracking and momentum measurement. SciFi is based on scintillating fiber technology with a silicon photomultiplier (SiPM) readout that covers the entire LHCb acceptance. These design choices were made to achieve a single hit position resolution better than 100 μm and a single hit reconstruction efficiency better than 99%. SciFi is also designed to have a low enough occupancy so that the hit efficiency is not impacted. The base elements of the SciFi subdetector are 250 μm diameter plastic scintillating fibers arranged in multi–layered fiber mats. Figure 2.7 shows a sketch of the SciFi geometry with 12 detection planes arranged in three stations (T1, T2, T3) with four layers each in an X–U–V–X configuration.

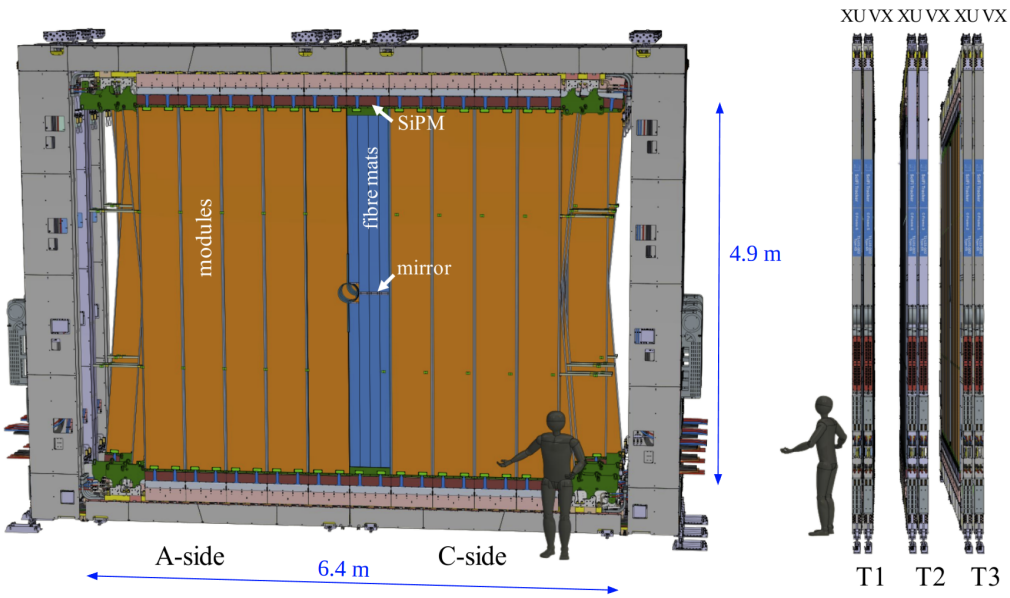

Figure 2.7: SciFi tracker (left) front and (right) side views [59].

External X layers and inner U and V layers have their fibers oriented vertically and rotated by $\pm 5^\circ$ and are responsible for measuring the horizontal and vertical deflections of charged particle tracks caused by the field of the magnet, respectively. Each station is built from modules, where a module contains eight mats made of six staggered layers of fibers. Each of the 12 layers is characterized by a very low material budget, accounting for less than 1% of a radiation length. The photons from the scintillating fibers are detected by arrays of SiPMs. The FE electronics consists of three types of boards. The PACIFIC board performs the digitization of the analog signals received from the SiPM. It contains a 64–channel ASIC with analog input and digital output, implemented in a 130 nm CMOS process. Each channel contains an analog processing,

digitization, slow control, and digital output. PACIFIC output data are sent to the Clusterisation board that performs zero–suppression and clustering, grouping neighboring channels from the same SiPM array into clusters and calculating the cluster position. Cluster data from the Clusterisation board are serialized on the Master board and shipped over optical fibers. The Master board is also responsible for the control and clock signal distribution, monitoring, as well as the distribution of low and high voltages.

2.6 RICH

The main goal of the Ring Imaging Cherenkov (RICH) [62] subdetector is to discriminate charged hadrons, namely pions, kaons, and protons, representing a crucial component of the LHCb flavor physics program. The RICH system performs the hadron particle identification (PID) in the 2.6–100 GeV momentum range. Some of the key benefits of the RICH are the capability to distinguish between final states of otherwise identical topologies ($B^0_{(s)} \to \pi^+\pi^-$, K^+K^- and $K^+\pi^-$), to reduce the combinatorial background in decay modes with hadrons in the final state ($B^0_s \to \phi\phi$, where $\phi \to K^+K^-$) and to perform the flavor tagging of B^0_s mesons, relying on charged–kaon identification from the $b \to c \to s$ decay chain. RICH consists of two subdetectors, RICH1 and RICH2, as shown in Fig. 2.8.

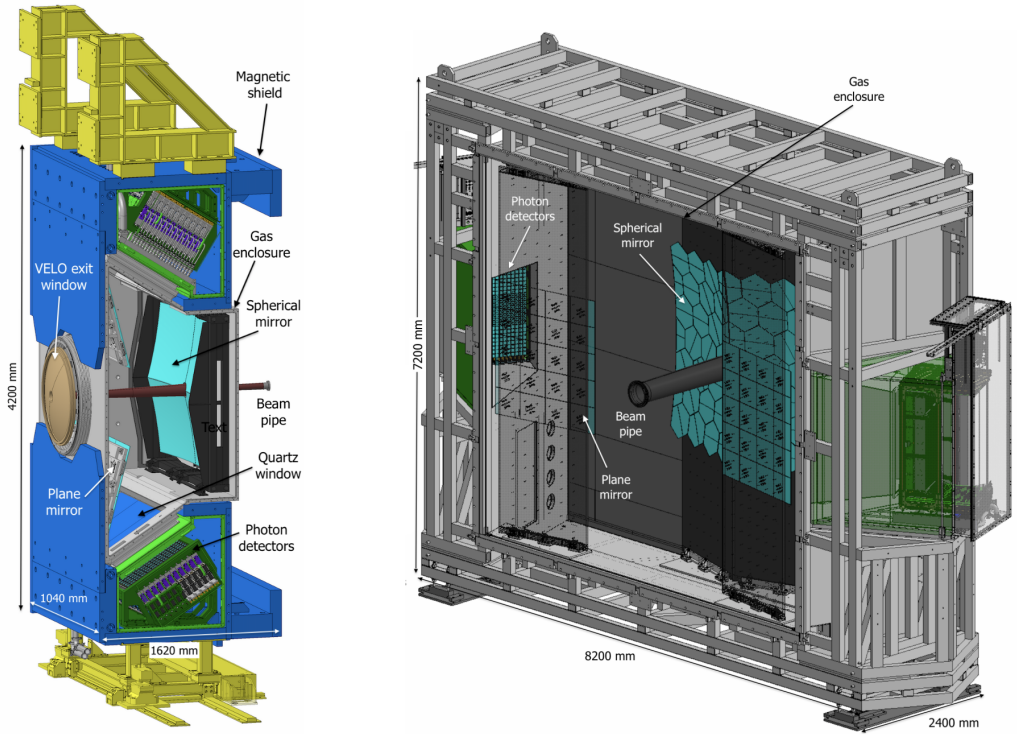

Figure 2.8: Schematic view of (left) RICH1 and (right) RICH2 subdetectors [59].

In both subdetectors the Cherenkov light produced inside fluorocarbon gaseous radiators is reflected outside the LHCb acceptance using a system of spherical and planar mirrors, focusing the ring images onto the photon subdetector planes. RICH1 is located upstream of the dipole magnet and employs a C_4F_{10} gas radiator, providing PID in the 2.6–60 GeV momentum range.

RICH2, located downstream of the magnet, is designed to provide PID for higher momentum particles, between 15 and 100 GeV, with a CF_4 gas radiator. To cope with the highly non–uniform occupancy, ranging from about 30% in the central region of RICH1 down to 5% in the peripheral region of RICH2, the photon detection planes are subdivided into two regions having different granularity. The multi–anode photomultiplier tubes (MaPMT)s provide good spatial resolution on a large active area, high detection efficiency, and very low background noise to allow single-photon detection despite the high occupancy. Given the high average hit rate in the high occupancy regions (10^7 hits/s per pixel on average), a radiation–hard fast readout front–end ASIC (CLARO) is used. MaPMTs are plugged into a baseboard, which propagates anode signals to the front–end board (FEB), which hosts the CLARO ASICs. The FEB output is sent to a subdetector module digital board that transports the digitized photon subdetector signals away from the high–radiation region.

2.7 Calorimeters

The LHCb calorimeter system [62] consists of an electromagnetic calorimeter (ECAL) followed by a hadronic one (HCAL). The main purpose of the calorimeter system is to identify hadrons, electrons, and photons, and to measure their energies and positions. The identification of electrons is a key element for flavor tagging in semileptonic decays, whereas the reconstruction with good accuracy of π^0 and prompt photons allows the study of B–meson decay channels. Both ECAL and HCAL detect particles via scintillation light from plastic scintillator modules, which is transmitted to PMTs through fibers. To account for different hit densities across the calorimeter surface, both ECAL and HCAL are segmented laterally in three and two regions, respectively, with increasing dimensions going from the beam pipe outward, as shown in Fig. 2.9.

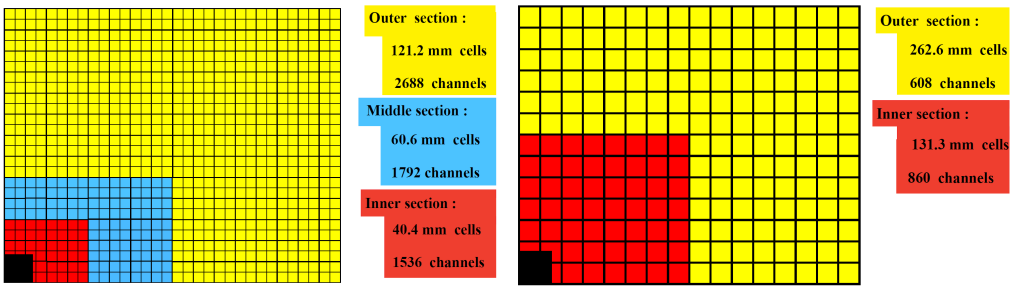

Figure 2.9: Sketches of (left) ECAL and (right) HCAL lateral segmentation [59].

ECAL cells have a shashlik structure with alternate scintillator and lead layers. The energy resolution of an ECAL cell is

$$\frac{\sigma(E)}{E} = \frac{(9.0 \pm 0.5)\%}{\sqrt{E}} \oplus (0.8 \pm 0.2)\% \oplus \frac{0.003}{E \sin\theta}, \tag{2.1}$$

where E is the particle energy in GeV and θ is the angle between the beam axis and a line from the LHCb interaction point to the center of the ECAL cell. The first term in Eq. 2.1 is stochastic due to the fluctuations related to the physical development of the shower, the second contribution is a constant term taking into account effects like mis–calibrations and non–linearities, whereas the third term is due to the noise of the electronics.

The HCAL is a sampling tile calorimeter, consisting of staggered iron and plastic scintillator tiles mounted parallel to the beam axis. The energy resolution is

$$\frac{\sigma(E)}{E} = \frac{(67 \pm 5)\%}{\sqrt{E}} \oplus (9 \pm 2)\%, \tag{2.2}$$

where E is the deposited energy in GeV. ECAL and HCAL calorimeters share the same electronics. Front–end boards (FEB) amplify, shape, and digitize PMT signals and then ship them to the back–end electronics, whilst a calorimeter control unit board distributes clocks and ECS commands to the FEBs. FEBs provide the digitized analog data in the form of transverse energy measurements. The central component of the FEBs is the ICECAL ASIC that performs the analog signal processing stage of the FEB.

2.8 Muon system

The main goal of the muon system [62] is to provide efficient muon identification, while keeping the pion misidentification probability low. Efficient muon identification with low contamination is required both for tagging and for clean reconstruction of muonic final states within B decays. As shown in Fig. 2.2, the LHCb muon system is composed of four stations, M2 to M5, located downstream of the calorimeter system. The stations are equipped with multi–wire proportional chambers (MWPC), interleaved with 80 cm thick iron absorbers to filter low–energy particles. Each station is divided into four regions, R1 to R4, of increasing area moving from the central beam axis outward, to uniformly distribute the particle flux and the channel occupancy across each station. MWPCs are made up of four independent layers, each consisting of anode–wires between two cathode–planes. The muon system electronics is designed to convert, format, and transmit the analog signals extracted from the subdetector. It hosts an amplifier–shaper–discriminator stage implemented in a dedicated ASIC, as well as a digital section that allows time alignment of the signals and logical combinations of readout channels.

2.9 Online and data acquisition

Unlike FE electronics that is custom–made for each subdetector, BE electronics is built on a common platform, the PCIe40 card [66]. The PCIe40 is a custom PCI–express card that fulfils several tasks, depending on the firmware loaded on the FPGA at the core of the board[5]. Figure 2.10 shows both a picture and a schematic view of the PCIe40 main components. The Arria 10 FPGA, located at the center of the card, is equipped with 48 bidirectional links running at up to 10 Gbit/s each. Links are connected to optical modules (MiniPod) driving the I/O to/from the detector, by means of eight MPO–12 connectors used for the fibers. Moreover, two bidirectional links running at up to 10 Gbit/s are devoted to time distribution. On the output side, each PCIe40 card can output data using two PCIe Gen3 x8 lanes. The card is fully instrumented with sensors that can measure all relevant voltages, currents, and temperatures. PCIe40 is designed to fulfill the functionality required for data acquisition and control. When configured for data acquisition purposes, the PCIe40 card takes the name of TELL40. Its purpose is to decode and process data coming from the detector before sending those to the server that hosts the card itself, via the PCIe bus. In this case, the optical links are used only in the receiver mode. The PCIe40 card used for control purposes (SOL40) [68] configures the FE

[5]The FPGA used for the PCIe40 card is the Intel® Arria® 10 described in Appendix A.

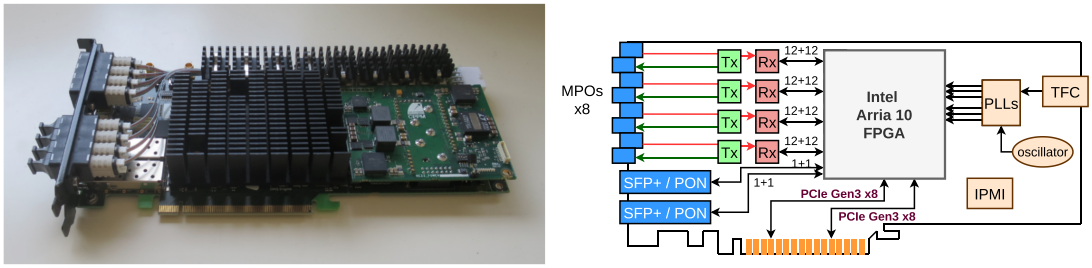

Figure 2.10: (left) Picture of the PCIe40 card and (right) diagram of the PCIe40 board and its primary components [59, 67].

electronics and transmits timing and fast control (TFC) commands to the FE and TELL40s. In this case, all bidirectional links are active. When configured as a SODIN [69, 70], the PCIe40 acts as a readout supervisor, interfacing the LHC machine clock to LHCb. Considering data paths first, around 500 TELL40 cards receive data via 10k optical links, carrying subdetectors data from the LHCb cavern to a data center located on the surface. Data packets are sent from the FE to the TELL40 boards asynchronously across all input links. The readout firmware of the TELL40 must be able to decode the data frames from the FE, realign them according to their bunch crossing identifier (BXID), build an event packet, and send it to the DAQ network, in which each TELL40 card represents a source point, managing data from a limited number of detector links. In order to perform selections, the LHCb software trigger requires complete events, containing the information of all subdetectors. The process of assembling all data fragments coming from different TELL40s and belonging to the same bunch crossing is called event building (EB) [71] and is performed on the ∼170 servers that host the TELL40 cards[6]. Apart from TELL40 cards, EB servers also contain GPUs running the first stage of the High Level Trigger (HLT1) application. As each server receives only the data from the subdetectors connected to the hosted TELL40s, all the EB nodes are interconnected with a high–performance network[7] that transmits the complete information to the node, which is in charge of the full event assembly. Each server acts in turn as data–source and data–sink in the event–building process, where cyclically every node acts as a full event builder (sink) and receives data from all other servers (sources). Instead of putting together single data packets, the EB builds data from several thousands bunch–crossings, to achieve optimal network performances. Built events are then stored in a shared memory buffer and sent to a GPU installed in each event–builder server that performs HLT1 reconstruction and selections. The events selected by HLT1 are sent to a temporary buffer storage, from where they will be accessed by the alignment and calibration processes and by the application performing second–stage selections (HLT2).

Figure 2.11 shows a schematic view of the entire LHCb online system. PCIe40 cards, in SODIN and SOL40 flavors, play a key role in the TFC system. TFC is responsible for controlling and distributing clock, timing, and trigger information, synchronous and asynchronous commands, across the readout chain. It also regulates the transmission of events through the readout chain taking into account TELL40 states, the LHC filling scheme, and calibration procedures, while ensuring a coherent data taking acquisition across all elements in the readout architecture. Figure 2.12 shows the TFC architecture and its data flow. The SODIN readout supervisor is the TFC master, sending necessary information and commands, while being interfaced to the LHC

[6]Each EB server contains up to three TELL40 cards, where each server hosts cards of a single subdetector.
[7]The 200 Gbit/s high dynamic range (HDR) InfiniBand technology is used for the EB network implementation.

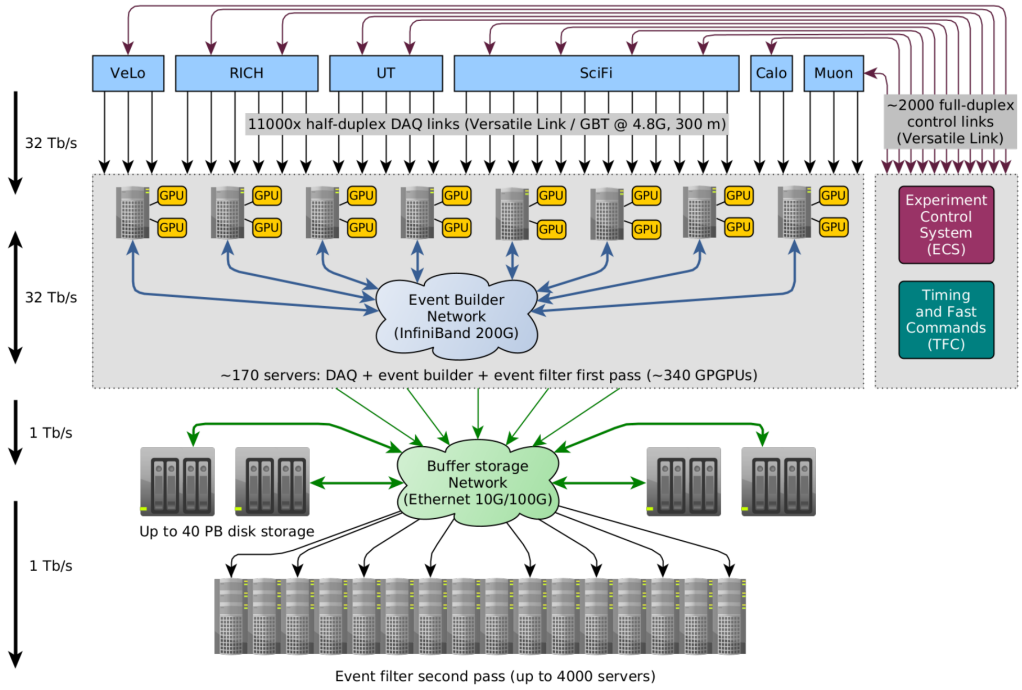

Figure 2.11: LHCb Upgrade online system [59].

40 MHz clock distribution system. Subdetector FE and BE readout electronics are connected to the SODIN via a network of bidirectional optical links using multiple SOL40 interface boards and passive optical splitters. These connections define the granularity of the partition, where a partition is a set of subdetector electronics and control resources that can be run independently of another set. The connections between the SOL40 and the FE boards use the GBT protocol for fast and slow control distribution to the FE and for slow control back from the FE. In contrast, the connections between SODIN, SOL40s, and TELL40s use Passive Optical Network technology (PON) and PON splitters. SODIN is also interfaced to the rest of the DAQ by its PCIe interface, sending information about the conditions under which the data are collected. Another key component of the online architecture is the experiment control system (ECS). It is responsible for the configuration, monitoring, and control of all areas of the experiment, including the slow control of high and low voltages, as well as monitoring and control of DAQ and trigger systems.

2.10 Trigger

The goal of the trigger system is to reduce the volume of data read from the detector, around 32 Tb/s at the nominal Run 3 conditions, to around 80 Gb/s, which can be recorded to permanent offline storage, while retaining as much signal as possible [71]. Given the high rate of potentially interesting events[8], instead of making trigger decisions based on partial event information[9], as it was done during Run 1 and 2, the reconstruction of the full event is exploited to achieve

[8]Over 300 kHz of bunch crossings contain a partially reconstructed beauty hadron and almost 1 MHz of bunch crossings contain a partially reconstructed charm hadron.

[9]During Run 1 and 2 the trigger decision was made on calorimeter and muon information, which were the only subdetectors that could be read out at the full 40 MHz speed.

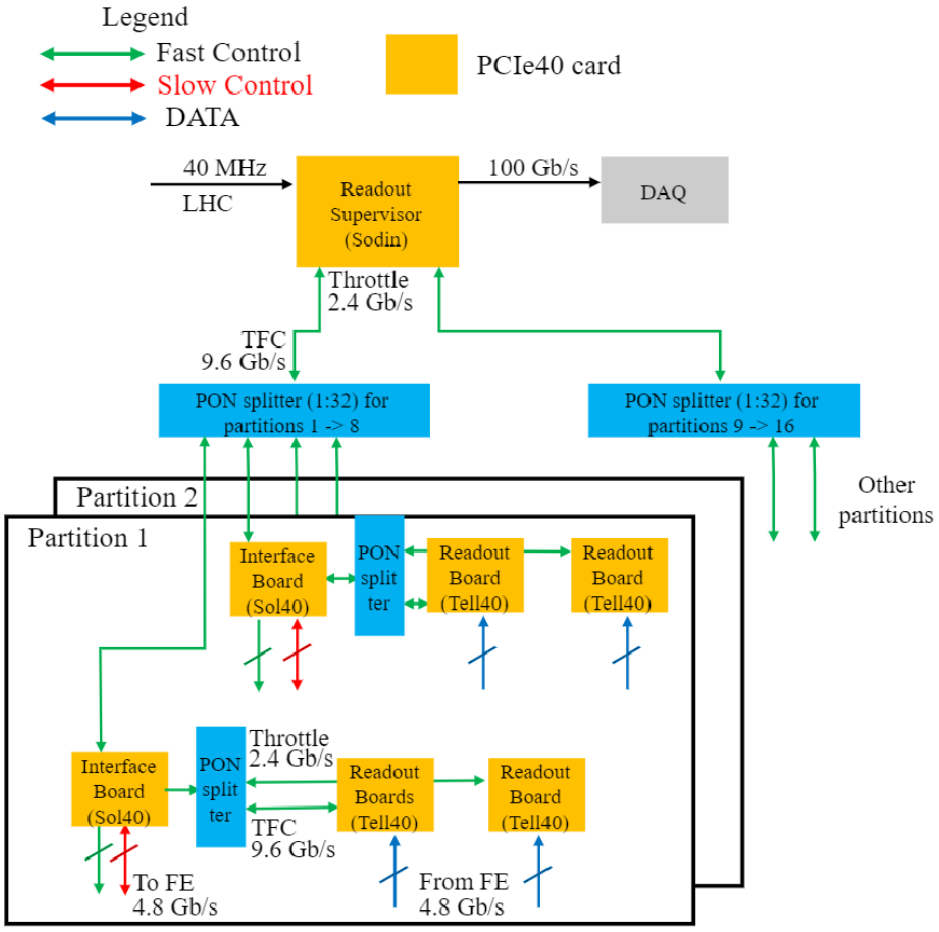

Figure 2.12: Logical architecture of the TFC system [59].

a data reduction factor of about 400. Given these constraints, the trigger system is divided into two main stages, named High Level Trigger 1 (HLT1) and High Level Trigger 2 (HLT2). HLT1 [42] focuses primarily on charged particle reconstruction and reduces the data volume by approximately a factor 20, while maintaining a high triggering efficiency. On the other hand, HLT2 performs the full offline–quality reconstruction, in addition to $\mathcal{O}(1000)$ selections. Between HLT1 and HLT2, a disk buffer (around 30 PB) holds the data while real–time alignment and calibration processes are being performed. It also allows events selected by HLT1 to be buffered for processing between LHC fills. The complete data flow from the detector to the offline storage is illustrated in Figure 2.13. HLT1 uses an array of GPUs mounted on the Event Builder servers to perform a fast event reconstruction, with the only purpose of reducing the event rate, while retaining as much signal as possible, to a level acceptable for HLT2. HLT1 focuses on reconstructing the trajectories of charged particles in order to measure their momenta with percent–level precision, while associating each particle with the corresponding pp collision and measuring its displacement from the primary vertex. The complete HLT1 reconstruction sequence is shown schematically in Fig. 2.14. HLT1 reconstruction and selections are preceded by a Global Event Cut (GEC) that removes a fraction of the events with the highest detector occupancy. These events require a large amount of computing time to reconstruct, while having worse detector performances. Subsequently, tracks are reconstructed in the VELO and used

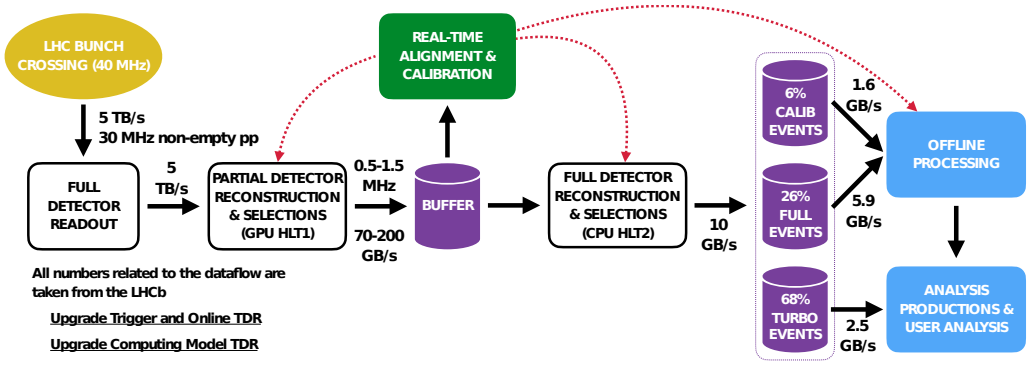

Figure 2.13: Schematic view of the online data flow [59].

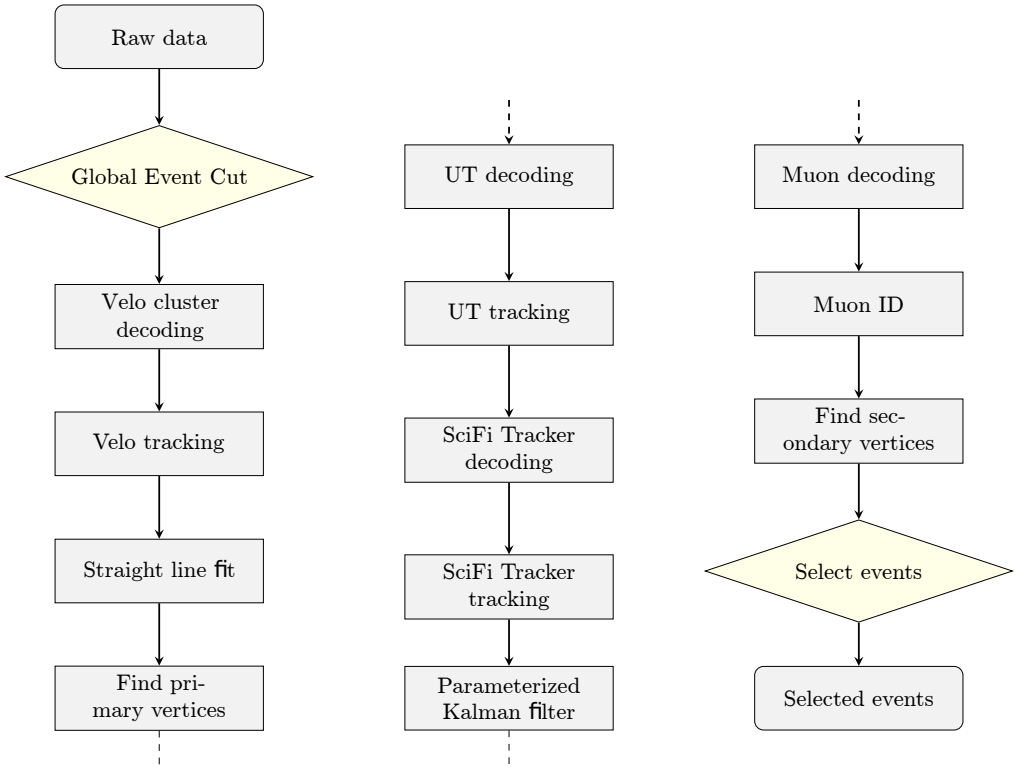

Figure 2.14: Baseline HLT1 sequence, with rhombi representing algorithms reducing the event rate, while rectangles represent algorithms processing data [59].

to find the positions of the primary vertices. Tracks are then extrapolated to the UT and SciFi, based on a minimum allowed momentum and/or transverse momentum. The particle displacement from the primary vertex is computed by means of a parameterised Kalman filter (see Sect. 6) that takes the momentum information as input. Tracks are then fitted to a common origin to form two–body displaced vertex candidates, which are used for selections. HLT1 selections are divided into four main categories. Primary inclusive selections cover the bulk of LHCb physics program, which, for example, include two–track vertex triggers, displaced single–track triggers, and displaced single muon triggers. Selections are also made for calibration

samples, for specific physics signatures not covered by the inclusive triggers, and technical triggers for luminosity determination, monitoring, calibration, and alignment.

HLT2 runs on an array of CPU servers, using the real–time alignment and calibration of the detector, to perform an offline–quality reconstruction, together with $\mathcal{O}(1000)$ selections. If an event is kept, the selection algorithm specifies which parts of the event need to be saved to permanent storage, exploiting the Turbo technique introduced during Run 2 [72]. The HLT2 reconstruction is divided into the following four main steps:

- charged particle pattern recognition, where VELO tracks are used to find the positions of the primary pp collisions. As a consequence, tracks that originate from the decays of long–lived particles, and are therefore displaced from the PV, are precisely identified. VELO tracks are then extrapolated into the UT and SciFi regions to measure their momentum;
- calorimeter reconstruction, where single–photon clusters, multiple–photon merged clusters and electron clusters are distinguished one from another using multivariate algorithms;
- particle identification, where a combination of the two RICH detectors, the ECAL, and the muon system are used to identify the five basic long–lived charged particle species – electron, muon, pion, kaon, and proton;
- Kalman fit, which allows the parameters of reconstructed tracks to be precisely measured.

2.11 VELO clustering before this thesis

Reconstructing VELO clusters from single–pixel hits is one of the first operations to be performed within the particle tracking sequence. As reported in Ref. [73], VELO clustering was supposed to occur within the HLT process, running on a CPU–based architecture. For this purpose, a first clustering algorithm was developed and optimized for the VELO sparse response [74]. Already from the first tests of the reconstruction software, based on detailed Monte Carlo (MC) simulated samples, VELO clustering required roughly 10% of the entire time needed for the HLT1 sequence. As the LHCb collaboration decided to move to HLT1 reconstruction and selections performed on GPUs, the clustering algorithm, together with a substantial part of the reconstruction software, was ported to the new architecture [42, 75]. With clustering originally planned to be part of HLT processes, VELO TELL40s had time reordering of VELO SPs as their main task, with the possibility of performing isolation flagging within the TELL40 itself, if enough FPGA resources were available. Time reordering consists in sorting VELO raw data coming from the detector by their timestap as they are not guaranteed to be ordered, due to the VeloPix readout architecture. Isolation flagging identifies SPs that do not have any active neighbor SP in order to ease clustering operations. More details about time reordering and isolation flagging can be found in Sects. 3.2.1, 4.4.3 and 5.3.1.

As I started the work described in this thesis, the algorithms to perform SP decoding and clustering within GPUs were available, together with the first functioning VELO firmware, excluding the isolation flagging. After the first exploratory tests using a prototype of the clustering firmware and the promising results of the first comparative performance studies, it was clear that a Retina–based VELO clustering algorithm and the corresponding firmware could be integrated within the VELO FPGA–based readout cards, freeing HLT from a significant amount of processing load. Thus, the following chapters present all the relevant steps that led, from the original idea, to a fully–functional and commissioned clustering architecture within the

LHCb silicon pixel detector. This is the first time that a FPGA–based algorithm, capable of reconstructing clusters from the hits of a HEP pixel detector, has been deployed to the detector readout cards, being able to run online at the unprecedented speed of 30 MHz input event rate, without time multiplexing. The original work detailed in this thesis is a significant and very challenging step towards innovative solutions aimed at fulfilling the ever–increasing processing and bandwidth needs of modern HEP experiments, by means of subsystems running at early DAQ stages and exploiting new heterogeneous computing architectures.

Chapter 3

A real–time 2D clustering algorithm

This chapter details the first implementation of a FPGA–based 2D clustering algorithm, capable of running in real time at the LHC crossing rate. The main components of the algorithm are described, together with the algorithm parameters that can be tuned for specific use cases. A specific reference is given to the application within the LHCb VELO detector data acquisition chain. The main limitations of the algorithm are also discussed.

3.1 Introduction

The clustering algorithm described in this thesis is capable of reconstructing two–dimensional clusters produced by charged particles hitting a silicon pixel detector in real time, at the LHC bunch–crossing rate. Clustering in two dimensions is an extremely parallelizable task, where the reconstruction of a cluster is independent of the presence and topologies of surrounding clusters. As described in Appendix. A, FPGA–based devices are widely spread in HEP experiments for readout purposes and they offer high design flexibility. For these reasons, the new algorithm presented in this thesis is designed to run on FPGA–based readout boards, taking individual pixel hits as input, reconstructing clusters, and returning their centers of mass. The clustering algorithm exploits the high level of specialization and parallelization of FPGA–based accelerators, to achieve the highest possible throughput, while maintaining a very low latency, and, given these features, it can be integrated within the readout chain of any pixel detector, regardless of its size and geometry. The algorithm has few parameters that can be tuned for specific applications, and a baseline VHDL version is available for download from a public code repository [76].
The final version of the algorithm, as well as the optimization work of the overall architecture, is detailed in this chapter. The algorithm fine tuning is performed by means of the official Geant–based LHCb simulation [77], taking into account the topology and occupancy distributions of clusters, and using several MC samples produced at the nominal Run 3 luminosity of 2×10^{33} cm^{-2}s^{-1}. Further details about the LHCb simulation and the physics performance of the algorithms can be found in Chaps. 5 and 6, respectively.

3.2 Algorithm implementation for LHCb VELO detector

The algorithm described in this thesis is specifically tailored to the LHCb VELO detector and its readout chain. As discussed in Sect. 2.3, VELO data are read as aggregated groups of 4×2 pixels, named SuperPixels (SPs), and subsequently sent to the clustering block, which takes as input the list of all active SPs found in a given event and returns the list of reconstructed clusters.

3.2.1 Isolation flagging

As a first operation, the clustering block compares the two–dimensional position of each SP with that of all other SPs in the same event within the same sensor. SPs are then sorted according to the presence of any active neighboring SP: a SP is flagged as 'isolated' if none of its eight SP neighbors has any active pixel. This information is particularly relevant in optimizing the performance of the cluster reconstruction process, allowing the implementation of a much faster algorithm for isolated SPs, which account for about 53% of the total number of SPs[1]. The isolation flag is used by the algorithm to differentiate the reconstruction mechanism between isolated SPs and non–isolated SPs, which have at least one active neighbor.

3.2.2 Isolated clustering

Isolated SPs are resolved directly into clusters using a look–up table (LUT). The table links each of the 2^8 possible pixel configurations, within a SP, to the center of mass of the corresponding reconstructed cluster(s). Within the firmware, the LUT is implemented as a single–port RAM, where the SP hitmap represents the address to be read from the LUT. The LUT is filled with precomputed two–dimensional coordinates of the center of mass of each SP configuration. The case of a double cluster in the same SP is also solved. This LUT–based reconstruction allows an extremely fast processing of isolated SPs, with a very limited amount of logic resources within the FPGA.

3.2.3 Non–isolated clustering

The cluster reconstruction of non–isolated SPs requires, instead, the parallel processing of an ensemble of SPs. A chain of independent matrices is used to reconstruct clusters from all non–isolated SPs coming from the same VELO sensor. This approach avoids the implementation of a single sensor–sized matrix in favor of smaller matrices that are handled more easily and faster inside the FPGA[2]. All matrices are created as empty entities and can contain up to nine contiguous SPs, organized in three rows and three columns. The first–arriving SP fills the center of an empty matrix and determines the physical location of the matrix inside the VELO detector, as well as the set of coordinates of the other SPs that can fill it. If a SP belongs to a matrix, it fills it; otherwise, it moves forward, checking the next available matrix of the chain or filling the center of an empty one. An explanatory graphical illustration of this mechanism is shown in Fig. 3.1. At the end of each event, in a fully parallel way, each pixel of a matrix checks if it belongs to a predetermined pattern of pixels, as shown in Fig. 3.2. If one of the patterns

[1] This estimate is performed using the official LHCb simulation tools. The LHCb software stack is detailed in Sect. 5.4.

[2] The expected occupancy of the VELO sensors is around 0.125% [60] in the regions closest to the beam pipe. A cluster is made of 2 pixels on average.

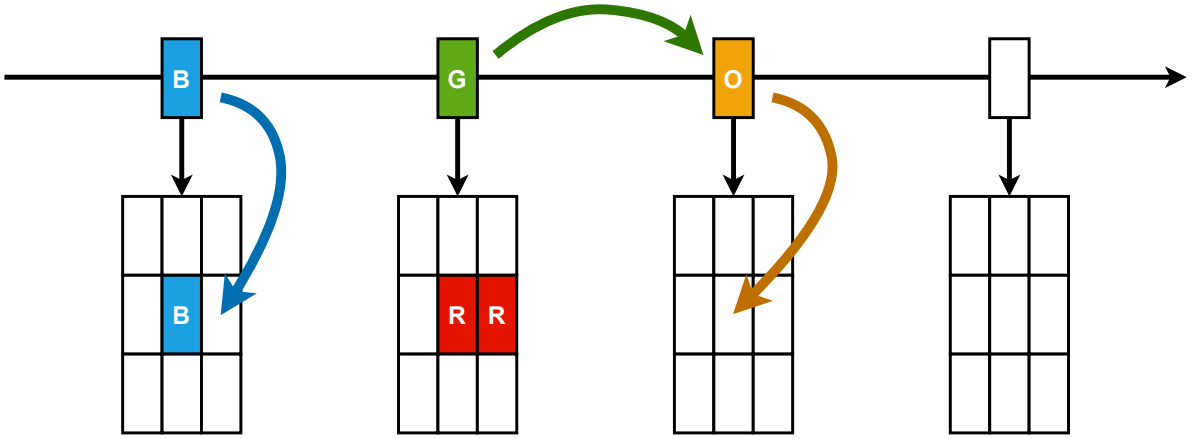

Figure 3.1: Sketch of the matrix filling mechanism with non–isolated SPs. SPs with same color (label) are neighbors with active pixels. The blue SP (B) fills the first matrix in the line that is already populated with one of its neighbors. The green SP (G) does not belong to any of the already populated matrices, so it moves forward. The orange SP (O) has reached an non–initialized matrix, so it fills the center.

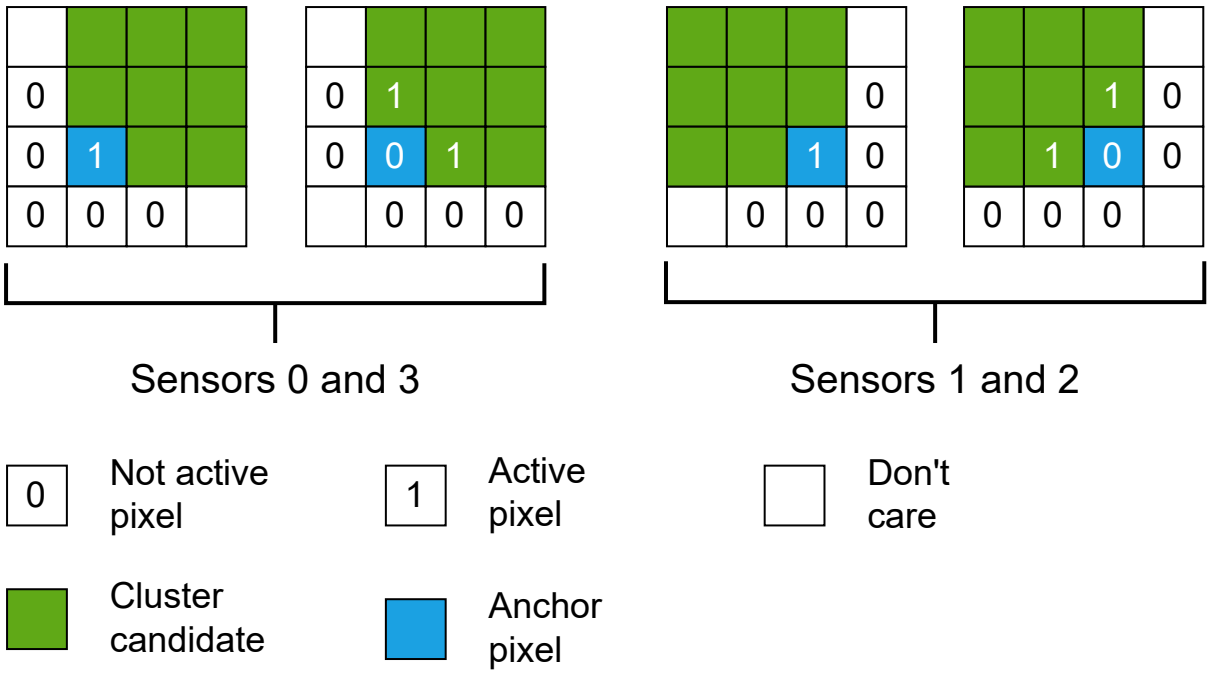

Figure 3.2: Pixel patterns seeding to a cluster candidate. Patterns are optimized for sensor mounting orientation. See Ref. [78] for further details.

is matched, the cluster candidate is recognized in the 3×3 grid (green pixels), as well as the anchor pixel (blue pixel), positioned at one of the corners of the 3×3 grid, depending on the orientation of the sensor. The centroid coordinates are determined using a LUT that links each of the possible 2^9 pixel configurations inside the 3×3 cluster candidate to the coordinates of its center of mass. The absolute position of the cluster candidate is obtained as a sum of three vectors of coordinates: the position of the matrix with respect to the detector, the position of

the anchor pixel with respect to the matrix, and the position of the reconstructed cluster with respect to the anchor pixel. In order to reconstruct cluster candidates that have an anchor pixel falling near the matrix edge, each matrix is surrounded by edges of registers fixed at zero, as shown in Fig. 3.3. These edges identify a set of pixels that are not used during the filling process, but are necessary to determine the 3×3 cluster candidate, even near to the matrix edges. An example of such a configuration is shown in the right part of Fig. 3.3. The width of the edges is determined by the VELO sensor number, the allowed patterns, and the cluster candidate topology (Fig. 3.2).

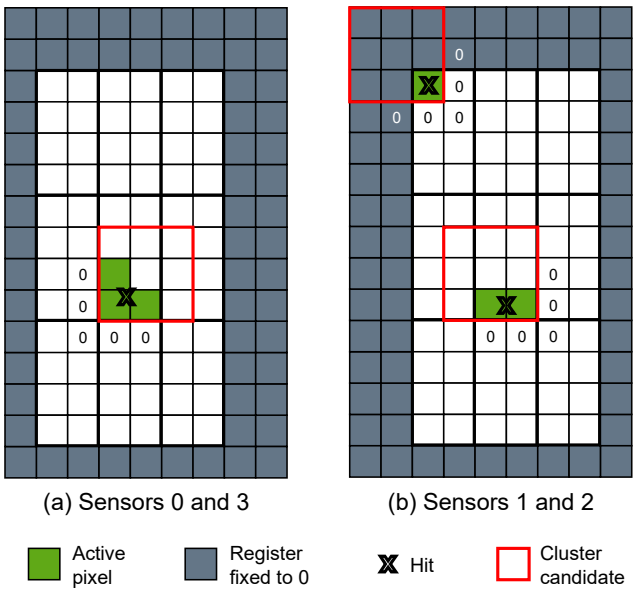

Figure 3.3: Matrix edges and pattern orientations: (a) for sensors 0 and 3 and (b) for sensors 1 and 2.

3.3 Parameters and limitations of the algorithm

The clustering algorithm has four main parameters that can be tuned to optimize its performance in terms of speed, efficiency, and quality of the reconstruction.

- The shape and size of the matrix are determined by how non–isolated SPs are arranged together. Bigger matrices can contain clusters made of a higher number of SPs. For the VELO clustering algorithm, matrices that can contain up to 3×3 SPs are implemented.
- The distribution of the number of SPs with neighbors per event determines the number of matrices that has to be instantiated. Increasing the number of instantiated matrices increases the number of SPs that can be accommodated. For the VELO clustering algorithm, 20 matrices are implemented for each VELO sensor.
- The size of the cluster candidates is determined by the distribution of cluster sizes shown in Fig. 3.4. The sizes of clusters created by individual charged particles crossing VELO layers are typically rather small (1–4 pixels in 96% of cases), with larger clusters being the product of merged hits or secondary emissions (δ–rays, etc.).

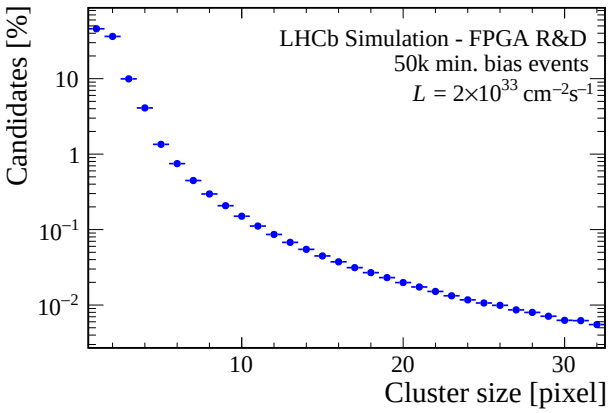

Figure 3.4: Distribution of cluster sizes in units of number of pixels.

- The geometry of the cluster matching patterns (Fig. 3.2) takes into account the fact that clusters are collections of an arbitrary number of pixels close together and separated from other active pixels. The "L" shaped sequence of inactive pixels with two different configurations of active pixels allows cluster seeds to be identified. The actual shape is chosen in order to optimize the cluster and track reconstruction efficiencies (see Sect. 6).

The tuning of the algorithm parameters needs to take into account the usage of FPGA logic and memory resources and the processing rate. Increasing the number of SPs that a matrix can contain linearly increases the amount of logic resources needed. The same linear relation is present between the number of instantiated matrices and the logic resources. However, an exponential relation links the sizes of the LUTs used for the cluster candidate reconstruction and the memory resource usage, as the number of possible pixel configurations increases exponentially with the number of pixels considered.

Since the algorithm is designed for a FPGA–based implementation, the number of instantiated matrices cannot be dynamically adjusted to cope with events with a higher number of (non–isolated) SPs. If the number of matrices is not enough to accommodate all the SPs, a fraction of those overflow the matrices. Overflow SPs can be discarded as a whole or some partial information can be extracted resolving them using a LUT, as if they were isolated. This inevitably increases the number of duplicated clusters, since more than one cluster is reconstructed from a single and compact conglomerate of pixels that is distributed over several SPs. For the LHCb–VELO implementation of the algorithm, overflow SPs are reconstructed as if they were isolated. See Sect. 6 for detailed studies on the algorithm physics performance.

3.4 Algorithm fine tuning

Since its first design, the clustering algorithm has been tuned to improve the overall reconstruction performance, taking into account the constraints given by the hardware implementation on the FPGA chip. These improvements concern a better cluster search pattern, as reported in Refs. [67, 79], an optimized cluster position calculation, and the removal of inefficient matrix edges. For instance, Fig. 3.5 shows the first version of the cluster search patterns. Starting from this first implementation, the following changes have been applied to optimize the algorithm response for what concerns search patterns:

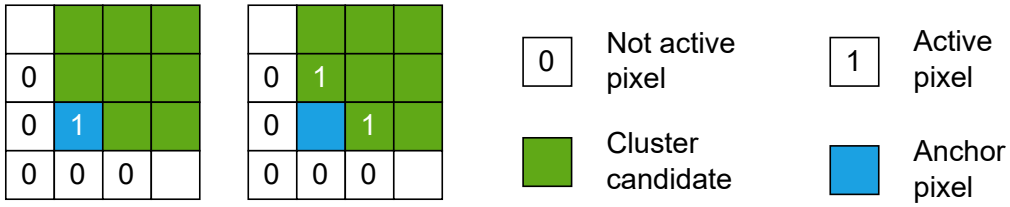

Figure 3.5: Original pixel patterns seeding to a cluster candidate, before the optimizations and fine tunings.

- Cluster search patterns have been updated to maximize cluster and track reconstruction efficiencies. As shown in Fig. 3.6 if two clusters end up being close enough and with specific topologies, only one of the two is reconstructed, whereas the other is not identified.

Figure 3.6: Example of cluster reconstruction inefficiency due to the original cluster search patterns, shown in Fig. 3.5.

This is due to the specific patterns that were originally used to search for clusters. In particular, the right pattern in Fig. 3.5 requires the bottom–left pixel to be zero in order to recognize a cluster. For specific cluster topologies and configurations, such as the one in Fig. 3.6, even if two clusters are present, only one is found. Not reconstructing a cluster may lead to tracking inefficiencies, where a track is not identified because clusters in one or more VELO modules are not reconstructed.

To avoid this type of inefficiencies, search patterns have been changed, as shown in Fig. 3.7. The bottom–left pixel, which was previously required to be zero, now moves to the 'don't care' category, and the anchor pixel is required to be zero.

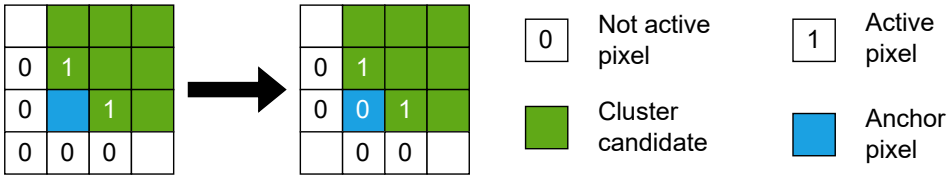

Figure 3.7: First optimization of the cluster search patterns adopted to avoid cluster reconstruction inefficiencies, as shown in Fig. 3.6.

- Search patterns were also optimized to avoid the reconstruction of multiple clusters from a single collection of pixels close together. Figure 3.8 shows an example of cluster splitting, where two clusters are reconstructed from a single one. Cluster splitting can lead to an increase in the number of clone tracks, which are tracks with many hits in common (see Sect. 6.3.1) but that differ by a small number of hits not being shared. Moreover, increasing the number of clusters has a negative impact on the time required to run the reconstruction, as the tracking algorithms need to run on a higher number of hits.

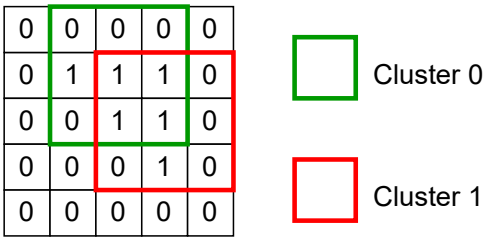

Figure 3.8: Example of cluster splitting where two clusters are reconstructed from a single collection of pixels close together.

This splitting behavior is due to the right pattern shown in Fig. 3.5 and the optimization described in Fig. 3.7 did not improve this specific case. In order to reduce cluster splitting, this specific search pattern has been updated, as shown in Fig. 3.9.

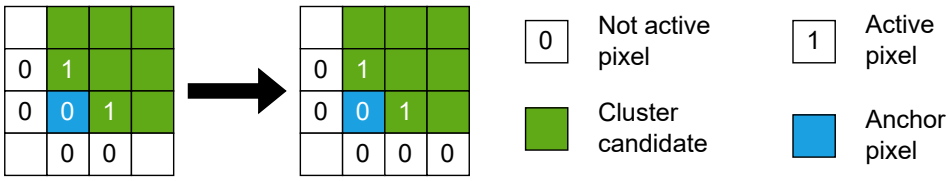

Figure 3.9: Second optimization of the cluster search patterns adopted to avoid cluster splitting, as shown in Fig. 3.8.

An additional pixel is required to be zero in the bottom–right part of the pattern. Cluster splitting is significantly reduced when applying this new pattern. In the example shown in Fig. 3.8 only the red cluster is reconstructed with the new optimized search pattern.

- Cluster search patterns have also been optimized to account for VELO sensor mounting orientation. Given a VELO module, two of its sensors are mounted on the front side of the layer, whereas the remaining two on the back side, having opposite local coordinate system origins (pixel $[0, 0]$), as shown in Fig. 3.10.

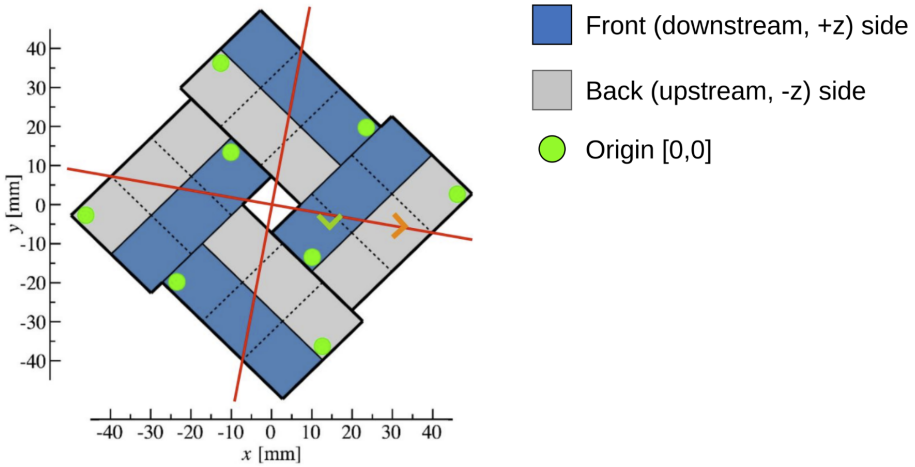

Figure 3.10: Sensor mounting orientation on a VELO layer. The origins of the local coordinate system are also highlighted as green dots. Red lines show particle tracks originating from the interaction vertex.

If the mounting orientation is not taken into account correctly, the number of split clusters being reconstructed increases. This is mainly due to the presence of large clusters elongated

in one direction, as shown in Fig. 3.11, produced by tracks grazing VELO sensors at small angles. While large clusters that develop from bottom–left to top–right are partially reconstructed, but no more than one cluster is identified[3], clusters that develop in the opposite direction, from bottom–right to top–left, lead to cluster splitting, due to the orientation of the cluster search patterns.

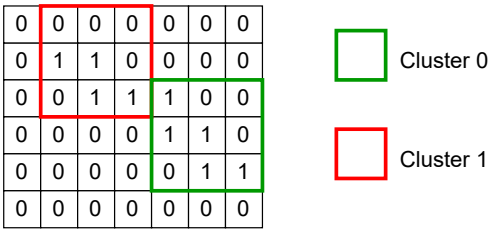

Figure 3.11: Example of a large cluster splitting due to an elongated cluster developing from bottom–right to top–left.

To account for this mirrored geometry, the cluster search patterns have been changed accordingly for sensors mounted on the back side, as shown in Fig. 3.12. This change does not affect the reconstruction of small clusters or clusters with an almost circular geometry. It affects large clusters lengthened in a diagonal direction for which a non–symmetrized cluster search pattern would lead to an increase in cluster spitting and, as a consequence, in clone and ghost track rates (see Sect. 6.3.1).

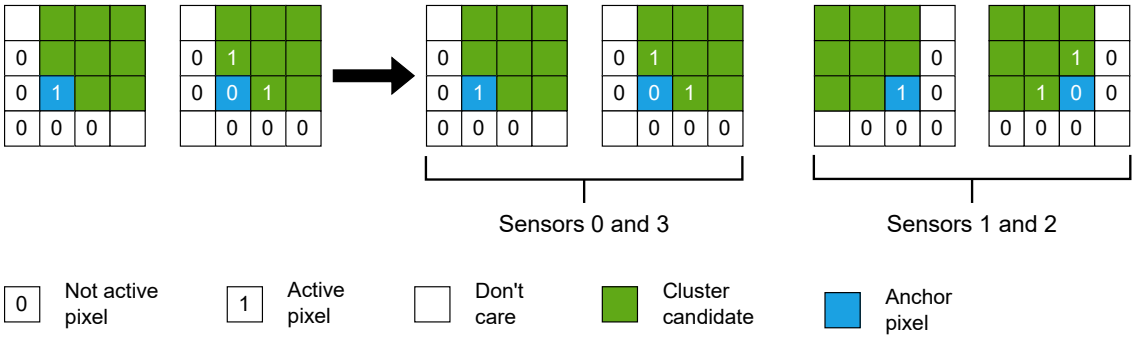

Figure 3.12: Third optimization of the cluster search patterns adopted to avoid large cluster splitting, as shown in Fig. 3.11, due to the VELO sensor mounting orientations.

Other than search patterns optimizations, the following tunings were applied to the algorithm.

- The cluster center position calculation has been tuned both for clusters originating from isolated and non–isolated SPs. In case of isolated SPs, the firmware has been adjusted to account for the possibility of two clusters in a single SP, as shown in the left part of Fig. 3.13. In the first implementation of the algorithm, only one cluster was reconstructed in this case. Even if isolated SPs containing two clusters are quite rare, introducing the possibility of reconstructing both of them allows the cluster reconstruction inefficiency to be reduced. The case of two independent clusters in the same 3×3 cluster candidate is also taken into account for the non–isolated case, as shown in the right part of Fig. 3.13. While in the first version of the algorithm only one cluster was reconstructed, having as

[3]Also bottom–left–to–top–right clusters can lead to cluster splitting if they are large enough to end up populating more than one matrix.

coordinates the average values of the coordinates of the single clusters, the updated version reconstructs both clusters correctly. In both cases the LUTs in the firmware have been updated.

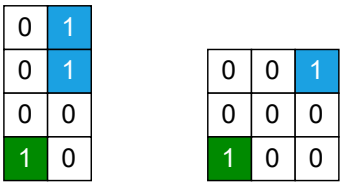

Figure 3.13: Examples of (left) an isolated SP containing two clusters and (right) a 3×3 cluster candidate containing two clusters.

- The cluster reconstruction for both isolated and non–isolated SPs has been tuned in such a way that the fractional parts of its center coordinates are no longer truncated to the lower one–forth step, as done in the first version of the algorithm, but rounded to the closest one–forth. A comparison between cluster coordinate truncation and rounding is shown in Fig. 3.14.

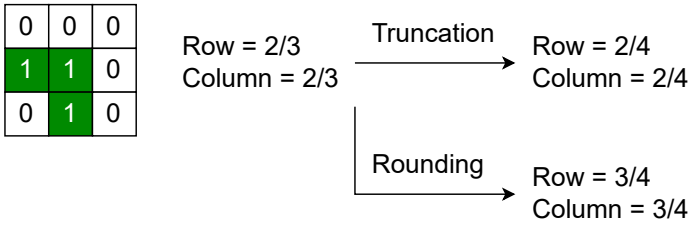

Figure 3.14: Comparison between cluster coordinates truncation and rounding.

This change was applied after an in–depth analysis to identify the cause of a bias observed in the primary vertex (PV) reconstruction position, as detailed in Sect. 6.3.4 and Appendix C. The truncation of the cluster coordinates to the lower one–forth step caused the reconstructed tracks to be shifted outward with respect to the interaction region. As a consequence, the reconstructed primary vertex is shifted towards negative z values. After applying this update, the shift in the PV z position is no longer visible.

- The matrix geometry has been modified from an initial 3×5 SP geometry to a more efficient 3×3 scheme. This allows the removal of inefficient matrix edges that could not seed clusters and the recovery of residual clustering and tracking inefficiencies, as shown in Fig. 3.15.

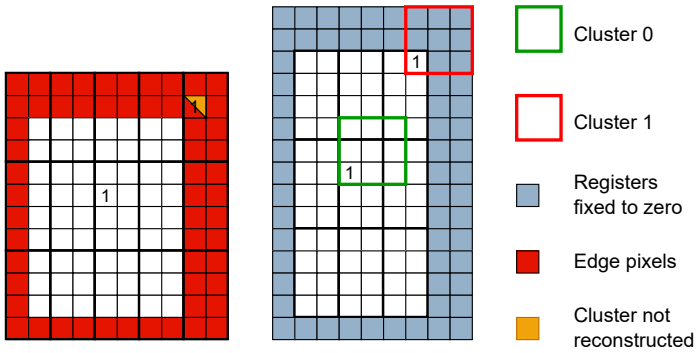

Figure 3.15: Fine tuning of the matrix geometry to improve cluster reconstruction efficiency.

In the first implementation of the algorithm, each matrix, appointed to non–isolated SP cluster reconstruction, had an edge of inefficient pixels that can be used to load SPs but could not trigger the reconstruction of a cluster as anchor pixels. As the algorithm was improved, the inefficient edges were removed in favor of a set of registers fixed to zero surrounding each matrix. Those registers are needed to extract a cluster in case the corresponding anchor pixel is on the edge of the matrix. Instead of keeping the same matrix geometry, making all pixels capable of seeding a cluster and surrounding it with fixed–to–zero registers, the matrix size has been reduced from 3×5 to 3×3 SPs so that the number of pixels that can seed a cluster remains comparable in the two cases[4]. This choice has been made based on firmware considerations: increasing the number of pixels that can seed a cluster would require more logic, as more pixels need to be checked at the same time according to cluster search patterns. Moreover, it would have put more pressure on timings as more fast logic would be needed to run in unison. More details about the firmware implementation, with special reference to timings and resource usage, can be found in Sect. 4.3, Sect. 5.3.5 and in Appendix B. However, reducing the matrix dimensions can cause SPs that make up large clusters to be more likely split over multiple matrices. This might have a negative impact on the tracking performance. For this reason comparison performance studies have been conducted, showing that the advantages of having a smaller but entirely active matrix have a net positive impact on physics performances. The corresponding increase in cluster splitting is marginal and does not have a noticeable effect.

This chapter ends the descriptive part of the thesis, where the motivations of the work, the environment in which it has been done and the clustering algorithm itself are detailed. The following chapters are devoted to an in–depth description of the firmware implementation of the algorithm, to its firmware and software integration within the LHCb readout chain, to the physics performance studies, and to the commissioning process. All the steps described play an important role in being able to collect data effectively with a significant part of the reconstruction process being performed online, for the first time.

[4]The number of pixels that can seed a cluster are 63 in the 3×5–SP matrix implementation and 72 in the 3×3–SP geometry. Making every pixel in the 3×5 matrix capable of seeding a cluster would have led to 120 potential anchor pixel positions seeding a cluster.

Chapter 4

VHDL firmware implementation

This chapter describes the firmware implementation of the clustering algorithm detailed in Chap. 3. A high–level description of the firmware as a whole is provided, together with a detailed view of the working principle of each component. Data formats and details of components not strictly related to cluster reconstruction, such as monitoring and error handling, are also discussed.

4.1 Key components of the architecture

The clustering architecture is built in a modular way, where each component is devoted to a specific task. First, a decoding and flagging stage splits data into separate streams, while flagging isolated SPs. Second, a pair of switch blocks sends data to the appropriate cluster processing component. Reconstructed clusters are then finally encoded with the chosen output format. Figure 4.1 shows a simplified view of the basic blocks that make up the architecture, whereas a more detailed view of the firmware structure is shown in Fig. 4.2.

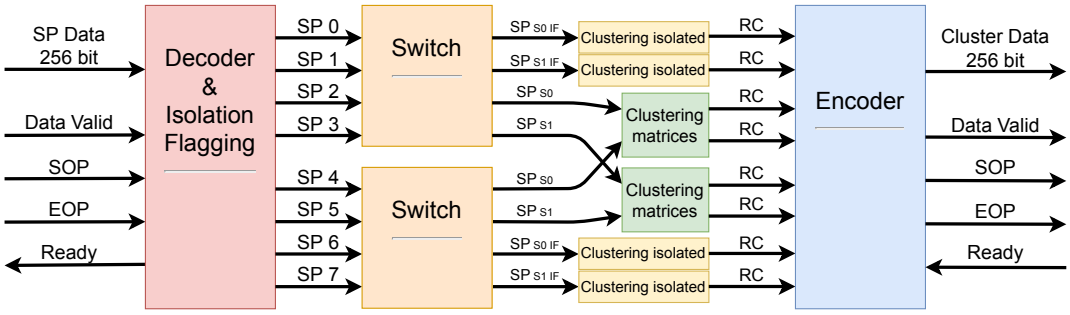

Figure 4.1: VELO data are received as 256–bit words, each containing eight SPs. A "Data Valid" signal states whether the incoming data are valid. Start of package (SOP) and end of package (EOP) signals delimit the start and the end of the data corresponding to each event. The clustering block sends a ready signal to the previous architecture component when it is ready to accept data. The "decoder and isolation flagging" block splits the 256–bit bus into eight 32–bit wide busses, each containing one SP. It also flags SPs that do not have any active neighbour SPs (isolation flag). A pair of switches arranges SPs by sensor (S0/S1) and by isolation flag (IF). The "clustering isolated" and "clustering matrices" blocks reconstruct clusters, that are encoded back into 256–bit words by means of an encoder.

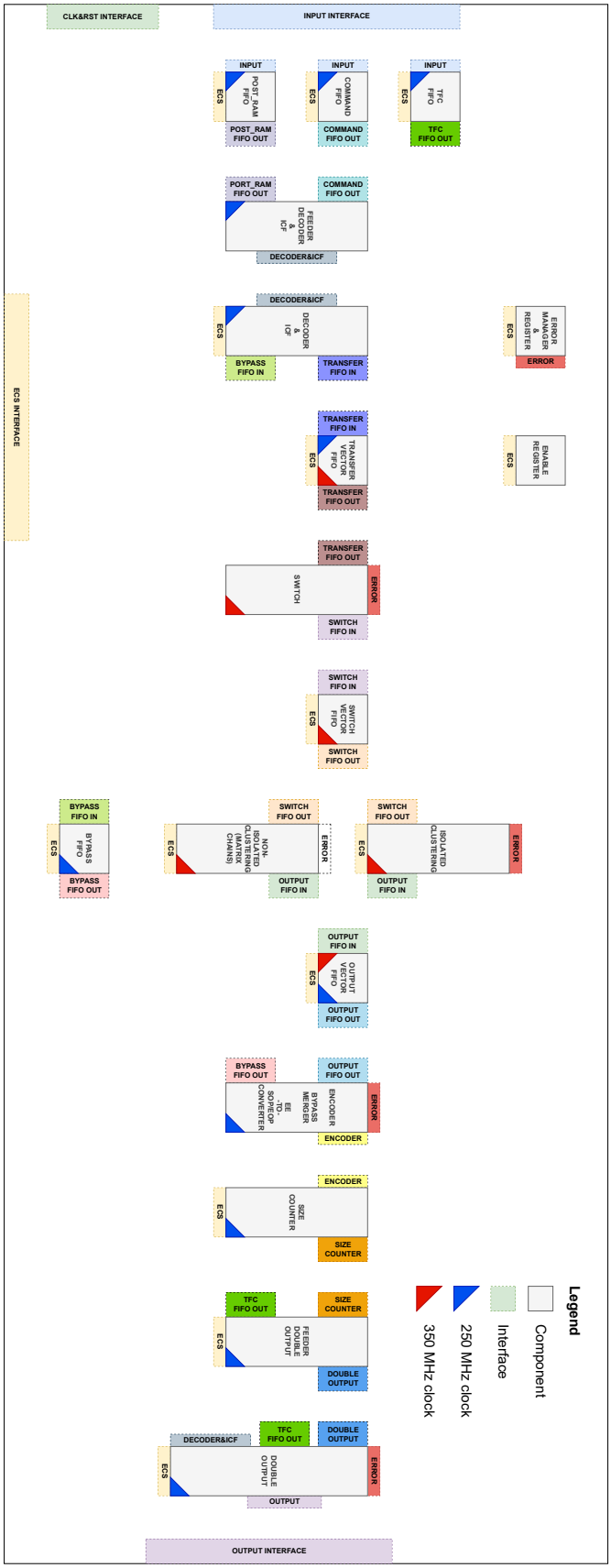

Figure 4.2: Overall view of the clustering firmware. Components (logic or memory) are represented as solid–line boxes, whereas input–output interfaces are represented as dashed–line boxes. Red and blue triangles state whether a component belongs to the 350 MHz clock domain region or to the 250 MHz one, respectively. FiFos with different read and write clocks are marked with both red and blue triangles.

The clustering architecture can be described in terms of the following schematic elements:

- input–output interfaces – formats and protocols used to exchange data between the clustering and the surrounding components;
- clock domains – different clocks used to drive the clustering logic, with appropriate clock crossing between clock domains;
- logic and memory components – the actual components responsible for the cluster reconstruction and buffering;
- monitoring – auxiliary components used to monitor the health of each firmware component during data taking;
- error handling – dedicated error detection and recovery logic assigned to all the main components.

4.2 Input–output interfaces

As with the majority of firmware components within the LHCb firmware, the clustering receives data on a 256–bit bus. Given that each SP is encoded in a 32–bit word, the data bus can carry up to eight SPs per clock cycle, where the input–interface clock frequency, as well as the output one, is 250 MHz. Each 256–bit word comes with a binary valid signal that states whether or not a word is valid. Data words are grouped into packets, each carrying data from a specific event. The first word of a packet is identified with a start–of–packet (SOP) binary signal and the last word with an end–of–packet (EOP) binary signal. The first word of a packet comes with some auxiliary signals, the FTYPE signal (8 bits) identifying the type of the packet for a specific subdetector, the FSIZE signal (16 bits) stating the size of a packet in bytes, and the BXID (12 bits) signal carrying the bunch crossing identifier[1]. Together with data–related information, a timing and fast control signal (TFC, 64 bits) is propagated together with a corresponding binary valid signal. The TFC architecture is described in Sect. 2.9. Figure 4.3 shows an example of input–output interface signals, where a valid data packet, containing SPs, goes into the clustering block (Fig. 4.3 left) and a corresponding valid data packet, containing clusters, leaves the same block (Fig. 4.3 right). In the following, the word metadata is used to refer to signals that do not carry data (SP or cluster), such as FTYPE and FSIZE.

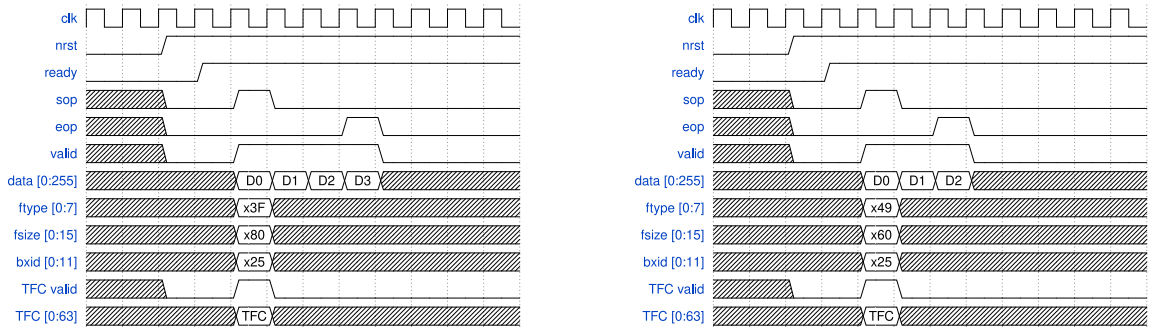

Figure 4.3: Example of (left) SP input data and (right) cluster output data.

[1]Section 5.3.1 details how the VELO firmware deals with detector latency.

4.2.1 Data formats

Active SPs are encoded by 32–bit words, as shown in Fig. 4.4. Each word contains the pixel hitmap (8 bits), the SP position within the sensor (15 bits), and the sensor identifier within the sensor pair (1 bit). Each VELO sensor is made of 256×768 pixels. Each SP is composed of 4×2 pixels, such that 6 bits are needed to specify the SP row, whereas 9 bits are required for the column. One extra bit is needed to identify the source sensor, as each data chain receives SPs from two sensors.

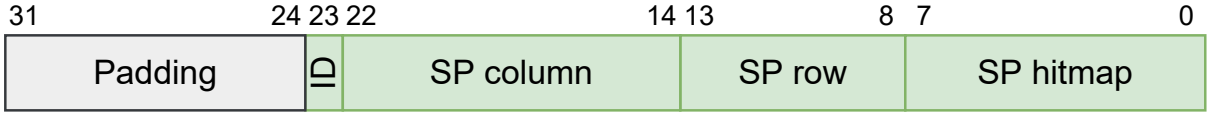

Figure 4.4: Data formats for VELO SuperPixels.

Clusters are encoded in 32–bit output words, as sketched in Fig. 4.5. Of these, 22 bits are used to specify the position of the cluster centroid, with 18 bits specifying the position of the pixel where the cluster centroid is located (Integer column and Integer row), and additional 4 bits are used to specify the position of the centroid within the pixel, in units of 1/4 of a pixel (Frac col and Frac row). Analogously to SP data, 1 bit is used to identify the sensor (ID). Eight additional bits are used to encode a cluster–topology identifier (Topology ID) and the reconstruction–quality flags (Flags). The topology identifier is used to distinguish cluster topologies that share the same centroid position within the pixel, so that the full cluster topology can be retrieved. If the cluster is reconstructed from an isolated SP (bit 30 = 1 in Fig. 4.5 top), six bits are used to store the topology identifier, whereas five bits are needed to store the identifier for clusters reconstructed through the matrices (bit 30 = 0 in Fig. 4.5 bottom).

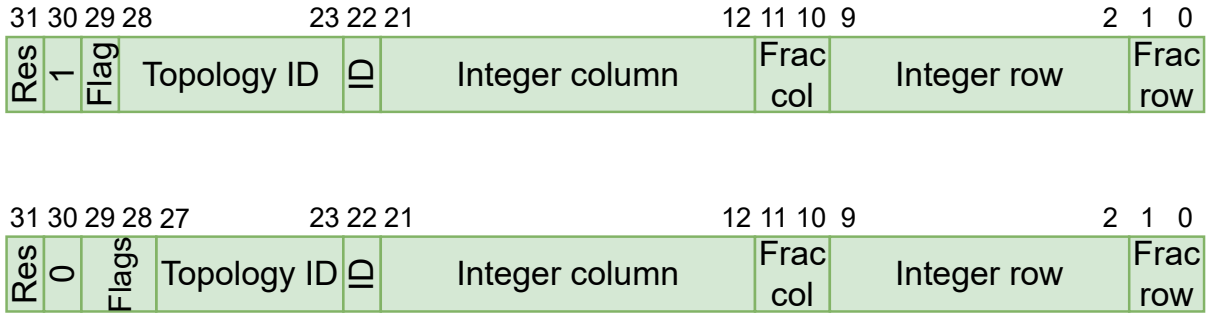

Figure 4.5: Data formats for (top) clusters reconstructed from isolated SPs and (bottom) clusters reconstructed from not isolated SPs. Bit 31 (Res) is reserved for internal use.

The cluster topology information is used both for monitoring purposes and for the ultimate optimization of the tracking performance, as the uncertainty associated with the 2D position of a cluster depends on its topology. The reconstruction–quality flags allow clusters from isolated SPs to be distinguished from clusters built from SPs overflowing the maximum number of instantiated matrices (which are arbitrarily treated as isolated). For clusters reconstructed within a matrix, these flags specify whether a cluster was fully contained in the 3×3 grid (Fig. 4.6 left) and whether the grid touched the boundary of the host matrix, which potentially means that the reconstructed cluster is a fragment of a larger one (Fig. 4.6 right).

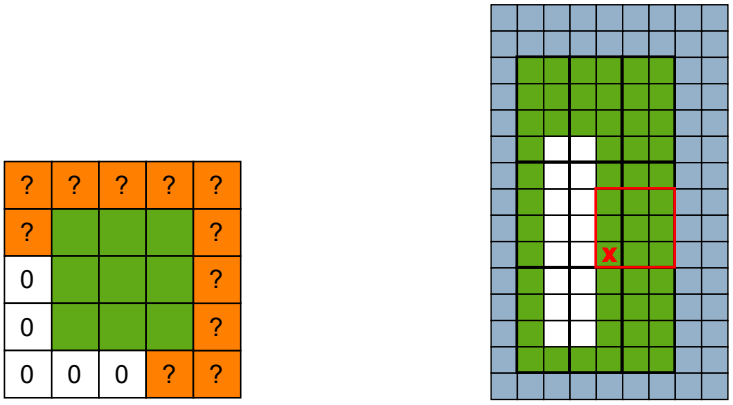

Figure 4.6: Graphical explanation of `self-contained` and `edge` flags. Gray pixels outside the matrix are fixed to zero. A cluster is flagged as `self-contained` if all surrounding pixels (orange pixels in the left image) are not active. A cluster is flagged as at the `edge` of the matrix if its anchor pixel is one of the green pixels in the right image.

Table 4.1 shows the returned values of the reconstruction quality flags, if a cluster is reconstructed from (left) an isolated SP or (right) within the matrix chain.

Meaning	Flag
Isolated	1
Overflow	0

Meaning	Flag
Self–contained & edge	11
Self–contained & not–edge	10
Not–self–contained & edge	01
Not–self–contained & not–edge	00

Table 4.1: Values and meanings of the reconstruction quality flags.

Topology ID

The topology identifier and the fractional parts of the position of the cluster center allow the derivation of the 3×3 or 2×4 full cluster topology. As shown in Fig. 4.7, only 320 3×3 topology configurations, out of the 512 (2^9), lead to a reconstructed cluster. This is due to the shape of the patterns used to find the anchor pixel in the cluster search (Fig. 3.2).

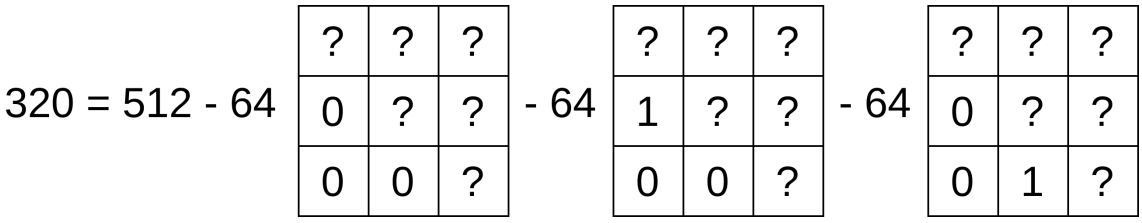

Figure 4.7: Count of the 3×3 topologies that lead to a cluster.

Table 4.2 shows all the possible fractional part configurations of the cluster centers for 3×3 topologies, as well as the number of appearances of each configuration. The majority of the

N° topologies	Frac. part (x,y)	N° topologies	Frac. part (x,y)
71	(0.00, 0.00)	14	(0.00, 0.50)
27	(0.75, 0.00)	14	(0.50, 0.00)
27	(0.00, 0.75)	12	(0.25, 0.25)
25	(0.25, 0.00)	10	(0.50, 0.50)
25	(0.00, 0.25)	8	(0.50, 0.75)
23	(0.75, 0.75)	8	(0.75, 0.50)
22	(0.25, 0.75)	6	(0.50, 0.25)
22	(0.75, 0.25)	6	(0.25, 0.50)

Table 4.2: Number of 3×3 topologies with the same fractional parts.

clusters has (0.00, 0.00) as fractional part; however, 28 configurations, out of 71, lead to a single cluster in the given 3×3 pixel grid. The remaining 43 configurations contain two independent clusters that are reconstructed using two different patterns (associated to two different anchor pixels), with a topology belonging to the 28 configurations leading to a single cluster. Figure 4.8 shows two examples of these occurrences, where the two topology configurations, associated with the reconstructed clusters, are the same (top) or different (bottom). In conclusion, in order to unambiguously identify 28 different topology configurations at most, five bits are needed to encode the full topology, as shown in Fig. 4.5 (bottom).

Similar arguments can be made to determine the minimum number of bits needed to identify 2×4 topology configurations that share the same fractional parts of the cluster center position. As above, Table 4.3 shows all the possible fractional part configurations of cluster centers for the 2×4 topologies, as well as the number of appearances. Only 12 configurations, out of a total

N° topologies	Frac. part (x,y)	N° topologies	Frac. part (x,y)
48	(0.00, 0.00)	11	(0.25, 0.75)
45	(0.50, 0.50)	11	(0.75, 0.25)
32	(0.50, 0.00)	11	(0.75, 0.75)
20	(0.50, 0.75)	8	(0.25, 0.00)
20	(0.50, 0.25)	8	(0.25, 0.50)
14	(0.00, 0.50)	8	(0.75, 0.00)
11	(0.25, 0.25)	8	(0.75, 0.50)

Table 4.3: Number of 2×4 topologies with the same fractional parts.

of 48, have (0.00, 0.00) as fractional part with a single reconstructed cluster. The remaining 36 configurations contain two independent clusters. On the other hand, out of the 45 configurations having (0.50, 0.50) as fractional part, 36 have a single reconstructed cluster. Thus, in order to unambiguously identify at most 36 different configurations that share the same fractional part, six bits are needed to encode the full topology as shown in Fig. 4.5 top.

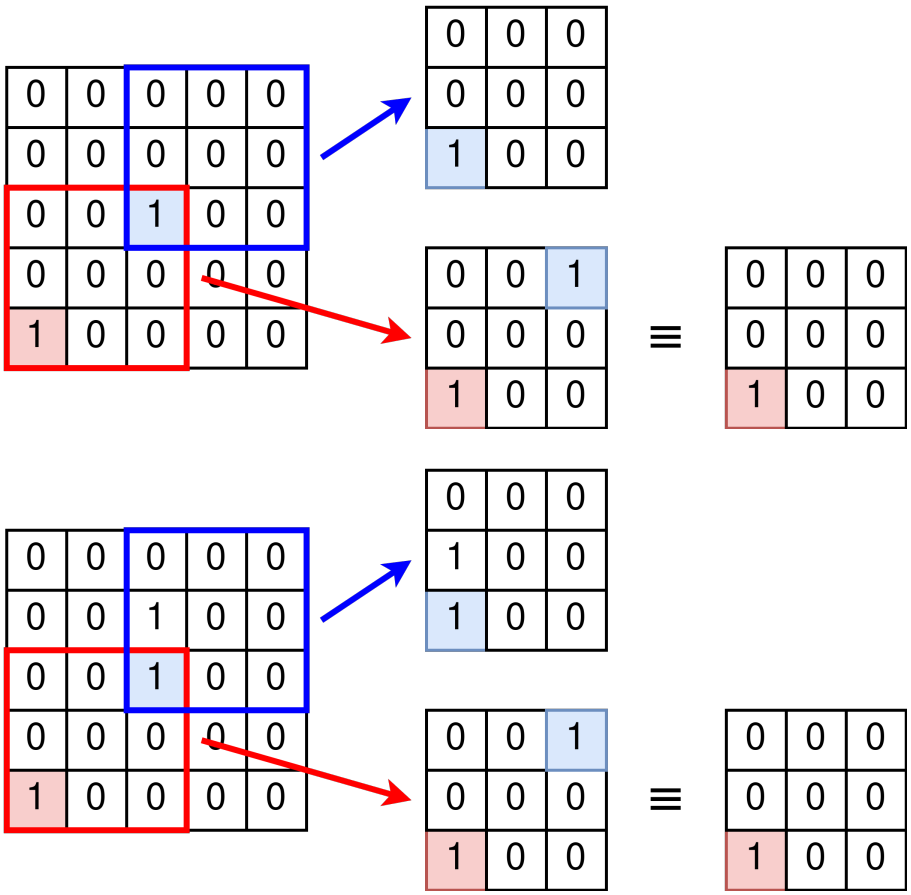

Figure 4.8: Example of a 3×3 topology configuration with (0.00, 0.00) fractional parts containing two independent reconstructed clusters (red square). The two reconstructed clusters (top) have the same topology, and (bottom) do not have the same topology.

4.3 Clock domains

Logic and memory components within the clustering architecture are driven by two clock sources, with frequencies of 250 MHz and 350 MHz. The clocks are created using PLLs that take the LHC 40 MHz external clock as input. Figure 4.2 shows the clock domain to which each component belongs using blue and red triangles, for 250 MHz and 350 MHz clock, respectively. These values have been chosen to ensure that the system as a whole can provide a throughput in excess of 30 MHz[2], while still respecting the timing constraints due to internal signal propagation in all of its parts (see Appendices A and B). The optimal working point of each firmware component has been evaluated using firmware simulation tools and official LHCb simulated data samples. Further details on the tests performed with the first functioning prototype of the clustering architecture are described in Sect. 5.1. Dual–clock FiFos are placed between firmware components running at different clock speeds to allow data synchronization between different clock domains.

[2]40 MHz is the crossing frequency of the LHC RF–buckets, however not all buckets are filled with protons (bunches). The averaged bunch–bunch collision frequency over the LHC cycle is 30 MHz.

4.4 Logic and memory components

This section describes the firmware implementation details for each component within the clustering architecture. The overall clustering architecture is built in a modular way, where each component performs a specific task. This approach eases the design, allowing separate components, devoted to different tasks, to be built and tested separately. It also allows the processing of one component to be decoupled from the processing of close–by components, introducing decoupling FiFos between them. Apart from clock crossing synchronization, decoupling FiFos absorb different processing rates of consecutive components limited in time, by reducing the overall rigidity of the system to the data flow[3]. FiFos allow also long data paths to be reduced, giving the firmware compiler more freedom to place components within the FPGA chip, resulting in lower compilation times and better timings. Each FiFo receives read and write requests, together with the data to be written, from the surrounding components, and exposes the read data and status signals. Status signals are used to show whether a FiFo is full or empty to avoid writing or reading, respectively. Within the clustering firmware an almost–full signal is used instead of the full one, which is raised if the occupancy of the FiFo is higher than a certain threshold. In this way, the component writing data to a FiFo does not have to stop writing as soon as it receives the full signal but can still write for a few clock cycles more, typically.

4.4.1 Input FiFos

Incoming data and metadata to the clustering are stored in the following three input FiFos:

- the postRAM FiFo stores the input SPs and can contain up to 1024 294–bit words, where each word is made of 256 bits of SP data and 38 bits of metadata;
- the TFC FiFo contains up to 512 64–bit TFC words, one for each event, received with the SOP signal. This FiFo buffers the input TFC words to be read at the end of the clustering processing block, keeping the event synchronicity;
- the command FiFo in which up to 512 32–bit commands are written. Each word is a subset of the TFC word and contains the command and the BXID of the event to which the command has to be applied. As for the TFC FiFo, a word is written at the start of each event, received with the SOP signal. See Sect. 4.4.3 for more details.

These three FiFos represent the input buffering stage of the clustering and are read by the subsequent components.

4.4.2 Decoder feeder

PostRAM and command FiFos are read by the decoder feeder, which, as the name suggests, sends data to the following block, the decoder. Figure 4.9 shows a block diagram of the decoder feeder that is built on a simple finite state machine that issues read requests to the postRAM and command FiFos, sending read data and corresponding valid signals to the decoder. The FSM receives a hold signal from the decoder stating that the decoder is not ready to accept any more data because it is busy elaborating previously received data. The FSM issues read requests if FiFos are not empty, reading the command FiFo once per event.

[3]Considering two firmware components placed one after the other having the same average throughput, a decoupling FiFo in between can store data output from the first component while the second one is temporarily busy and can provide previously buffered data to the second component even if the first one is not outputting data.

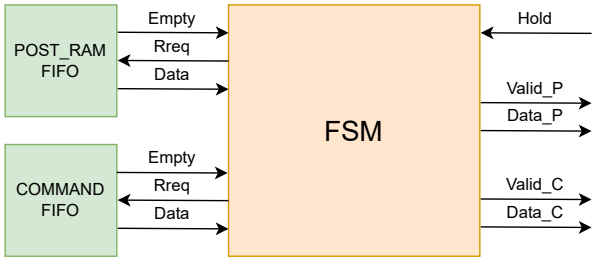

Figure 4.9: Block diagram of the feeder reading the postRAM and the command FiFos. Data and valid signals related to the postRAM FiFo (Valid_P and Data_P) and the ones related to the command FiFo (Valid_C and Data_C) are sent to the decoder. A hold signal is received from the decoder stating the it is not ready to accept data.

4.4.3 Decoder block

Data output from the decoder feeder are received by the decoder, a block diagram of which is shown in Fig. 4.10. As a first step, the decoder reads the command, originally stored in

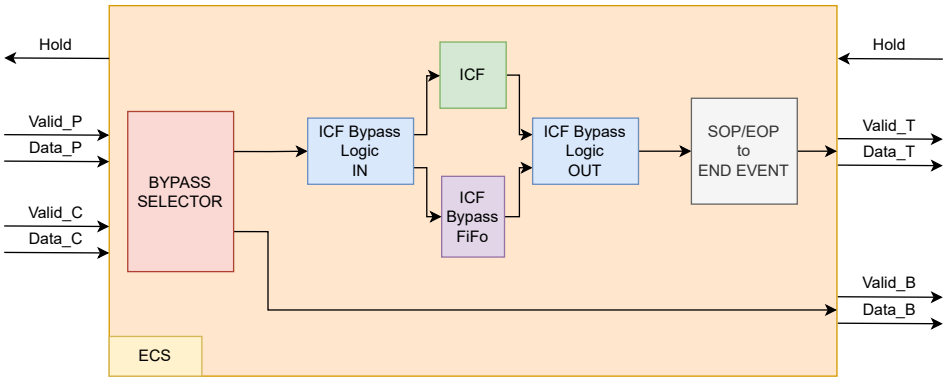

Figure 4.10: Block diagram of the decoder, that is responsible for bypassing input data not containing SPs, for performing the isolation cluster flagging, for splitting a single 256–bit stream into eight 32–bit streams while moving from a SOP/EOP logic to an EndEvent one.

the command FiFo, and decides whether the input data contain valid SPs or control words to be bypassed. In the latter case, the bypass selector sends the input data to the bypass FiFo, together with a valid signal (Valid_B and Data_B signals in Fig. 4.10). At the same time, it sends a dummy word to be propagated throughout the cluster processing to be received by the component responsible for bypass FiFo reads, so that it knows when to perform a read request to the bypass FiFo, keeping the right event synchronicity. Valid SPs, together with dummy words, are sent from the bypass selector to the Isolation Cluster Flagging (ICF) bypass logic. As detailed in the following section, the ICF can perform isolation flagging if the input packet contains less than 144 SPs. If this is not the case, the entire packet is bypassed, saved in the ICF Bypass FiFo, and a dummy word is propagated through the ICF block. As the ICF block performs the flagging on input SPs, it sends its output to the output part of the ICF Bypass logic. The purpose of this component is to propagate flagged SPs and to identify bypass dummy words, in which case it performs a read request to the ICF Bypass FiFo. Flagged and bypassed data are sent to the SOP/EOP–to–EndEvent converter. This block is responsible for splitting

the 256–bit words into eight 32–bit streams so that individual SPs can be reconstructed into clusters. It is also responsible for converting the SOP–EOP protocol to the EndEvent (EE) protocol used within the clustering architecture: a 32–bit EE word is interposed between SPs of different events in all the eight data streams. Each EE word carries a specific flag to distinguish it from SPs and an event identifier (5 bits) that can be used during subsequent data processing stages to cross–check data synchronization. Fig. 4.11 shows an example of input and output data to the SOP/EOP–to–EndEvent converter.

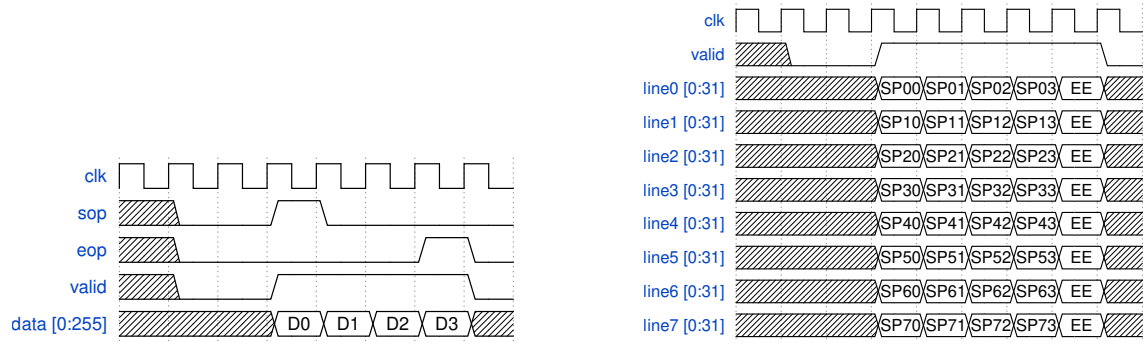

Figure 4.11: Example of (left) input and (right) output data to the SOP/EOP–to–EndEvent converter shown in Fig. 4.10, where the 256–bit bus is split into eight 32–bit streams and the SOP/EOP control logic is converted into a EndEvent logic.

Isolation flagging

SPs are flagged with an isolation bit inside the decoder block, as shown in Fig. 4.10. The flagging process includes five steps: read, buffer, load, flag, and write, arranged in a pipeline, as shown in Fig. 4.12. First, all SPs of a given event are read and stored in registers. The maximum number

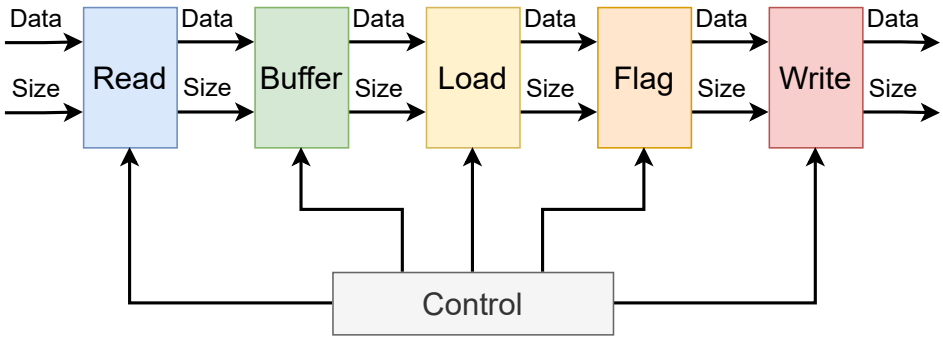

Figure 4.12: Block diagram of the isolation flagging.

of SPs that can be stored in the read registers is not dynamically adjustable. It has been set to 144 based on the distribution of the expected number of SPs in the most crowded VELO module, requiring more than 98% of the LHCb simulated events to be accommodated in the read registers. In events where the number of SPs exceeds the size of the read registers, SPs are not sent to the flagging process, but are instead bypassed and sent directly to the cluster reconstruction, using the bypass mechanism previously discussed.

As soon as all SPs belonging to one event are received, the content of the read registers is copied to the buffer. This data exchange decouples the reading and flagging operations, allowing SPs of an event to be read in while the flagging of the previous event is still ongoing. The flagging process compares the coordinates of each SP with those of the other SPs in the same event. A status vector is used to store the isolation flag for each SP: if two SPs are found to be neighbors, the corresponding bits in the status vector are set to one. SP comparisons are not performed all in a single clock cycle. At each clock edge, the load block extracts two subsets of 16 SPs each from the buffer (Fig. 4.12). For each SP in the two subsets, it also computes the set of coordinates to be matched by the neighbors by one–unit additions and subtractions of the coordinates of the SP row and column. The two SP subsets, together with the coordinates of the neighbors, are passed to the flag block that performs the 16×16 comparisons on the two subsets. For each SP of the first subset, the flag block checks if the SP row is equal to one of the rows of the SPs in the second subset or to the row above or below; the same check is performed on columns. If both row and column checks yield a positive result, the two SPs are flagged as neighbors, and the corresponding bits in the status vector are set to one. On each clock cycle, the load block selects a different pair of SP subsets from the buffer, sending them to the flag block until all possible combinations of 16–SP subsets have been checked. The described architecture allows the same logic resources to be re-used while updating the SP subsets to be flagged at each clock cycle. To make comparisons between n 16–SP blocks, $n(n+1)/2$ clock cycles are needed.

The number of parallel comparisons performed for every clock cycle is the result of a trade–off between resource usage and throughput and is based on the constraints of its use within the LHCb experiment. As soon as all comparisons are completed, the contents of the flagging registers and the status vector are copied to the write block, thus decoupling the flag and write processes. The write block is responsible for adding the isolation flag to the SP words and for sending flagged data to the next component, the switch. The data exchange within the read–load–flag–write pipeline is regulated by back–pressure: if a component cannot accept the data of an event because it is still processing the previous event, the control unit keeps the previous component on hold.

4.4.4 Transfer and Bypass FiFos

Data output from the decoder are stored in two FiFos: the transfer FiFo and the bypass FiFo. The first one contains flagged SPs, while the second one stores bypassed words. Both are vector FiFos, meaning that they consist of more FiFos grouped together. In both cases, eight FiFos that can contain up to 1024 32–bit words are instantiated. Unlike a single 256–bit FiFo, a collection of FiFos allows read operations to be performed independently on each FiFo. This is a key feature of the following components, such as the switch, which require each SP to be accessible individually. The transfer and bypass FiFos send their almost–full signals to the decoder to be used as hold signals: if either of the FiFos is approaching its depth limit, the decoder stops sending data to them.

4.4.5 Switch

The SPs stored in the transfer FiFo are not ordered by sensors within a VELO half–module nor by isolation flag. The switch, placed after the decoder, arranges SPs by sensor and isolation flag, feeding them to the appropriate cluster–reconstructing blocks. Fig. 4.13 shows the two switching units, each of them handling the data of four of the eight FiFos within the transfer vector FiFo.

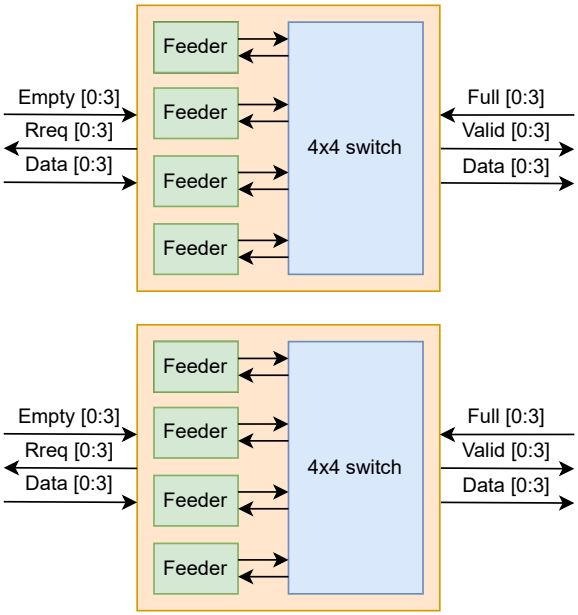

Figure 4.13: Block diagram of the switch units.

Within a switch unit, four feeders send read requests to the transfer FiFos, according to their empty state and the state of the 4×4 switch. If a FiFo contains data and the corresponding input line of the 4×4 switch on which the feeder should write the data is not busy, the feeder performs a read request to the corresponding FiFo. Each of the two switching units performs a 4→4 switching, allowing every input data word to be directed to any of the four output streams according to its sensor number and isolation flag, regardless of the origin input stream. The basic components of the switch are the splitter and the merger (Fig. 4.14). The former has one input and two outputs, and it sends input data to one of the two outputs according to their isolation flag or origin sensor. The latter has two inputs and one output, and it routes two inputs in a single output line. Two splitters and two mergers combine to form a 2→2 dispatcher. To implement a switch with $2n$ inputs/outputs, N two–way dispatchers connected together are needed, where

$$N(n) = 2N(n-1) + 2^{(n-1)}.$$

The block diagram of the splitter is shown in Fig. 4.15. The splitter is based on a finite state machine (FSM). The next state is determined by the R0 register state, the arrival of valid input data, and the hold state of the following processing block. On the arrival of valid input data, the FSM decides between sending it directly to the output or storing it in the R0 register, based on the input hold signal. In the latter case, a latch–enable (LE) write signal is sent to the register. A multiplexer controlled by the FSM routes data to the output. If a SP is received, then one of the two valid signals is set to one, according to the routing scheme (isolation flag or sensor). If an EE signal arrives, it is sent to both outputs. The input hold signal determines whether data can be sent to the output. An output hold is generated as long as the R0 register is full, since no more data can be accepted as input, given the possibility of an input hold signal assertion.

The block diagram of the merger is shown in Fig. 4.16. As for the splitter, a FSM determines if input data can be sent directly to the output or must be stored in appropriate registers (R0 and R1). If an EE word arrives on one of the inputs, it is stored until a second EE word arrives

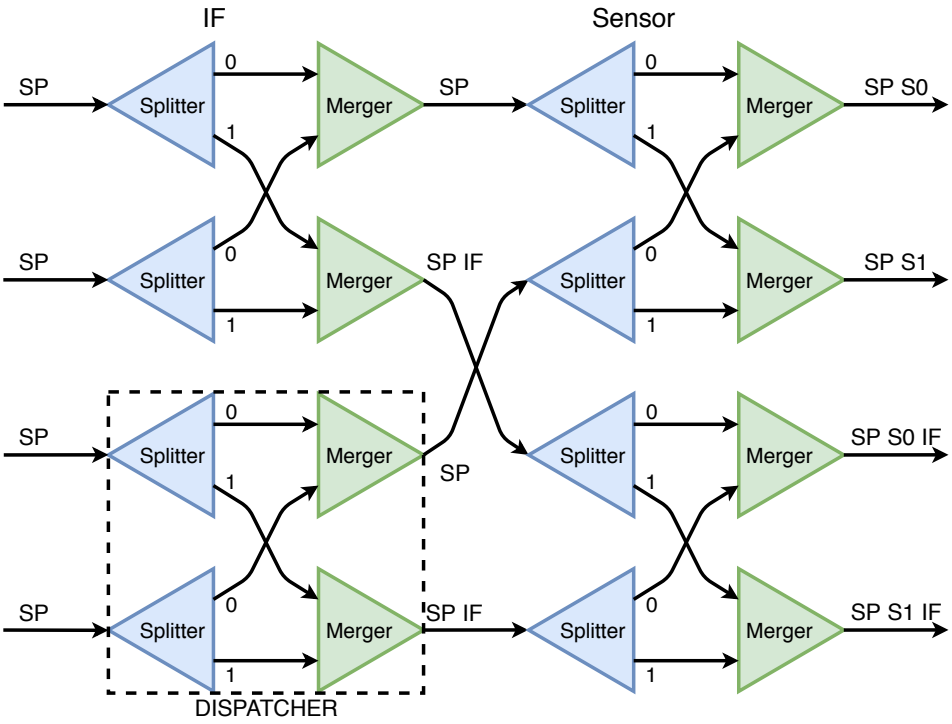

Figure 4.14: Block diagram of a 4 to 4 switching unit.

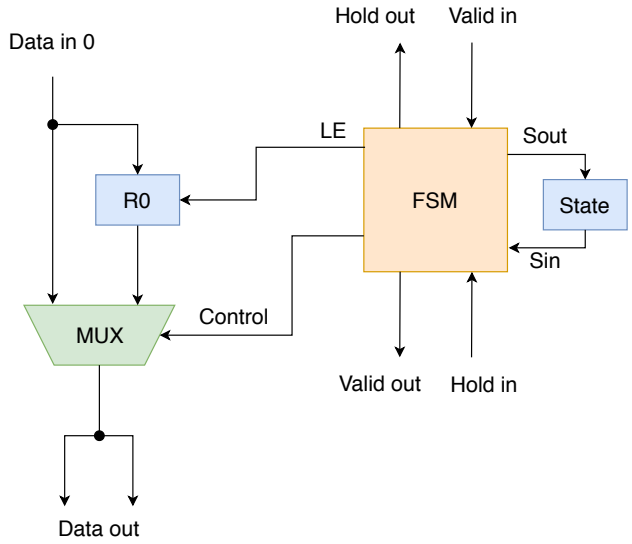

Figure 4.15: Splitter block diagram. R0 and State are registers, MUX is a multiplexer and FSM is a finite state machine that manages hold, valid, control and latch enable (LE) write signals.

on the other input. The two EE words are then compared and, if their event IDs match, a single EE word is output; otherwise, a sync error signal is set to one.

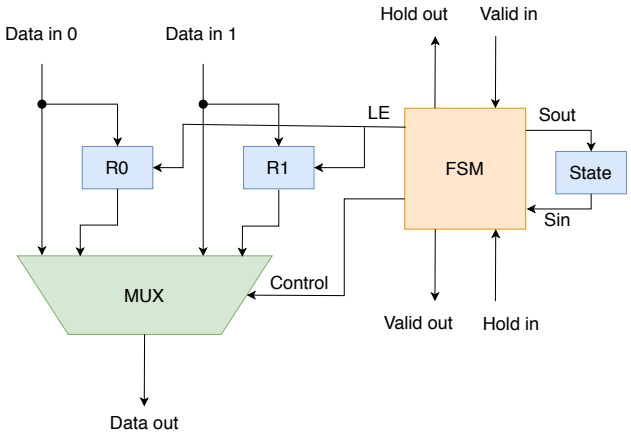

Figure 4.16: Merger block diagram. R0, R1 and State are registers, MUX is a multiplexer and FSM is a finite state machine that manages hold, valid, control and latch enable (LE) write signals.

4.4.6 Switch FiFo

SP data processed by the switch are stored in the switch FiFo. Like transfer and bypass FiFos, the switch FiFo is also a vector component made of eight FiFos that can contain up to 512 32–bit words. Out of the eight FiFos, two contain isolated SPs from sensor zero, two contain isolated SPs from sensor one, two contain non–isolated SPs from sensor zero, and two contain non–isolated SPs from sensor one.

4.4.7 Cluster reconstruction – isolated SPs

All isolated SPs, identified by the switch and stored in the four corresponding switch FiFos, are read by the cluster–reconstructing component shown in Fig. 4.17. Four feeders read the SP data

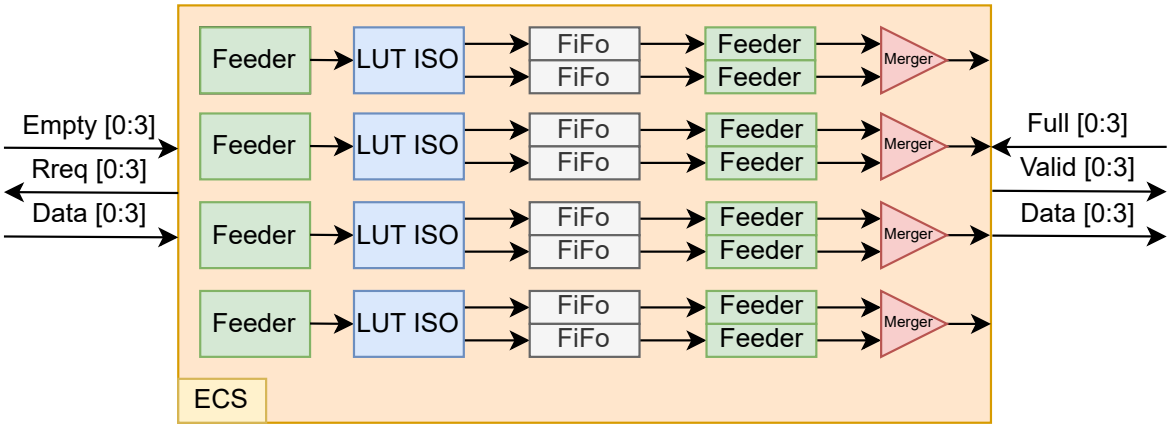

Figure 4.17: Sketch of the component reconstructing clusters from isolated SPs.

from the four FiFos, two containing SPs from isolated SPs of sensor zero and two of sensor one. SPs are sent to four corresponding LUTs that reconstruct the clusters from the pixel content of each SP, as shown in Fig. 4.18 (left). Each LUT reconstructs the cluster centroid from the

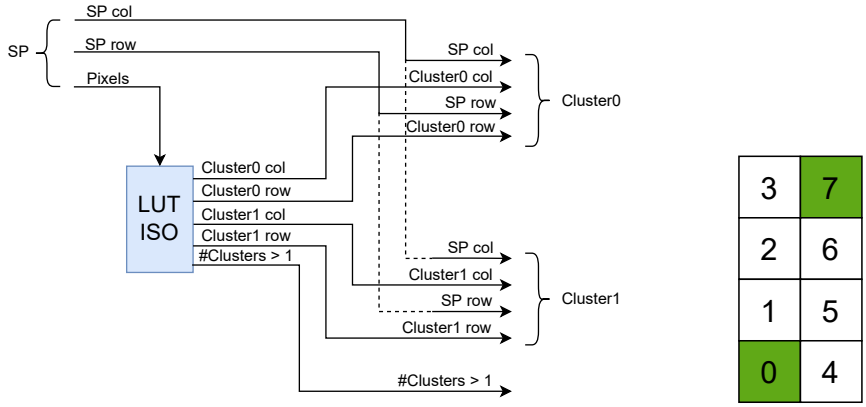

Figure 4.18: (left) Cluster reconstruction of isolated SPs by means of a LUT and (right) example of isolated SPs containing 2 clusters.

active–pixel hitmap extracted from the SP word. The cluster word is built by combining the LUT output with the original SP row and column. As shown in Fig. 4.18 (right), isolated SPs can contain two individual clusters. If this is the case, the LUT reconstructs the two clusters and a corresponding bit is set to one (#Clusters in the left part of Fig. 4.18). Each isolated LUT has 256 addresses, each of which contains a 27–bit word. The SP hit map is used as the address to access the LUT, using the pixel order shown in Fig. 4.18 (right). The 27–bit output word from the LUT contains one bit stating if there are two clusters within the same isolated SPs and two 13–bit words, each carrying the seven–bit position of the cluster and the six–bit topology identifier. Reconstructed clusters are written in FiFos (Fig. 4.17), placed after the LUTs. Each FiFo is read by a feeder, and the eight lines coming out of the feeders are merged into four, using the same switch merger shown in Fig. 4.16, and sent to the output.

4.4.8 Cluster reconstruction – non–isolated SPs

Non–isolated SPs are resolved into clusters using two instances, one for each of the two sensors, of the same firmware component shown in Fig. 4.19. Input data are read from switch FiFos using two feeders and sent to the matrix chain. In order to ensure a high throughput, each matrix receives data from two parallel input lines. Each input line is combined with a hold signal, that is propagated backwards through the whole chain to control the data flow by back–pressure, avoiding data loss. As the first SP populates a matrix, a set of coordinates is calculated and stored, to be matched with all further SPs arriving at the same matrix. The initialization of an empty matrix is done using only one of the two input lines, since only a single SP can enter the center of the matrix at a time. A second SP coming simultaneously from the other parallel line would need the coordinates of the free slots to fill the matrix, which cannot be immediately available due to timing constraints. For this reason, input line 0 (Fig. 4.19) has the priority over input line 1 during a matrix initialization. In order to maintain good load balance, input lines are swapped when going from one matrix to the next: line 0 of a matrix feeds line 1 of the next matrix and vice versa. When EE words have arrived on both input lines, the content of the matrix is moved to the cluster finder block. An error is raised if two different EE signals are detected.

The cluster–finder block processes the content of the corresponding filled matrix. Figure 4.20 shows the logic of how clusters are reconstructed, starting from the matrix pixel content. Each

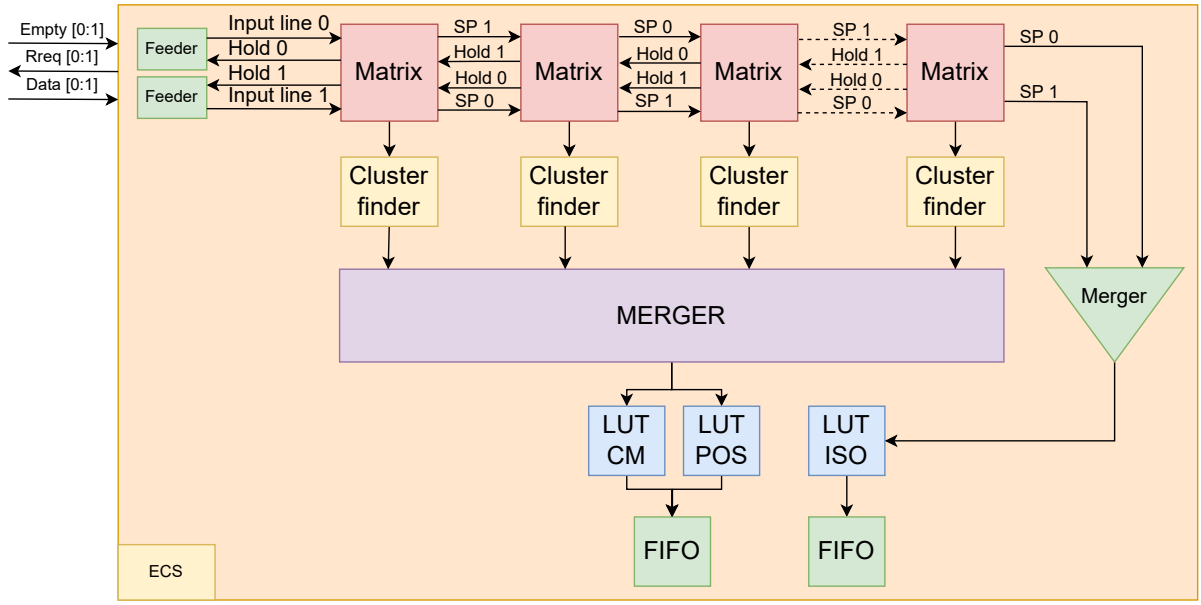

Figure 4.19: SP distribution in a matrix chain. Clusters are reconstructed through the cluster finder block and merged into a FiFo.

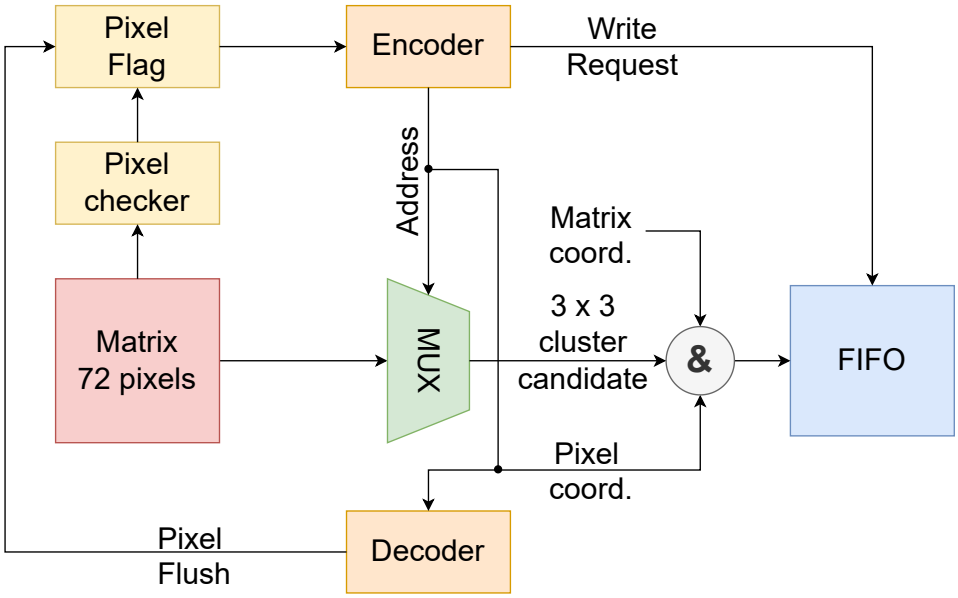

Figure 4.20: Cluster finder block diagram and its data flow.

pixel in a matrix checks if it belongs to one of the L–shaped patterns of the algorithm through the pixel checker block. This process is performed in parallel at full speed for each pixel in each matrix. When a pattern match is found, an anchor pixel in the matrix is identified. As a consequence, the bit in the pixel flag vector corresponding to the position of the anchor pixel in the matrix is set to one. An encoder reads the content of the pixel flag vector and passes the addresses of all anchor pixels found to a multiplexer, one at a time. The multiplexer extracts

the 3×3 cluster candidate corresponding to the address received from the encoder. As soon as an anchor pixel has been processed and the corresponding cluster candidate found, the decoder block receives the pixel address from the encoder and resets the corresponding bit to zero in the pixel flag vector. The reset operation is performed by means of the pixel flush signal. For each cluster, a word containing the matrix coordinates, the anchor pixel position, and the 3×3 cluster candidate is written to the matrix FiFo. A merger (purple rectangle in Fig. 4.19) reads the reconstructed clusters from the matrix FiFos. Fig. 4.21 shows how the merger is built. A series of feeders perform read requests to the matrix FiFos and the read data are fed to a chain of 2–to–1 mergers (the same switch merger component shown in Fig. 4.16). The 20 input lines are merged into a single output line.

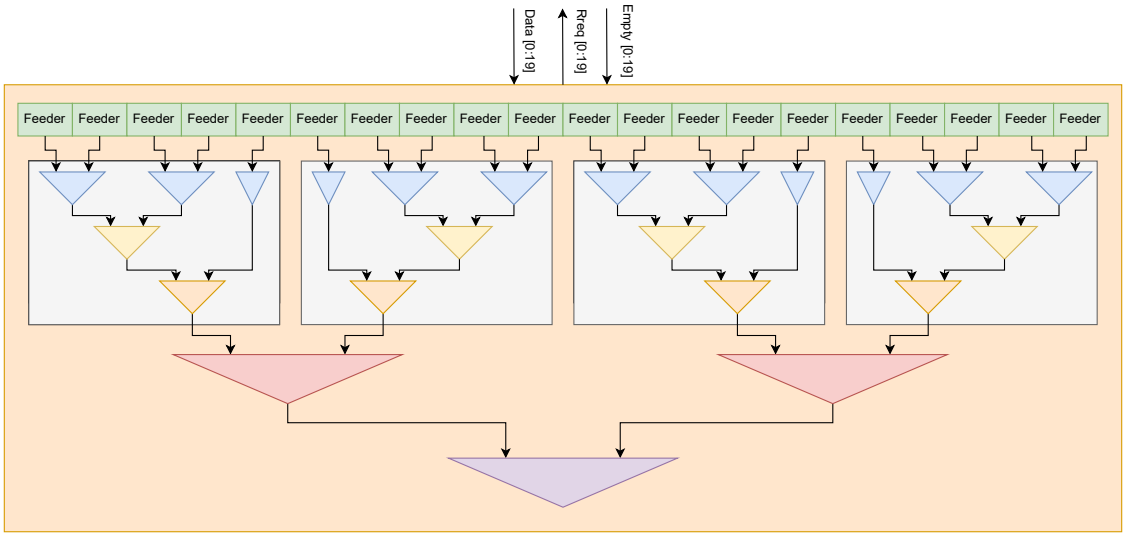

Figure 4.21: Block diagram of the merger collecting reconstructed clusters from matrices.

The merger output is sent to two LUTs, one computing the center of mass (LUT CM) of the cluster, starting from the 3×3 cluster candidate and the other (LUT POS) computes the position of the anchor pixel within the matrix. The center–of–mass LUT has 512 addresses, one for each of the possible pixel configurations within the 3×3 candidate. Each address contains a 11–bit word, where six bits are used for the cluster center position and five bits are used for the topology identifier. The position LUT has 128 addresses to be able to extract the anchor pixel position from the 72 possible configurations inside the matrix. Each address contains a 7–bit word, of which four are used to identify the row and three the column. The cluster position is obtained by combining the matrix position in the detector, the anchor–pixel position in the matrix, and the center of mass position within the 3×3 cluster candidate. Cluster words are then saved into a FiFo that contains all the clusters from non–isolated SPs of a VELO sensor that do not overflow the matrix chain.

The two data lines at the end of the matrix chain that carry overflow SPs are merged into a single line. Overflow SPs are reconstructed as if they were isolated by means of a LUT with the same content of the one used for isolated–SP reconstruction, and the reconstructed clusters are stored in a FiFo.

4.4.9 Cluster output FiFo

Reconstructed clusters are stored in the output FiFo, which is a vector component that contains eight FiFos. Each of the eight FiFos can store up to 1024 32–bit words. Out of the eight FiFos, four are dedicated to store clusters from isolated SPs, two for sensor zero and two for sensor one. The remaining four FiFos contain clusters reconstructed from non–isolated SPs, two for sensor zero and two for sensor one, where, for a given sensor, one FiFo stores clusters output from the matrix chain, whereas the other contains clusters reconstructed from overflow SPs.

4.4.10 Encoder, bypass merger and event logic converter

Reconstructed clusters are read from the output FiFo by means of a feeder and sent to the encoder. The output of the encoder is sent to the bypass merger, which reads the bypass FiFo and incorporates the previously bypassed data into the data stream. The EE–to–SOP/EOP converter converts the EndEvent logic used inside the cluster–reconstruction components back to a SOP/EOP logic. These operations are performed by the same firmware entity sketched in Fig. 4.22. The encoder, as the name implies, is devoted to encoding the eight separate 32–bit

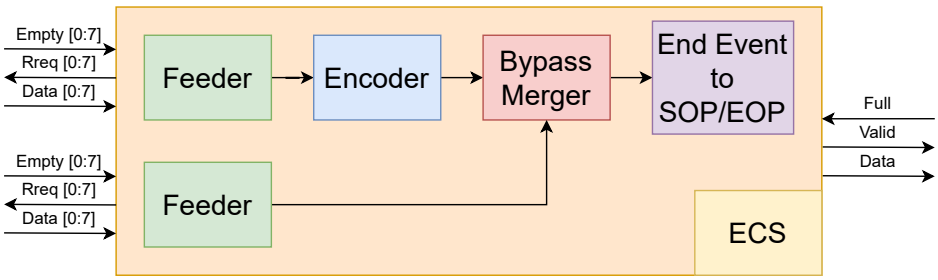

Figure 4.22: Structure of the firmware component performing data encoding, bypass merging and EE–to–SOP/EOP conversion.

data streams into a single 256–bit bus to comply with the required output format. The encoder architecture has been designed as a trade–off between speed and bandwidth optimization. The encoder is required to output a 256–bit word at each clock cycle to maintain a throughput larger than 30 MHz. Given the speed constraint, the SP packing performed by the encoder is not optimal in each event, interleaving zero–padded words between 256–bit words to match the output width. To build the complete 8→1 encoder, seven 2→1 encoders are instantiated, as shown in Fig. 4.23. The 2→1 encoder block puts together two input data lines (N + N bits) into a single output (2N bits) by means of buffer registers (R0, R1, and R3) and a control FSM (Fig. 4.24). If two cluster words are received and no hold signal is asserted by the subsequent block, the two words are packed together and sent out. If a single cluster is received, it is stored in the R3 register and matched to the next input cluster. If a hold signal is received, the incoming cluster is stored in the R0 or R1 register, depending on its input line. In case an odd number of words is received within an event, a zero–padded word is added to match the 2N output width. When two EE signals are received, they are compared and, if they match, sent out. Otherwise, an error signal is generated.

The bypass merger, shown in Fig. 4.25, is responsible for reading the data from the encoder output, while incorporating the previously bypassed data into the data stream at the appropriate time (see Sect. 4.4.3). As in the case of the encoder, a FSM manages read request operations to the bypass FiFo, sending the appropriate data to the output and handling hold signals. As soon

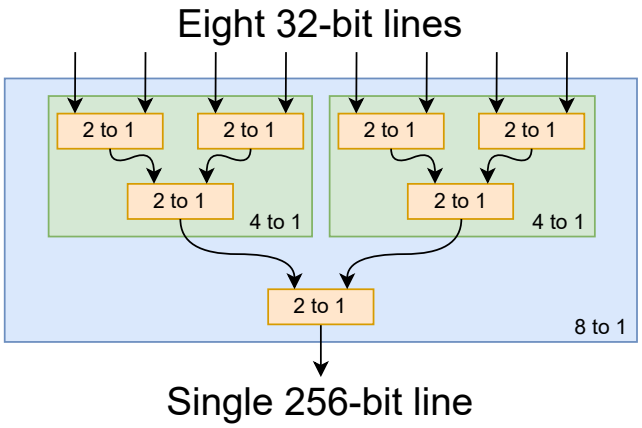

Figure 4.23: Structure of a 8→1 encoder built from 2→1 encoders.

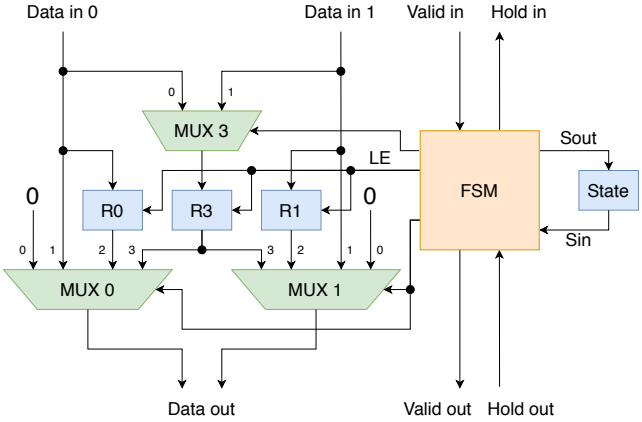

Figure 4.24: 2–to–1 encoder block diagram. R0, R1, R3 and State are registers, MUX0, MUX1 and MUX3 are multiplexers and FSM is a finite state machine that manages hold, valid and latch enable (LE) write signals.

as the R1 register is free and the bypass FiFo is not empty, the FSM performs a read request to the FiFo, so that a bypassed word is ready in the register to be sent to the output. Whenever cluster or EE words are received, they are sent to the output, if the input hold signal is low, otherwise, input data are stored in the R0 register. As soon as a dummy bypass word is received, the bypassed data are sent to the output, keeping the correct event synchronicity. The FSM issues the latch enable signals if data need to be stored in the R0 or R1 registers, while keeping track of whether they already contain data using the state register. The FSM also triggers the Control signal to decide which data should be sent to the output. Before sending the data to the next component, an EE–to–SOP/EOP converter changes the logic used to define the start and end of an event from EE words to SOP and EOP signals. Its architecture is similar to the 2→1 encoder and bypass merger: a FSM manages the internal operations, sending as an output both data and SOP/EOP signals, taking into account the ready state of the following component. Each time an EE signal is received, an EOP is issued to be propagated with the last word of the previous event, and a SOP signal is associated with the first word of the following event.

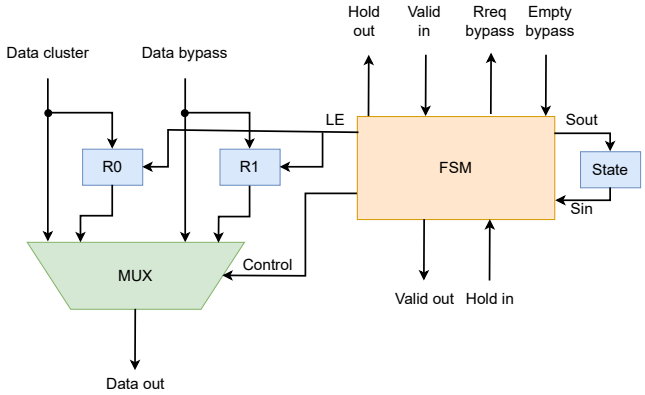

Figure 4.25: Block diagram of the bypass merger. R0, R1 and State are registers, MUX is a multiplexers and FSM is a finite state machine that manages hold, valid, latch enable (LE) and read request signals.

4.4.11 Size counter

Data from the EE–to–SOP/EOP converter are stored in a FiFo that can contain up to 1024 258–bit words. These words contain reconstructed clusters (256 bits), the SOP, and the EOP signals (one bit each). As data are written to the FiFo, they are also counted to compute the FSIZE of the event. The computed size is stored in a separate FiFo that can contain up to 1024 16–bit words. Fig. 4.26 shows the firmware component responsible for the FSIZE computation.

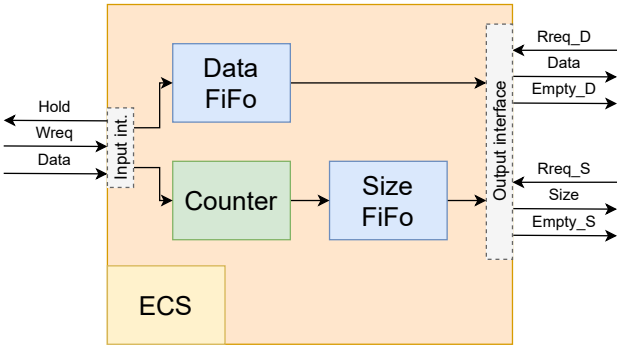

Figure 4.26: Block diagram of FSIZE counter firmware component.

4.4.12 Double–output feeder

The double–output feeder reads data from the data and size FiFos within the size counter. When the following component is ready to accept data, the feeder performs read requests to the data FiFo and, once per event, to the size FiFo. At the same time, it also issues read requests to the TFC FiFo (Sect. 4.4.1) that buffers TFC words, right at the beginning of the clustering block.

4.4.13 Double–output

During normal data–taking operations, only reconstructed clusters are sent to the output. In this case, clusters and the corresponding metadata are stored in a cluster FiFo, which can contain up

to 512 350–bit words, made of 256–bit cluster words, 64–bit TFC words, 16–bit FSIZE words, 12–bit BXID words, SOP and EOP signals. The cluster FiFo is read by a feeder and its content is sent to the output. For debugging and monitoring purposes, the firmware is equipped with a double–output mode that allows both clusters and SPs of the same event to be sent to the output, when enabled. Fig. 4.27 shows the structure of the double–output component. The

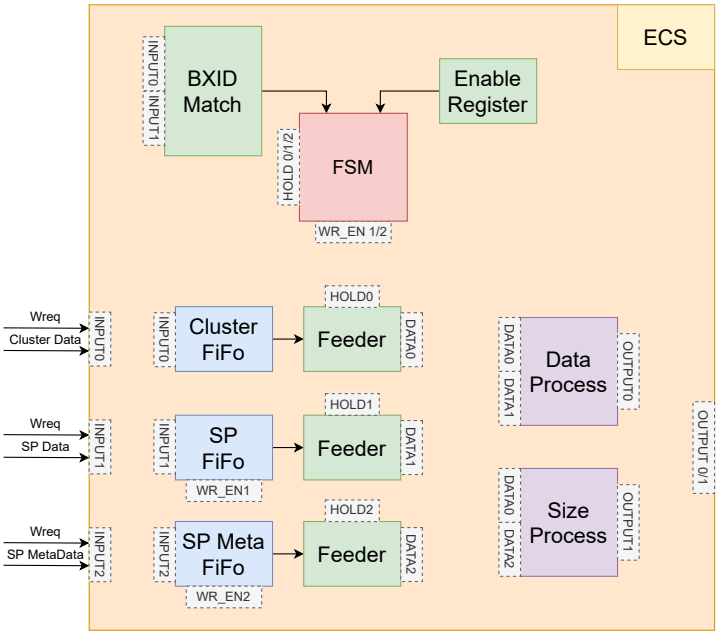

Figure 4.27: Block diagram of the double output firmware component.

double–output mode is enabled through ECS, by writing to the corresponding enable register. In this case, SP data and the corresponding metadata are written to the SP and SP Meta FiFos. The first one can contain up to 1024 294–bit words, whereas the second one 1024 24–bit words. Once the double–output mode is enabled, the BXID of the first SP packet is stored so that it can be matched with the BXID of the corresponding cluster packet (BXID match). When the match is found, the data process starts to concatenate cluster and SP data: first, all the 256–bit words containing reconstructed clusters are sent out, followed by SP words. The first cluster word is accompanied by the SOP, BXID, TFC, FTYPE, and FSIZE signals. The last SP word goes along with the EOP signal. A dedicated FTYPE value (0x61) is assigned to double–output packets so that they can be distinguished from SP and cluster packets. The unused bit 31 of each 32–bit SP within the 256–bit word is set to one to distinguish clusters from SPs. The size of the entire packet is computed by the size process that adds cluster and SP sizes, taking care of the padding. Since the size has to be provided with the first word in the packet, SP sizes are stored in the SP meta FiFo that is read independently from the SP FiFo containing data. A FSM manages the overall operation of the double–output component, guiding the read requests from the feeders to the FiFos and enabling the write operations to the SP FiFos, when the double–output mode is on.

4.5 Monitoring and error handling

The clustering architecture has several blocks whose behavior affects the functioning of the entire data processing chain. Therefore, a monitoring procedure is implemented to probe each block throughout the entire reconstruction process to ensure correct data handling. Monitoring is performed through ECS read and write requests, as shown in the left and right waveforms of Fig. 4.28, respectively. The occupancy levels of all FiFos in the design, as well as their maxima

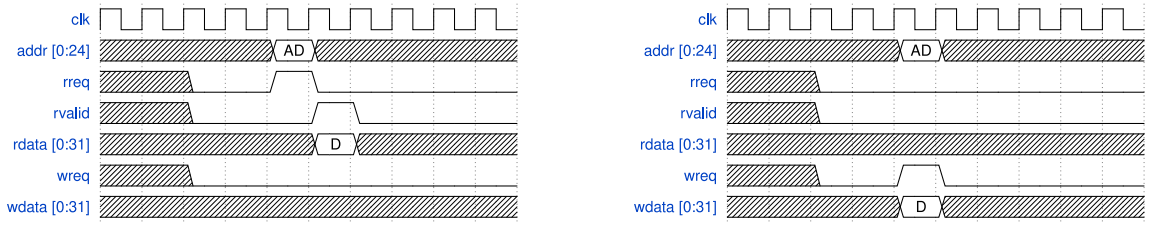

Figure 4.28: Example of (left) an ECS read operation and (right) an ECS write operation.

over a certain time interval, are periodically read to check for, and diagnose, possible slowdowns of any processing blocks. The fraction of SPs that overflow the matrix chain is also monitored.

Each processing block in Fig. 4.2 is also equipped with an error–checking logic that monitors two types of errors. The first type corresponds to a data loss, occurring when a block receives valid data in input and the register in which data should be written is already full. The second type occurs when mismatching EE signals are received, indicating a loss of synchronization in the input data. In both cases, a signal is generated and an error word is output, containing a code to trace back the origin of the error for debugging purposes. A reset signal needs to be sent to the clustering logic and memories to recover from both error types. The clustering block may also receive an error word from the previous firmware components. Error words are identified by a specific FTYPE (0x1D). In case an error word is received, the corresponding BXID is saved in a FiFo within the error manager, and the data are bypassed. Before being sent to the output, the BXID of the data is compared to the one stored in the error manager FiFo. If a match is found, the appropriate error FTYPE is propagated. The error BXID buffering mechanism described here is implemented locally because the BXID signal is not propagated throughout the entire cluster reconstruction chain; instead, it is recovered at the end of the chain, reading the TFC FiFo.

Chapter 5

Firmware and software integration

This chapter details the integration process of the clustering firmware within the VELO DAQ chain, as well as all the changes and updates of the LHCb software stack, needed to decode, process, and simulate FPGA VELO clusters instead of SPs. It covers all the steps of this process, from the first tests of the first functioning clustering firmware on a prototyping board to the final measurement of a significant improvement of the HLT1 reconstruction throughput.

5.1 Highlights of the development stage

The first functioning prototype of the clustering firmware consisted of the core components between the decoder and the encoder stages, as shown in Fig. 4.2, excluding the ICF. The firmware was designed, extensively tested, and debugged in simulation over the course of one year. Figure 5.1 shows an example of the QuestaSim® simulator used to test the behavior of each firmware component and the interplay between components. Once the overall behavior

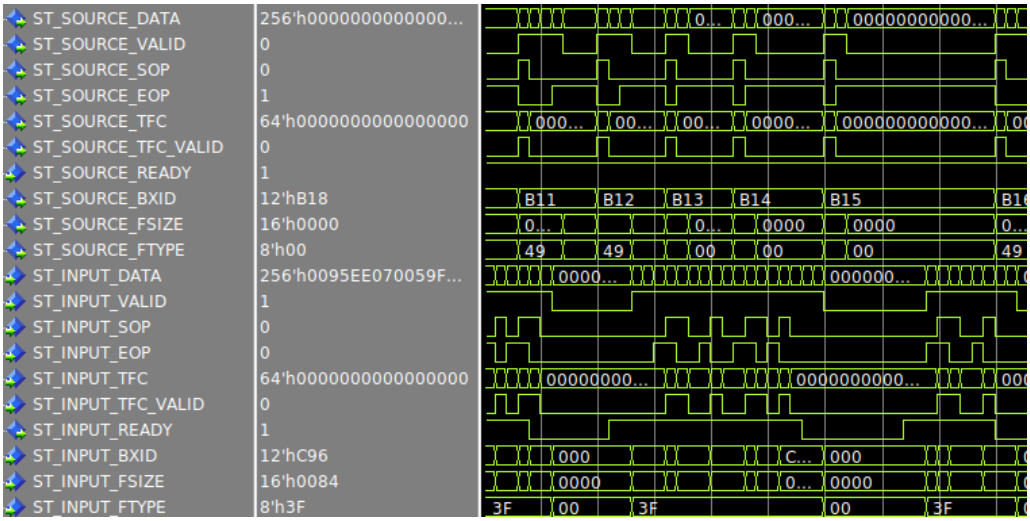

Figure 5.1: Display of the QuestaSim® firmware simulation tool, showing input (ST_INPUT) and output (ST_SOURCE) interfaces of the clustering firmware. Meanings and purposes of signals are detailed in Sect. 4.2.

was checked to be correct, the firmware was compiled using the Quartus® software tool. As an output of the compilation, the following main parameters are reported:

- resources required; the compiler reports the amount of logic and memory resources that are needed to implement the firmware on the FPGA. The first prototype required 24% of ALM logic and 6% of M20K memories to perform cluster reconstruction of a VELO module. See Appendix A for further details on FPGA building blocks. The resource values need to be kept as low as possible in order to be able to integrate the clustering firmware within the VELO DAQ firmware (Sect. 5.3);
- timings; unlike simulation tools, the firmware compiler takes into account the actual FPGA fabric and its clock resources when placing the firmware into the FPGA blocks. As a consequence, it reports potential timing issues related to one or more clock sources used to drive the logic. Reports show the maximum clock frequencies at which the firmware can be run, together with the critical paths that limit its frequency, and some optimization suggestions. See Appendix B for further details on timing issues.

5.2 First tests on hardware

A key step during the firmware development stage was hardware testing, where the compiled firmware was loaded into a FPGA and tested using simulated data as input. Hardware tests are essential for the development, as they allow the firmware to be run on a real chip for longer time with respect to the simulation, which may highlight bugs and weaknesses of the design. Hardware tests allow also the firmware throughput to be measured in real time and to stress test the chosen design, finding its limits, for example, by progressively increasing the clock frequencies at which it runs. Figure 5.2 shows the prototyping system used for the first tests of the clustering firmware, together with its main components. The system is a DNS5GX_F2 (DN0237) from Dini group enclosed within a 4U rack–mount chassis. It consists of a FPGA–based main board along with up to four modular I/O boards. The main board supports two Intel® Stratix® V GX FPGAs[1], together with a Marvell Discovery processor used for system monitoring and control. An additional Stratix V configuration FPGA provides configuration and I/O control for the main board peripherals and subsystems. The main advantage of testing the clustering firmware on this type of board is the high degree of control and monitoring allowed both via software and by directly probing relevant signals with an external oscilloscope. Another key advantage is given by the similarities between the FPGA mounted on the DNS5GX_F2 prototyping board and the one used for DAQ purposes on the PCIe40 card, as shown in Table 5.1. Hardware tests were performed loading simulated SP data into input RAMs[2] which are read in loop to continuously feed data into the cluster reconstruction block. The clusters reconstructed in hardware were read and compared to the output of the high–level C++ simulation of the algorithm, run on the same set of input SPs. The quality of the reconstructed clusters was verified. Further details on the software emulation of the clustering algorithm are presented in Sect. 5.4. In addition to reconstruction quality checks, hardware tests allow the average reconstruction throughput to be measured. To be able to run the firmware online, it is mandatory that this throughput is higher than 30 MHz, within the VELO DAQ cards. To measure the throughput the system was set up in order to send a pulse signal on a pin on the board every 256 reconstructed events. These

[1]The FPGA model number is 5SGXEABN2F45C3.
[2]Hence the name of postRAM for the first clustering buffering FiFos in Fig. 4.2.

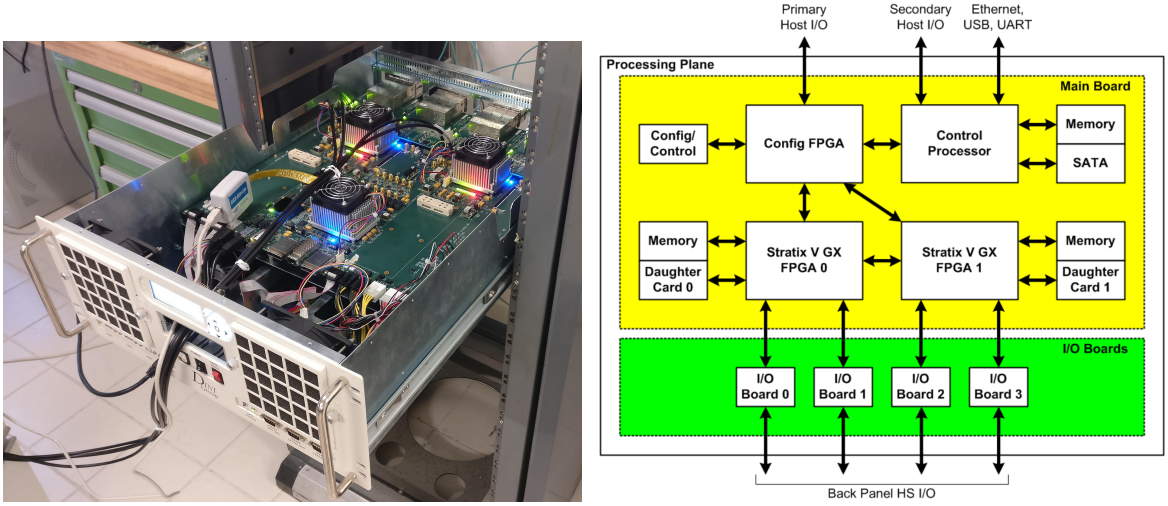

Figure 5.2: (left) Top view of the DINI prototyping board used for hardware testing and (right) block diagram of the main components of the board.

	5SGXEABN2F45C3	10AX115S4F45E2SG
Family	Stratix	Arria
Node	28 nm	20 nm
ALM	359200	427200
M20K	2640	2713
DSP	352	1518
CLK	650 MHz	644 MHz

Table 5.1: Comparison of Intel® Stratix® V and Arria® 10 FPGAs used in the prototyping board and in the PCIe40 board, respectively.

signals were measured and stored using a digital oscilloscope and the corresponding average throughput was measured. Figure 5.3 shows an oscilloscope screenshot used for throughput measurements. It turns out that the firmware can process events with up to an average of 32 SPs per VELO half–module, using a 350 MHz clock rate, which corresponds to a speed of 0.96 GSpixel/s or 7.7 GPixel/s, given that each SP is made of eight pixels. This condition is met for the entire VELO detector, where the average occupancy is 26 SPs per event, near the nominal interaction point. An average event processing rate of 38.9 MHz is measured on minimum–bias LHC collision events, in the most crowded VELO module, at the Run 3 nominal instantaneous luminosity of 2×10^{33} cm^{-2}s^{-1} . The measurement is also performed on pp collisions with higher than average track multiplicity, containing reconstructible $B_s^0 \to \phi\phi$ decays, as a sample of typical data that the LHCb DAQ would select and save on permanent storage. The measured throughput of 30.9 MHz is still higher than the average LHC bunch crossing rate and ensures that even a random fluctuation due to the occurrence of several high–occupancy events in a row poses no risk of clogging the pipeline. Input RAMs were set up to inject an event every 25 ns, in order to test the back–pressure mechanism and the resistance to trains of more consecutive events. Moreover the high–occupancy test shows that the firmware, and in particular its buffering stages (FiFos), can cope with a high number of input SPs per event, thus

being resistant to potential large tails in the distribution of the number of clusters per event. Therefore, the clustering firmware is expected to run safely throughout the entire Run 3 physics data taking.

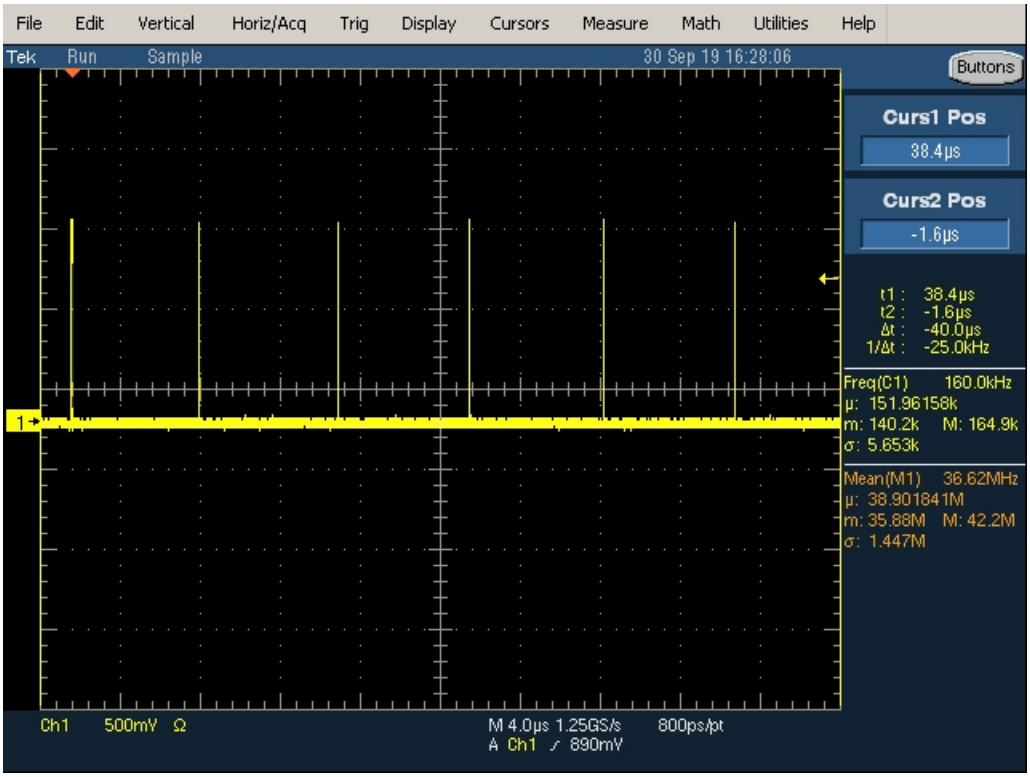

Figure 5.3: Oscilloscope screenshot used for throughput measurements. The system is set up such that a signal is generated every 256 reconstructed events.

5.3 Firmware integration

The first working firmware prototype allowed us to measure its performance in terms of cluster reconstruction quality and throughput, ensuring that it can run online. Given also the relatively low amount of FPGA resources required, we integrated the clustering within the VELO firmware, taking all the necessary steps to make our implementation LHCb–compliant.

5.3.1 VELO firmware

This section is intended to provide more details on the VELO TELL40 firmware [67] that are key to understanding the steps taken to integrate the clustering within the VELO DAQ chain. The key characteristics of the VELO detector are described in Sect. 2.3. Figure 5.4 shows the main components of the VELO TELL40 firmware, including the clustering reconstruction. The VELO TELL40 firmware is structured into two independent data streams. Each stream receives data from 10 optical links connected to the FE electronics of a VELO half–module and sends out reconstructed clusters via a PCIe Gen 3 $\times 8$ bus. A low–level interface surrounds the core of the firmware, providing functions such as transceiver interfaces for the optical links and PCIe

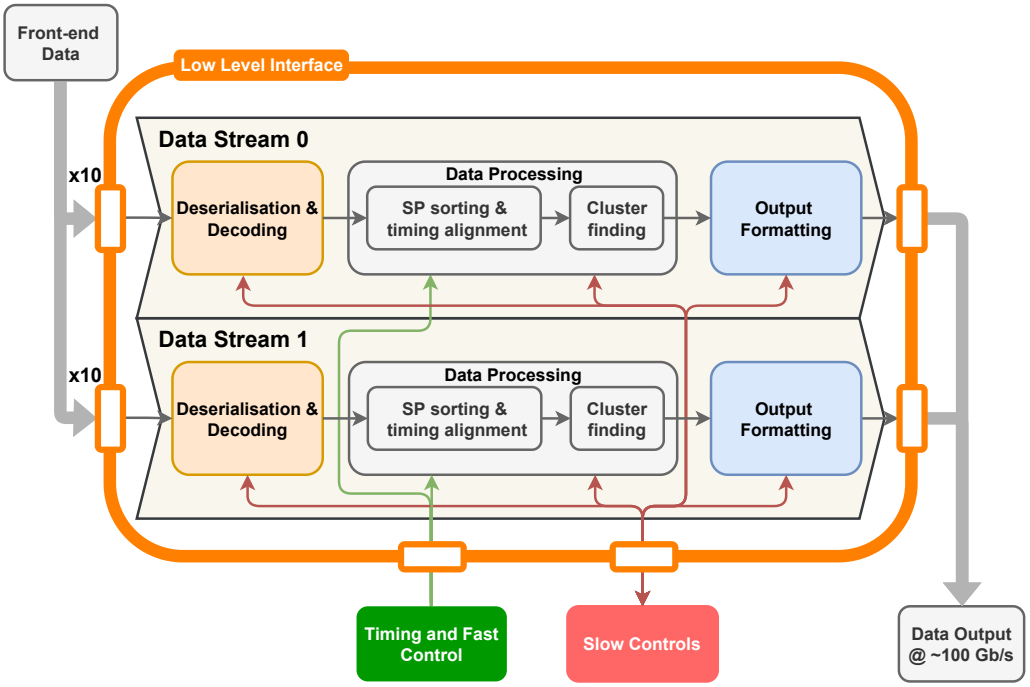

Figure 5.4: Block diagram of the VELO TELL40 firmware. Data from the detector front–end are received via 20 optical links and is sent to two independent data streams. Data are then decoded, processed, formatted, and sent to the host server via the PCIe bus. Data processing include SP extraction, timestamp sorting and timing alignment.

bus, slow control capabilities for reading and writing registers in the firmware, and the interface to the timing and fast control system. These components, together with the Output Formatting block[3], are the same for all the LHCb subdetectors, whereas the Deserialization & Decoding and the Data Processing block are VELO specific. As a first step, data are decoded and deserialized. A 128–bit GWT frame is received from each of the 20 input links every 25 ns. Each frame can carry up to four SPs. Together with SP data, the frame contains parity information to identify possible transmission errors and four header bits. Since the GWT frame arrives as a serial stream, a bit–slip operation is performed on data to identify the header bits. When the header is identified, the link is locked. Deserialized data are then sent to the data processing block, where several operations occur. First, the pre–router component extracts the four SPs from each GWT frame, discarding the empty ones, and performing a retiming from the 40 MHz input rate to the 160 MHz one, so that SPs can be treated individually. The pre–router also adds three bits to each SP word in order to identify its origin VELO chip. The following component, the router, performs the timing sorting of the SPs. This operation is required since the VeloPix chip has a columnar readout logic, where data fragments traverse the length of the column from their origin. This implies that data fragments produced at the top and bottom of the pixel matrix have significantly different readout latencies and data from different events might be read out in a mixed order. The router takes the 9–bit timestamp assigned to each SP within the VeloPix and performs a time sorting. The sorting operation is first performed on the first four most

[3]The Output Formatting block formats the data into LHCb event fragments, adding metadata information as the event number and the source identifier to trace back the origin of the data. These metadata are essential for event–building purposes where information from multiple subdetectors and different events is put together.

significant bits by means of switching blocks, interleaved with buffering FiFos. The remaining five least significant bits are sorted using a series of multiplexers. Sorted SPs are stored in 512 RAMs, one for each of the possible values of the 9–bit timestamp. Time–sorted SPs are sent to the post–router that performs the alignment with the LHCb TFC timing system, moving from the partial VELO 9–bit identifier to the 12–bit LHCb one[4]. The clustering firmware is inserted right after the post–router block so that, instead of sending out SPs, clusters are reconstructed from them and passed to the Output Formatting stage.

5.3.2 Clustering within VELO

In order to fully integrate clustering within the VELO firmware, several changes were required that led to the complete firmware sketched in Fig. 4.2. The main changes applied are the following.

- SOP–EOP to EndEvent logic conversion. As described in Sect. 4.4.3, the clustering processing uses an EndEvent logic, placing special EE words between SPs or clusters belonging to different events. This allows the independent treatment of data lines while maintaining event synchronization and being able to check for potential data corruption. However, the standard LHCb implementation, and hence the VELO one, to distinguish data belonging to different events is to propagate Start Of Packet and End Of Packet signals. In order to comply with the SOP–EOP logic, two conversion layers were added to the decoder and to the encoder stages to move from SOP–EOP to EE and vice versa, respectively.

- Bypass non–SP data. Input data to the clustering block might not contain valid SPs, in which case data need to be bypassed and retrieved at the end of the clustering process with the right timing. For this purpose, a bypass mechanism has been put in place, where the decoder decides whether or not an input data packet contains meaningful SPs based on TFC information. In the latter case, input data are written to a bypass FiFo, and a dummy placeholder word is propagated throughout the clustering chain. Once the dummy word enters the encoder, it is recognized and non–SP data are retrieved from the bypass FiFo and added to the output stream.

- Compute FSIZE. To be able to efficiently perform event building and access built data, the size of the data in bytes needs to be computed within the FPGA and sent out as metadata. The size counter component was added after the encoder stage to count the number of reconstructed clusters and provide the correct value of FSIZE for each packet.

- Remap ECS addresses. Since the first development stage, the clustering block has an ECS interface to be able to read and write register inside the firmware itself. During the integration process, the ECS addresses were remapped in order to avoid conflicts with other addresses already defined in the rest of the VELO firmware. The remap was done assigning to the most significant eight bits of the ECS address the role of component identifier in the firmware, with different sets of this eight–bit identifier for the different components.

- Add metadata signals. In addition to the core input data containing SPs, their valid signal, SOP, and EOP, the LHCb firmware requires additional metadata such as BXID and

[4] 12 bits are needed to uniquely identify each of the 3654 possible positions of the bunch inside the beam.

FTYPE, together with the aforementioned FSIZE and TFC. These ports were added to both input and output interfaces and appropriate buffering and propagation mechanisms were introduced in the clustering chain.

- Introduce the double output mode. As described in Sect. 4.4.13, a double output component is placed at the end of the clustering firmware so that, once enabled, it outputs both clusters and input SPs of the same event for debugging purposes. This extension required an additional buffering stage to hold SPs while waiting for the corresponding clusters to be reconstructed. It also required an additional control stage, activated via ECS, to enable/disable double–output operations.

The integration of the clustering firmware within the VELO framework, described in this section, was done together with Luca Giambastiani, whom I trained. More details about this topic can be found in Luca's Master thesis [78]. Besides the aforementioned changes, a reorganization and optimization of the isolation cluster flagging process was needed to be able to fit it into the FPGA, given the limited amount of resources, and to ensure that it can sustain high enough input event rates. Changes to the ICF are described in the following section.

5.3.3 Isolation cluster flagging

As described both in Sects. 3 and 4, being able to distinguish isolated SPs from not isolated ones is a key component of the clustering algorithm, allowing for a fast and efficient cluster reconstruction of the non–negligible fraction of isolated SPs. Isolation cluster flagging is performed by comparing the 2D coordinates of all SPs in each event, belonging to the same VELO sensor. If SPs are found to be neighbors, an appropriate flag is assigned to the SP word. These operations require the SPs to be buffered while the coordinate comparison is performed. Since the number of registers storing the SPs cannot be dynamically adjusted, its value was determined in simulation so that ∼99% of events can be completely contained in the ICF registers. This value turned out to be 144 SPs. During the clustering integration stage, the ICF was restructured and optimized both in terms of resource usage and throughput. Figure 5.5 shows the effects of the different optimizations on the consumption of ALM logic resources.

The first ICF implementation (black curve) required around 24% of the FPGA logic to flag up to 64 SPs per sensor pair. This performance per ALM was pour due to the flagging being done in a single clock cycle. Furthermore, timings were not met, and it was not possible to run the ICF at a high enough clock frequency to sustain the 30 MHz input event rate. Several optimizations were applied, initially trying to maintain the overall design choices. Implementing mutual SP flagging, where if SP A is a neighbor of B, also the opposite relation is true (black–to–blue transition in Fig. 5.5) and adding coordinate pre–computation while disabling decimal to Gray conversion[5] (blue–to–light–blue transition in Fig. 5.5) led to an improvement of about 57% of the resource usage. However, resource needs were still too high and a high enough throughput was yet to be reached. These considerations led to a complete redesign of the ICF, as described in Sect. 4.4.3, with the main changes being the concurrent read–flag–write operations and the reuse of the same logic to flag SPs belonging to the same event. The green curve in Fig. 5.5 shows the resource needs for the final ICF version, which exhibits a linear behavior as a function of the maximum number of SPs that can be flagged per event, contrary to the quadratic behavior of the first implementations.

[5] During the first ICF development stages, the decimal to Gray code conversion of SP word was investigated to ease the 2D coordinate comparisons. However, it turned out to have no significant benefit.

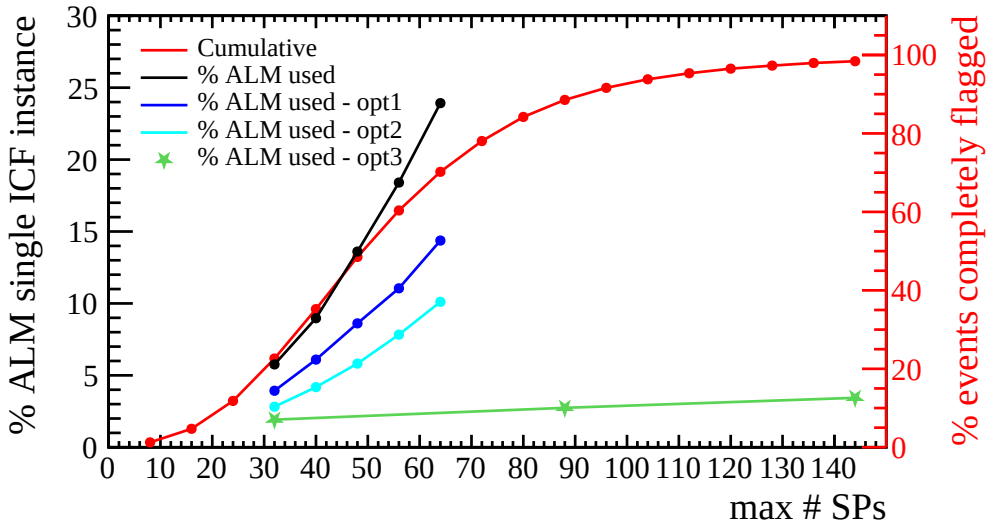

Figure 5.5: Percentage of FPGA ALM logic required to implement one instance of the ICF as a function of the maximum number of SPs that the ICF can handle per event. Two instances, one per data stream, are required to perform isolation flagging of a VELO module. Black, blue, light blue and green curves represent different ICF firmware iterations and optimizations, that are described in the text. The red curve, referring to the right axis, shows, as a fraction of the maximum number of SPs that the ICF can store, the percentage of events that can be completely flagged.

5.3.4 Simchecker

Being able to check the cluster reconstruction quality in QuestaSim® as new features and fixes are added to the firmware is paramount to ensure a stable development and maintenance cycle. For this reason we developed a simulation checking tool (simchecker) that allows a bit–level comparison to be performed between clusters reconstructed in firmware and the output of a software algorithm emulating the firmware expected behavior, integrated within the LHCb software stack[6]. The steps performed by the simchecker to verify the accuracy of the firmware output are the following:

- SPs and clusters produced via the full LHCb simulation are dumped to a file;
- the simulation output file is formatted appropriately to create both input files to be loaded into the firmware input RAMs and check files containing the expected clusters;
- the firmware simulation is run over the SP input files, and the reconstructed firmware clusters are saved to a file. All these operations are performed using QuestaSim®;
- comparison checks are run over the software–produced clusters and the firmware ones. Errors are raised if not all firmware–reconstructed clusters are present in the software list, and vice versa.

Input and check files are produced once for all VELO modules, whereas checks are repeated each time a change is added to the firmware. QuestaSim® configuration and the consequent checks have been wrapped into a set of scripts that can be executed to run the test automatically over the data from a user–defined set of VELO modules.

[6]The software side of the clustering integration is discussed in Sect. 5.4.

5.3.5 FPGA resources

Compiling through Quartus® allowed us to measure the amount of FPGA resources needed to implement the VELO firmware on the Arria 10 chip. Figure 5.6 shows a detailed report on resource usage. Overall, the VELO firmware requires 73% of the logic and 71% of the M20K

	Resource	Usage	%
1	Logic utilization (ALMs needed / total ALMs on device)	264,800 / 427,200	62 %
2	ALMs needed [=A-B+C]	264,800	
1	[A] ALMs used in final placement [=a+b+c+d]	311,220 / 427,200	73 %
2	[B] Estimate of ALMs recoverable by dense packing	51,015 / 427,200	12 %
3	[C] Estimate of ALMs unavailable [=a+b+c+d]	4,595 / 427,200	1 %
3			
4	Difficulty packing design	Low	
5			
6	Total LABs: partially or completely used	37,396 / 42,720	88 %
7			
8	Combinational ALUT usage for logic	357,842	
9	Memory ALUT usage	2,720	
10			
11			
12	Dedicated logic registers	430,297	
13			
14	Virtual pins	48	
15	I/O pins	385 / 960	40 %
16			
17	M20K blocks	1,913 / 2,713	71 %
18	Total MLAB memory bits	87,040	
19	Total block memory bits	24,959,392 / 55,562,240	45 %
20	Total block memory implementation bits	39,178,240 / 55,562,240	71 %
21			
22	Total DSP Blocks	0 / 1,518	0 %
23			
24	IOPLLs	8 / 16	50 %
25	FPLLs	10 / 32	31 %
26	Global signals	34	
1	-- Global clocks	31 / 32	97 %
2	-- Regional clocks	1 / 16	6 %
3	-- Periphery clocks	2 / 384	< 1 %
27	JTAGs	1 / 1	100 %
28	ASMI blocks	0 / 1	0 %
29	CRC blocks	0 / 1	0 %
30	Remote update blocks	0 / 1	0 %
31	Oscillator blocks	0 / 1	0 %
32	PCIe Hard IPs	2 / 4	50 %
33	HSSI RX PCSs	41 / 72	57 %
34	HSSI PMA RX DESERs	41 / 72	57 %
35	HSSI TX PCSs	41 / 72	57 %
36	HSSI PMA TX SERs	41 / 72	57 %
37	HSSI CDR PLL	72 / 72	100 %
1	-- CDR PLLs for Unused RX Clock Workaround	31 / 72	43 %
38	HSSI ATX PLL	2 / 24	8 %
39	Impedance control blocks	0 / 16	0 %
40	Average interconnect usage (total/H/V)	32.5% / 33.3% / 31.2%	
41	Peak interconnect usage (total/H/V)	80.5% / 84.6% / 82.2%	
42			
43	Programmable power technology high-speed tiles	11,514 / 18,244	63 %
44	Programmable power technology low-power tiles	6,730 / 18,244	37 %
45			
46	Programmable power technology high-speed LAB tiles	9,533 / 13,898	69 %
47	Programmable power technology low-power LAB tiles	4,365 / 13,898	31 %
48			
49	Maximum fan-out	115360	
50	Highest non-global fan-out	38493	
51	Total fan-out	3678675	

Figure 5.6: Quartus® report table summarizing the amount of FPGA resources used for the VELO firmware implementation.

memories available on the FPGA chip. More detailed reports allow also the resource consumption of the single components to be checked. Clustering requires 31% of logic and 11% of M20K memories. With respect to the resource needs of the first prototype, described in Sect. 5.1, the additional logic resources required come from the ICF that has been integrated within the clustering. Moreover, the additional memory resources required come mainly from the buffers used for metadata, bypass, and double output purposes.

Quartus® reports allow also the placement of the different firmware components within the FPGA chip to be visualized using a tool called Chip Planner. Figure 5.7 shows the placement of the VELO firmware on the FPGA highlighting the different components with different colors. Components like the Low Level Interface and the PCIe block are placed by the compiler close to the edges where the transceivers are located. On the contrary, components such as the clustering that do not need to communicate directly with the outside are placed away from transceivers.

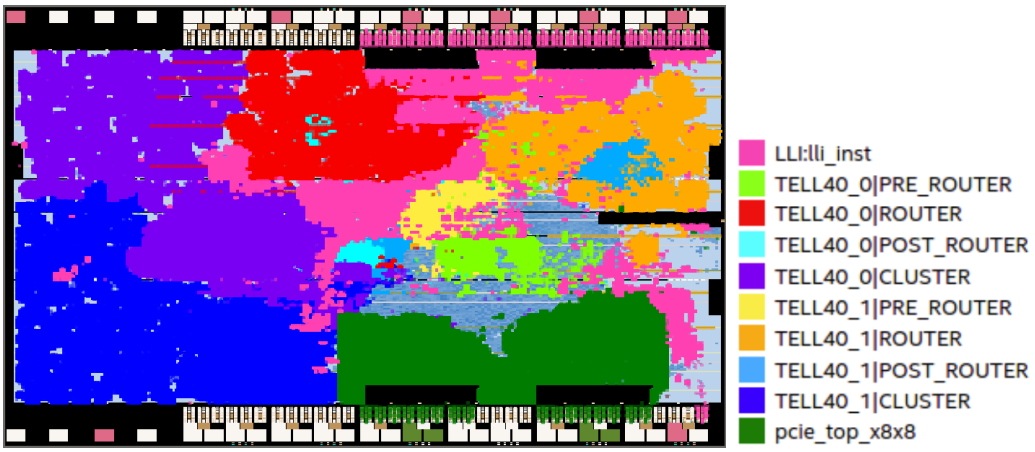

Figure 5.7: Quartus® Chip Planner view identifying different components of the VELO firmware placed within the FPGA chip. Details about different VELO firmware components can be found in Sect. 5.3.1

Figure 5.8 shows the utilization of routing resources within the FPGA, in a color–coded scale. Regions close to LLI transceivers are particularly crowded due to the high signal fan–out.

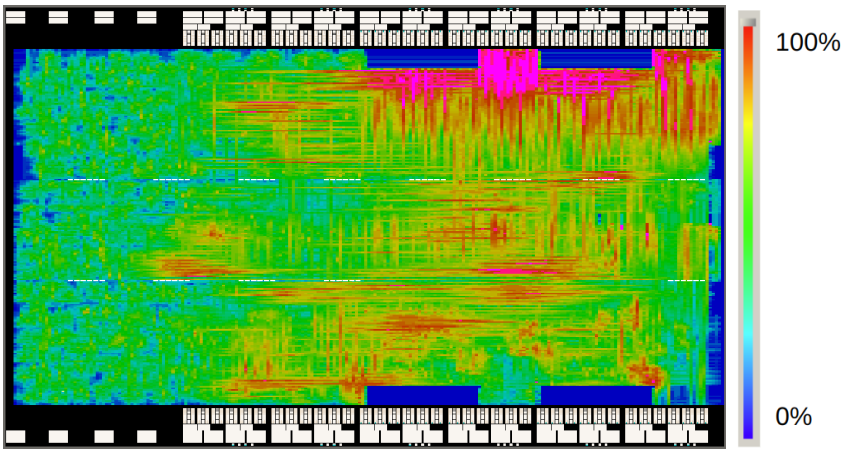

Figure 5.8: Quartus® Chip Planner view of routing resource utilization in different parts of the FPGA.

5.3.6 Timing optimization

During the firmware integration process, several changes were applied to the firmware to mitigate the impact of timing violations (see Appendix B). The changes mainly concerned the move from combinatorial unbuffered logic to buffered pipelined logic to ease the compiler job of placing the firmware components while being compliant with clock constraints. These changes had an effect on the overall clustering throughput; however, they led to a more stable system. Figure 5.9 shows the average event throughput that the clustering can sustain as a function of the VELO module. Even with the timing optimization changes, the throughput is always higher than the 30 MHz limit. The throughput was measured in QuestaSim® by feeding the clustering with Run 3 simulated data. As it can be observed, the lowest throughput value is around module 15, located near the nominal interaction point. In this region, modules have the highest occupancy, and also the fraction of non–isolated SPs is higher with respect to the external modules. The bottleneck in this configuration is represented by the matrices used for non–isolated SP cluster reconstruction.

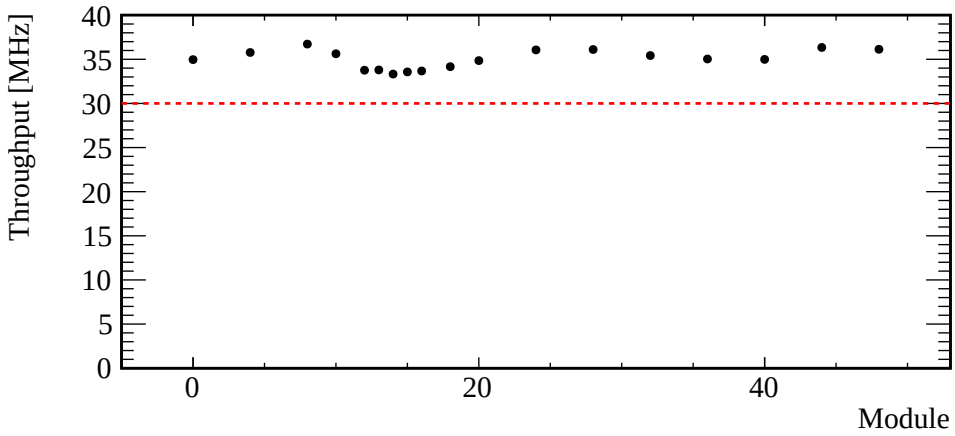

Figure 5.9: Average event throughput that the clustering can sustain as a function of the VELO module at the nominal LHCb Run 3 luminosity of 2×10^{33} cm^{-2}s^{-1}. The 30 MHz lower limit, below which the throughput must not fall, is highlighted with a red dashed line.

5.4 Software integration

This section describes the work done on the software side, developing algorithms to emulate the FPGA clustering and decode its output within the LHCb software stack [72, 73]. This work was needed to study the performance of the FPGA clustering and compare them with a more standard CPU algorithm, as detailed in Sect. 6. The main steps to adapt the VELO decoding sequence taking into account the cluster format are also detailed. First, an overview of the LHCb software stack is given.

5.4.1 LHCb software stack

Figure 5.10 shows the main steps within the LHCb data flow. In the case of real data taking, built events (see Sect. 2.9) are sent to the two–stage trigger (see Sect. 2.10). Selected events then go through an offline data processing called sprucing, which applies further selections to data

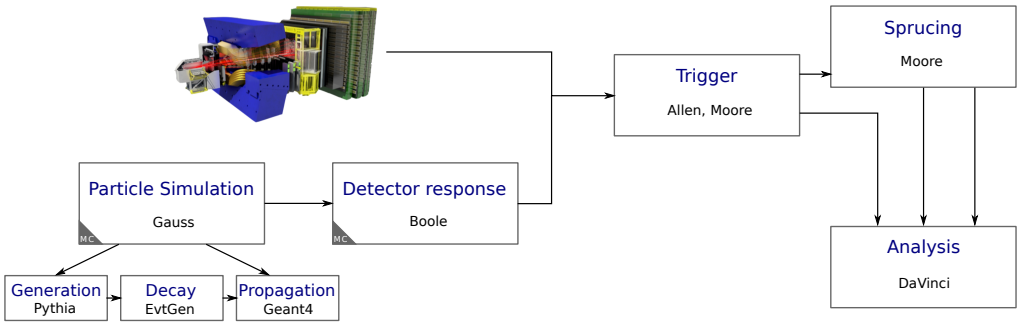

Figure 5.10: Sketch of the LHCb data flow, including the simulation part [59].

while being able to tune the amount of event information to be persisted in the final output. In case of simulated data, *pp* interactions are modeled, and the resulting particles are propagated inside the detector volumes with which they interact. The detector and readout electronics response is also modeled and implemented in the simulation. Simulated data are then passed to the same trigger–sprucing–offline analysis chain as the real data. The data flow and the corresponding operations are managed by several software packages, all built on the `Gaudi` core software framework [80]. The `Gauss` package handles the particle production and propagation within the detector, `Boole` manages detector digitization and signal handling, `Allen` performs the HLT1 trigger operations, whereas `Moore` runs HLT2 together with Sprucing operations. `DaVinci` is used for offline analysis.

5.4.2 Simulation side

In order to produce VELO clusters, accurately emulating the FPGA firmware behavior, and perform quality checks and tests (see Sect. 6), we added the following algorithms to the LHCb software stack:

- `VPRetinaMatrix` emulates the behavior of the firmware matrix (see Sect. 4.4.8), reconstructing clusters from non–isolated SPs. It defines the matrix geometry, together with functions to initialize an empty matrix, to check if a SP belongs to an already initialized matrix, and to eventually add it to the matrix. `VPRetinaMatrix` also implements the cluster search mechanism, using the patterns described in Sect. 3.2.3, both for light and full VELO clusters[7].

- `VPRetinaClusterCreator` flags SPs according to their isolation and creates VELO light clusters from SPs using the isolation flag information. If a SP is isolated, the corresponding cluster(s) is(are) reconstructed using a LUT initialized with the same content as the one implemented in firmware (see Sect. 4.4.7). If the SP is not isolated, it is sent to a chain of matrices, implemented as a vector of `VPRetinaMatrix` objects. The `chain_length` property of the algorithm allows the number of matrices per sensor to be set. By default, this is set to the actual number used in the firmware (20 matrices).

[7] A light VELO cluster is an object containing the position of the cluster center of mass in the LHCb coordinate system and identified with a unique LHCbID. A full VELO cluster is an object with all the characteristics of a light one, but it also contains the list of all the pixels from which the cluster is made up. Light clusters are used to perform fast–track reconstruction, whereas full clusters are used for finer reconstruction, including the error on the position of the particle obtained from the cluster topology and dimensions. Full clusters are also used for MC matching during simulation studies.

- `VPRetinaFullClustering` performs the same operations as of `VPRetinaClusterCreator` but outputting full clusters instead of light ones. It has the same `chain_length` property as `VPRetinaClusterCreator`.
- `VPClusterEfficiency` is used to perform cluster reconstruction quality studies, including efficiency and residual checks (see Sect. 6).

`VPRetinaClusterCreator` and `VPRetinaFullClustering` can be run as part of the reconstruction sequence, in order to reconstruct clusters from simulation files that contain only SPs. Alternatively the `VPRetinaClusterCreator` algoritm can be run directly in `Boole`, so that VELO clusters are added to the produced simulation files. The following options are available in `Boole` configuration file:

- `RetinaCluster` option allows files to be produced with VELO clusters instead of SPs;
- `PreserveSP` option preserves SPs, preventing the previous option from removing them.

An additional script was made available in order to add VELO clusters to already available simulation files, produced before the introduction of the FPGA VELO clustering algorithm. Changes were also applied to the `VPSuperPixelBankEncoder` algorithm that emulates the VELO TELL40 behavior if a SP–only firmware is loaded. The output format of the `VPSuperPixelBankEncoder` algorithm was updated to reflect the actual hardware behavior.

5.4.3 Firmware–software differences

Great care has been taken to ensure a perfect bit–by–bit correspondence between the output produced by the firmware and by its software simulation when fed exactly with the same sequence of SPs. This is key for both the simchecker checks (see Sect. 5.3.4) and physics performance checks (see Sect. 6). However, the algorithm has some sensitivity to the exact order in which the SPs are fed into the clustering process, and this order is unpredictable during the actual data taking from a real detector. This amounts to an intrinsic limitation in the accuracy of the simulation of the firmware output. This discrepancy is relevant only for a small number of large clusters consisting of several SPs, since SPs that are part of the cluster may be split over matrices in different ways depending on their arrival order, as shown in Fig. 5.11. In events

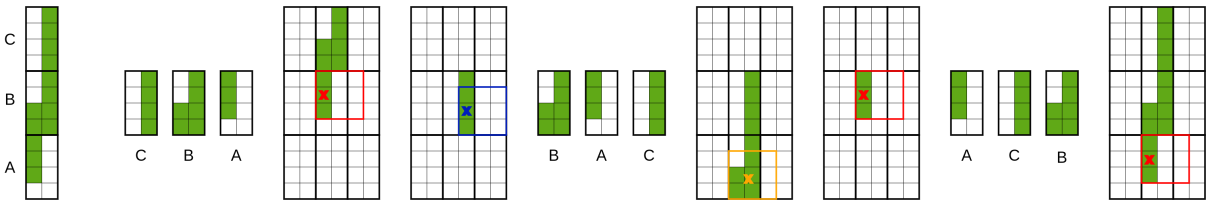

Figure 5.11: Example of different ways in which a large cluster can be split depending on the SP input order to the matrix. Active pixels are marked in green, reconstructed clusters are represented by red, blue and yellow boxes. Each color corresponds to different cluster.

with many SPs with neighbors, the number of matrices implemented in the firmware might not be enough to accommodate all the SPs. SPs that overflow the matrices are reconstructed as isolated, both in the firmware and software implementations of the algorithm. Again, the SP input order determines which SPs overflow the matrices.

These small differences are not relevant for physics performance checks, where a large number of events are analyzed. However, in the simchecker case all possible outcomes of the clustering algorithm as a function of the input SP order are considered to avoid raising false errors or not identifying true ones.

5.4.4 Decoding side

Similarly to what has been done on the simulation side (see Sect. 5.4.2), new decoding algorithms have also been added or existing ones have been changed to accept VELO clusters instead of SPs. The main algorithms are the following:

- `CalculateNumberOfRetinaClustersPerSensor` is the HLT1 algorithm that counts the number of clusters contained in each data bank. Its purpose is to determine the amount of GPU memory required for VELO decoding, before performing the decoding itself;
- `DecodeRetinaClusters` performs the HLT1 VELO cluster decoding, moving from a set of 32–bit cluster words to a container of clusters, the position of which is specified within the LHCb global coordinate system. The decoding algorithm has been written to perform efficient accesses to GPU memory, while discarding empty clusters added during encoding firmware operations (see Sect. 4.4.10);
- `DecodeRetinaClustersContracts` runs some checks on the decoded clusters output by the `DecodeRetinaClusters` algorithm. The implemented contracts involve checks on the coordinates to be within the VELO acceptance. These contracts are usually run during the decoding development stage and are not run during data taking to avoid adding extra work to the GPUs;
- `VPRetinaClusterDecoder` is the HLT2 algorithm that decodes the VELO data bank and outputs a set of light clusters;
- `VPRetinaFullClusterDecoder` is the HLT2 algorithm that decodes the VELO data bank and outputs a set of full clusters;
- `VeloClusterTrackingSIMD` is the baseline HLT2 VELO tracking algorithm. A template has been added to decode VELO clusters and perform tracking from them instead of decoding SPs and reconstructing clusters, before moving to tracking;
- `VPRetinaTopologyID` defines the function to compute the topology identifier during the cluster reconstruction process when clustering is performed in simulation and the function to decode the topology identifier from the cluster word when clustering is run on the FPGA. For more details on the topology ID, see Sect. 4.2.1;
- `VPRetinaClusterDoubleOutputChecker` decodes VELO data collected in double–output mode (see Sect. 4.4.13). It reconstructs clusters from SPs using the same working principle as the `VPRetinaClusterCreator` algorithm and compares them with the clusters reconstructed in firmware.

5.5 Throughput and bandwidth gains

One of the key advantages of performing two–dimensional VELO cluster reconstruction at early data acquisition stages, within detector readout cards, is the significant load reduction on GPUs and CPUs devoted to HLT processing. As a consequence, GPUs and CPUs, characterized by more complex architectures than FPGAs, can be exploited in an optimal way, focusing their efforts on more articulated tasks such as track reconstruction and selections. Having developed the full set of VELO cluster decoding algorithms, it was possible to measure the performance gains obtained by offloading cluster reconstruction to readout FPGAs instead of performing this operation within HLT1. The throughput gain was measured both for the GPU and CPU implementations of HLT1. In the GPU case, a 11% throughput gain was measured both on Nvidia RTX 2080 Ti gaming cards and on Nvidia A5000 cards, which are the actual GPU cards mounted on Event Builder servers to perform HLT1 reconstruction and trigger decisions. The same throughput gain was measured to be about 8% in the CPU implementation of HLT1. The difference in throughput gain between GPU and CPU implementations is due to the different architecture and degree of parallelization, together with the higher amount of optimizations performed in the GPU case. Moreover, both CPU and GPU implementations expect to receive SPs with the isolation bit already available, whereas the FPGA clustering determines it by itself in the firmware. Lastly, the CPU implementation expects SPs to be spatially sorted per row and per sensor, which is not the case for the TELL40 output, where in each VELO data bank, SPs of two sensors are mixed. Given these assumptions made by the GPU and CPU implementations, the real throughput gains are likely higher than the ones reported here.

As an additional advantage, outputting clusters instead of SPs allows for a reduction in VELO data size of approximately 14%. Given that both SPs and clusters are encoded into 32–bit words, the bandwidth reduction can be measured by counting the number of clusters produced from a given number of SPs. A first estimate can be made considering that about 50% of SPs are isolated and 50% have neighbors. With the exception of isolated SPs that contain two clusters, which is a less common case, from an isolated SP, one cluster is reconstructed. Therefore, reconstructing clusters from isolated SPs does not lead to a bandwidth reduction. Instead, out of the 50% of non–isolated SPs, the vast majority creates two–SP clusters, where the cluster is spread over two SPs. In this case, out of two SPs, a single cluster is reconstructed, leading to a bandwidth reduction. With the previous assumptions, the maximum theoretical bandwidth reduction is about 25%. However, the trade–offs introduced in the encoder design, for which empty clusters are interleaved between other clusters, together with the precise measurement of the fraction of isolated–vs–non–isolated SPs and the correct estimate of the number of non–isolated SPs per cluster, lead to a measured bandwidth reduction of about 14%. This additional bandwidth reduction allows resources to be saved both in the DAQ chain and in the data storage, sending a preprocessed information out of readout cards to the Event Builder and to HLT1 processes.

5.6 More on clustering algorithms

As the LHCb HLT design evolved from a CPU–only implementation to a GPU–CPU mixed architecture, two main implementations of the VELO clustering were developed. On one hand the sparseCCL algorithm [74] was developed for the CPU architecture and optimized for analyzing many small and sparse images at high throughput, as in the case of HEP vertex detectors. When tested on an Intel® Xeon Gold 6126 @2.6GHz, this algorithm is capable of handling 173 MSpixels/s. On the other hand, the mask clustering algorithm [75] was introduced as the

HLT1 architecture was moved from CPUs to GPUs, exploiting the high degree of parallelization of GPUs, while keeping memory consumption low. When tested on a Nvidia Quadro RTX 6000, the mask clustering algorithm proved capable of processing 50.6 MSpixels/s. Instead, as detailed in Sect. 5.2, the FPGA–based clustering algorithm can process up to 32 SPs at a 30 MHz input event rate, leading to a 960 MSpixel/s processing speed. It is also worth noting that the FPGA implementation requires only a fraction of the electrical power required by GPUs to perform the same task, as discussed in Sect. 7.8.

While comparisons against sparceCCL and mask clustering algorithms take into account implementations tackling the same problem but running on different architectures, it is also worth comparing the FPGA VELO clustering with similar clustering algorithms running on FPGA architectures. The most advanced FPGA–based clustering algorithm [81] that can be found in literature tackling a similar problem to the FPGA VELO clustering has been developed within the Fast TracKer project [35], with an eye to the trigger upgrade of the ATLAS experiment. This system can run at about 100 KHz input event rate, with the deployment of about 4 parallel firmware copies, processing about 12.5 MPixel/s each. It runs on a Xilinx Spartan–6 LX150 T FPGA, occupying 5739 FlipFlops, 10583 LUTs, 15 8kb–BRAMs, 21 9kb–BRAMs to reconstruct 144×328 pixels. For comparison, VELO FPGA clustering requires 132500 ALMs and 300 M20K memories and it is capable of processing an input event rate of 38.9 MHz, while reconstructing clusters from 512×768 pixels. In both cases, the detector occupancy is expected to be on the order of 0.1%. Therefore, the algorithm presented in this thesis is a significant advancement over the previous state–of–the–art in HEP.

In a broader perspective, the VELO FPGA clustering algorithm can be seen as a special case of Connected Component Labeling with Center of Gravity calculation (COG), a computation that often occurs in image processing systems with the purpose of identifying connected sets of pixels belonging to the same visual feature. The main difference being the modest size of the features of our interest that we could contain within a 3×3 matrix, and their sparseness, which makes our problem somewhat simpler. However, this greater simplicity comes with a 'frame rate' requirement (30 MHz) that is orders of magnitude larger than typical image processing rates (<1 kHz). In recent years, this type of image processing tasks are also increasingly being moved from CPUs to dedicated FPGA firmware to achieve greater speed and efficiency, and it may be interesting to compare those solutions to the present work. As an example, we take the FPGA implementation described in Ref. [82]. There, frames of 640×480 pixels are processed at a 730 Hz rate, by a Zynq AP–SOC 7045 FPGA, running a 225 MHz clock, without COG. This system compares well with our case, where each of the 104 instances of our firmware processes a matrix of 512×768 pixels, and clock frequency and resource usage are also quite similar. However, our frame rate is larger by a factor of nearly 10^5. This difference is likely due to the sequential structure of image processing firmwares that proceed by a raster scan rather than by a massively parallel calculation; but is definitely also a consequence of the greater simplicity of our problem in terms of cluster size and occupancy. In fact, cases of FPGA–based CCL implementations that reach a throughput comparable to that of our architecture are based on splitting the image into smaller parts that are analyzed in parallel and later coalesced [83]; an approach that bears some resemblance to our use of sparse matrices. However, all the above examples assume that image data arrive as an ordered sequence of pixels and do not provide detailed topology analysis of the found clusters, so they could not be straightforwardly applied to our problem. Conversely, the smallness of the components addressed by our system may not be of interest in general image processing application; nevertheless, it cannot be excluded that some of the ideas described in this article could find some use in image processing tasks, at least in some specific instances.

Chapter 6

Physics performances

This chapter details the studies performed to ensure that the quality of the VELO FPGA cluster reconstruction is sufficiently performing to meet the very demanding physics requirements of the LHCb track reconstruction. Comparisons with the CPU implementation of the clustering algorithm are performed both at the low level, measuring the cluster efficiency, and at high level, studying tracking resolutions, as an output of the full reconstruction sequence.

6.1 Motivation

As discussed in Chap. 3, the FPGA implementation of the clustering algorithm performs some approximations to be able to reconstruct clusters in a highly parallel way, in excess of 30 MHz. A key step before deploying the firmware to the DAQ system is to ensure that the algorithm, as it has been designed, does not degrade the track reconstruction efficiency nor the quality of the reconstruction. This is particularly important because the FPGA cluster–finding architecture was designed with the intent of replacing the raw pixel data with reconstructed hit coordinates at the detector readout level. As a consequence, the raw pixel data are discarded and cannot be recovered at any later stages. For this purpose, extensive studies are performed, comparing the FPGA implementation of the VELO clustering to the CPU one, in order to spot possible issues and to tune the algorithm parameters as needed. For the sake of generality, comparisons are made with a CPU–based clustering algorithm that is free from implementation–specific constraints. The actual HLT1 implementation at LHCb is GPU–based, but its performance is indistinguishable from the CPU version we take as reference [42, 84]. Details about the CPU– and GPU–based VELO clustering algorithms are discussed in Sect. 2.11. Performance studies are first done by comparing cluster reconstruction at low level, both in terms of efficiency and residuals. The track reconstruction efficiency is also studied together with the momentum, impact parameter, and primary vertex resolutions.

6.2 Clustering performance

The key differences between the FPGA and CPU algorithms that can potentially affect reconstruction performance are the cluster finding mechanism, based on pattern matching, the maximum cluster size in the FPGA algorithm (limited to a 3×3 pixel grid), and the constraints due to the FPGA sparse–matrix filling scheme. Those differences can potentially lead to inefficiencies,

cluster splitting, or incomplete reconstruction of some clusters. An example of partial cluster reconstruction is illustrated in Fig. 6.1a, where the red pixel is left out of the reconstructed cluster. The shift of the reconstructed hit position may lead to a degradation of the precision on the reconstruction of the particle trajectory, or even to a loss of efficiency if the associated track is not reconstructed at all. Figure 6.1b shows an example of cluster splitting, where the algorithm finds two clusters, with a pixel in common, from six contiguous active pixels. Therefore, it is

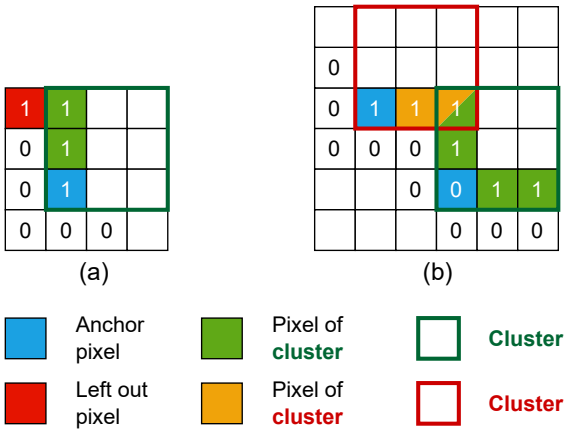

Figure 6.1: Example of corner cases of the FPGA clustering algorithm: (a) partial cluster reconstruction and (b) cluster splitting.

very important to compare FPGA and CPU clustering algorithms at a low level, considering their capabilities of reconstructing the true position of a Monte Carlo simulated charged particle hitting a layer of the VELO detector (MC hit). If the simulated particle releases enough energy to light up one or more pixels, these pixels are associated with the hit itself. Since clustering algorithms accept the digital response of the detector as an input, only MC hits that have at least one pixel associated with them are considered. This is done to factor out possible detector inefficiencies and resolution degradation in hit position measurement.

In order to study the physics performance, the software simulation of the FPGA–based clustering algorithm described in Sect. 5.4.2 is used. The CPU–FPGA comparisons presented in this section are performed on a minimum bias[1] sample of about 50k pp bunch crossings[2], at instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}, corresponding to a total of 10.4×10^7 SPs and 7.1×10^7 clusters, generated at the foreseen LHCb–Upgrade running conditions with a center of mass energy of 14 TeV. Figure 6.2 shows the number of SPs per event as a function of the VELO module number, considering both entire modules and half–modules[3]. The corresponding distributions for clusters are shown in Sect. 7.12.

[1] Minimum bias (MB) events are inelastic events selected with a loose (minimum bias) trigger configuration, with as little bias as possible.
[2] This corresponds to an average number of pp interactions per bunch crossing $\nu = 7.6$.
[3] As discussed in Sect. 5.3.1 a VELO module is read by a single TELL40, containing two clustering instances, each processing SPs from one half–module.

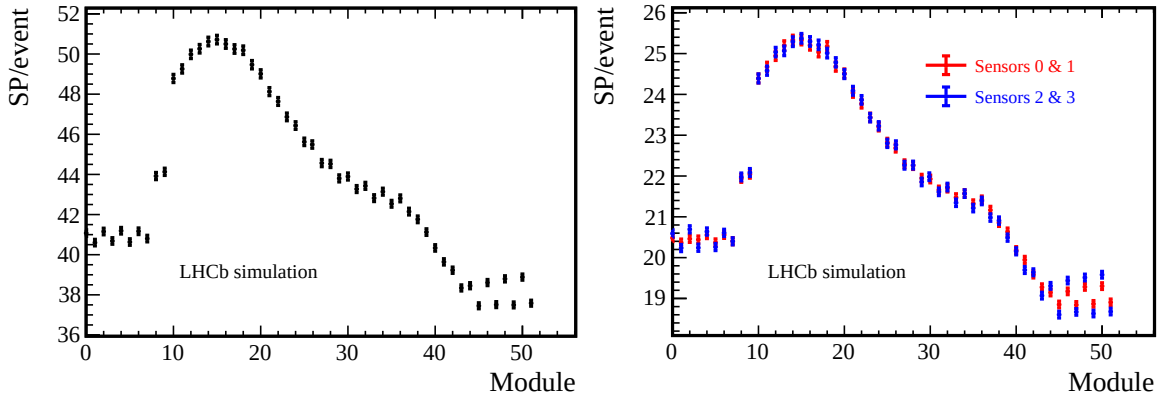

Figure 6.2: Distributions of the number of VELO SPs as a function of the module (left) grouping SP per module and (right) grouping SPs per half-modules.

6.2.1 Cluster efficiency

The efficiency (ϵ) in reconstructing VELO clusters is defined as the ratio between the number of MC hits matched to a cluster ($N_{\text{linked MC hits}}$) and the total number of MC hits ($N_{\text{MC hits}}$),

$$\epsilon \equiv \frac{N_{\text{linked MC hits}}}{N_{\text{MC hits}}}, \tag{6.1}$$

where an MC hit is matched to a cluster if they share at least one pixel. Table 6.1 shows a summary of the samples used and the overall efficiencies. The efficiency in reconstructing VELO clusters of the FPGA–based algorithm is approximately 99.8% and almost indistinguishable from that of the CPU algorithm. Overall, the FPGA clustering algorithm produces 2.97%

	MagDown	*MagUp*
Events	38477	49157
CPU clusters	54231010	69191020
FPGA clusters	55835697	71251193
CPU clustering efficiency	100.00%	100.00%
FPGA clustering efficiency	99.82%	99.82%
CPU clustering efficiency – VELO reconstructible	100.00%	100.00%
FPGA clustering efficiency – VELO reconstructible	99.92%	99.92%

Table 6.1: Summary of the MC samples used and of the overall clustering efficiencies.

more clusters than the CPU one, whereas the clustering efficiency difference between the CPU and FPGA algorithms is 0.18% and 0.08% when considering all clusters and clusters from reconstructible VELO tracks only, respectively[4]. An MC track is defined as a reconstructible VELO track if it has at least 3 VELO sensors with at least 1 MC hit each, as defined in Ref. [85].

Figure 6.3 shows the clustering efficiency, for both the FPGA and CPU clustering algorithms, as a function of the radius and module number of the hit and of η, ϕ, p, and p_{T} of the corresponding

[4]Numbers is Table 6.1 are obtained with both polarities of the magnet, magnet down (MD) and magnet up (MU). Plots in the following are obtained using the MU sample as the results are the same for the two polarities.

track. In each figure, red and blue curves represent the FPGA clustering efficiencies considering all clusters (including those from non–reconstructible VELO tracks) and selecting only clusters from VELO reconstructible tracks, respectively. The black curve represents the corresponding CPU efficiency, considering all types of cluster. The 68% confidence intervals for the efficiencies are calculated using the Wilsonian approach [86]. In order to identify possible efficiency dependencies

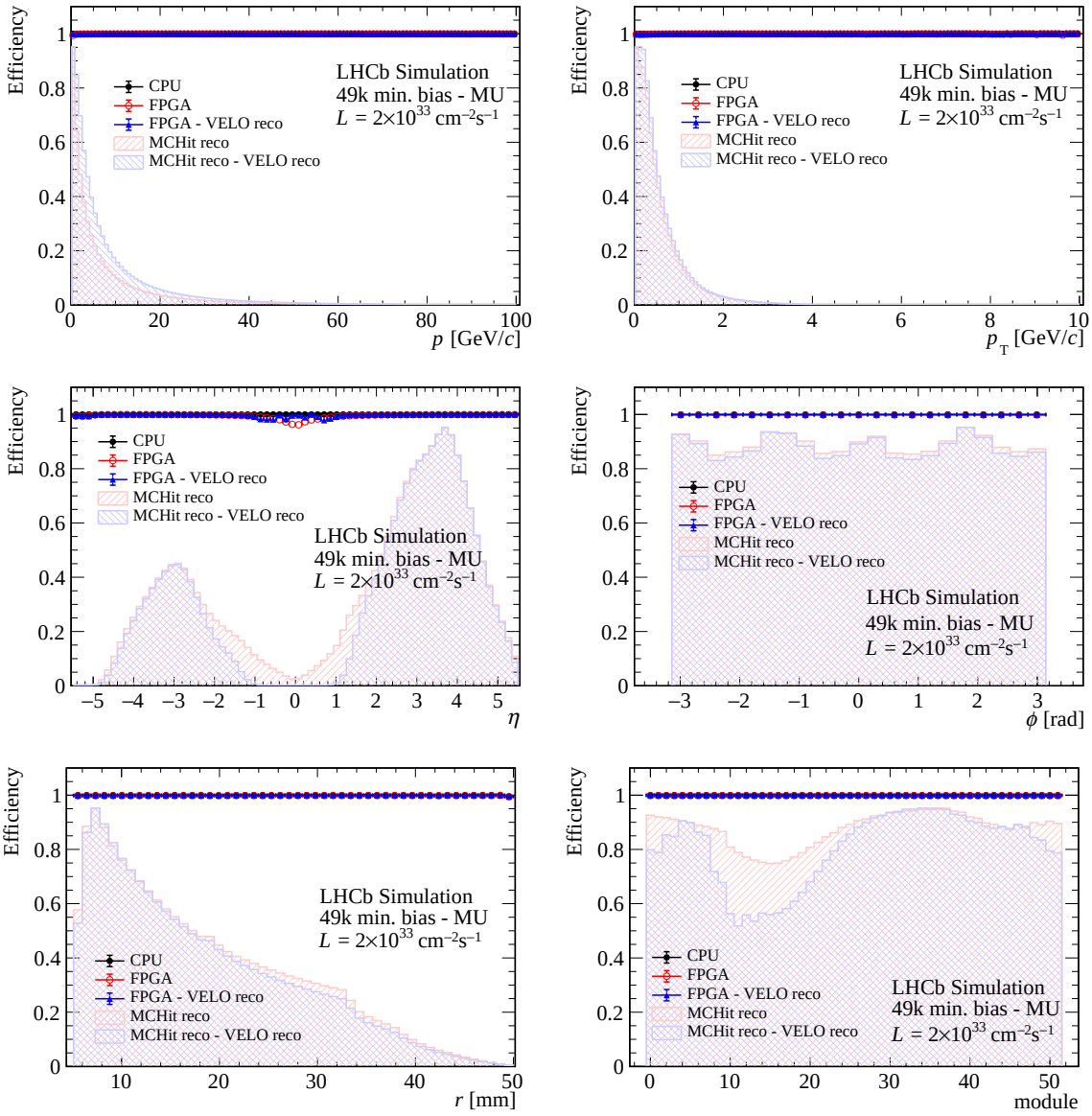

Figure 6.3: Comparison of the clustering efficiency of all clusters and clusters from VELO reconstructible tracks when using the FPGA and CPU clustering algorithms, as a function of various kinematic variables.

on the same kinematic variables, the clustering inefficiency is plotted as a function of the same variables in Fig. 6.4. It turns out that the FPGA cluster finder is slightly less efficient than the CPU algorithm, when all types of clusters are considered inclusively (the inefficiency is approximately 0.18% on average). Due to the intrinsic features of the FPGA algorithm, this inefficiency is due to large clusters linked to more than one MC hit. However, when selecting

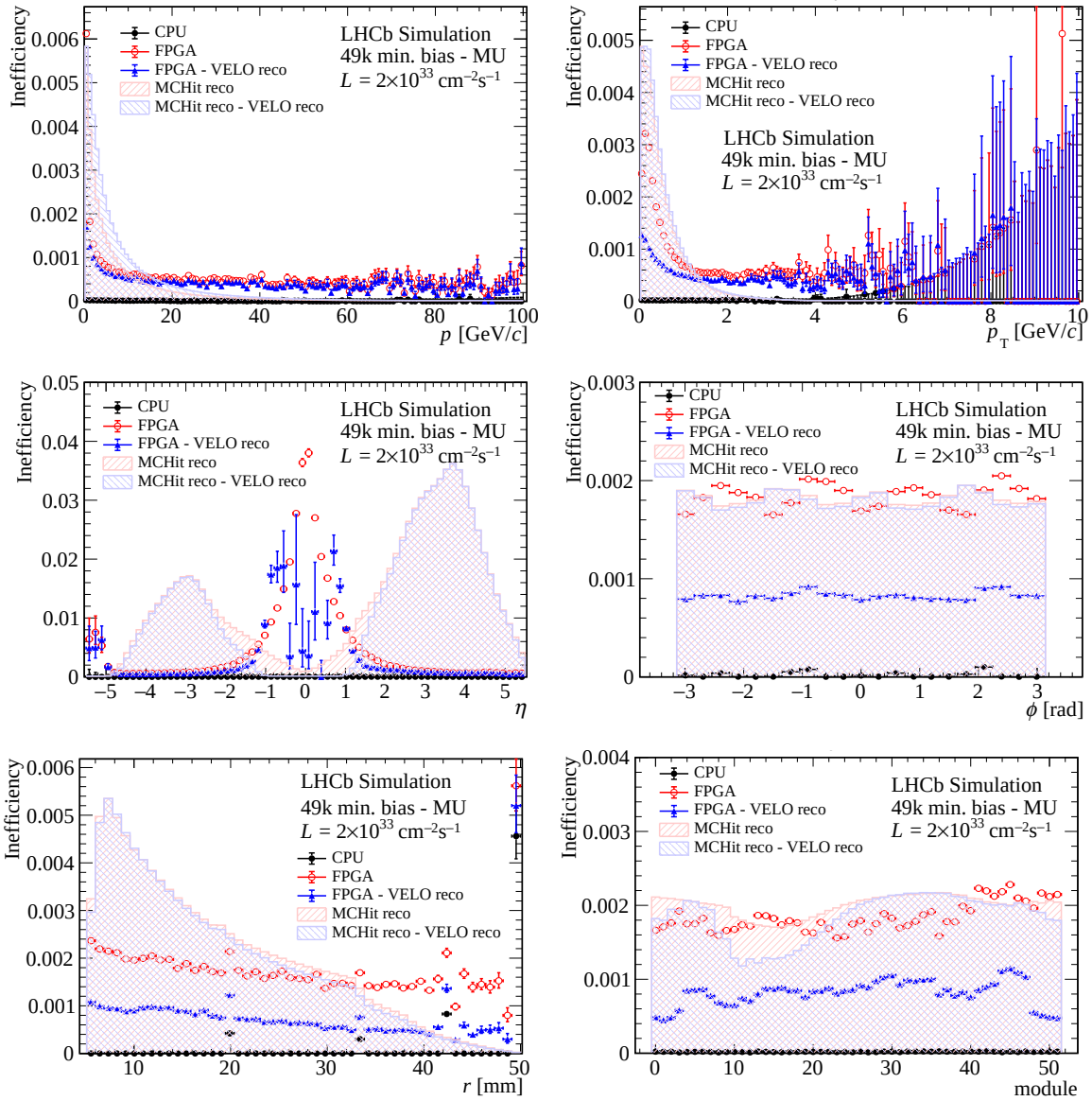

Figure 6.4: Comparison of the clustering inefficiency as a function of momentum, transverse momentum, pseudorapidity, ϕ, radius and VELO module, (red) for FPGA considering all clusters, (blue) for FPGA selecting only clusters from VELO reconstructible tracks and (black) for CPU considering all clusters. The red and blue histograms show the momentum, transverse momentum, pseudorapidity, ϕ, radius and VELO module distribution of MC hits from all types of tracks and selecting only VELO reconstructible tracks, respectively.

only MC hits from VELO reconstructible tracks, the overall inefficiency decreases to 0.08%. Although not exactly uniform, this inefficiency does not show any "hot spots" in particular places, varying around 0.2% over most of the space; the largest effects are observed at low momenta, where the maximum effect reaches 0.6%. An exception to that is a peak of large inefficiency at pseudorapidity close to zero, where, however, the distribution of MC tracks has very few entries. These tracks graze VELO sensors at a very low angle and likely produce very spread–out clusters. For this reason, they are unlikely to be accurately measured regardless of the clustering algorithm. Anyhow, all inefficiency effects decrease dramatically for VELO tracks, being nearly uniformly distributed, where the average inefficiency is approximately 0.08%. Given that physics decay candidates are built out of reconstructible tracks and that they can typically count on many measurement layers, we should expect negligible effects on physics efficiencies and resolutions. However, this will be explicitly studied in the following sections. In particular, it is important to double check for possible effects on the vertex reconstruction, where VELO–only tracks play a role (Sect. 6.3.4). The largest effects observed on VELO–only tracks, especially at low pseudorapidity, are consistent with the fact that they correspond to a higher proportion of "large" clusters, linked to more than one MC hit, due to the wider range of dip angles in crossing VELO sensors. FPGA clustering has been optimized for clusters containing at most nine pixels, and large clusters lead to lower efficiency for the FPGA algorithm relative to the CPU one.

6.2.2 Cluster residuals

In addition to cluster efficiencies, cluster residual distributions are key to determine the quality of input data to the track fit. The residual is defined as the distance between the position of the reconstructed cluster and the true coordinates of the hit generated by the passage of the particle in the corresponding detector layer. A comparison between residual distributions of reconstructed clusters, between the CPU and FPGA algorithms, is shown in Fig. 6.5.

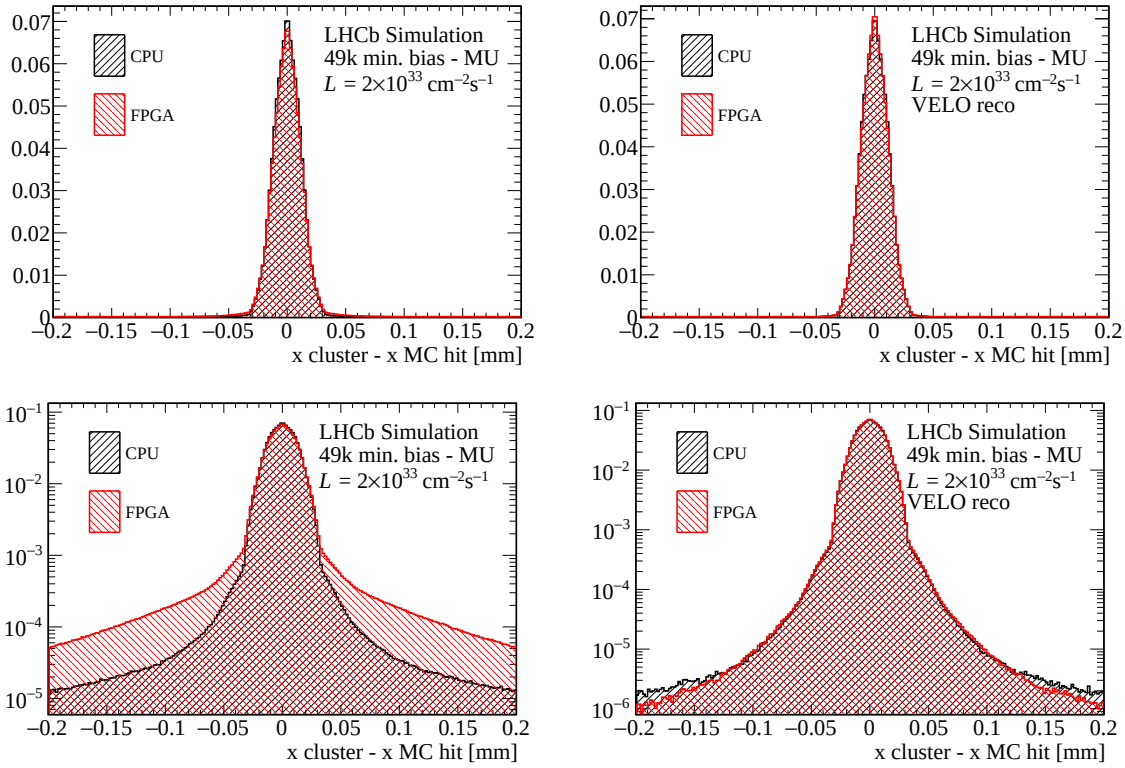

Figure 6.5: Comparison of the normalised distributions of the cluster residuals (left) for all clusters, including those produced by non–reconstructible tracks and (right) for clusters produced by VELO reconstructible tracks. Top (bottom) plots are displayed in linear (logarithmic) scale.

The overall difference between CPU and FPGA distributions is visible only in logarithmic scale. When considering all types of clusters, including those from non–reconstructible tracks, the FPGA distribution shows higher tails starting at two orders of magnitude below the peak value. These tails are due to split–up clusters. The difference becomes negligible when selecting clusters from VELO reconstructible tracks, where the FPGA distribution shows smaller tails due to clustering inefficiencies. In both cases no effect due to the FPGA rounding of the cluster center is observed[5]. Figure 6.6 shows residual distributions for special regions: long pixels and sensor edges. Pixels connecting VELO chips are designed to be 2.5 times longer (137.5 µm) than standard pixels (55 µm). Sensor edges are defined as a 3–pixel wide area around each sensor. In both cases, no peculiar behavior is spotted. Moreover, the efficiency of reconstructing clusters in these special regions is the same as the one measured over the entire VELO.

[5]As discussed in Sects. 3.4 and 4.2.1, the position of the cluster center is specified with 1/4 of a pixel position. This could have had a negative impact on the residual distribution. From the studies reported in this section and in the following, no effect is observed due to rounding.

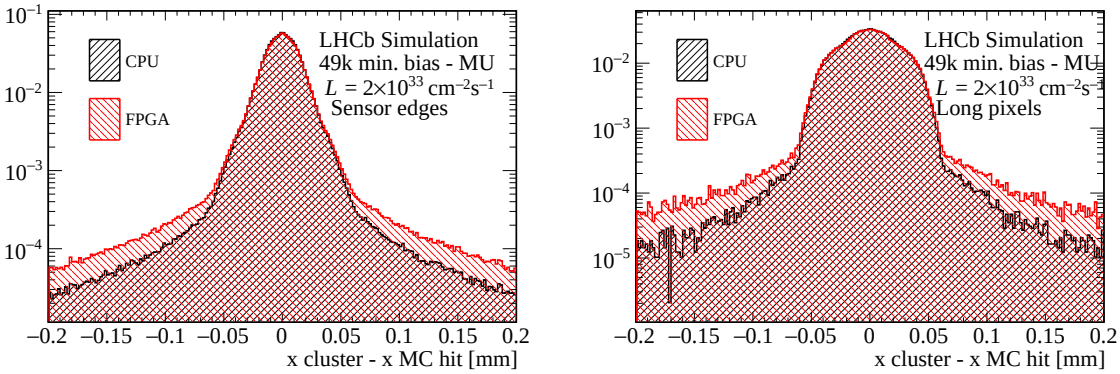

Figure 6.6: Comparison of the cluster residual distributions for (left) clusters produced by MC hits at the edge of VELO sensors and (right) clusters produced by MC hits on long pixels. Distributions are normalized to their respective integrals.

6.2.3 Cluster inefficiency sources

After several improvements and refinements of the FPGA clustering algorithm (see Sect. 3.4), the cluster inefficiency has been reduced below 0.2%. The residual inefficiency is due to MC hit merging, where more than one MC hit is linked to the same cluster. Figure 6.7 shows the number of MC hits linked to CPU clusters not reconstructed by the FPGA, as a function of the number of pixels in the CPU cluster itself. Since the two–dimensional distribution of CPU–reconstructed MC hits missed by the FPGA algorithm starts from two MC hits linked to the same cluster, it can be deduced that all the clusters not reconstructed by the FPGA originate from MC hit merging, where the number of MC hits linked to the CPU cluster is greater than one. Some hot spots are present when two MC hits are linked to the same CPU cluster, as shown in Fig. 6.7.

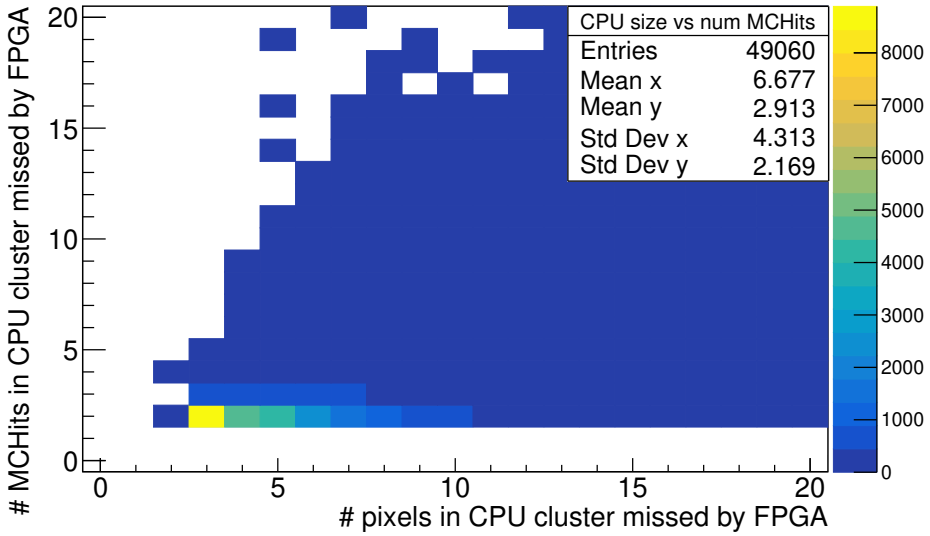

Figure 6.7: Number of MC hits linked to CPU clusters not reconstructed by the FPGA as a function of the number of pixels in the CPU cluster.

Figure 6.8 illustrates some examples of MC hits not identified by the FPGA algorithm because of hit merging. It is worth noting that, in these particular cases, the CPU algorithm reconstructs a single cluster, averaging the positions of the two hits.

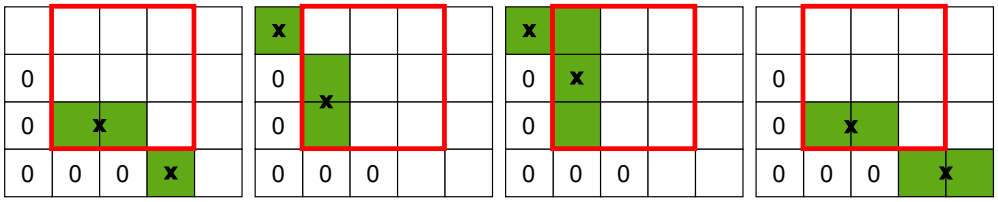

Figure 6.8: Examples of MC hits missed by the FPGA clustering due to MC hit merging. Red squares are the 3×3 cluster candidates.

Figure 6.9 shows the comparison between CPU cluster residual distributions of all clusters and clusters linked to MC hits not reconstructed by the FPGA. As it can be observed, the cluster reconstruction quality of clusters linked to MC hits missed by the FPGA algorithm is bad. Therefore, the loss of a small fraction of poor–quality reconstructed MC hits is not harmful.

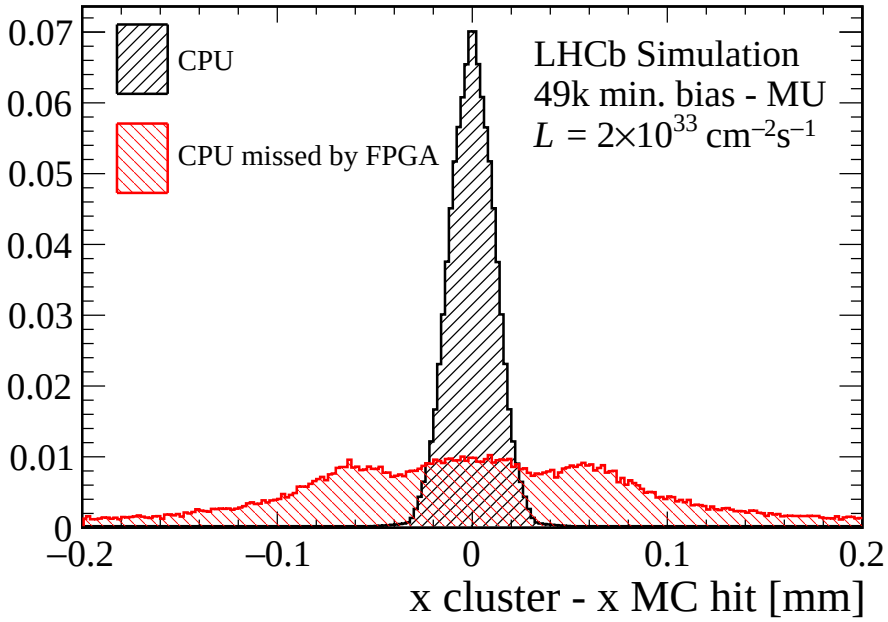

Figure 6.9: Cluster residual distributions comparing all CPU clusters with CPU clusters liked to MC hits not reconstructed by the FPGA.

6.3 Tracking performance

Extensive studies are also performed to measure the quality of the full track reconstruction, when FPGA VELO clusters are used. The trajectories of charged particles traversing the tracking system are reconstructed from hits in some of the three tracking detectors, which are the VELO and the Upstream Tracker (UT), located upstream of the magnet, and the Scintillating Fiber (SciFi) detector located downstream of the magnet [61]. Figure 6.10 shows a pictorial representation of the track type definitions within LHCb. The tracks reconstructed using only

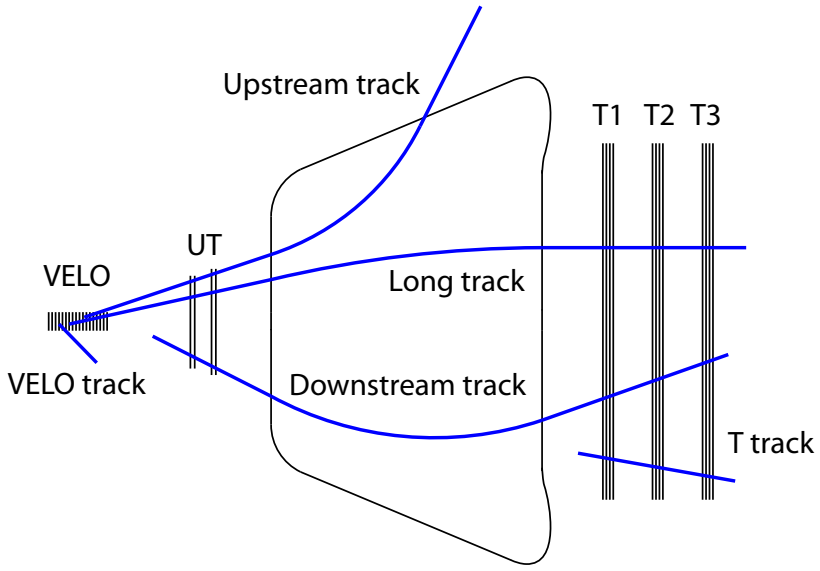

Figure 6.10: Pictorial view of the track types in the LHCb tracking system.

hits from the VELO detector are called VELO tracks[6]. VELO tracks with $2 < \eta < 5$ can also have hits in the SciFi detector and optionally in the UT. These tracks are called "long tracks". As they traverse the whole magnetic field of the LHCb detector, they have the most precise measurement of the momentum and therefore are key for physics analyses.

6.3.1 Tracking efficiency

Tracking reconstruction efficiency is defined in Eq. 6.2 as the ratio between the number of reconstructed tracks matched to an MC particle and the number of reconstructible MC particles,

$$\epsilon \equiv \frac{N_{\text{MC-matched}}}{N_{\text{MC-reconstructible}}}. \tag{6.2}$$

An MC particle is called VELO reconstructible if it has at least one hit on three different VELO sensors. An MC particle is called long reconstructible if it is VELO reconstructible and if it has

[6]VELO tracks can also have $\eta < 2$, in which case they are used only for the primary vertex reconstruction.

at least one x and one stereo cluster in each of the SciFi tracker stations. The association of a reconstructed track with an MC particle is defined in terms of the number of shared hits. A reconstructed VELO track is matched to an MC particle if at least 70% of its hits are matched with the hits of the MC particle. A reconstructed long track is matched to an MC particle if at least 70% of its hits are matched with the hits of the MC particle in each VELO, SciFi, and UT subdetectors[7]. Matching and reconstructibility conditions of the other track types shown in Fig. 6.10 are extensively discussed in Ref. [61].

The comparison between the CPU– and FPGA–based reconstruction performances for VELO tracks and for VELO segments of long tracks is reported in Tab. 6.2. It also reports the relative fraction of reconstructed clone tracks with respect to the total number of tracks in the category to which they belong and the relative fraction of ghost–reconstructed tracks with respect to the total number of tracks. A clone is defined as any additional reconstructed track matching an already truth–matched MC track, whereas a ghost is a reconstructed track not associated with any true MC track [85]. The efficiencies and clone fractions are almost indistinguishable

Track type	Quantity	CPU clusters [%]	FPGA clusters [%]
All VELO tracks	efficiency	98.254 ± 0.007	98.254 ± 0.007
	clone	1.231 ± 0.006	1.234 ± 0.006
Long tracks	efficiency	99.252 ± 0.006	99.252 ± 0.006
	clone	0.806 ± 0.006	0.806 ± 0.006
	ghost	0.848 ± 0.003	0.928 ± 0.003

Table 6.2: VELO tracking efficiency, relative fraction of clone and ghost tracks, comparing CPU and FPGA clusters.

when comparing CPU and FPGA algorithms for VELO and long tracks, not displaying any perceptible systematic difference. The fractions of ghost tracks differ at the per–mille level. This difference is due to tracks in the pseudorapidity region below 1.5. These tracks graze VELO sensors at a very low angle and produce very large clusters, increasing the possibility of cluster splitting. For this reason, it is unlikely that the position of the particle hitting the detector and creating the cluster is accurately measured, regardless of the clustering algorithm.

In order to identify possible "hot spots" of inefficiency, the tracking efficiency is checked as a function of p, p_{T}, ϕ, η of the track and as a function of the number of primary vertices in the event. This is done separately for the following track categories:

1. all HLT1 VELO tracks (Fig. 6.11);
2. all long tracks for HLT1 (Fig. 6.12) and for HLT2 (Fig. 6.15);
3. long tracks originating from b–hadron decays, with momentum greater than 3 GeV/c and transverse momentum greater than 500 MeV/c for HLT1 (Fig. 6.13) and for HLT2 (Fig. 6.16);
4. long tracks from electrons for HLT1 (Fig. 6.14) and HLT2 (Fig. 6.17).

[7]The condition on the UT hits applies only if UT hits are reconstructed for the track.

In addition, Fig. 6.12 and 6.15 include a plot of efficiency as a function of DOCA_z[8], for the special category of long tracks corresponding to particles produced in the decay of s–hadrons (for all other tracks the distribution of DOCA_z is concentrated near zero, with larger values dominated by secondary particles from interactions with the detector material).

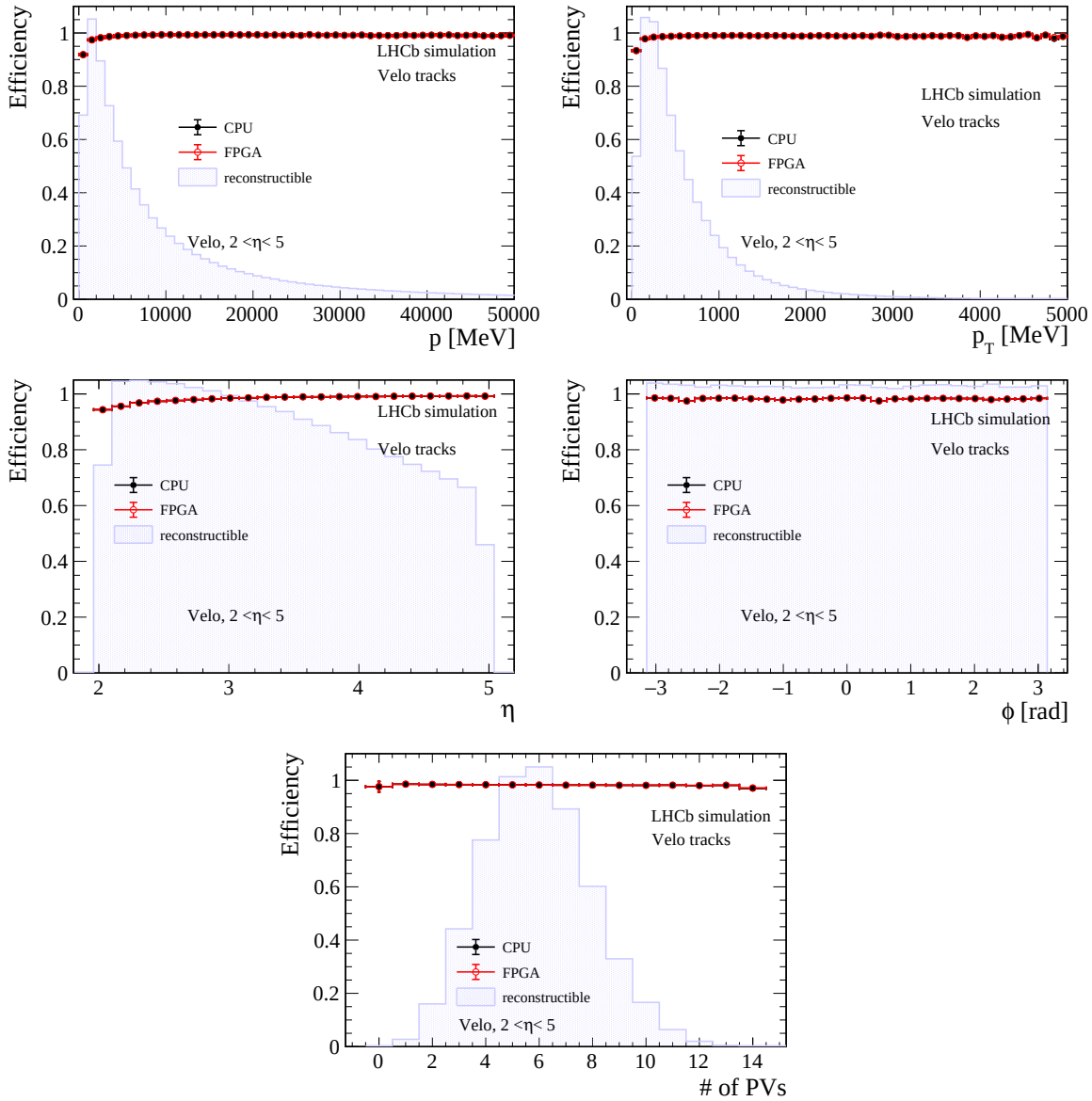

Figure 6.11: Comparison of the HLT1 reconstruction efficiency of all reconstructible VELO tracks when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$.

[8] DOCA_z is defined as the distance of closest approach of a track to the beamline, along the z direction.

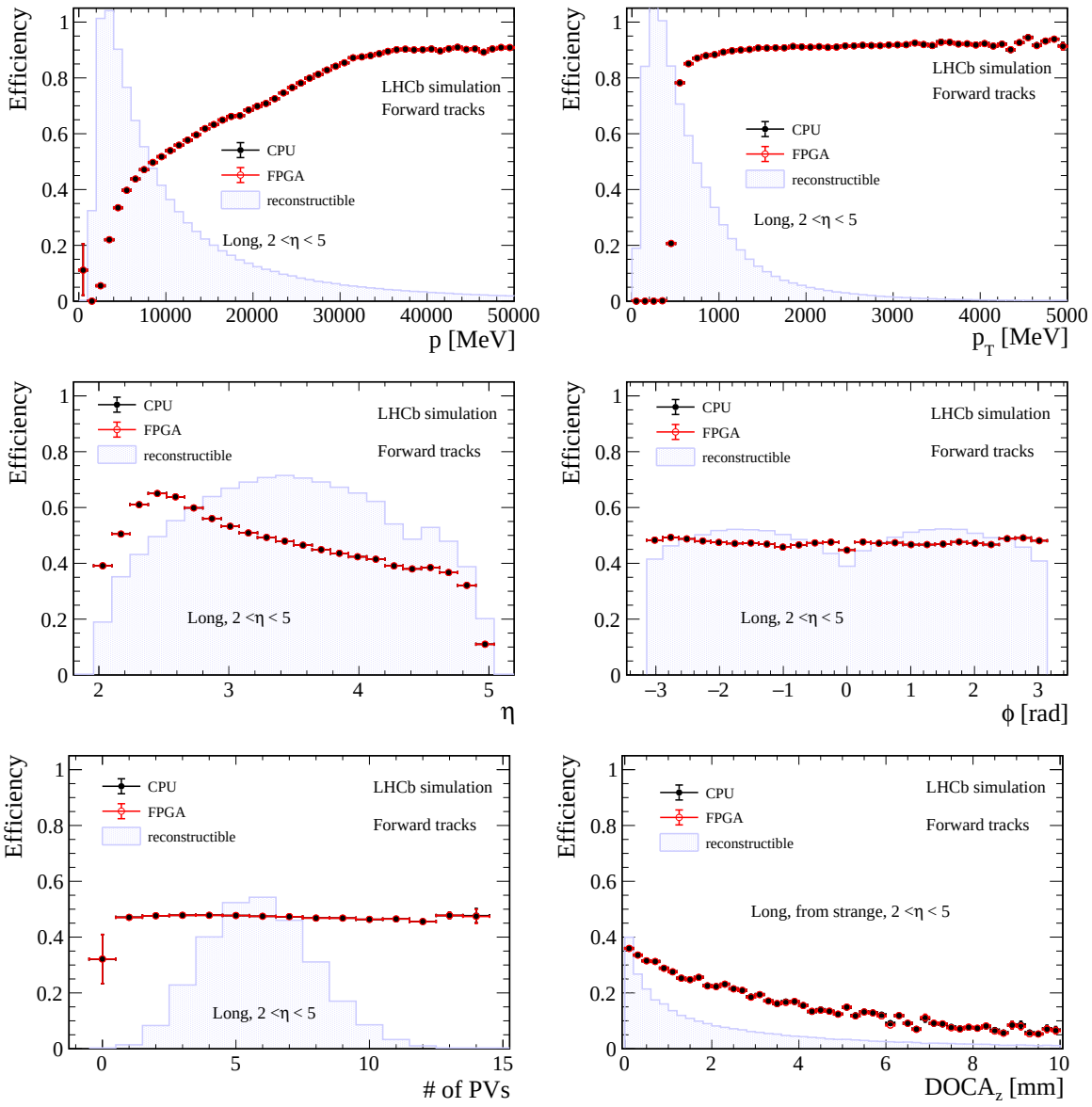

Figure 6.12: Comparison of the HLT1 reconstruction efficiency of all reconstructible long tracks when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}. The DOCA$_z$ plot at the bottom right is restricted to the sample of long tracks from s–hadron decays.

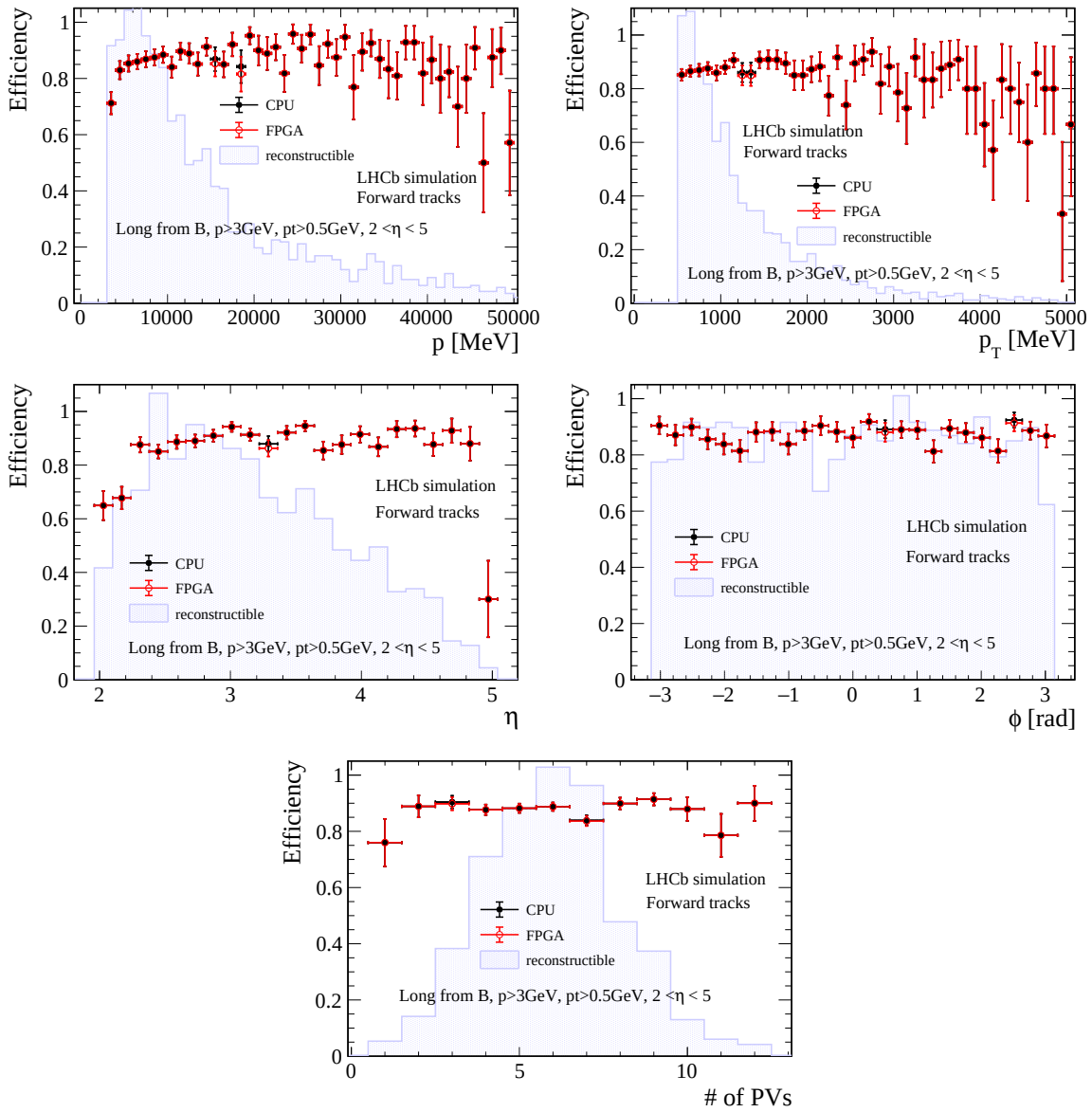

Figure 6.13: Comparison of the HLT1 reconstruction efficiency of all reconstructible long tracks produced in the decay of a b–hadron and with momentum (transverse momentum) greater than 3 GeV/c (0.5 GeV/c), when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

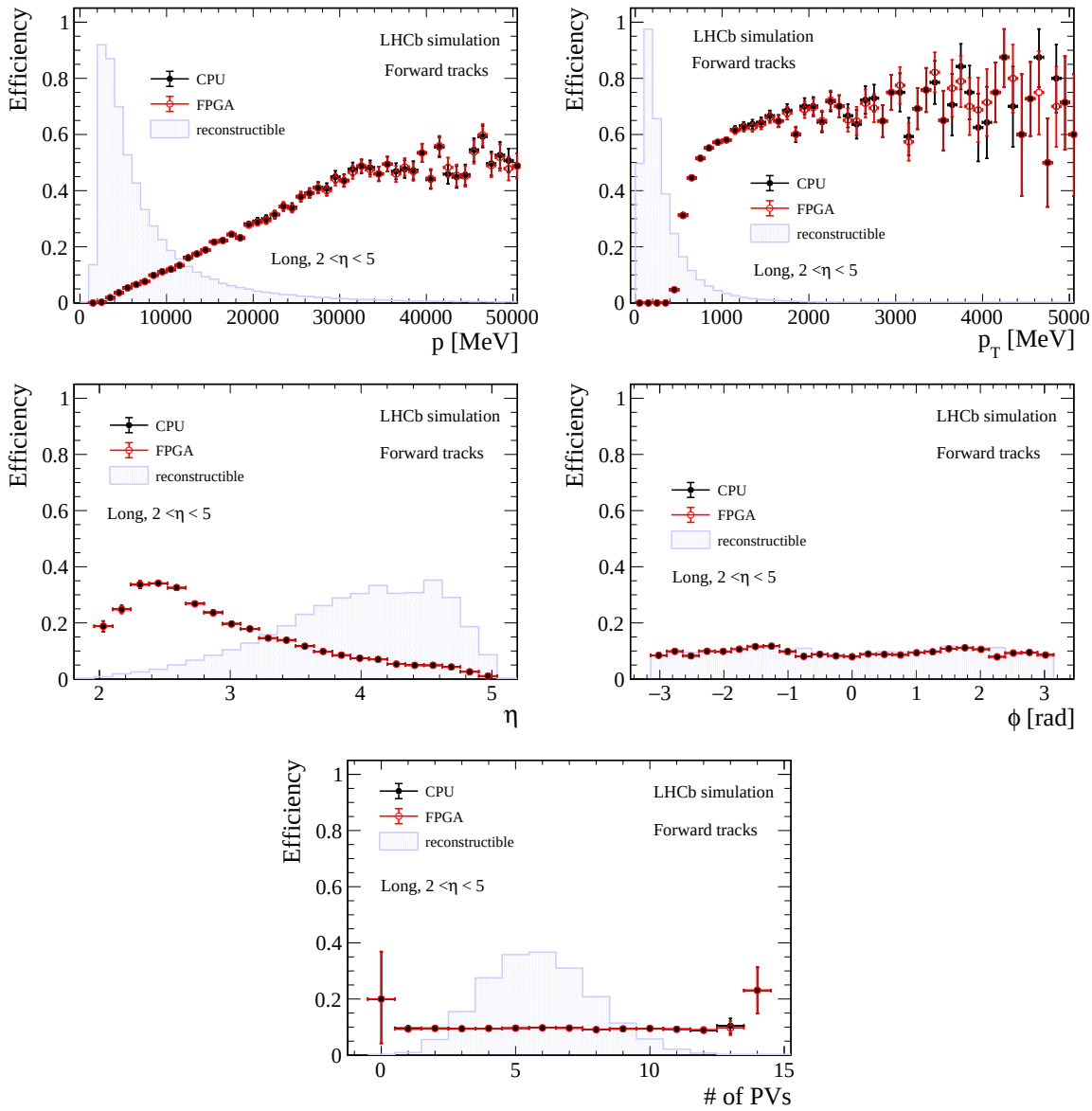

Figure 6.14: Comparison of the HLT1 reconstruction efficiency of all reconstructible electron long tracks when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

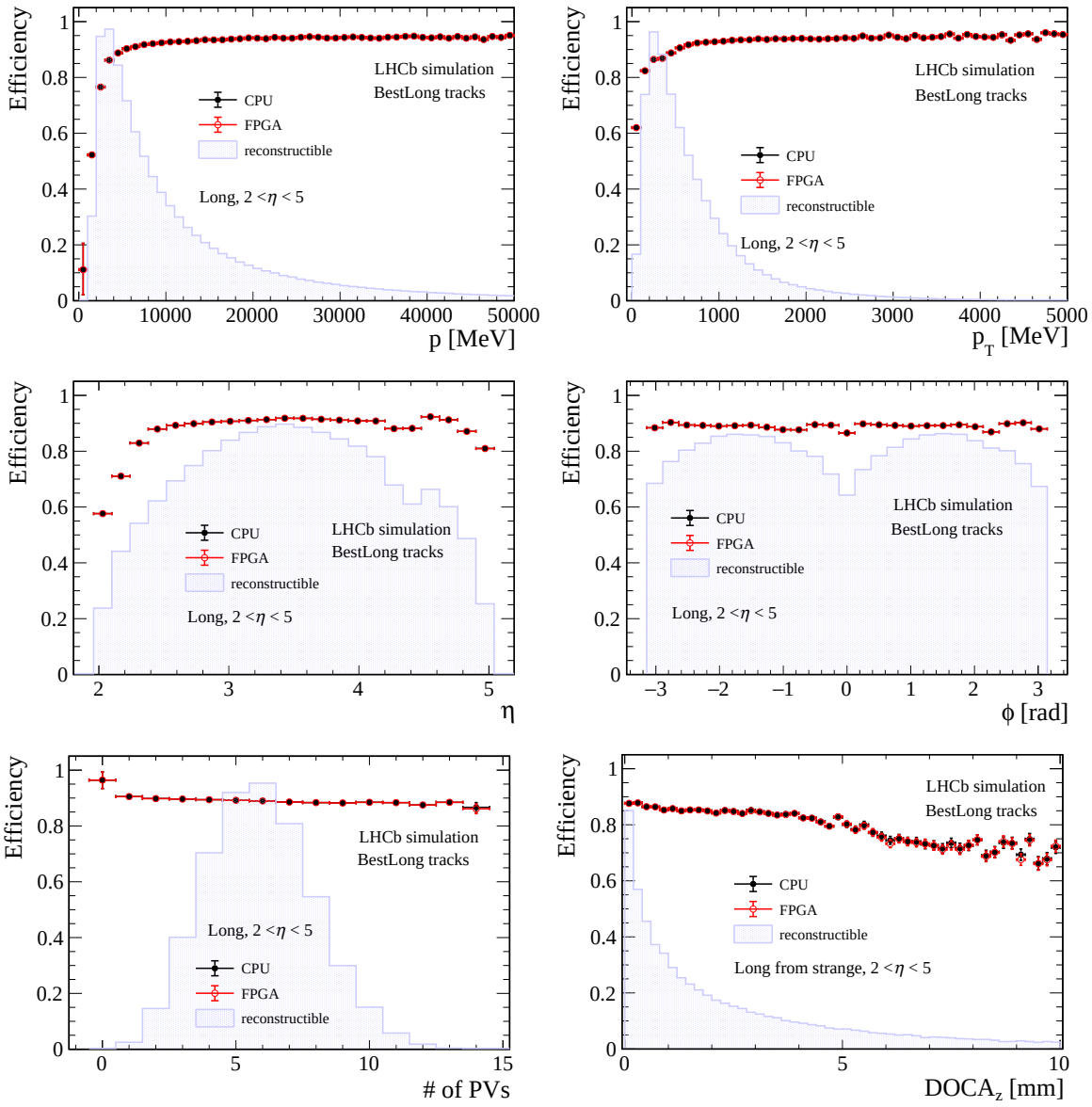

Figure 6.15: Comparison of the HLT2 reconstruction efficiency of all reconstructible long tracks when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}. The DOCA$_z$ plot at the bottom right is restricted to the sample of long tracks from s–hadron decays.

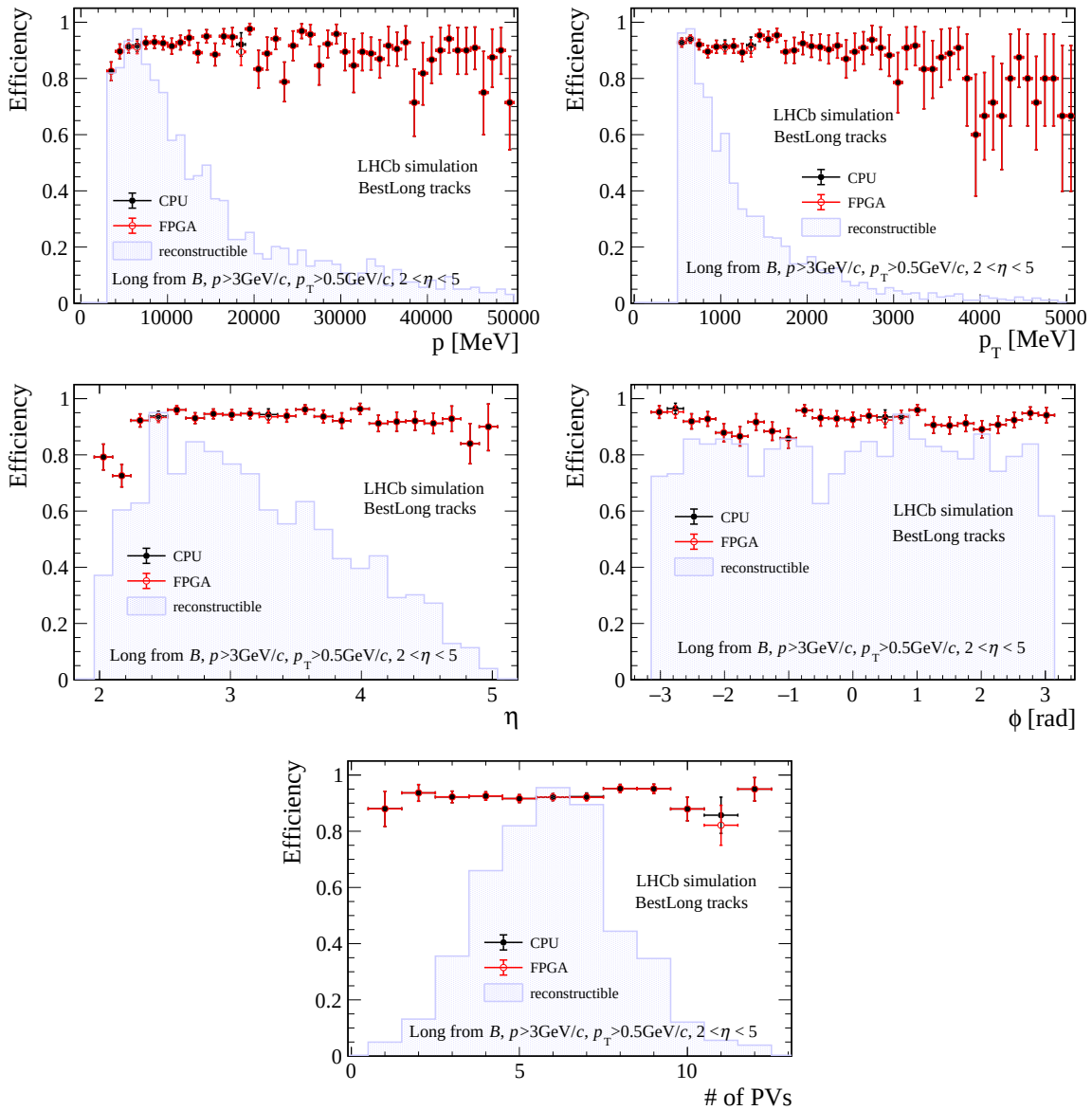

Figure 6.16: Comparison of the HLT2 reconstruction efficiency of all reconstructible long tracks produced in the decay of a b–hadron and with momentum (transverse momentum) greater than 3 GeV/c (0.5 GeV/c), when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

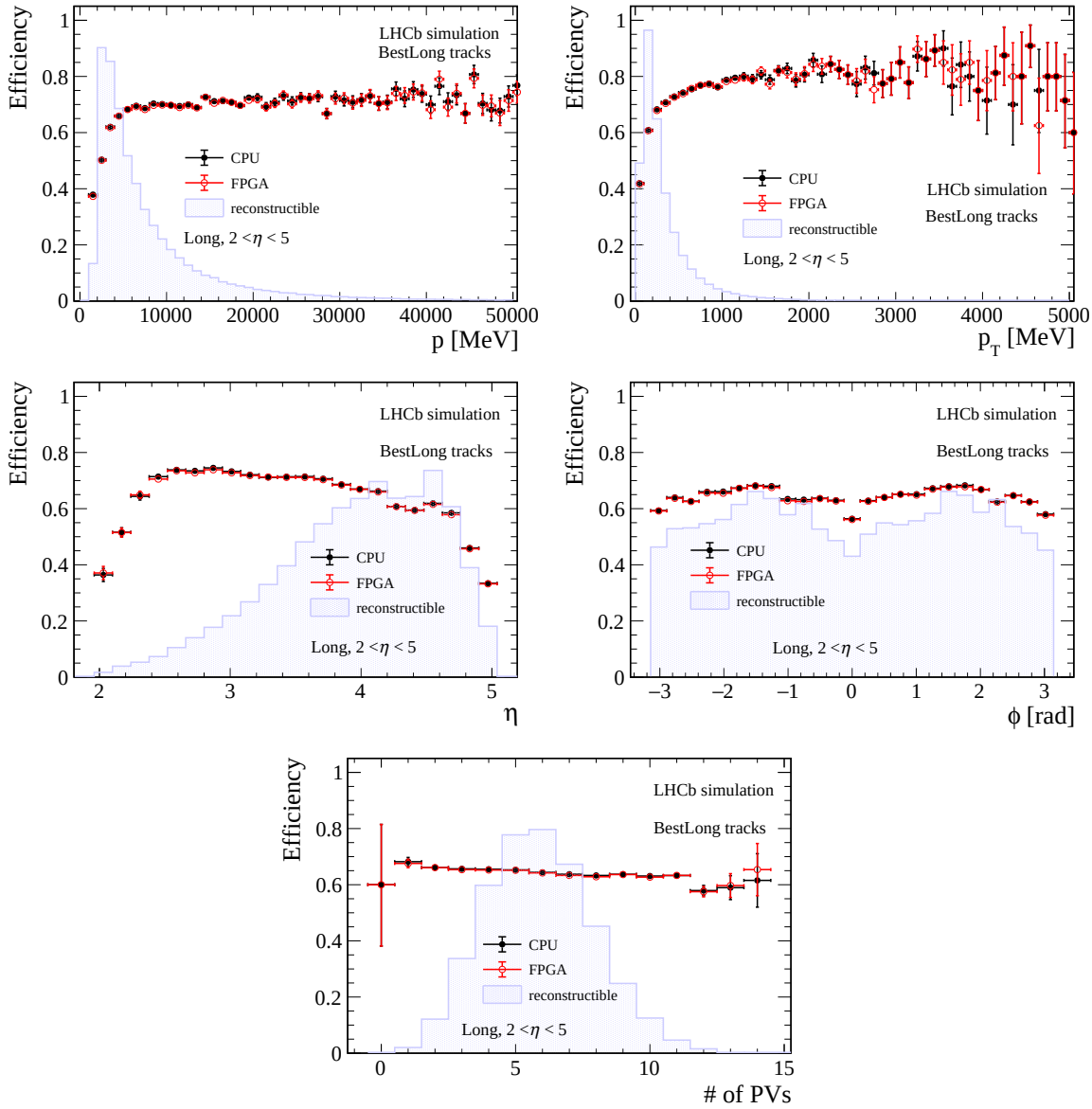

Figure 6.17: Comparison of the HLT2 reconstruction efficiency of all reconstructible electron long tracks when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

As anticipated by cluster–level studies in Sect. 6.2, the tracking efficiencies obtained with FPGA–based clustering are nearly indistinguishable from those using CPU–based clustering.

Another key quantity to monitor when comparing CPU and FPGA performances is the ghost rate of forward tracks. It is defined as the fraction of ghost tracks reconstructed over all reconstructed tracks, and it is shown in Fig. 6.18 as a function of p, p_{T}, η of the tracks and as a function of the number of PVs. A track is classified as a ghost track if it is not matched with any MC particle. Ghost tracks are reconstructed due to the mismatch of hits from separate MC particles or from detector noise or spillover [85]. Also, the ghost rate turns out to be indistinguishable between the two clustering approaches.

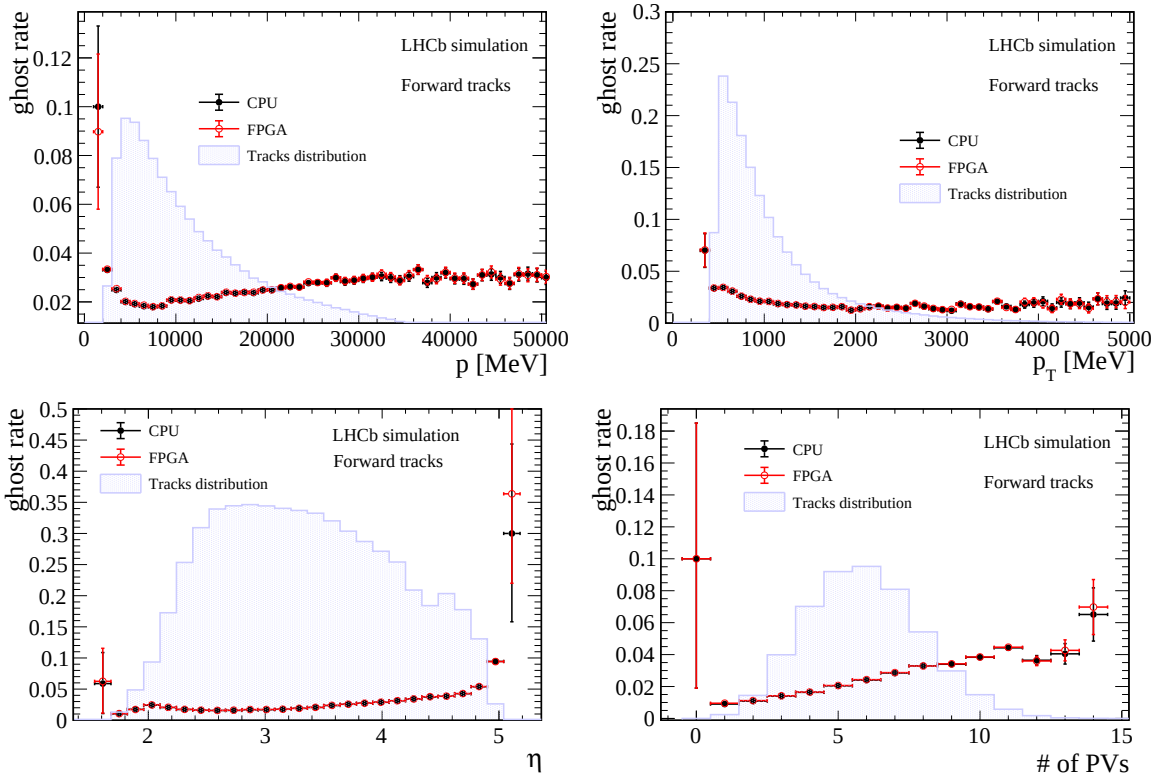

Figure 6.18: Comparison of HLT1 ghost rate of long tracks reconstructed using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

6.3.2 Electron tracking efficiency

When performing physics comparison studies between different VELO clustering algorithms, special care has been taken over electron reconstruction quality. The precise identification and characterization of electrons are particularly important for the LHCb physics program, especially for measurements related to Lepton Flavor Non–Universality tests. Like heavy charged particles, electrons release energy via the ionization process as they travel through matter. However, due to their small mass, the energy loss due to bremsstrahlung[9] effects become more relevant with respect to heavier charged particles. Moreover, electrons can originate from gamma conversion of photons interacting with matter. All of the above reasons suggest that electrons are more difficult to track and measure accurately with respect to other particles. Given their importance within the LHCb physics program and their peculiar way of interacting with matter, it is particularly important to study tracking reconstruction efficiencies focusing on electrons only. This is also motivated by the fact that electrons tend to produce larger clusters with respect to other particles, and this could lead to higher tracking inefficiencies due to the limit on the maximum cluster size (3×3 pixels). Therefore, the tracking reconstruction efficiency for long electron tracks is tested for both HLT1 and HLT2 reconstructions using a 10k–event MC sample of $B^0 \to K^{*0} e^+ e^-$ decays and a 40k–event MC sample of $B^0 \to K^{*0} \gamma$ decays, where the photon is converted into an electron–positron pair when interacting with matter, at the nominal Run 3 instantaneous luminosity of $2 \times 10^{33}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$. These decay modes belong to the $b \to sll$ category, which is one of the golden search paths for Lepton Flavor Universality violation.

[9] Bremsstrahlung consists of the emission of electromagnetic radiation due to the interaction of a charged particle with the nucleus electric field. The emission probability varies as the inverse square of the particle mass.

$B^0 \to K^{*0} e^+ e^-$

Figures 6.19 and 6.20 show HLT1 and HLT2 reconstruction efficiency comparisons on the $B^0 \to K^{*0} e^+ e^-$ sample, when using CPU and FPGA clustering algorithms.

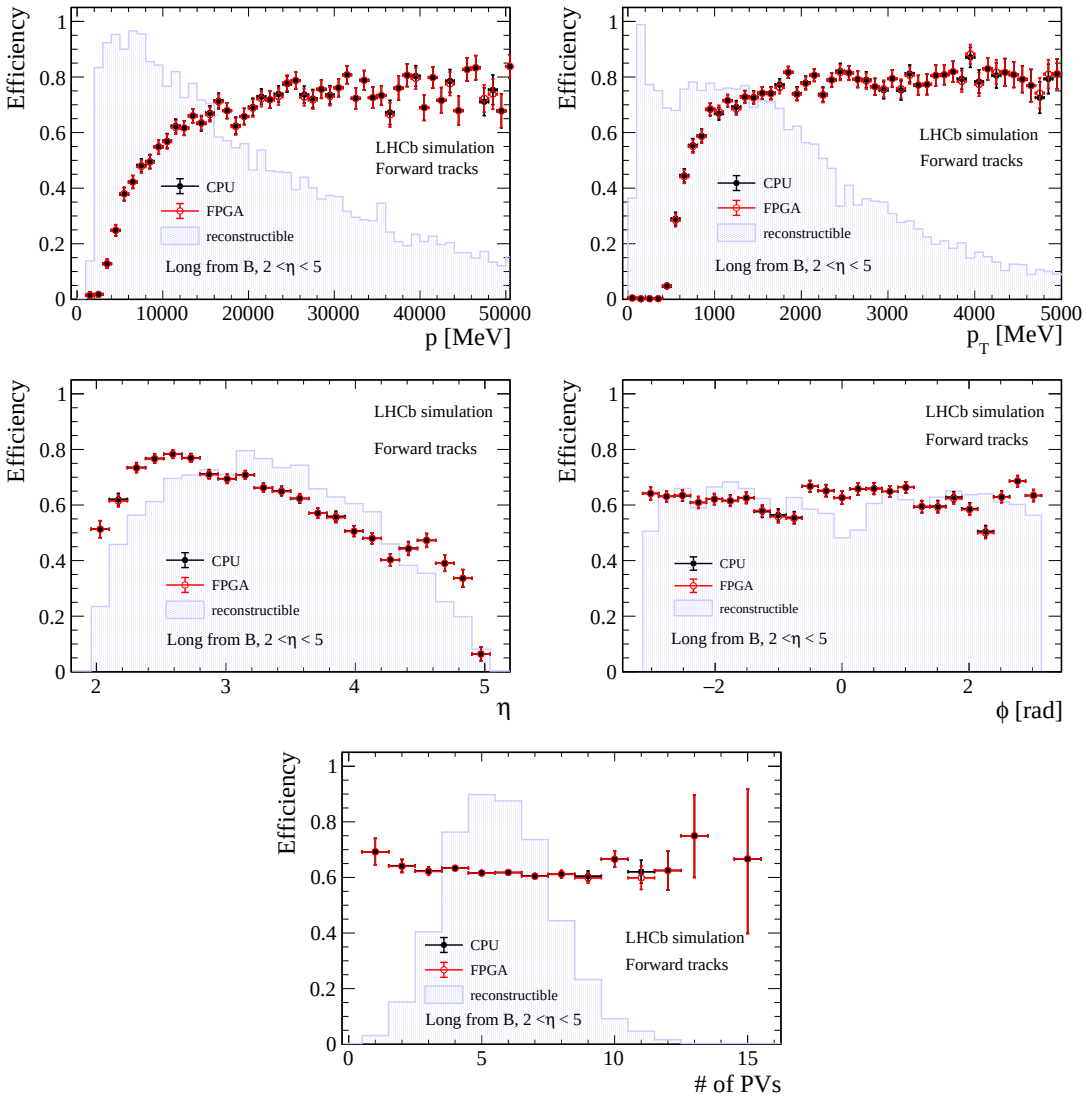

Figure 6.19: Comparison of the HLT1 reconstruction efficiency of all reconstructible electron long tracks from $B^0 \to K^{*0} e^+ e^-$ decays, when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

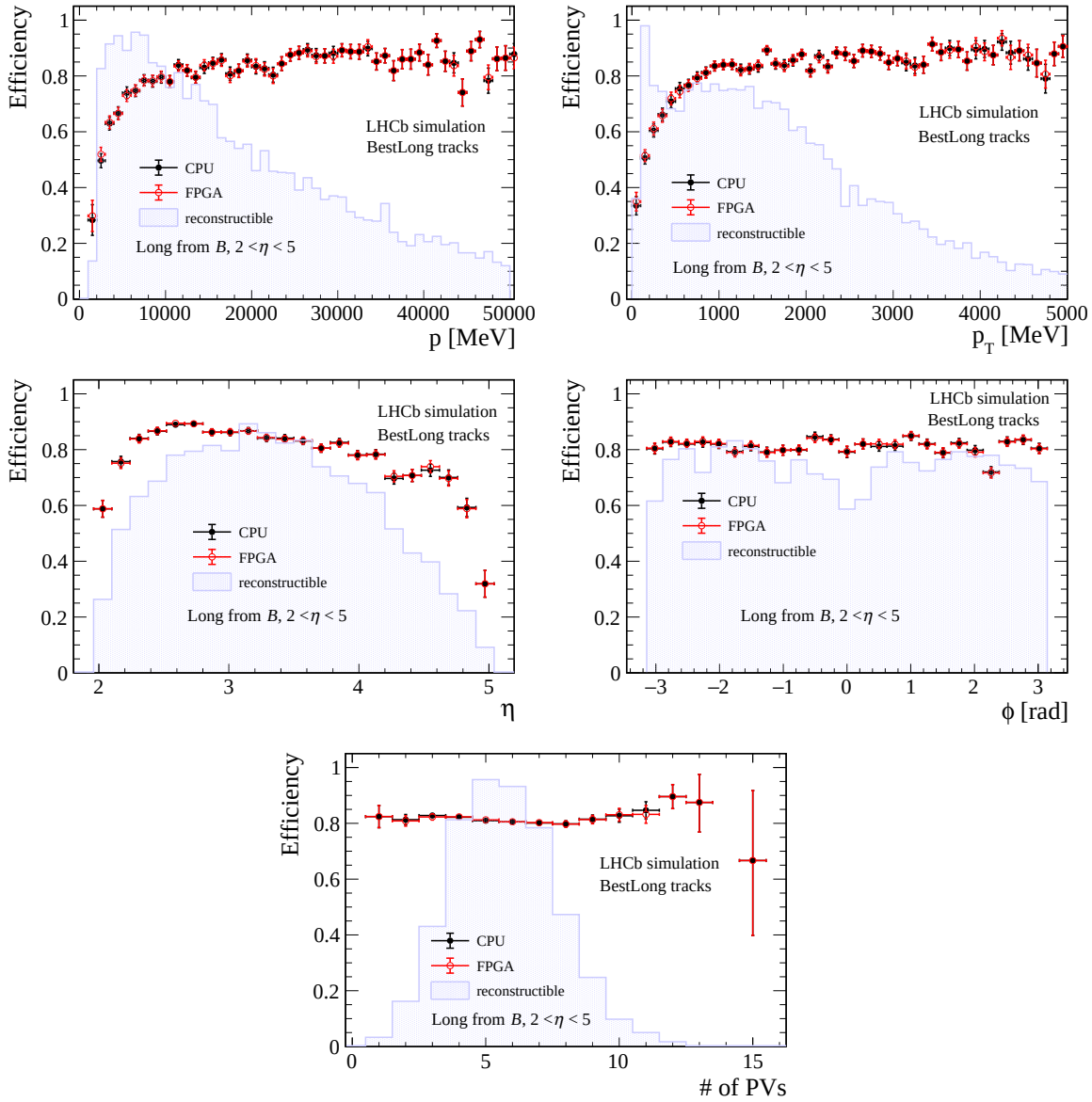

Figure 6.20: Comparison of the HLT2 reconstruction efficiency of reconstructible electron long tracks from $B^0 \to K^{*0} e^+ e^-$ decays when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

$B^0 \to K^{*0}\gamma$

Figures 6.21 and 6.22 show comparisons of HLT1 and HLT2 reconstruction efficiency on the $B^0 \to K^{*0}\gamma$ sample, when using CPU and FPGA clustering algorithms.

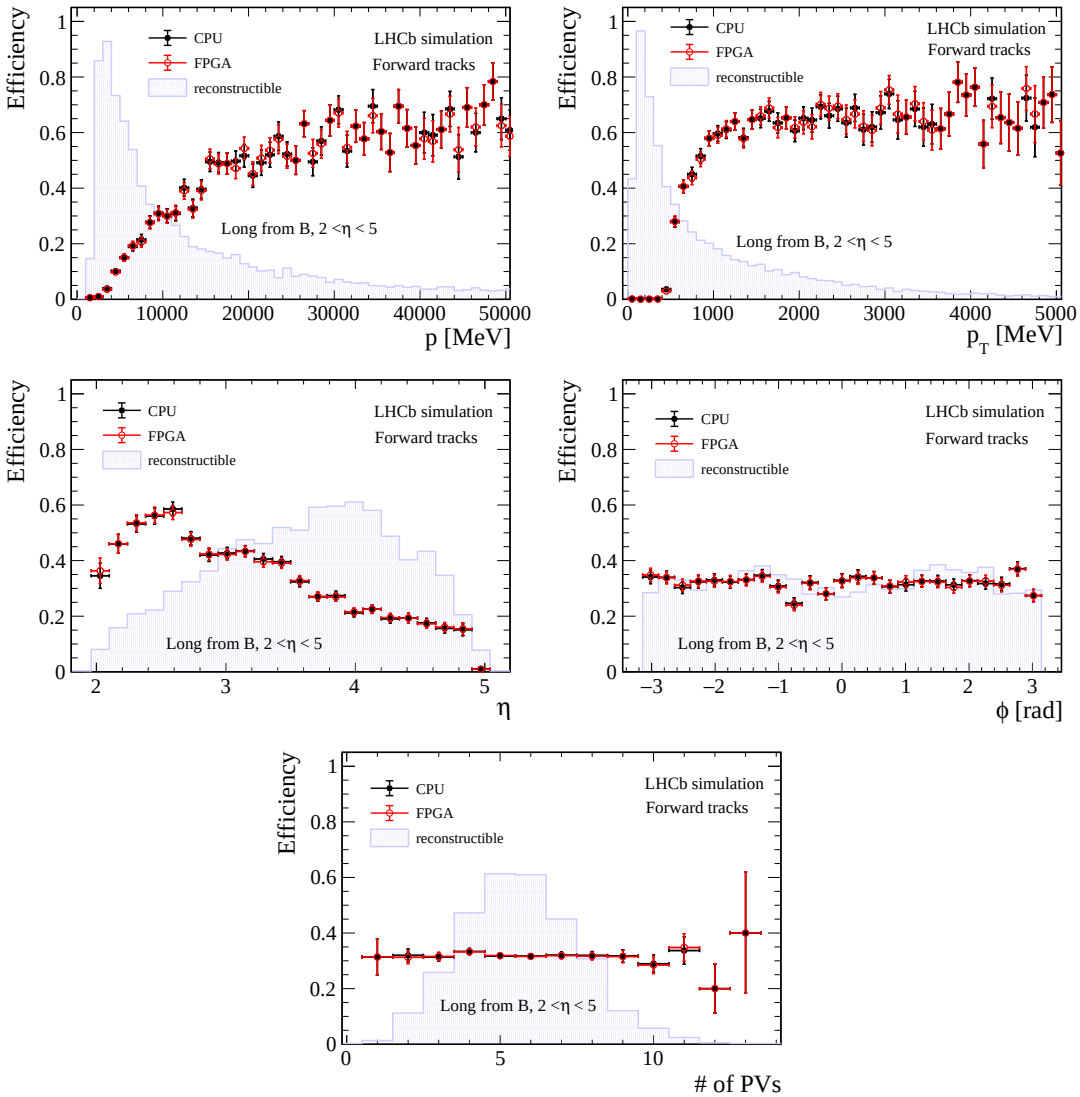

Figure 6.21: Comparison of the HLT1 reconstruction efficiency of all reconstructible electron long tracks from $B^0 \to K^{*0}\gamma$ decays, when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

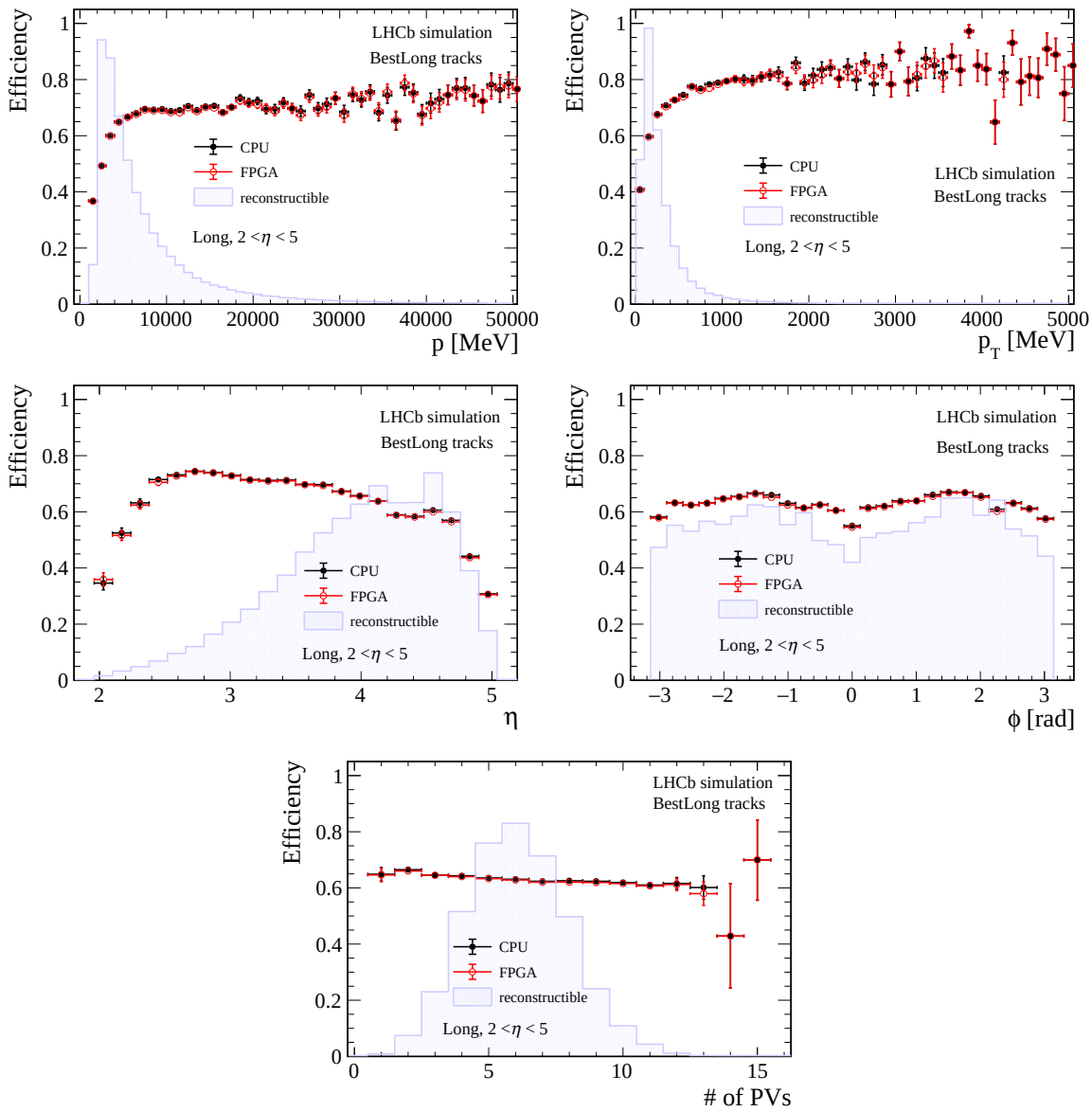

Figure 6.22: Comparison of the HLT2 reconstruction efficiency of all reconstructible electron long tracks from $B^0 \to K^{*0}\gamma$ decays when using CPU and FPGA clustering algorithms, as a function of various kinematic variables, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

In both $B^0 \to K^{*0}e^+e^-$ and $B^0 \to K^{*0}\gamma$ cases, the efficiencies obtained with CPU and FPGA clustering algorithms are nearly indistinguishable over all kinematic variables analyzed. This is a further confirmation that the approximations performed within the FPGA–based VELO clustering algorithm do not affect the reconstruction quality, also for these particularly important decay modes.

6.3.3 Impact parameter and momentum resolution

One of the key observables that need to be measured with high accuracy during the track reconstruction process is the impact parameter (IP). The IP_x observable is defined as the x component of the vector linking the PV to the intersection of the track with the plane transverse to the z axis and passing through the PV,

$$\mathrm{IP}_x \equiv (x - x_{\mathrm{PV}}) - -(z - z_{\mathrm{PV}})\frac{p_x}{p_z},$$

where all variables are referred to reconstructed quantities and (x, y, z) is the point of closest approach of the track to the PV (IP_y is the analog of IP_x with the substitution $x \to y$). Measuring the IP and its resolution with high precision is key for an efficient trigger and to determine the mean lifetime of particles. This is particularly important for time–dependent measurements, such as the study of very fast flavor oscillations of B_s^0 mesons. The IP resolution with respect to the PV is checked using HLT1 VELO Kalman–fitted tracks, by analysing separately IP_x and IP_y distributions. IP_x and IP_y resolutions, estimated using the standard deviation of a Gaussian function fitted to their distributions in the range $[-300, 300]$ µm, are displayed in bins of $1/p_{\mathrm{T}}$ and η in Fig. 6.23. The performances of the CPU and FPGA clustering algorithms are almost indistinguishable. One of the key ingredients for the high–precision IP determination is the Kalman filtering stage [87]. During Kalman filtering, the information from measurements at detector planes is combined to obtain optimal estimates of the track parameters, while rejecting fake tracks. Therefore, IP resolution is also checked using HLT2 full Kalman–fitted tracks.

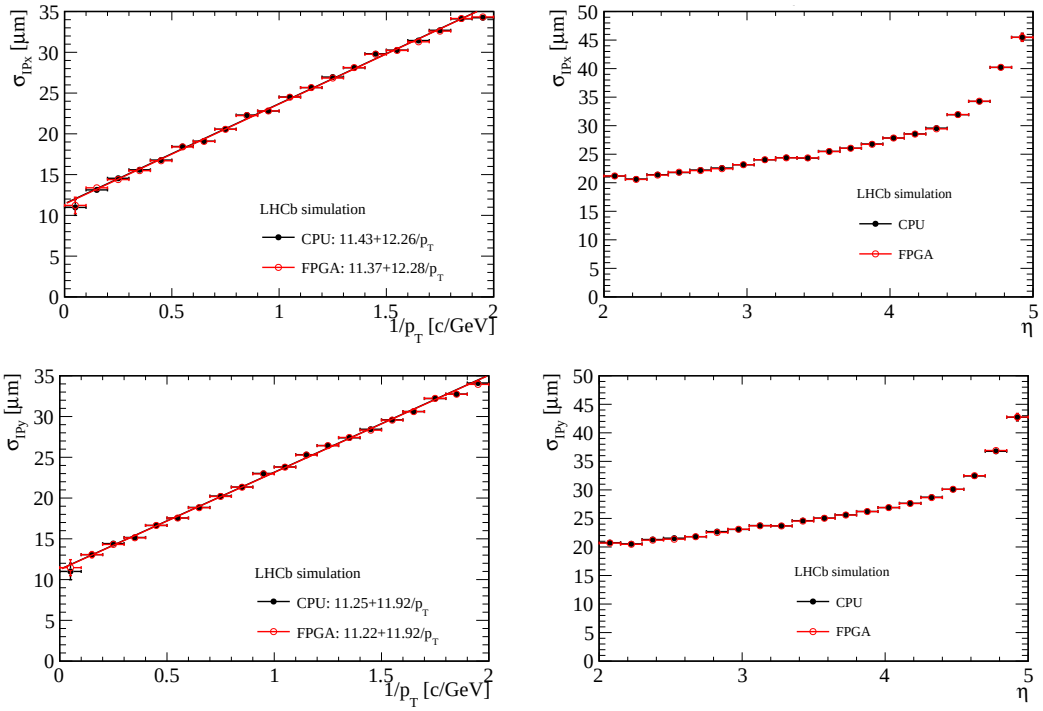

Figure 6.23: Comparison of the resolution of (top) IP_x and (bottom) IP_y of HLT1 Kalman–fitted VELO tracks when using the FPGA and CPU clustering algorithms, (left) as a function of the inverse of the true transverse momentum and (right) as a function of the true pseudorapidity of the track, at Run 3 instantaneous luminosity of 2×10^{33} $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$.

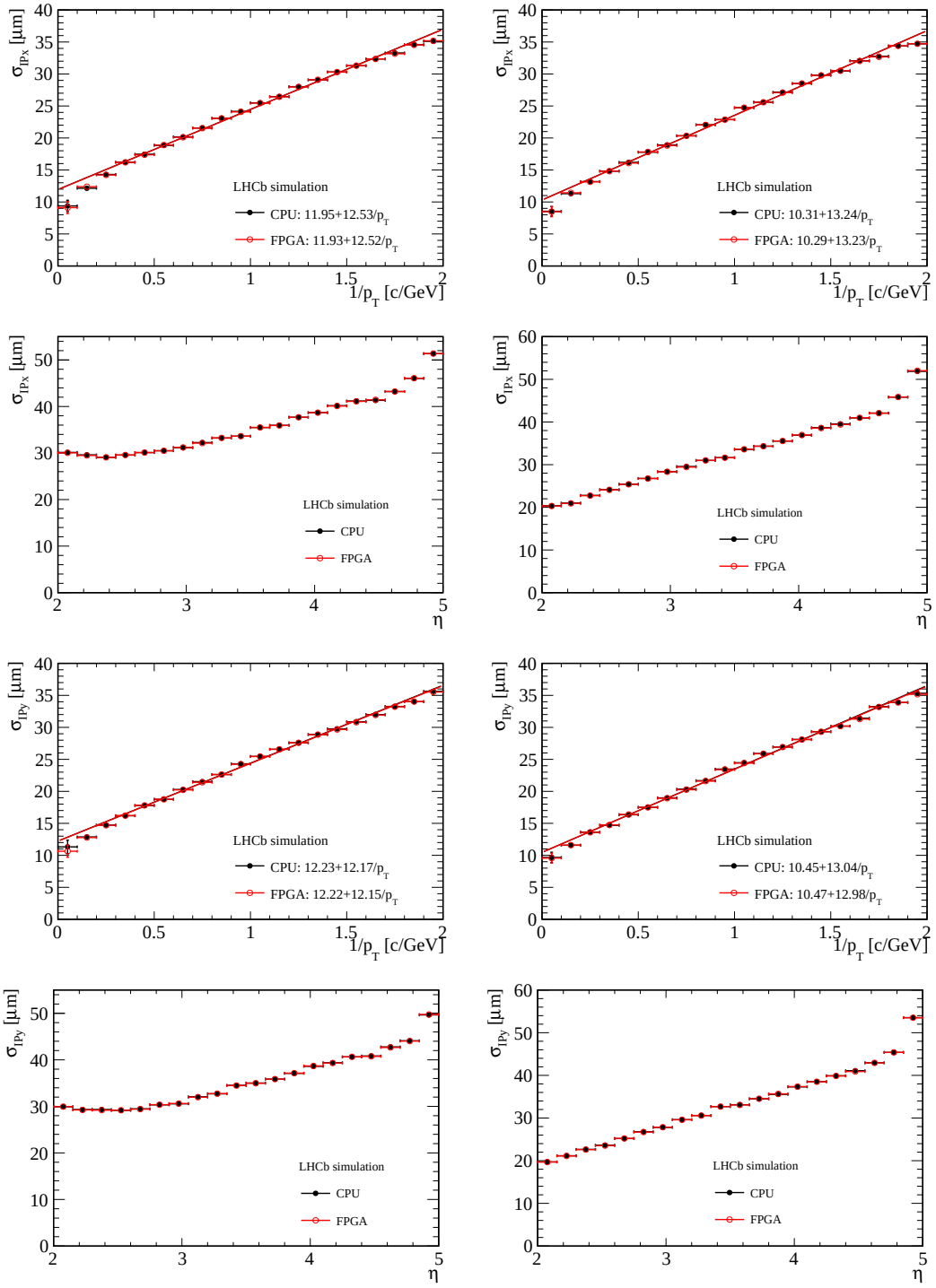

Figure 6.24: Comparison of (top–two) IP$_x$ and (bottom–two) IP$_y$ resolution of HLT2 full Kalman–fitted tracks using FPGA and CPU clustering algorithms, as a function of the inverse of the true transverse momentum and of the true pseudorapidity of the track, (left) considering all track types and (right) only long tracks, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

Resolutions on IP_x and IP_y are shown in Fig. 6.24 as a function of $1/p_\mathrm{T}$ and η for all track types and only long tracks, separately. Also in this case performances of the CPU and FPGA–based clustering algorithms are almost indistinguishable. As a possible improvement, the cluster topology, as provided by the firmware, can be used to assign more realistic uncertainties on the positions of the hits. This would improve tracking performances as a whole and, in particular, the IP resolution.

In the effort to identify possible reconstruction quality degradation while using FPGA VELO clusters, the momentum resolution is also studied. The relative resolution on the momentum magnitude for long tracks in the range $2 < \eta < 5$ is displayed as a function of the true momentum and pseudorapidity in Fig. 6.25 and Fig. 6.26, using HLT1 and HLT2 reconstruction, respectively. Resolution is defined as the sigma of a Gaussian function fitted to the $\mathrm{d}p/p$ distribution in the range $[-10\%, 10\%]$ ($[-5\%, 5\%]$) for the p (η) observable. As observed for track reconstruction efficiency and IP resolution, momentum resolution performance is indistinguishable when comparing CPU and FPGA VELO clusters.

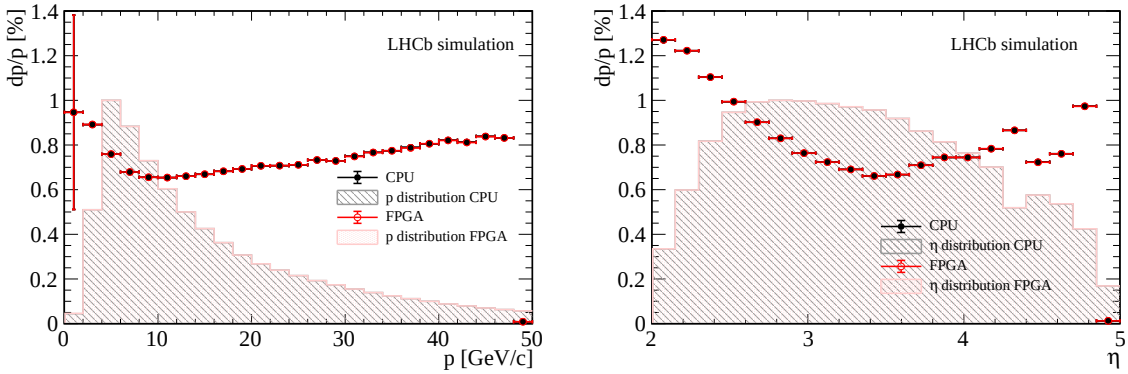

Figure 6.25: Comparison of the HLT1 resolution on the momentum of long tracks when using the FPGA and CPU clustering algorithms, (left) as a function of the true momentum and (right) as a function of the true pseudorapidity of the tracks, at Run 3 instantaneous luminosity of 2×10^{33} $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$.

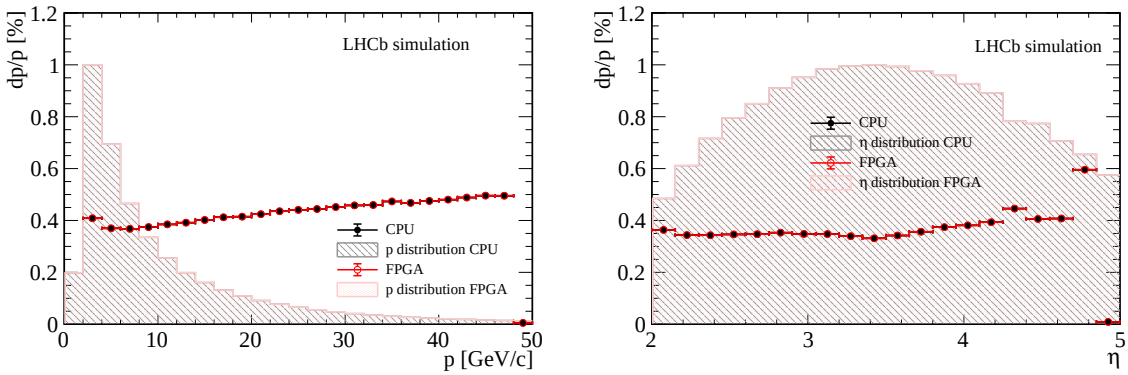

Figure 6.26: Comparison of the HLT2 resolution on the momentum of long tracks when using the FPGA and CPU clustering algorithms, (left) as a function of the true momentum and (right) as a function of the true pseudorapidity of the tracks, at Run 3 instantaneous luminosity of 2×10^{33} $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$.

6.3.4 Primary vertex reconstruction

The primary vertex (PV) is another key observable, used both for trigger applications and for precise measurement of time–dependent quantities, such as the particle time–of–flight and the flight distance between the primary and secondary vertices. Therefore, both the PV reconstruction efficiency and its resolution are checked during CPU–FPGA performance comparison studies. The primary vertex reconstruction efficiency is defined in Eq. 6.3

$$\epsilon \equiv \frac{N_{\text{MC-matched}}}{N_{\text{MC-reconstructible}}} \tag{6.3}$$

where $N_{\text{MC-matched}}$ is the number of reconstructed primary vertices that are matched to an MC PV, and $N_{\text{MC-reconstructible}}$ is the number of MC PVs with at least four reconstructed VELO tracks. MC matching is performed by distance, requiring that the reconstructed PV lies at a distance along the z axis less than 2 mm or $5\sigma(z_{\text{PV}})$, whichever is less, from the MC PV, where $\sigma(z_{\text{PV}})$ is the uncertainty of the reconstructed position of the PV along the z axis. Figure 6.27 shows the efficiency comparison for the two clustering methods, both as a function of the number of reconstructed tracks associated with the corresponding MC PV and of the z coordinate of the MC PV. The reconstruction efficiency as a function of z_{PV} is rather flat in both cases, but

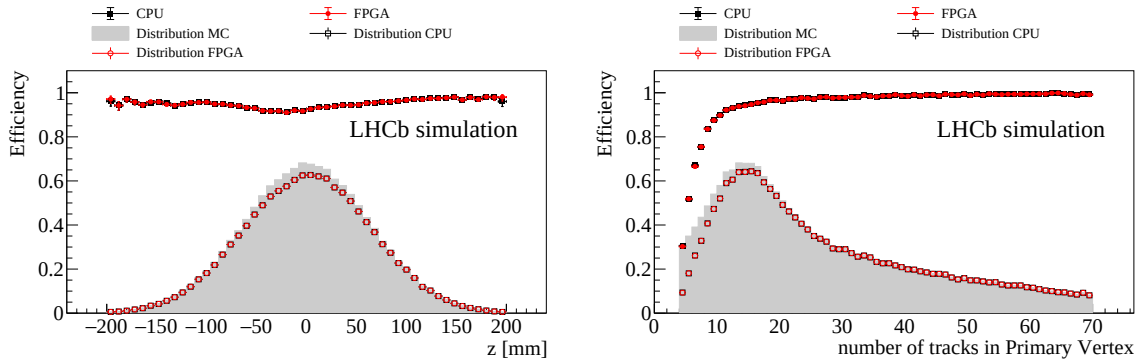

Figure 6.27: Comparison of the reconstruction efficiency of the PVs with the FPGA and CPU clustering algorithms, as a function of (left) the true z position of the PV vertex and (right) of the number of reconstructed tracks of the PV, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

both clustering methods exhibit a small drop in efficiency for values of z_{PV} between -60 and 0 mm. The resolution and bias of the PV reconstruction is quantified along each coordinate axis, based on the distribution of the residuals of the reconstructed position of the PV minus the true one ($\Delta x \equiv x_{\text{reconstructed}} - x_{\text{MC}}$, *etc.*). To prevent our estimates from being confounded by the presence of few outliers far in the tails of these high–statistics distributions, we adopt a more robust procedure to measure resolutions than the raw root mean square (RMS). First, the RMS of the bulk of the distribution is estimated from the values of the 25$^{\text{th}}$ and 75$^{\text{th}}$ percentiles of the distribution. Second, a Gaussian fit is performed with a range limited to ± 4 RMS around zero, and the sigma of this Gaussian is taken as a measure of the resolution. Figure 6.28 shows the results as a function of z_{PV} and the number of reconstructed PV tracks. The performances of the CPU and FPGA–based clustering algorithms turn out to be barely distinguishable. The resolutions along all the axes show a small knee in the region $-60 < z_{\text{PV}} < 0$ mm that is characterized by a lower reconstruction efficiency (see Fig. 6.27). Further checks on the PV reconstruction are displayed in Appendix C. In particular, a bias on the reconstruction of z_{PV} had

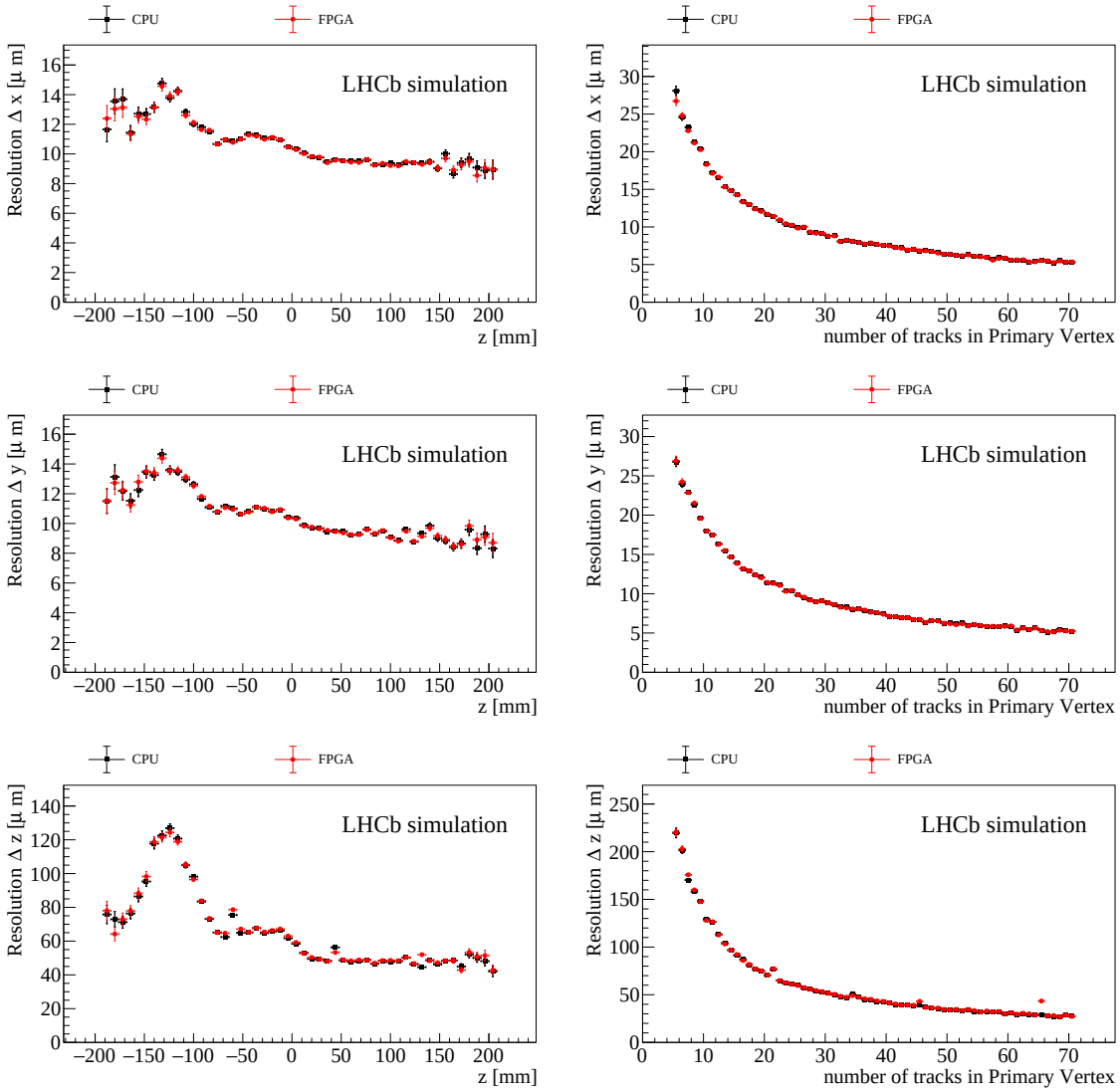

Figure 6.28: Comparison of the resolution in reconstructing the PVs with the FPGA and CPU clustering algorithms, as a function of (left) the true z position of the PV vertex and (right) of the number of reconstructed tracks, for (top) the x coordinate, (center) the y coordinate and (bottom) the z coordinate of the PV, at Run 3 instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}.

initially been spotted, limited to the low–efficiency and low–resolution region $-60 < z_{\mathrm{PV}} < 0$ mm. The size of the bias was about $-3\,\mu$m, which corresponds to approximately 4% of the z_{PV} resolution. The origin of the bias was traced back to the first version of the rounding of the coordinates of the cluster center position. Cluster coordinates are encoded in the FPGA output with two fractional binary digits, corresponding to a resolution of one fourth of a pixel. In the first implementation of the code, the values of the coordinates were truncated, causing a bias on the measurement of the cluster positions. In the current implementation, instead, a rounding to the closest one–fourth step is applied, and the bias is no longer visible. In conclusion, all studies have shown that FPGA–reconstructed clusters lead to a track reconstruction quality that is effectively indistinguishable from the software reconstruction.

6.4 Robustness to large clusters split–up

This section details all the studies performed to determine the robustness of the FPGA cluster reconstruction algorithm to the amount of large cluster split–up. As discussed in Sect. 3.3, the clustering algorithm reconstructs clusters with dimensions up to 3×3 pixels. Any cluster larger than that would be partially reconstructed. Furthermore, due to the limited dimensions of the matrix, SPs that make up a single cluster may end up in different matrices, thus leading to cluster splitting. Moreover, the cluster split–up or partial reconstruction occurrence rate in real data might not be accurately predicted by simulation. For these reasons, we performed additional studies to identify possible issues as the abundance of large clusters increases. Here, the comparison with the CPU case is investigated, using the same reconstruction code and the same simulated sample as in Sect. 6.2.1. In the following, a CPU cluster is referred to as a "large cluster" if it contains more than 9 pixels. This is the maximum number of pixels that a FPGA cluster can include, while any cluster with more than 9 pixels is reconstructed with only a subset of its pixels, and in some cases it is split into more clusters.

The left plot in Fig. 6.29 shows the distribution of the fraction of large clusters per event, divided into four equally populated quantiles. The rightmost quantile is further divided into two regions to investigate the effect of events with a high fraction of large clusters on the reconstruction quality. Moving from the leftmost quantile to the rightmost tail, the large cluster fraction increases from 0.42% to 1.83%, spanning over a factor of four. The bin corresponding to zero fraction contains events that do not have large clusters. These events have a smaller number of clusters compared to the average event, as shown in the right plot of Fig. 6.29.

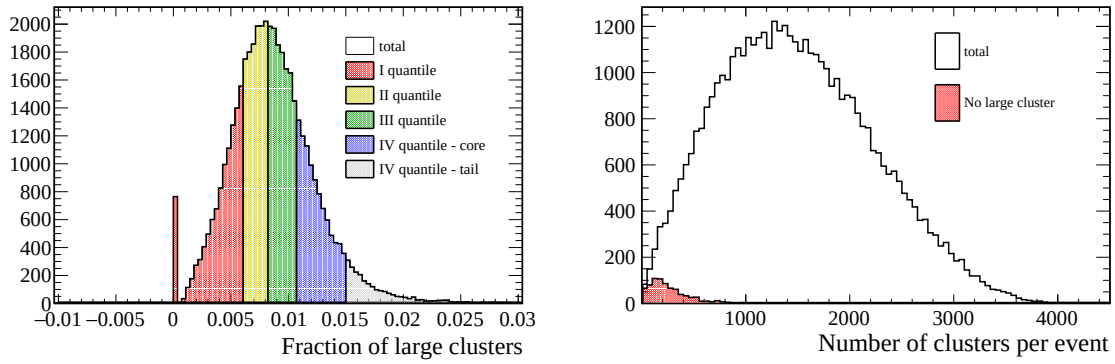

Figure 6.29: (Left) Distribution of the fraction of large clusters ($N_{\text{pixels}} > 9$) per event, split into four equally populated quantiles. The quantile corresponding to the highest fractions is further divided into two regions for a better identification of the effects in the right tail. (Right) Distribution of the number of CPU clusters per event, (black) for all events and (red) only for events without any large clusters.

Figure 6.30 shows the ghost fraction, defined as the ratio between the number of ghost tracks and the number of reconstructed tracks, as a function of the large cluster abundance. Each point is centered on the average fraction of each of the five highlighted regions in Fig. 6.29. CPU and FPGA ghost rates follow the same trend, with the absolute difference between the two always below 0.1%. Figures 6.31, 6.32 and 6.33 show tracking reconstruction efficiencies, clone rates, and hitEffFirst3 for (left) VELO and (right) long tracks, respectively. The tracking reconstruction efficiency is defined in Sect. 6.3.1. The clone rate is defined as the ratio between the number of clone tracks and the number of reconstructed tracks. HitEffFirst3 is the VELO hit efficiency using the hits on the first 3 layers, which are key to precisely identify the origin

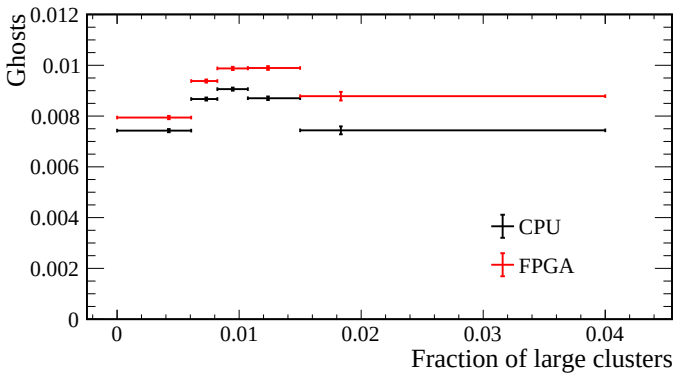

Figure 6.30: HLT1 ghost rate as a function of large cluster abundance.

vertex of the track. Hit efficiency is defined as the ratio between the number of hits shared between the matched and the reconstructed track and the number of hits of the matched track.

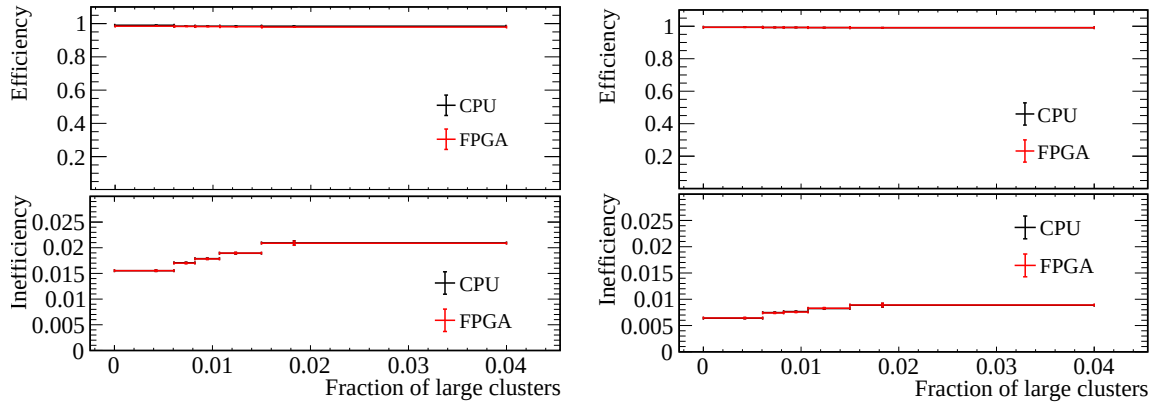

Figure 6.31: (top) HLT1 efficiency and (bottom) inefficiency (left) for VELO tracks and (right) for long tracks as a function of large cluster abundance.

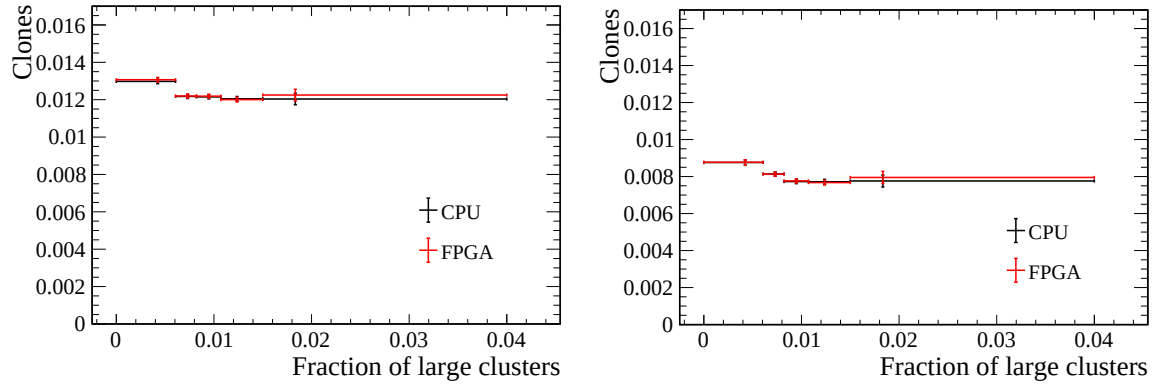

Figure 6.32: HLT1 clone rate (left) for VELO tracks and (right) for long tracks as a function of large cluster abundance.

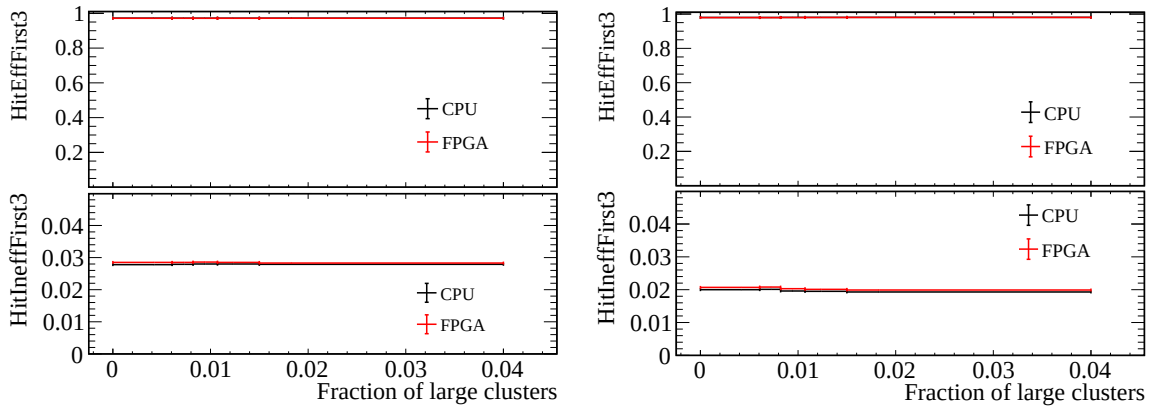

Figure 6.33: (top) HLT1 HitEffFirst3 and (bottom) HitIneffFirst3 (left) for VELO tracks and (right) for long tracks as a function of large cluster abundance.

All studies show permille–level differences when comparing FPGA and CPU algorithms over the large–cluster fraction distribution in Fig. 6.29 (left). We conclude that the small differences observed between FPGA and CPU clustering algorithms do not show any tendency to enlarge when the fraction of large clusters increases, neither for the ghost rate nor for the clone rate or tracking efficiency. Therefore, there is no reasons of concern that a possible larger than expected proportion of large clusters might cause an unanticipated drop in performance.

6.5 Robustness to VELO occupancy

Another important aspect to be checked is the level of robustness of the FPGA clustering with respect to the increase in the VELO occupancies. The simulation could underestimate the occupancy of clusters (and SPs) in the VELO sensors; therefore, it is of crucial importance to check if the reconstruction degrades in terms of cluster efficiency and tracking performance when the occupancy increases.

6.5.1 Clustering efficiency

The FPGA clustering efficiency depends on the occupancy of each VELO module because of the following main reasons:

- the probability of having not isolated SPs (and so the cluster dimensions) increases with a higher number of SPs;
- as the number of SPs increases, the number of matrices instantiated inside the FPGA might not be enough to accommodate all of them.

The efficiency (or inefficiency) of the clustering algorithm is measured using simulated samples by selecting regions with a different average number of passing tracks and, consequently, a different number of SPs and reconstructible clusters. In order to determine the local occupancy, for each VELO sensor a 2D histogram[10] is filled with the positions of the reconstructible MC

[10]The 2D histogram is divided into 0.5 mm × 0.5 mm bins.

hits, by integrating over the whole set of reconstructed available events. Therefore, the local occupancy is determined as the ratio of the number of counts in each bin to the area of the bin, divided by the total number of reconstructed events. Subsequently, a 1D histogram in occupancy bins is filled with the number of reconstructible MC hits. The same procedure is repeated with non–reconstructed MC hits. Inefficiency is then measured as the ratio between these two 1D histograms. Figure 6.34 shows a comparison between CPU and FPGA clustering inefficiencies as a function of VELO occupancy, where the FPGA inefficiency is plotted considering both all clusters and selecting only clusters from VELO reconstructible tracks. The difference between FPGA and CPU clustering algorithms does not show any relevant tendencies to increase when the local VELO occupancy increases.

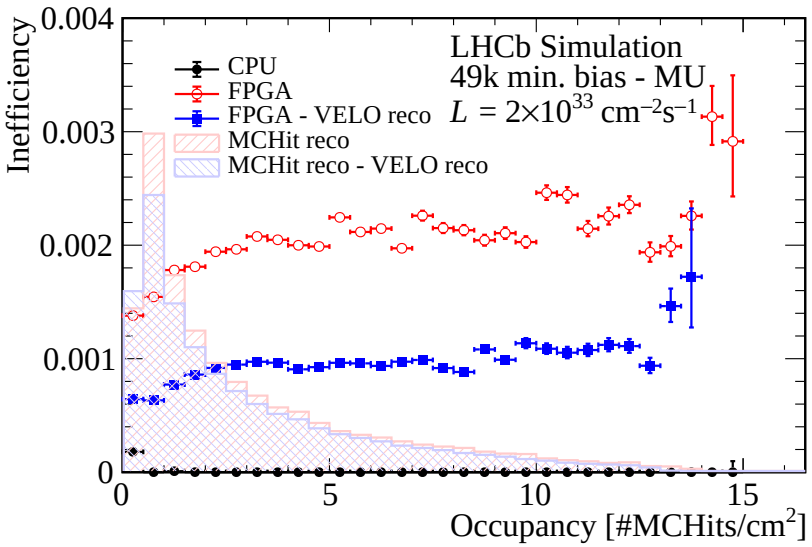

Figure 6.34: Clustering inefficiency as a function of the local VELO occupancy.

6.5.2 Tracking performances

Tracking performance is also studied as a function of the total number of SPs per event in the VELO detector. The distribution of the number of SPs per event, divided into four equally populated quantiles, is shown in Fig. 6.35. The rightmost quantile is further divided into two regions to investigate the effect of events with a high number of SPs on the reconstruction quality. Moving from the leftmost quantile to the tail on the right, the number of SPs spans almost a factor of five.

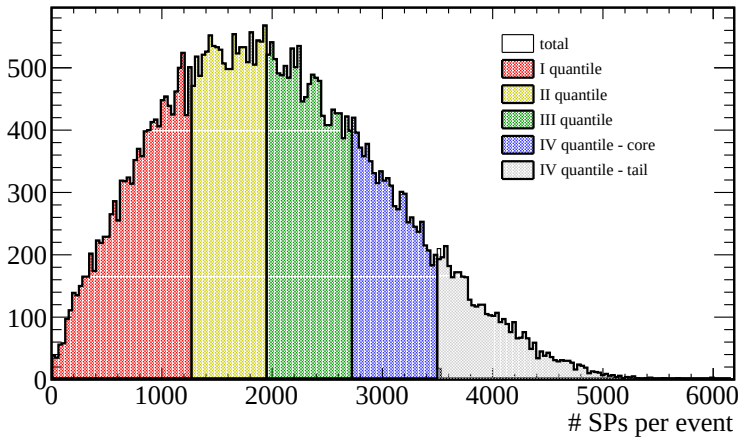

Figure 6.35: Distribution of the number of VELO SPs per event.

Figure 6.36, instead, shows the ghost fraction as a function of the number of SPs. Each point is centered on the average fraction of each of the five highlighted regions in Fig. 6.35. CPU and FPGA ghost rates follow the same trend, with the absolute difference between the two staying always below 0.1%.

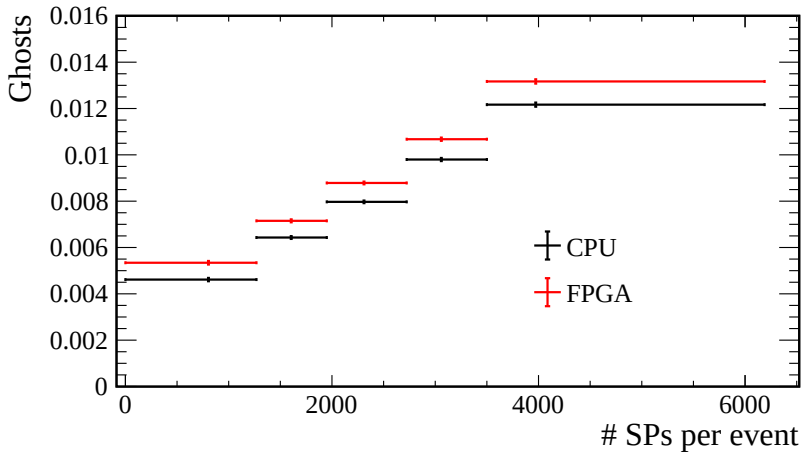

Figure 6.36: HLT1 ghost rate as a function of the number of SPs per event.

Finally, Figures 6.37, 6.38 and 6.39 show tracking reconstruction efficiencies, clone rates, and hitEffFirst3 for (left) VELO and (right) long tracks.

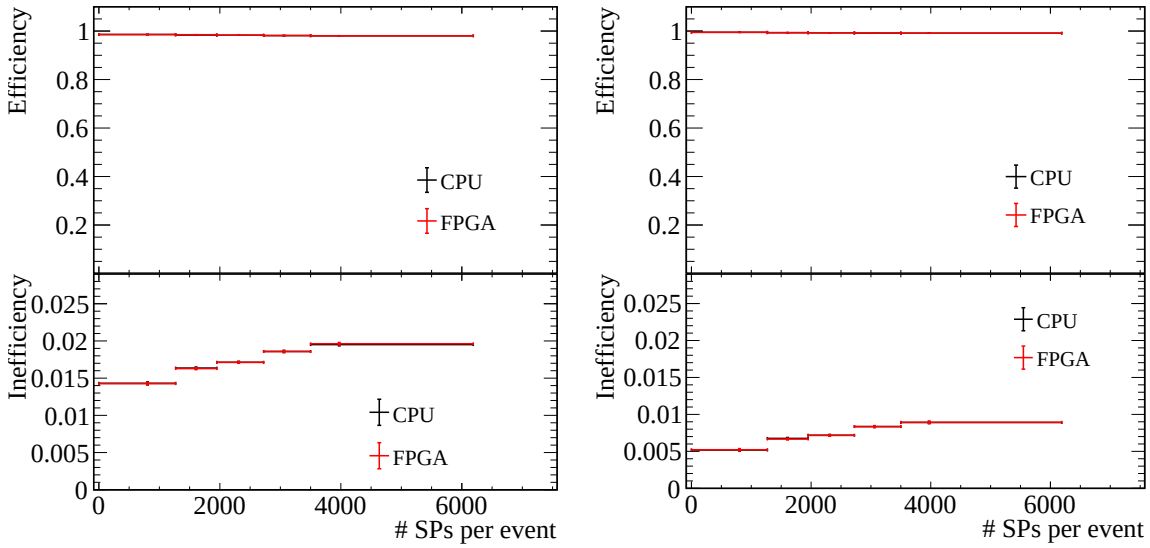

Figure 6.37: (top) HLT1 efficiency and (bottom) inefficiency for (left) VELO tracks and (right) long tracks as a function of the number of SPs per event.

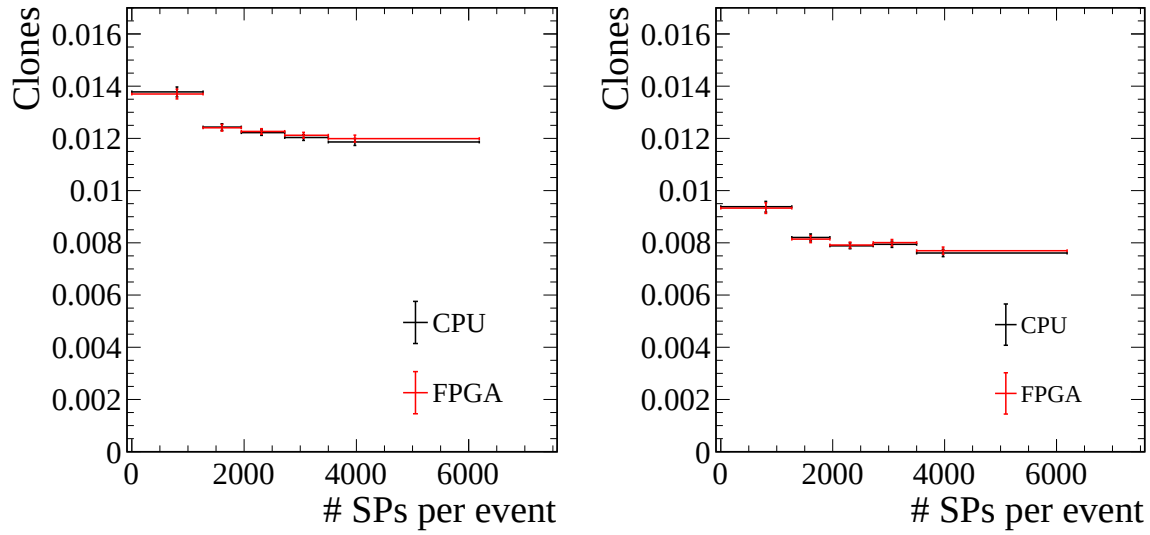

Figure 6.38: HLT1 clone rate (left) for VELO tracks and (right) for long tracks as a function of the number of SPs per event.

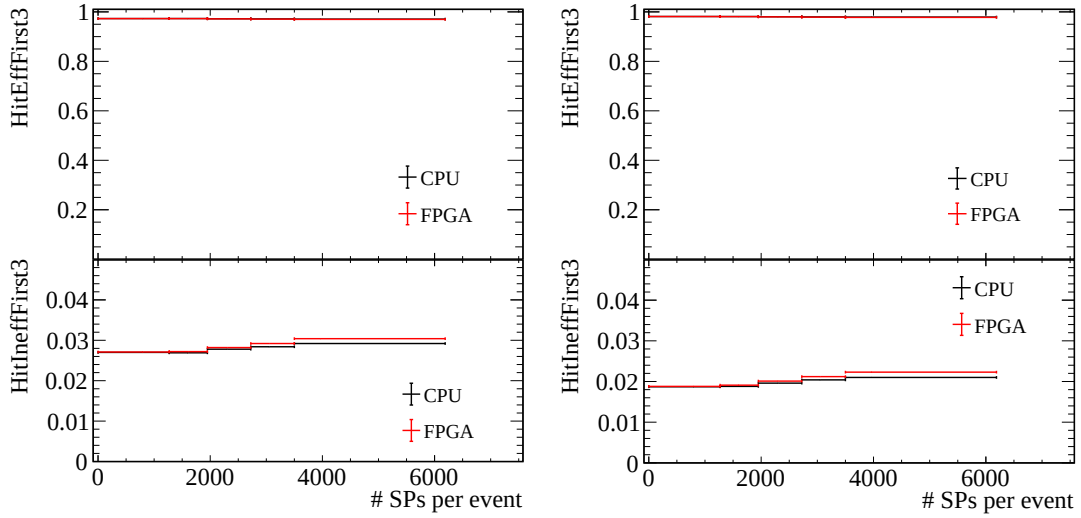

Figure 6.39: (top) HLT1 HitEffFirst3 and (bottom) HitIneffFirst3 (left) for VELO tracks and (right) for long tracks as a function of the number of SPs per event.

All presented studies show permille–level changes spanning over the number of SPs per event distribution in Fig. 6.35. The FPGA behavior follows the CPU trend, within permille level differences. As in the case of robustness to large cluster splitting, the small differences observed between FPGA and CPU clustering algorithms do not show any tendency to enlarge when the number of SPs is increased, neither for ghost rate, nor for clone rate or tracking efficiency.

6.6 Summary of the performance studies

The performance studies summarized in this chapter aim at characterizing the FPGA–clustering algorithm in detail. The cluster reconstruction efficiency and quality show that none of the approximations applied (see Chap. 3) has a significant impact on the reconstruction of the cluster itself, and thus on the accurate measurement of the particle hit position, with marginal residual inefficiencies due to hit merging. This is also reflected in higher–level quantities, such as tracking efficiency, impact parameter and momentum resolutions, and primary vertex efficiency and resolution. In–depth comparisons with a full–fledged CPU–based clustering algorithm show that the two algorithms have indistinguishable responses for different types of decay modes, including b–hadrons, s–hadrons and LFU golden channels such as $B^0 \to K^{*0} e^+ e^-$ and $B^0 \to K^{*0} \gamma$. The search for significant differences between CPU and FPGA implementations returns null results, also when taking into account that the simulation might not perfectly mimic the detector real behavior, both in terms of cluster size and occupancy. This is not obvious given the profound differences between the two clustering implementations running on very different architectures. The choices made during the design and implementation of FPGA algorithm do not impact the performance quality and allow the cluster reconstruction to be run with a high degree of parallelization, reaching offline–like reconstruction quality, while running online, on the spare space of detector readout cards. Given the indistinguishable performance of the FPGA implementation relative to the CPU one, the algorithm presented in this thesis has been chosen as the baseline option for LHCb Run 3 data taking. The commissioning steps required to fully validate the clustering firmware response are detailed in the next chapter.

Chapter 7

Commissioning and final thoughts

This chapter describes the commissioning steps that led to a fully functional clustering firmware, deployed on VELO TELL40 cards. Starting from the outcome of the integration process, where the clustering firmware was added to the VELO DAQ chain, several tests have been performed, gradually getting closer to data taking conditions. During this test campaign, fixes and refinements have been applied to the firmware. The conclusions and final considerations of the work detailed in this thesis are also presented.

7.1 Introduction

The successful firmware and software integration process, together with the detailed performance studies, has demonstrated that the VELO clustering reconstruction can be moved from the HLT farm to a preprocessing stage, being deployed within the detector readout cards, with benefits for the entire DAQ chain, both in terms of throughput and bandwidth. Despite the fact that the attention and close scrutiny put in identifying possible bugs and pitfalls did not show any criticality, the tests performed are based on the expected behavior of the detector and its front–end and back–end electronics, modeled in detail within the LHCb MC simulation. Therefore, in–depth stress tests of the clustering architecture, using the first real collision data, are of utmost importance to validate simulation–based test outcomes. This test campaign goes by the name of commissioning and took, as for the other subdetectors and subsystems, the entire 2022 data taking period. It involved hardware tests, starting from a single board up to the full–scale production system, software–based monitoring tests, both at low and high level throughout the DAQ chain, and firmware tests, validating the clustering response at bit level. Commissioning also allowed the overall electrical power reduction to be measured, while moving clustering from the GPU to the FPGA architecture, which is especially relevant for future HEP experiments. The validation of new diagnostic tools built directly within the firmware, exploiting real data, was also performed during the commissioning period. Such tools include per–bunch luminosity monitoring, of which a preliminary calibration is provided. Moreover, the 2022 commissioning period also led to the first physics results, including invariant mass peaks, a hint of which is presented in this chapter.

7.2 MiniDAQ tests

The LHCb upgrade required the design and construction of a new detector and the related electronics, as described in Sect. 2. To facilitate the commissioning process, the LHCb Online team made available a development platform to test the monitoring and control of the new subdetector front–end electronics, together with the new readout electronics and the TFC system. This development platform goes by the name of MiniDAQ [88]. A MiniDAQ consists of a dual–socket server[1] equipped with PCIe40 cards and comes with the software suite to control and monitor the hardware. The MiniDAQ used for cluster firmware tests has two PCIe40 cards, one acting as a TELL40 and one as a SODIN and SOL40 card. The two cards are used to control and collect data from a complete VELO module. The module, together with its optical and power board (see Sect. 2.3), is powered by a bench power supply while the sensors and the ASICs are passively cooled, instead of using active CO_2 cooling. A first set of tests was performed using the digital test pulse mode, where individual pixels can be turned on. This type of test allows the behavior of the DAQ chain to be studied even in the absence of real *pp* collisions, knowing the exact position of each active pixel. The digital test–pulse test was performed to compare the position of the reconstructed clusters with the input SPs, under different input rate and occupancy conditions. A similar type of test was performed using the internal data generator of the TELL40 card, where data do not come from the VELO module itself but are generated within a specific firmware block in the TELL40 card. The internal generator can be set up in one of two modes: a pseudo–random mode generates SPs spread randomly over the entire module and a slow–control mode where the user defines a set of SPs to be used and writes them in appropriate firmware registers, using the ECS. These modes allow tests of the firmware even without a front–end connected and working, giving maximum flexibility to the user. During these tests, both with the generator and the actual module, no issue was identified.

7.3 First tests on VELO TELL40s

Having tested that the VELO firmware was behaving fine within the MiniDAQ setup, we moved to the actual TELL40s within which we loaded the complete VELO firmware, including the clustering. During one of the first tests involving the entire DAQ chain, from the detector to the event building stage, we loaded a pixel hitmap onto one of the VELO modules, using digital test pulses. The pixel hitmap is shown in the upper part of Fig. 7.1. The corresponding SPs were then sent out of the detector to the TELL40 readout card, where the cluster reconstruction occurs. Data are then propagated to the event builder and finally saved to a file. The content of the file was then decoded using the algorithms described in Sect. 5.4.4 and the positions of the reconstructed clusters were plotted (bottom image of Fig. 7.1) and compared to input SP hitmaps. This was the first key test that demonstrated the proper functioning of the FPGA–based VELO clustering both at the firmware level, being able to reconstruct the correct clusters starting from the input SPs, and at the DAQ higher level, where data containing clusters were correctly propagated, built, and decoded, using the production–grade software that would be used for data taking.

[1] The MiniDAQ server is a DELL® PowerEdge® R740, equipped with two Intel® Xeon® Silver 4210.

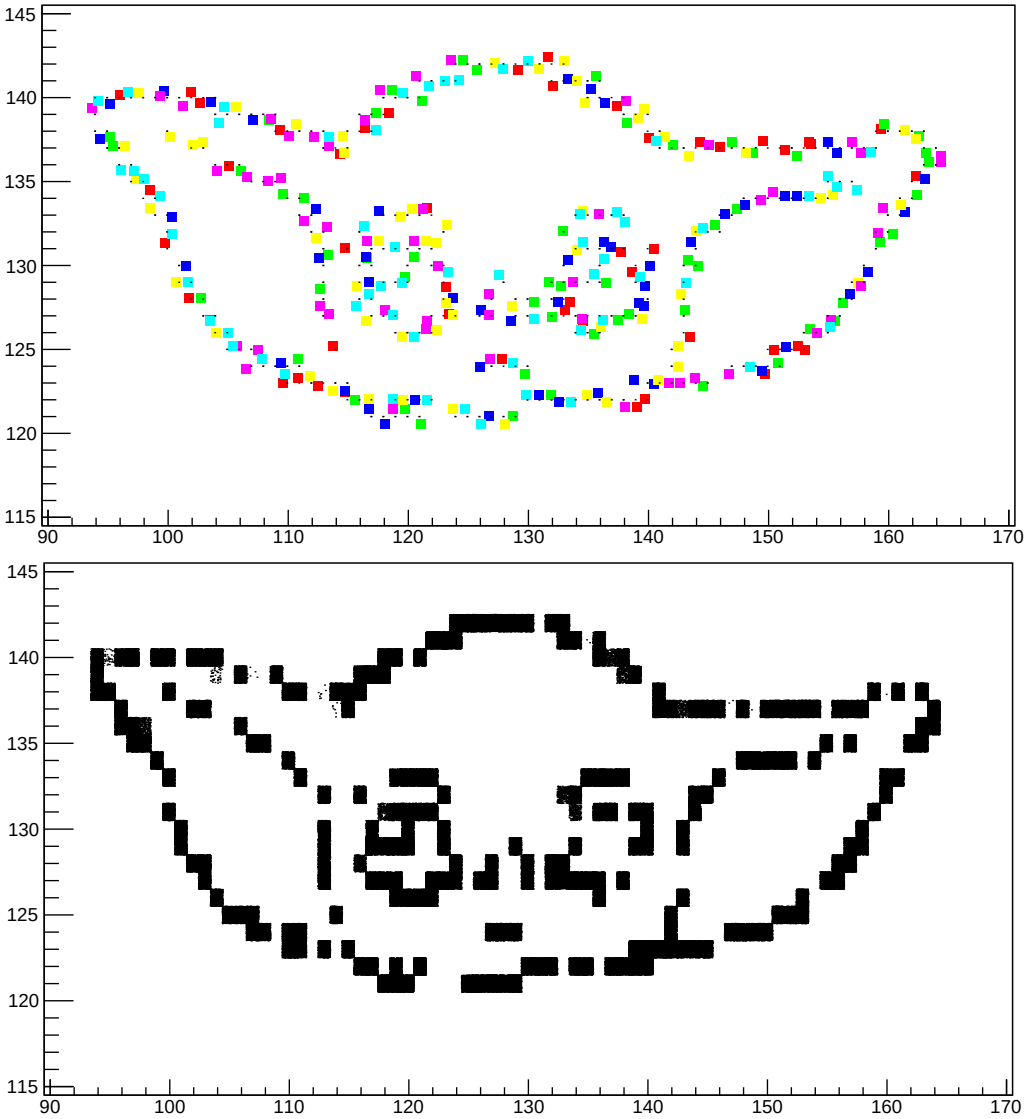

Figure 7.1: Outcome of one of the first tests performed on VELO TELL40 cards, showing (top) an image of the active pixels loaded onto one VELO half–module and sent as input to the firmware cluster reconstruction, where each of the six different colors indicates the origin ASIC, and (bottom) the two–dimensional positions of the reconstructed clusters. The x and y axes are the column and row of the VELO ASIC, respectively, in units of pixels.

7.4 WinCC slow control

As mentioned in Sect. 2.9, one of the key components of the data acquisition architecture is the experiment control system. The system is implemented in different firmware and software layers. The layer with which the user interacts is the WinCC OA SCADA[2] that is interfaced with the front–end electronics via a message broker called DIM [89]. WinCC is used for all the slow control related tasks, including the DAQ monitoring at the firmware level. Having integrated the clustering architecture within the DAQ chain, the WinCC VELO project needed to be

[2]SCADA stands for Supervisory Control And Data Acquisition.

updated to include the necessary monitoring components for the clustering part. To facilitate firmware monitoring and debugging during data taking, the readings of the relevant registers within the firmware have been structured in a panel, as shown in Fig. 7.2. Within the panel, the user can follow the data flow from the input side, where SPs are received, through the cluster reconstruction steps, to the output. The panel shows the current occupancy levels of several FiFos within the firmware (see Fig. 4.2), together with enable and error LEDs and SP overflow counters.

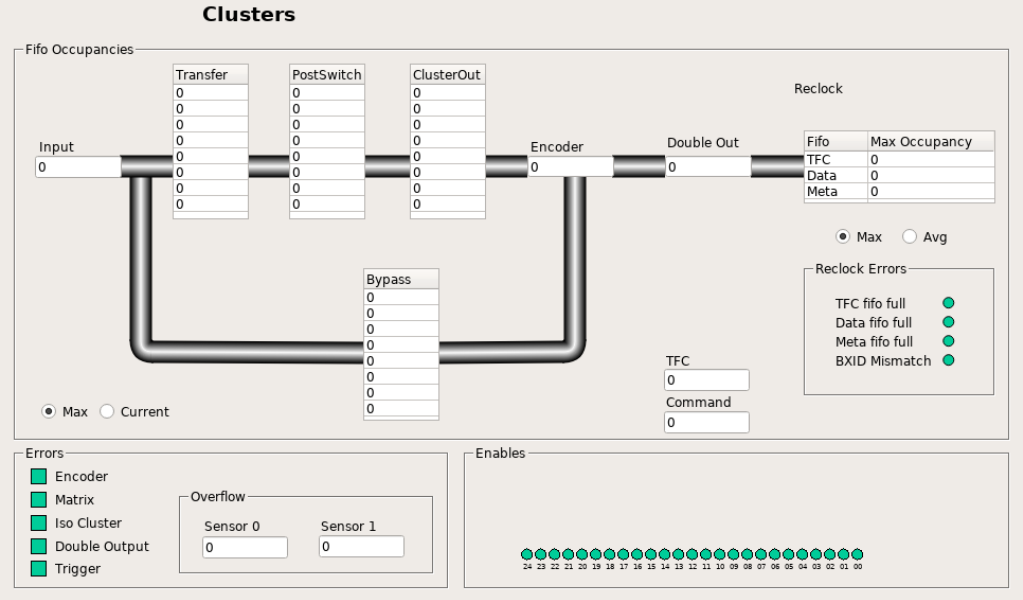

Figure 7.2: Cluster monitoring panel within the VELO WinCC project.

7.5 Online monitoring

Being able to quickly identify possible detector issues or software bugs during data taking is key for the success of the LHCb experimental physics program. For this reason, a small subset of the data is reconstructed on the LHCb Online computing farm. As an output of the reconstruction, a set of plots and histograms is created, allowing detector experts to identify possible pitfalls. The core software that performs the just–mentioned operations is called *Monet* [90], which is a python–based web application that allows data quality plots to be visualized.

During the commissioning stage the VELO part of Monet processing has been updated to take into account the different output format that contains clusters instead of SPs. This update required the replacement of SP–decoding and software clustering algorithms with cluster–decoding algorithms (see Sect. 5.4.4). After some local tests, the updated Monet configuration was then deployed to the production system. Examples of plots produced by Monet are shown in Fig. 7.3. In the left plot of Fig. 7.3, the cluster hitmap of VELO module 20 is represented in the absolute LHCb coordinate system. Brighter regions of the hitmap are closer to the beam pipe and thus have more clusters. In the right plot of Fig. 7.3, the distribution of the primary vertex position along the x axis, measured using only C–side VELO modules, is shown. Given that during this run, the VELO was opened, with A–side sitting at +28 mm and C–side sitting

at −28 mm, the PV position measured with the C–side only is centered at +28 mm, due to the reconstruction software assuming that the VELO was closed at 0 mm.

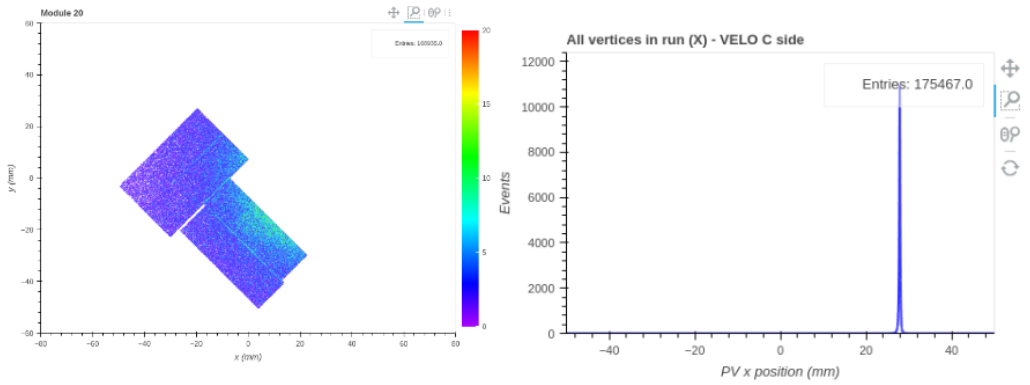

Figure 7.3: Two plots available on Monet during data taking, showing (left) the cluster hitmap on one VELO module and (right) the distribution of primary vertices horizontal positions. The average PV position around −28 mm is due to the VELO open position.

7.6 Debugging

After the first tests on VELO TELL40s using test pulses on the detector side and having set up all the necessary tools to monitor the DAQ behavior, both at low and high level, the firmware, including cluster reconstruction, was tested during *pp* collisions under different conditions. To stress test the system, the input data rate to the TELL40 cards was increased from a few kHz up to 20 MHz and an issue was observed around 18 MHz where the DAQ system was going into error. At the same time, the WinCC cluster panel (see Fig. 7.2) was showing an error at the encoder level. To debug this issue, the firmware was recompiled adding a SignalTap[3] instance to monitor the behavior of several signals inside the FPGA. The debug firmware was then loaded into the MiniDAQ setup, and using the TELL40 internal generator, the issue was reproduced under controlled and known conditions. Using SignalTap, it was possible to identify the matrix (see Sect. 4.4.8) as the faulty component, as data were received correctly and in sync by the matrices, but after the matrix cluster reconstruction, data from different events were mixed. An overflowing FiFo, within each matrix, was causing the loss of EndEvent signals, which are then responsible for the error in the encoder that identifies data from different events being mixed. Being able to reproduce the issue using the internal firmware generator allowed us to recreate and study it in QuestaSim® simulation. Using the simulation tool, the appropriate fix was identified and implemented. Having fixed this issue, the clustering firmware did not show any input–rate–related issues during the following data taking.

When analyzing the cluster data collected during the first collisions, we observed some peculiar behaviors clearly related to a bug in the firmware. Under specific conditions, duplicated clusters within the same data bank and clusters with the wrong ASIC ID[4] were observed. As in the

[3]SignalTap is a FPGA debugging tool that captures and displays user–specified signals within a firmware design. Similarly to an oscilloscope, the developer can specify one or multiple trigger conditions based on a set of signals and, if the conditions are matched, the signals within one or more firmware components are shown while the FPGA is running.

[4]The two most significant bits of the cluster column bit–field (see Sect. 4.2.1) can be used to identify the ASIC

high–throughput issue, these problems were firstly analyzed and then solved using the SignalTap debugging tool. Duplicated clusters were due to input data not being correctly bypassed (see Sect. 4.4.10) and being interpreted as SPs. Changing the bypass conditions accordingly solved the issue. The wrong ASIC ID issue, however, was not caused by the clustering itself, but was due to a data mishandling within the router (see Sect. 5.3.1), which was promptly fixed.

7.7 Double output checks

As described in Sect. 4.4.13, the clustering firmware is equipped with a double–output mode that, when activated, allows both reconstructed clusters and input SPs to be output. This mode was used during the first period of Run 3 data taking for debugging purposes, in order to ensure that the reconstructed clusters are the correct ones, given the input SPs. A typical VELO data bank collected when the double–output mode is enabled is the following:

```
Bank:  0x101B (subsystem:  2 'VELO_A', number:  27)
Size:  56B (48B payload)
Type:  97 'VPDoubleOutput'
Version:  4
0x0000 | 94 D2 02 30 74 12 03 20 00 00 00 00 00 00 00 00 7C 02 03
00 88 E2 02 00 C4 A2 8A 60 00 00 00 00
0x0020 | 28 27 06 80 04 E8 05 80 20 A9 05 80 02 6C 15 80 |
```

The sourceID of the bank (`0x101B`) indicates that the data come from VELO side A, in particular module 13 and data flow number one. The size of the bank (`56B`) shows the number of bytes that make up the bank itself. The type (`97`) and the version (`4`) allows the format version and the type of data contained in the bank to be identified. After the metadata information, the actual data are present. Excluding empty clusters, the decoded data are the following:

```
Clusters                    SPs
3002D294 (row=165, col=45)  80062728 (row=156, col=48)
20031274 (row=157, col=49)  8005E804 (row=160, col=46)
0003027C (row=159, col=48)  8005A920 (row=164, col=44)
0002E288 (row=162, col=46)  80156C02 (row=176, col=170)
608AA2C4 (row=177, col=170)
```

To distinguish between clusters and SPs when the double output mode is active, the most significant bit of each SP is set to one. Figure 7.4 shows a graphical representation of active pixels (green squares) and the corresponding reconstructed clusters (red crosses).
The collected data have been analyzed with the `VPRetinaClusterDoubleOutputChecker` algorithm described in Sect. 5.4.4. One million events have been analyzed for a total of about 1.4 billion clusters. No errors or discrepancies have been observed between the firmware behavior and its software emulation.

within a sensor from which a cluster or SP comes from. Given that there are three ASICs per sensor, these two bits can take values 0, 1 and 2. A value of 3 was observed, clearly indicating an issue in the firmware.

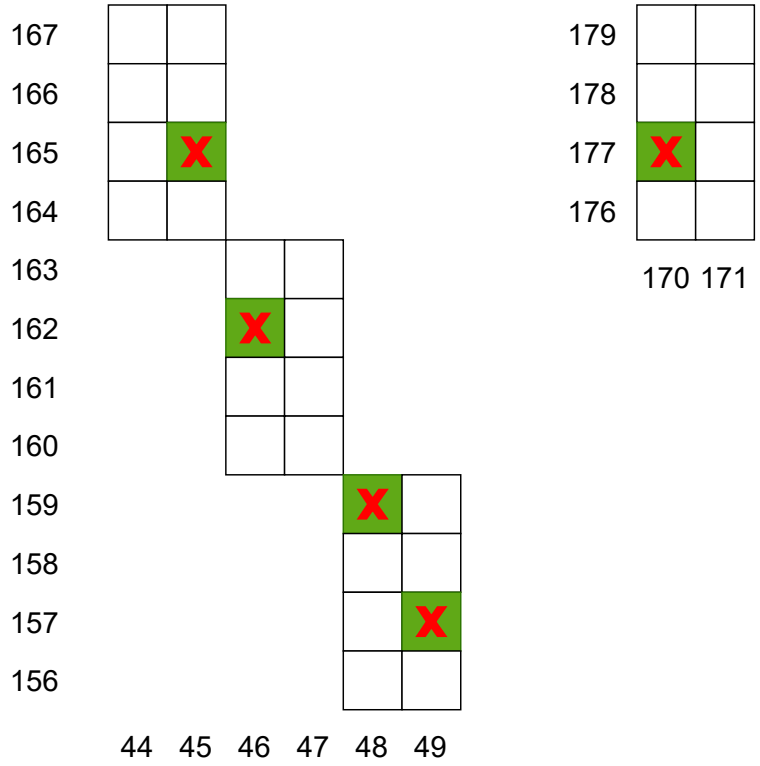

Figure 7.4: Graphical representation of the position of SPs and clusters from the VELO data bank above. Active pixels are drawn as green squares and reconstructed clusters are represented as red crosses. The rows and columns values are also shown for reference.

7.8 Power consumption

During Run 2 operations LHCb power demand was around 5.5 MW. The experiment power consumption is expected to increase by roughly 1.5 MW during Run 3 due to the increased data processing power, to be compared to the five–fold increase in luminosity. Given that LHCb is currently proposing to build a second upgrade [27], which would lead to an order of magnitude increase in data rate and hence to a significant increase in the processing power required, a power consumption optimization strategy is paramount for the future of LHCb.

Moving part of the reconstruction sequence to a preprocessing stage performed on FPGAs might be a valuable option in terms of power consumption. A hint for this already comes from the VELO cluster reconstruction in Run 3. Adding clustering to the FPGA processing increases its power consumption, which has been quantified by measuring the current on the 0.9 V power input line. Figure 7.5 shows the FPGA power consumption of all VELO TELL40s at the nominal event rate of 30 MHz. The measurement was performed with and without cluster processing. The average power consumption of a TELL40 FPGA when processing an event rate of 30 MHz with the readout firmware only is 6.1 W; this increases to a total of 8.6 W for the full firmware, including the clustering block. Given the 52 VELO TELL40 cards, performing clustering within the readout requires roughly 130 W. The same measurements have been repeated for different values of the input event rate, as shown in Fig. 7.6, where the average FPGA power consumption increases slowly as a function of the input rate. For comparison, we estimate the power needed

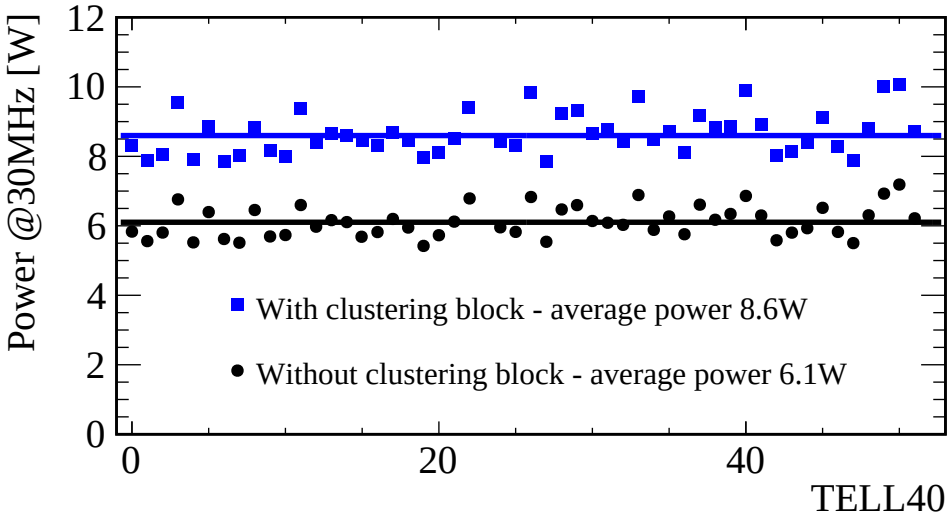

Figure 7.5: Power consumption of individual VELO TELL40 FPGAs processing data at an event input rate of 30 MHz. The average value over all 52 FPGAs is also superimposed with an horizontal line. Measurements using the firmware without the cluster finding block (outputting SPs instead of clusters) are also reported for comparison.

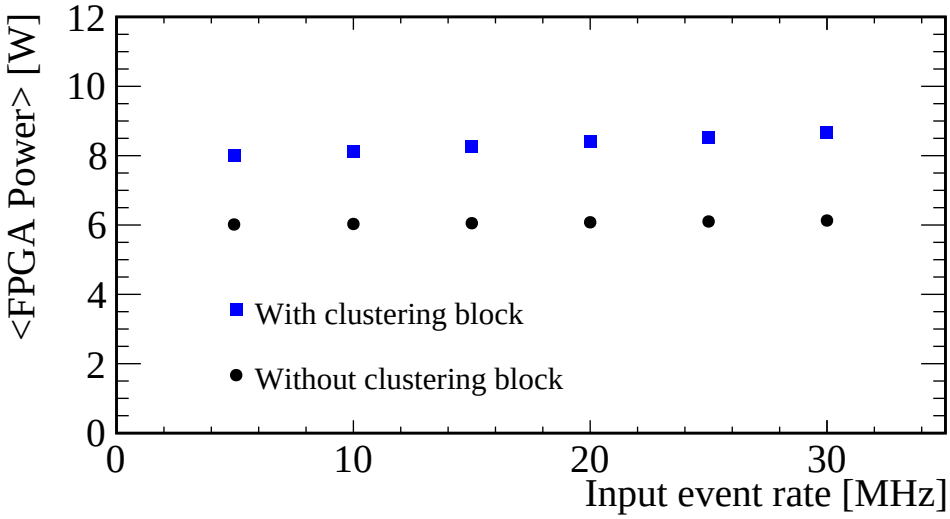

Figure 7.6: Average power consumption over all 52 FPGAs as a function of the input event rate. Measurements using the firmware without the cluster finding block (outputting SPs instead of clusters) are also reported for comparison.

to perform clustering on GPUs during the HLT1 reconstruction. Knowing the GPU power usage (230 W), the number of GPUs (236) required to process a 30 MHz input event rate and the fraction of time spent in cluster reconstruction (11%), running cluster reconstruction on GPUs requires about 6 kW. Therefore, reconstructing clusters on readout FPGAs requires $\mathcal{O}$(50x) less power than performing the same task on GPUs. This is in agreement with the measurements presented in Ref. [91].

FPGA-based clustering is just a first step towards track reconstruction being performed at a preprocessing stage, the benefits of which are discussed in depth in Chap. 1. However, this result clearly suggests that running part of the reconstruction on FPGAs at early DAQ stages has a substantial benefit in terms of reduction in power consumption.

7.9 VELO closure with clusters

As described in Sect. 2.3, the two sides of the VELO detector are movable and can be retracted to protect the detector during beam injection, or in general during non–stable beam conditions. Being able to move the two halves allows also the detector to be centered around the beam, fill by fill. The VELO closing procedure has been updated from the Run 2 one and validated with the new detector to be able to safely close the detector, starting from the so–called garage position at a 54 mm opening[5]. The main steps performed during closing are the following:

- once LHC declares the stable beam condition, the VELO high voltage is turned on and the monitoring task, devoted to the closing of the VELO, starts to receive data and reconstruct tracks and primary vertices out of them;
- the primary vertex positions are measured with respect to each half of the detector and used to fill histograms, from which the position of the detector relative to the beam is determined;
- the closing software, implemented in WinCC, takes as input the position of the beam and a set of safety criteria, such as the beam condition monitors [92], to decide whether and how to move the VELO halves, both in x and y;
- the motion is done in steps and at each step the beam conditions and the PV position are checked before moving on. At each point, if safety conditions are not met or if PVs are not provided correctly, the closing software asks the operator to open the VELO.

After the first VELO closures performed with the TELL40 cards loaded with the SP firmware, we moved on with the first closure using the firmware with clustering included. As a safety measure, before the closure two data sets were collected, one with the SP–only firmware and one with the cluster one, to compare the PV position distributions, based on which the closing steps are performed[6]. The distributions are shown in Fig. 7.7 and since no significant differences were observed between SP– and cluster–based PV distributions, we moved on with the closure. The closure was done in steps corresponding to (54, 40, 30, 26, 12, 10, 6, 4, 2, 1 and 0) mm opening values. At each position, the PV position distributions were checked, together with the safety criteria, before moving on to the next steps. The entire closure procedure was carried out without issues or problems.

[5]The opening is defined as the distance along x between modules on opposite sides of the VELO.

[6]Between the two data sets collection both the TELL40 firmware and the monitoring software were updated in order to produce and decode clusters instead of SPs.

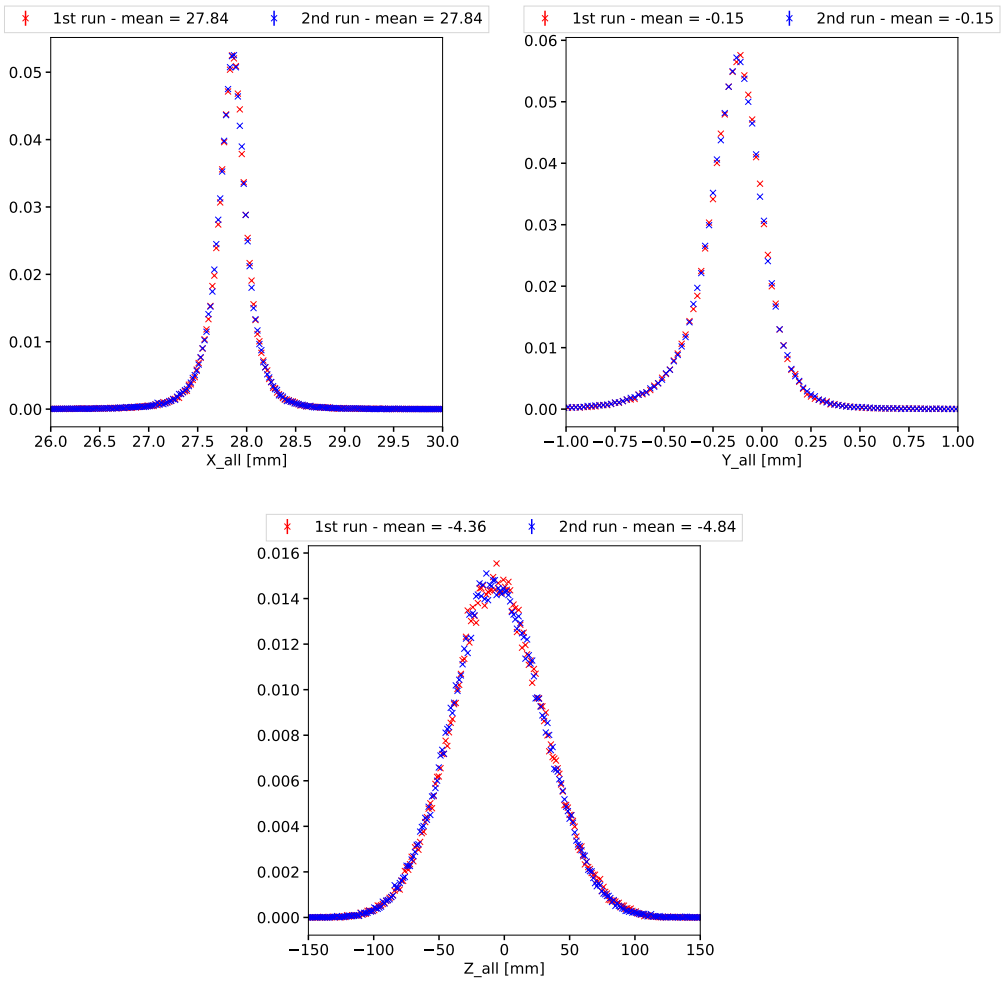

Figure 7.7: PV position distributions along the x (top–left), y (top–right) and z (bottom) axes, comparing data collected with the SP–only firmware (red) with cluster ones (blue).

7.10 New opportunities with VELO clusters in real time

With cluster reconstruction occurring inside VELO TELL40 readout cards online, at 30 MHz, it becomes possible to accumulate useful statistics in the firmware itself. The main advantages of this type of monitoring are the very high rates at which statistics are accumulated, while being accessible via ECS, without disturbing data acquisition, even when HLT1 is not running. This low–level monitoring can provide diagnostics, such as the identification of misbehaving regions of the detector and the measurement of spillover. It can also be a useful source of data for beam parameter measurements, such as the beam line position, which can be monitored at the full 30 MHz bunch crossing rate. An example of cluster–based firmware–level monitoring that has already been implemented and used during Run 3 data taking is for instance instantaneous luminosity measurement in real time, complementary to the Plume subdetector [93] built specifically for this purpose. Moreover, being performed in real time, the clustering can provide a measurement of the instantaneous luminosity per bunch which is affected by the individual bunch intensities and, to a lesser extent, by the beam emittances. These quantities being challenging

to track, a bunch by bunch measurement of μ can provide an interesting feedback to the LHC, directly from the low–level firmware cluster counting. In this section the key components of the firmware luminosity monitor are summarized.

The working principle of luminosity monitoring within the VELO TELL40 cards is the linear relation between the mean number of reconstructed clusters and the luminosity at which data are collected. Detailed simulation studies have been performed to determine the best way to measure luminosity in terms of linearity with respect to the number of reconstructed clusters, stability with respect to the position of the luminous region, and taking into account that the required firmware resources need to be limited in order to fit within the FPGA chip. The implemented solution involves the addition of extra logic, placed at the end of the cluster reconstruction chain, to select and count the number of reconstructed clusters within specific regions of each VELO module (see Fig. 7.8), using a minimal amount of both logic and memory resources. Cluster counts are performed independently for each of the four possible bunch–crossing[7] types in order to subtract the detector noise and background. Furthermore, changes to both ECS–related firmware and software components were applied to be able to read, archive, and reset cluster counts. In order to provide luminosity measurements, cluster counts need to be calibrated. The

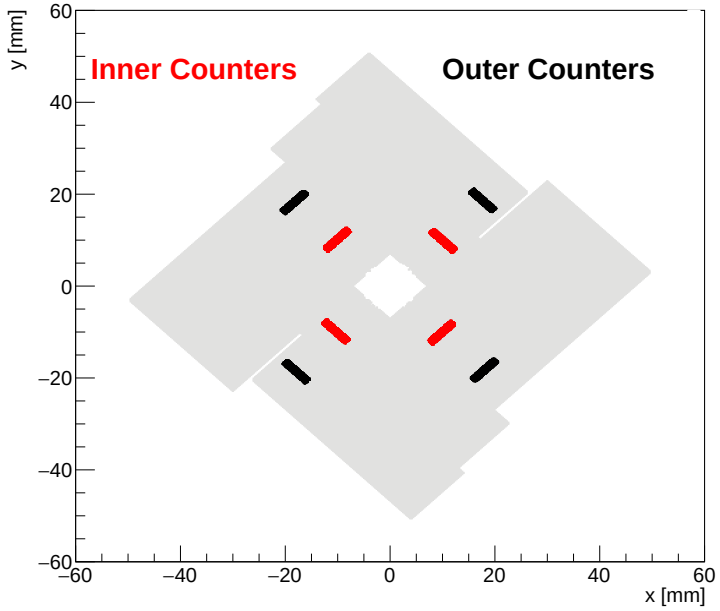

Figure 7.8: Positions of outer counters (black regions) used for luminosity monitoring and inner counters (red) used for beam position monitoring within a VELO station.

calibration procedure has been performed during a Van der Meer scan [94], where the beams have been gradually shifted one with respect to the other, as shown in Fig. 7.9, and, as a consequence, the number of reconstructed clusters changes accordingly. A Van der Meer scan offers the ideal conditions under which an observable, linearly proportional to the instantaneous luminosity, can be calibrated and normalized under well–defined and controlled beam settings.

To obtain a first preliminary determination of the cross section, σ, defined as the pro-

[7]Depending on the LHC filling scheme four bunch–crossing types are possible: beam–beam where both beams are colliding at the interaction point, beam–empty and empty–beam where only one beam is present and empty–empty where none of the beams are present.

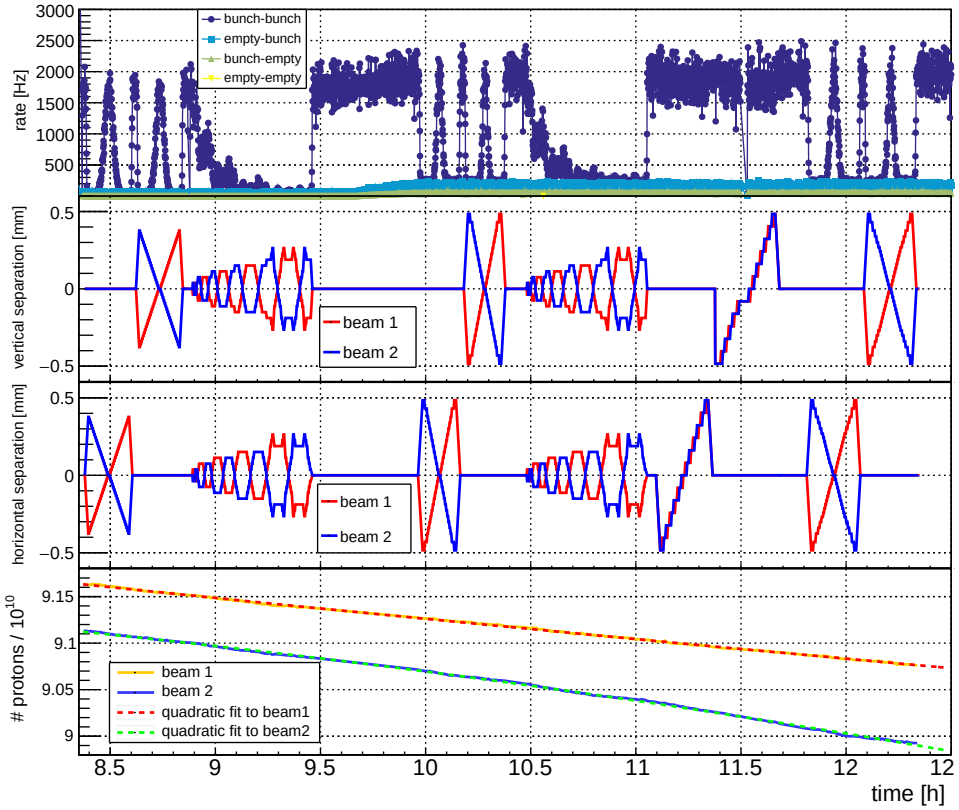

Figure 7.9: (top) Rate of cluster counts for each bunch–crossing type. (middle plots) Horizontal and vertical separations of beams. (bottom) Number of protons per bunch. All quantities are plotted as a function time during the Van der Meer scan.

portionality constant between the luminosity and the number of reconstructed clusters, the background–subtracted and bunch–population–normalized cluster rate is plotted as a function of the beam separation and fitted with a Gaussian function. This fit is performed both for one–dimensional scans, as shown in Fig. 7.10, where beams are shifted in the horizontal and vertical planes separately, and for the two–dimensional scan, as shown in Fig. 7.11, where beams are displaced in the horizontal and vertical planes at the same time. From the fits the standard deviations, σ_x and σ_y, are obtained, and therefore the cross section, as shown in Eq. 7.1 [95, 96],

$$\sigma = \frac{2\pi n_{clusters}(\Delta_x = 0, \Delta_y = 0)\sigma_x\sigma_y}{N_1 N_2}, \tag{7.1}$$

where $n_{clusters}(\Delta_x = 0, \Delta_y = 0)$ is the number of clusters reconstructed in the null horizontal and vertical beam separations and N_1 and N_2 are the numbers of protons per bunch for beam 1 and 2, respectively. The cross section is derived from Eq. 7.1, for both 1D and 2D Van der Meer scans, leading to the following values:

$$\sigma_{1D} = (215.4 \pm 0.6) \cdot 10^{-24}\ \text{cm}^2$$

$$\sigma_{2D} = (218.6 \pm 0.4) \cdot 10^{-24}\ \text{cm}^2$$

The low–level monitoring work described in this section was done together with Daniele Passaro, whom I trained. More details about this topic can be found in Daniele's Master thesis [97].

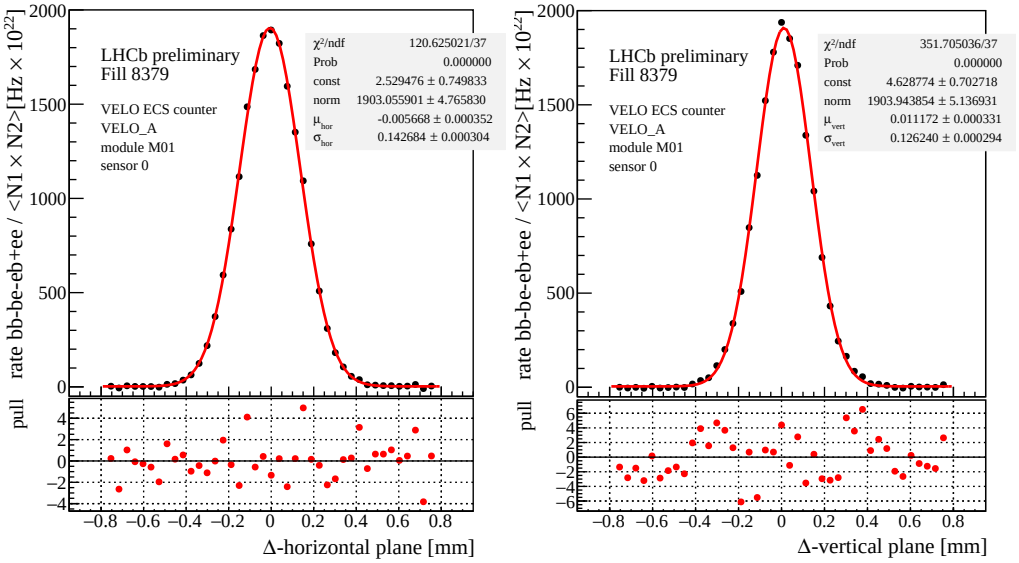

Figure 7.10: Cluster count rate with background being subtracted and normalized with the number of protons per bunch plotted as a function of (left) the horizontal and (right) the vertical beam separations. The normalized cluster rate is fitted with a Gaussian function. Pulls are shown.

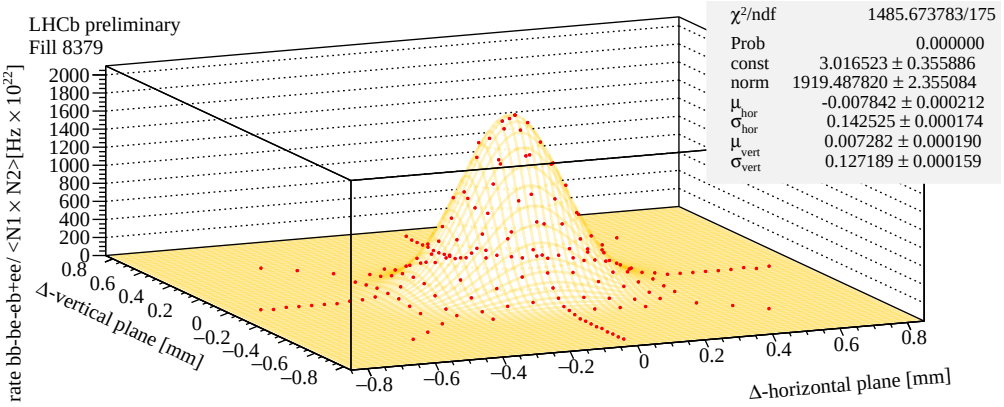

Figure 7.11: Cluster count rate with background being subtracted and normalized with the number of protons per bunch plotted as a function of the horizontal and vertical beam separations. The distribution is fitted with a two–dimensional Gaussian function.

7.11 μ scan

During the first data taking period of LHCb Run 3 commissioning, several tests were performed to study the behavior of the detector, the DAQ chain, and the reconstruction sequence. One of the key tests was the so called μ scan[8], where the luminosity was constantly increased, bringing the beams closer to each other [98]. This is a key test for both the VELO detector and its

[8] μ is defined as the average number of visible pp interactions per bunch crossing, and it is a measurable value. ν is defined as the average number of pp interactions per bunch crossing and it is used mainly for simulation purposes. The relation between μ and ν is $\mu \simeq 0.7\nu$.

DAQ, including clustering, where the number of detector hits increases with luminosity, and so does the number of SPs per event at the clustering input. During the test, the behavior of the clustering was monitored in terms of both FiFo occupancies and number of reconstructed clusters. No DAQ error was triggered due to the clustering. Furthermore, the μ scan represented a unique opportunity to test the linearity of the cluster counts as a function of the increasing luminosity. The firmware counters described in Sect. 7.10 were used to monitor the firmware behavior. As the μ and thus the luminosity increased the number of reconstructed clusters also increased linearly, as shown in Fig. 7.12. This is a key indication that neither the detector nor

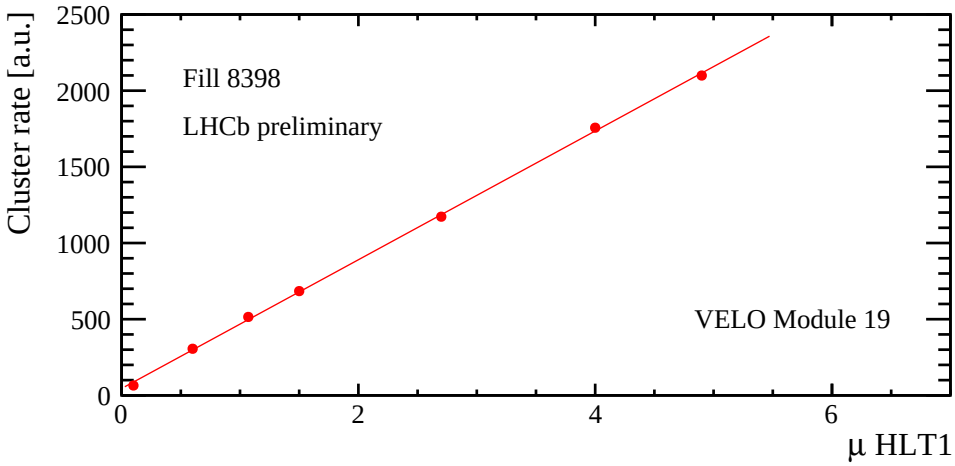

Figure 7.12: Cluster rate as a function of the number of visible interactions per bunch crossing, μ. μ is measured within HLT1 processing using a set of counters that take as input the information of several subdetectors, such as the number of VELO tracks, the number of vertices, the number of SciFi clusters, the ECAL transverse energy and the number of Muon hits.

the firmware were approaching a saturate response as luminosity increased. The linear behavior also shows the capability of the clustering firmware to measure the luminosity in a wide range of μ values. Firmware cluster counts, read periodically via ECS, are currently being added to the list of observables monitored during data taking to precisely measure the instantaneous luminosity, representing an additional and complementary source of low–level monitoring.

7.12 First physics results

As described in Sect. 3, the clustering algorithm has several parameters that can be tuned to improve its performance. During the design and optimization phases, these parameters were set based on the analysis of several official LHCb simulation samples. Therefore, the first real data collected are of paramount importance to check if the actual detector response is well described by the simulation, and thus verifying if the clustering algorithm parameters are correctly tuned. Figure 7.13 shows a comparison between the simulated number of isolated and neighbor clusters per event as a function of the sensor pair and the real values, measured during the μ scan (see Sect. 7.11), at different luminosities. As it can be observed, the simulation reproduces the actual detector behavior quite accurately, in particular for modules in the forward and backward regions of the VELO detector. Near the nominal interaction region (sensor pair $\sim$ 30), the main data–simulation discrepancy is observed, with more clusters from SPs with neighbor

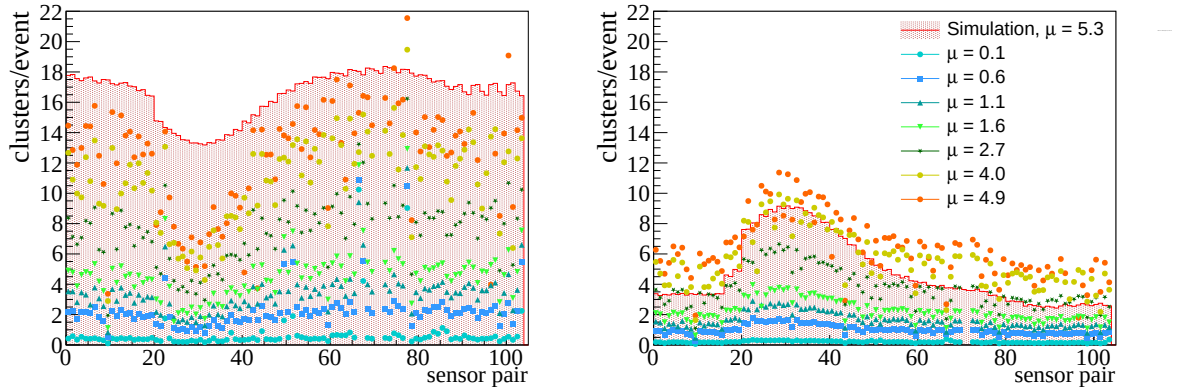

Figure 7.13: Data and simulation distributions of (left) the number of isolated and (right) non–isolated clusters per event as a function of the sensor pair number. Simulated distributions are taken at μ=5.3, whereas data distributions are within the 0.1 and 4.9 μ range.

being reconstructed with respect to clusters resolved from isolated SPs. Figure 7.14 shows a comparison between simulated and real data cluster size distributions, for VELO modules 6 and 14, upstream ($z = -212.50$ mm) and on ($z = -12.50$ mm) the nominal interaction point, respectively. As with the cluster–number comparison, real data have been collected during the μ scan (see Sect. 7.11), at different luminosities. Real data show larger clusters with respect to simulation, with this behavior visible both close and far away from the nominal interaction point. Given the maximum cluster size limited to 3×3 pixel, it is important to periodically monitor these low–level quantities, to ensure a good reconstruction quality. It is worth mentioning that these data have been collected with a brand new detector that itself needs to be commissioned and optimized. Therefore, we might expect these distributions to change as detector thresholds and other parameters are optimized in the future. Given the good simulation–data matching, the algorithm parameters seem to have been appropriately set for the first period of LHCb Run 3 data taking, leaving space for further fine tuning that will improve

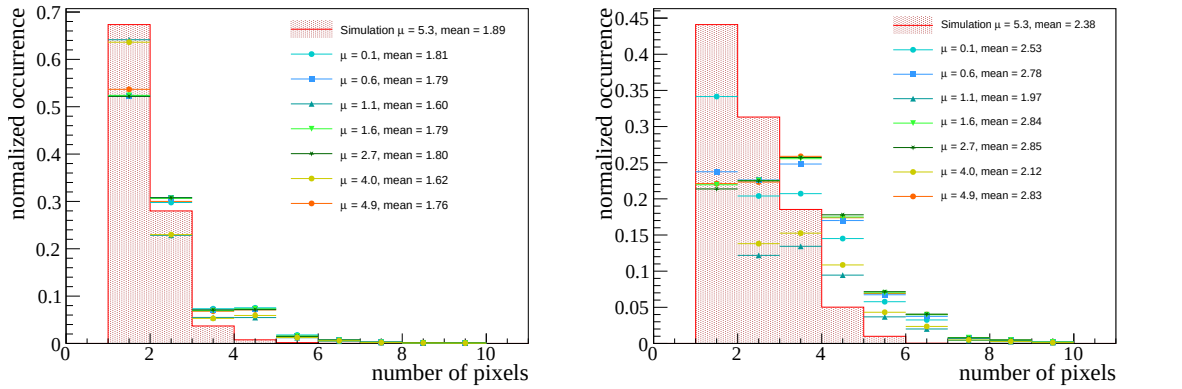

Figure 7.14: Comparisons of data and simulation cluster size distributions in pixel units for (left) module 6 ($z = -212.50$ mm) and (right) module 14 ($z = -12.50$ mm). Simulated distributions are taken at μ=5.3, whereas data distributions are within the 0.1 and 4.9 μ range.

the algorithm response. The fact that the clustering algorithm was configured correctly is also visible in the plots shown below. In addition to firmware, software, and structure tests, during the Run 3 commissioning the first physics results were also obtained. Figure 7.15 shows mass peak plots of $D^{0+} \to D^0\pi^+$, $D^0 \to K^-\pi^+$ and $J/\psi \to \mu^+\mu^-$ decay channels, respectively[9]. To produce these plots, a preliminary configuration of HLT2 was used. This configuration will be fine–tuned during the rest of Run 3 data taking, to obtain the best reconstruction quality. All

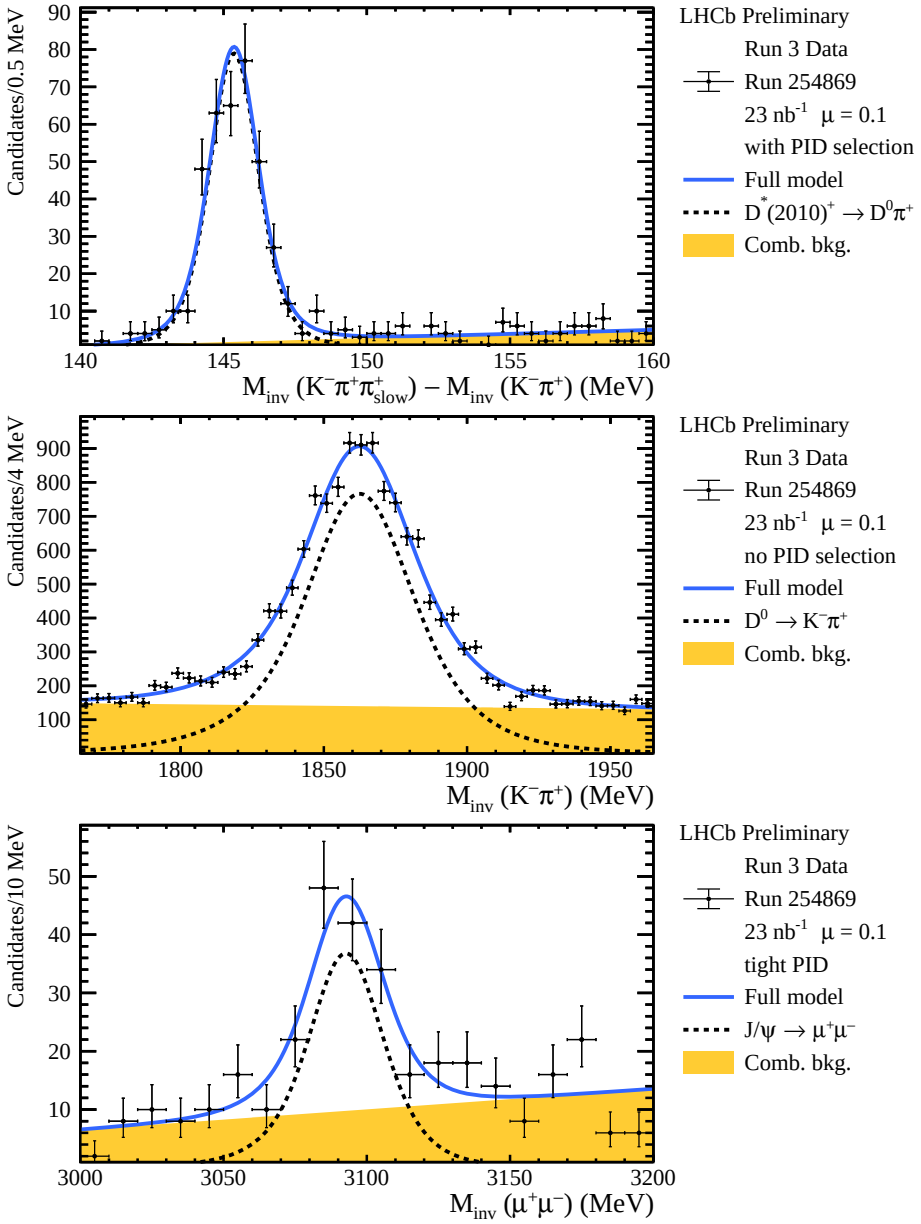

Figure 7.15: Invariant mass peaks of $D^{0+} \to D^0\pi^+$, $D^0 \to K^-\pi^+$ and $J/\psi \to \mu^+\mu^-$. A preliminary configuration of the HLT2 tracking sequence is used. See Ref. [99].

[9]Since these are the very first mass plots obtained with a completely new detector and reconstruction software, which are currently under commissioning, they are presented here only for illustrative purposes.

subdetectors, from tracking to particle identification, DAQ systems, event building, and HLT processing, have been shown to properly function in the upgraded LHCb environment, both in terms of hardware and software. Apart from some tuning and optimization, that will be applied as data are collected and analyzed, the new Upgraded LHCb experiment has demonstrated to be capable of taking good–quality data, with a completely renewed detector, a new trigger–less DAQ and a reconstruction, leveraging the heterogeneous–computing model.

7.13 Conclusions

This thesis describes a new two–dimensional cluster–finder algorithm capable of reconstructing particle hits on a pixel detector in excess of 30 MHz input event rate, at the LHCb Run 3 instantaneous luminosity of 2×10^{33} cm^{-2}s^{-1}. The algorithm is designed to run online on FPGA–based readout cards in order to take full advantage of the high degree of flexibility and parallelization provided by modern FPGA chips. The corresponding firmware architecture is implemented and optimized so that it requires a rather small amount of FPGA resources, both in terms of logic and memory, while providing an excellent reconstruction quality. Despite being of rather general applicability, both the algorithm and the firmware architecture are tailored and optimized for the silicon pixel vertex detector, VELO, of the LHCb Upgrade experiment.

The FPGA VELO clustering is the first processing stage of the LHCb "Retina" project, which aims at reconstructing tracks in real time, as a embedded system within the detector readout cards. It is therefore characterized by the same high throughput and low latency capabilities as all other "Retina" components. The first firmware prototype, written in VHDL language, was developed and tested within the LHCb–Pisa laboratory, and it has shown to be capable of reconstructing clusters on–the–fly at 30 MHz input event rate, when fed with LHCb–official simulated input data from the most occupied VELO module. Precise and extensive tests have been performed to ensure that the tracking performance of this novel system is indistinguishable from a full–fledged software implementation of the clustering algorithm. This was made possible by developing a C++ version of the algorithm, accurately emulating the firmware behavior, and fully integrated within the LHCb simulation software stack.

The sparse–matrix technique adopted in the FPGA VELO clustering proved successful in handling large detectors (order of 40 million pixels) with a modest amount of logic and memory resources. In order to process data from a VELO module, the clustering architecture, including isolation flagging, requires 132500 ALMs and 300 M20K memories, accounting for 31% of logic and 11% of M20K memory resources of an Arria–10 GX 1150 FPGA chip, respectively. Exploiting these resources, the firmware is capable of processing events with up to an average of 32 SPs per VELO half–module, using a 350 MHz clock, which corresponds to a processing speed of 0.96 GSpixel/s (or 7.7 GPixel/s). This is a significant advancement over the previous state–of–the–art in HEP, where the best performing two–dimensional cluster–finding system implemented in FPGAs, that can be found in literature, was developed for the Fast TracKer processor with an eye to the trigger upgrade of the ATLAS experiment [81]. This system can run at about 100 KHz input event rate, with the deployment of about 4 parallel firmware copies, processing about 12.5 MPixel/s each.

Given its high throughput, the low amount of resources needed, and the excellent tracking performance, the LHCb collaboration decided to adopt this new architecture as the default option for Run 3 data taking. As a consequence, the clustering firmware has been integrated within the VELO readout firmware, at no extra cost, and extensively tested, first on a test setup and then on the production system. Moreover, the corresponding software decoding and the

default reconstruction sequence have been adapted, both for GPU–based HLT1 and CPU–based HLT2 architectures, to decode directly VELO clusters, instead of reconstructing them in software. The algorithm and the corresponding firmware implementation have been fully commissioned during the first data–taking period of LHCb Run 3, including bit–level comparisons between the actual firmware output and the expected clusters to be reconstructed, given the input data. A series of VELO closures have been successfully performed using the clustering firmware and related decoding and reconstruction, obtaining the first Run 3 physics results with the FPGA VELO clustering performed within the readout chain. The good simulation–data matching also suggests that the algorithm parameters have been appropriately set for the LHCb Run 3 data taking, however there is still headroom for further fine tuning that might improve the algorithm response and may be necessary based on the real data, as the detector response is fully tuned and commissioned. During Run 3 commissioning period, the VELO DAQ system, including clustering, was stress tested both at high input event rate and at high luminosity, without showing any issue.

Performing cluster preprocessing on readout boards allows HLT1 to accept a 11% higher rate of events[10], as the ready–made hit coordinates accelerate the track reconstruction, also yielding the additional benefit of about 14% reduction in data flow. Being implemented and optimized for FPGAs, the clustering architecture also leads to a significant drop in electrical power consumption, as the VELO clustering implementation on FPGAs requires $\mathcal{O}(50\text{x})$ less power than the GPU one. This is especially relevant for future HEP experiments, where the demand for greener solutions will increasingly become an essential aspect of their design. Improvements in both output bandwidth and HLT throughput are also beneficial for the entire data acquisition and reconstruction chain, and will be increasingly so in the future when, instead of just reconstructing clusters, heterogeneous systems will be able to produce more complex objects, such as track primitives, in real time. The on–the–fly accessibility of all hits (or clusters) on a complex detector such as the VELO also allows the additional potential offered by FPGA reconstruction in real time to be exploited. In fact, the implementation of a FPGA–based system for real–time luminosity measurement and beam monitoring at the LHCb interaction point, transparently embedded within the detector readout cards, seems feasible already during the current Run 3, thanks to the innovative work done in this thesis. Additional functionalities for monitoring the operational status of the VELO sensors can be also implemented, addressing spillover effects and sensor detection efficiency.

In conclusion, the work described in this thesis is a significant first step towards a new computing paradigm in HEP, where offloading highly repetitive and parallelizable tasks to dedicated accelerators is paramount to sustain the ever–increasing trigger and data acquisition requirements. As recently demonstrated by several industrial applications, exploiting the flexibility and computational power of heterogeneous computing infrastructures is one of the most promising solutions to tackle high data volume–related issues, at low cost. This is key for the physics program of the upcoming Runs, at even higher instantaneous luminosities, where more and more data need to be collected, handled, and processed at the unprecedented speed of 30 MHz, to measure with ever increasing precision many statistically limited observables, probing the current understanding of fundamental interactions.

The work described in the thesis is the content of a paper that has been published in the IEEE Transactions on Nuclear Science journal [100].

[10] This value is an underestimation of the actual HLT1 throughput gain as the GPU–based clustering implementation expects SPs with the isolation flag, whereas the FPGA–based clustering computes it by itself.

Appendix A

Field Programmable Gate Array

Field Programmable Gate Arrays (FPGAs) are semiconductor integrated circuits in which the electrical functionality is customizable at configuration time. FPGAs represent a key component in the heterogeneous computing paradigm and complement other technologies such as CPUs and GPUs. FPGA devices target real–time computations where mixed precisions and different vectorization factors are present. Another key feature of FPGA devices is the ability to drive many and different types of I/O interfaces from PCIe busses to optical transceivers. All these features make FPGA devices particularly appealing for HEP purposes where digital hardware tailored to specific requirements is needed. FPGA chips are typically mounted on PCBs that allow electrical communication between the chip and other components such as oscillators, optical transceivers, PCIe busses, memory modules, sensors, programming headers, and power delivery circuitry. The ensemble of all the components takes the name of FPGA–based board. Figure A.1 shows the architecture of a modern FPGA chip (the Intel® Arria® 10 FPGA is taken as an example), which has the following main components:

- Core Fabric Logic is made of a high number of interconnected logic units programmable by the developer;
- M20K memory blocks can store up to 20 kbits in dedicated cells spread over the FPGA;
- Digital Signal Processing (DSP) blocks implement arithmetic logic such as floating point sums and multiplications;
- PLLs generate clocks with desired frequency from an external clock source. PLLs are used to drive both transceivers and internal logic;
- Transceiver Channels, Transceivers, and PCIe Hard IPs (Intellectual Properties) are located at the edges of chip to ease data exchange with other components mounted on the board.

FPGA core logic

The core component of the FPGA logic is the Logic Array Block (LAB). LABs are composed of basic building blocks named Adaptive Logic Modules (ALMs). Each ALM can be configured to implement logic functions, arithmetic functions, and register functions. LABs can be configured as memory LABs, in which case they take the name of MLABs. As shown in Fig. A.2, LABs are surrounded by routing wires and switches, which are used to exchange signals between them.

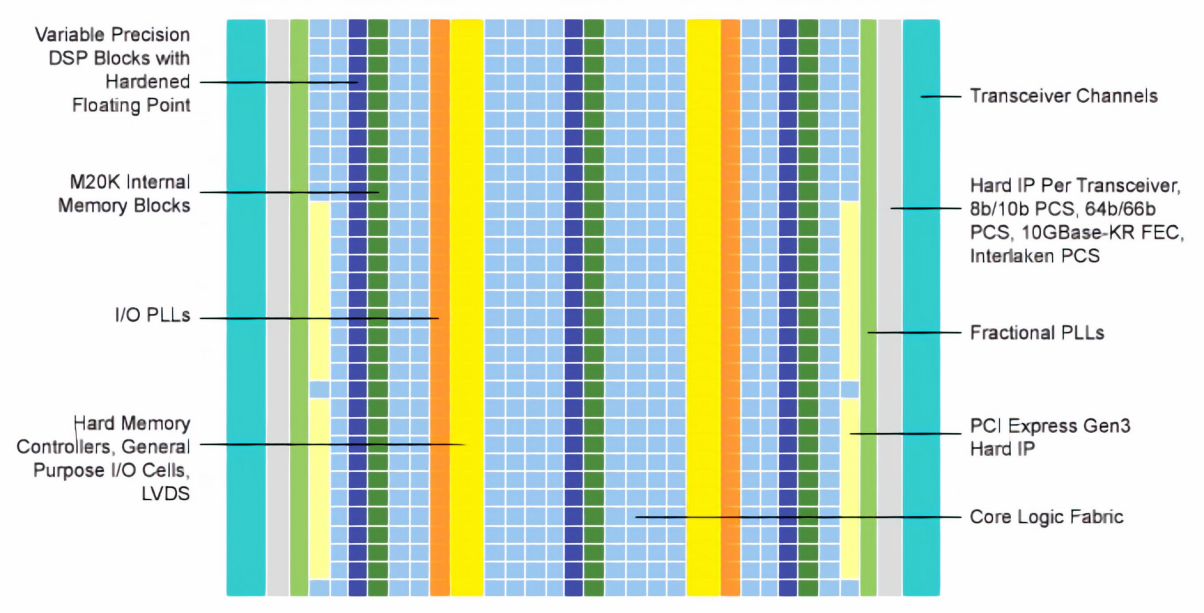

Figure A.1: Intel® Arria® 10 FPGA architecture [101].

Different types of interconnect are present in the FPGA chip. Local and Direct Link Interconnects allow fast communication between close by LABs and within ALMs of the same LAB. Row and column interconnects with variable speed and length allow data exchange throughout the entire chip. Fig. A.3 shows the main constituents of an ALM. Each ALM contains two combinational adaptive LUTs (ALUTs) and four registers. The two combinational (ALUTs) can be fed with up to eight inputs allowing the implementation of various logic functions. ALMs can be operated in the following modes:

- Normal mode, where two logic functions can be implemented using the eight ALM inputs. Several combinations of two functions with independent inputs can be implemented, such as two four–input, five–input, and six–input functions. Input signals can also be shared between LUTs to create functions with a higher number of inputs;
- Extended LUT mode, where a single logic function can be created with up to seven independent inputs;
- Arithmetic mode, where two sets of four–input LUTs are used along with two dedicated full adders. The two adders can add the output of two four–input logic functions. A carry chain is implemented between adders;
- Shared arithmetic mode, where a three–input sum can be implemented in the ALM, configuring the ALM with four four–input LUTs.

Regardless of the operating mode, the ALM output can drive the local, row, and column routing resources, directly from the LUT or adder output or using the registered values.

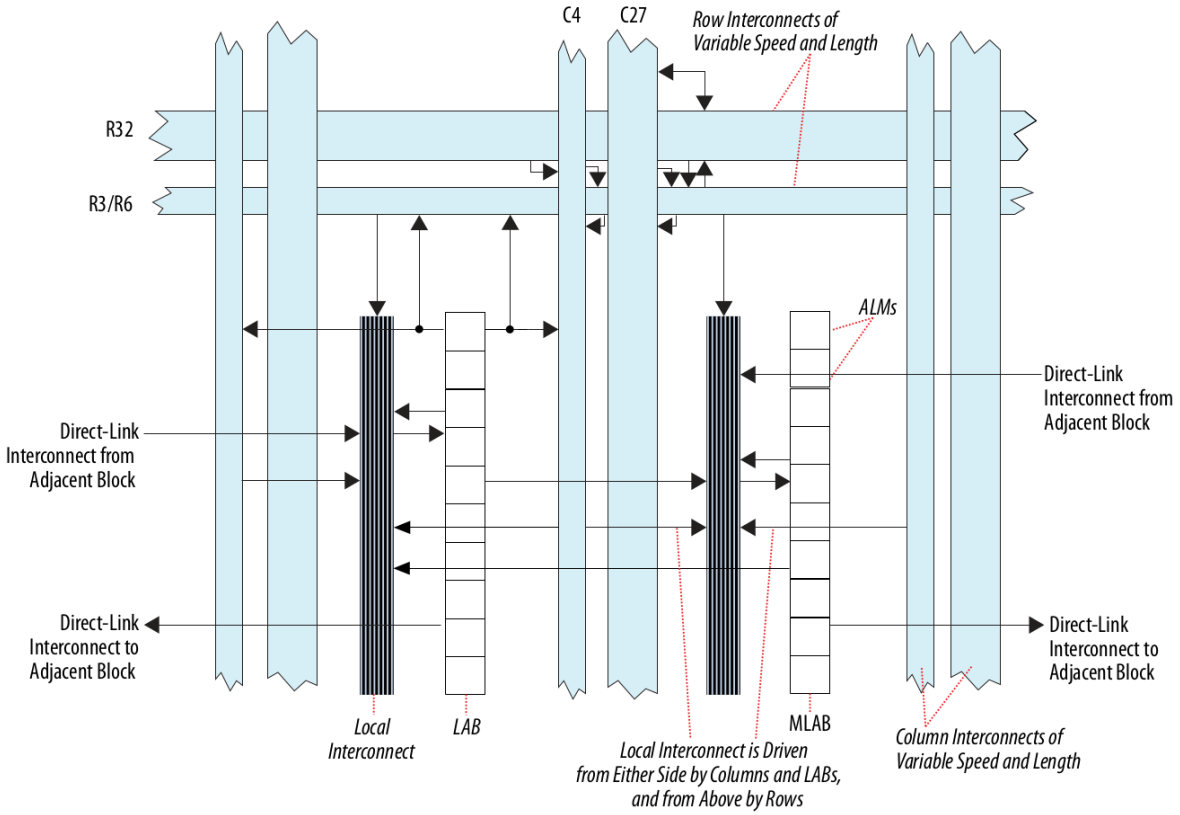

Figure A.2: Intel® Arria® 10 FPGA LAB and MLAB structure with the LAB interconnects [102].

Memory and DSP

Besides LABs that can be configured as memory elements, FPGAs have dedicated embedded memory blocks known as M20K blocks that can contain up to 20 kb. M20K blocks are ideal for larger memory arrays while still providing a large number of independent ports, whereas MLABs are ideal for wide and shallow memory arrays. Embedded memory supports the following operation modes:

- single–port RAM, where only a read or a write operation can be performed at a time;
- dual–port RAM, where any combination of two port read–write operations can be performed, both with single or dual clocks;
- shift register, where the memory block is used as a shift–register block to save logic cells and routing resources;
- ROM, where the content of the read–only memory is written to it at compilation time;
- FiFo, where the memory block is used as a buffer with the first data coming out being the first data written to it. Both single– and dual–clock FiFos can be implemented.

Alongside dedicated memory blocks, digital signal processing (DSP) blocks are also present within the FPGA chip. These blocks are specifically designed to perform fixed–point and floating–point arithmetic operations, such as additions and multiplications.

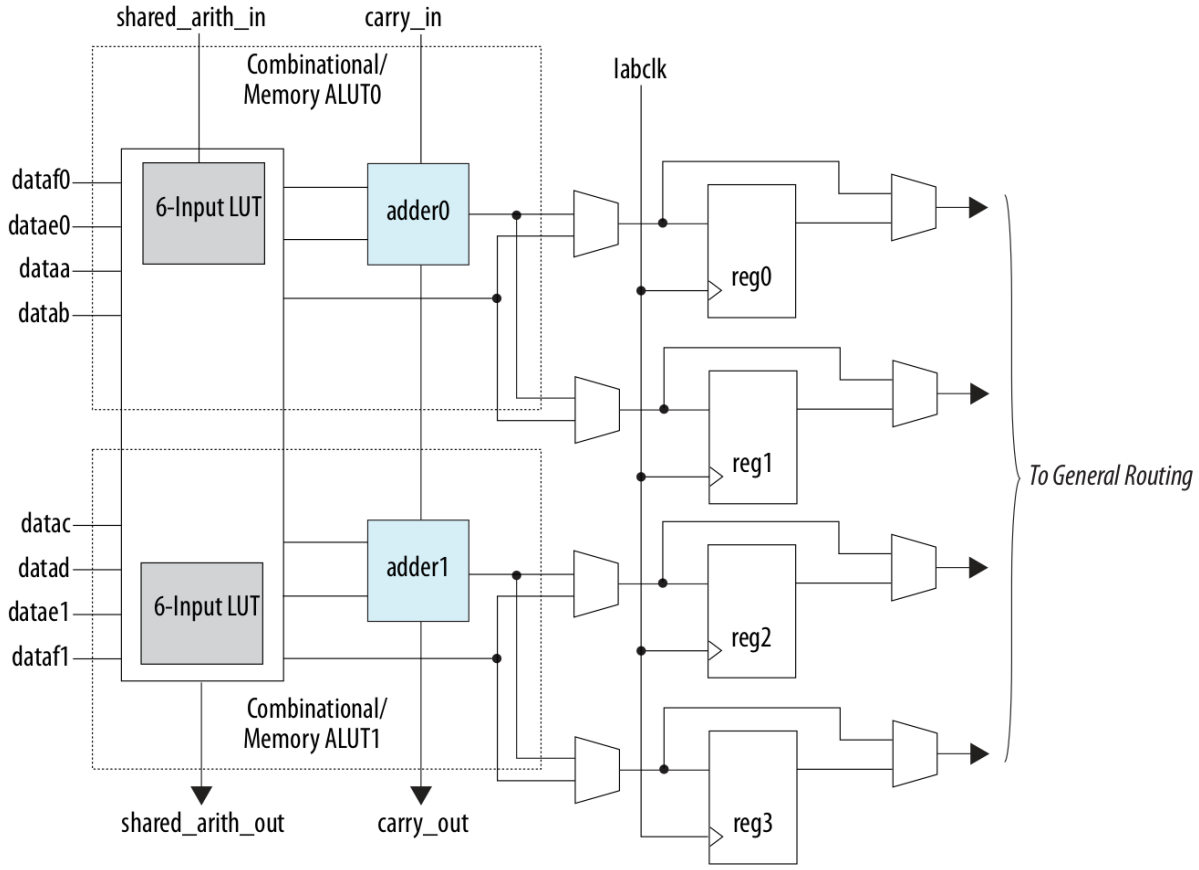

Figure A.3: Intel® Arria® 10 ALM High–Level Block Diagram [102].

Clocks and I/O

Clock resources within FPGAs are organized in hierarchical clock networks and phase–locked loops (PLLs). Global clock networks provide low–skew clock sources for ALMs, DSPs, embedded memory, and PLLs, whereas regional clock networks serve as clock sources for logic contained within limited chip regions. Periphery clock networks provide clock signals mainly for I/O purposes. FPGA chips also contain configurable PLLs that can be used to create clock signals with user–specific frequencies, reducing the number of oscillators required on the board. As mentioned above, a key feature of FPGAs is the ability to drive many and different I/O interfaces. I/O–specific locations within the FPGA chip are organized in banks, where the I/O standard of each bank can be specified by the user. Each bank contains several lanes connected to I/O pins, I/O PLLs, Serializer/Deserializer (SERDES) circuitry, and memory controllers. FPGAs also contain hard IP that are cores with a fixed physical implementation, targeting a specific I/O standard. Examples of commonly used hard IPs are PCIe and Ethernet standards.

Arria 10 resources

The characteristics described in the previous sections are common for the vast majority of FPGA chips available on the market. The specific FPGA chip mounted on LHCb data acquisition cards, which is the target device for the development of the architecture described in this thesis,

belongs to the Intel® Arria® 10 family. In particular, it is a 10AX115S4F45E2SG chip. The part number can be decoded as follows:

- 10A = family (Arria 10);
- X = family variant (with up to 17.4 Gb/s transceivers);
- 115 = device type (427200 ALMs);
- S = transceiver count (72);
- 4 = transceiver speed grade;
- F = package type (FineLine BGA);
- 45 = package code (1932 pins);
- E = operating temperature (0°C - 100°C);
- 2 = fabric speed grade;
- S = power profile (standard power);
- G = package material (RoHS).

Firmware design

The typical firmware development workflow for a FPGA is shown in Fig. A.4. Starting from a series of specifications and requirements, the firmware is developed using a hardware description language (HDL). HDL languages are used to describe the behavior of the actual hardware using a high–level language. The algorithm described in this thesis has been implemented using the VHDL language. Unlike CPU/GPU programming, FPGA programming involves the reconfiguration of the electrical paths inside the chip. As a consequence, firmware design requires the knowledge of the FPGA chip architecture and its components. A key step during the development process is the simulation of the code behavior. For this purpose, the code is wrapped within a test bench that provides all the required input signals. Firmware simulation allows the verification of the behavior of the different firmware components by looking at individual signals propagating inside the FPGA. The simulation tool used during this thesis is Siemens EDA® QuestaSim® software. Once the simulated behavior matches the desired one, the firmware can be compiled. Compilation consists of two main steps: the analysis&synthesis and place&route. During analysis and synthesis, the compiler examines the logical completeness and consistency of the code, while checking syntax errors. It also synthesizes and optimizes the design using algorithms to minimize gate count, remove redundant logic, and use the device architecture as efficiently as possible. Starting from the synthesis output, place&route operations are performed. The firmware components are "fitted" within the FPGA chip using the available resources, selecting the appropriate interconnection paths between them. The place and route procedure is carried out having as constraints the clock frequencies at which the logic operates. As an output of the fitting procedure, the logic and memory resources required by the firmware are measured. The result of the fitting procedure is then analyzed using the time analysis tools that check if timing violations are present. The compiler raises a timing violation to warn that it has not been able to fit the design inside the FPGA while following the constraints given by the user for the

specific use case (see Appendix B). The compiler used during the firmware development for this thesis is Intel® Quartus Prime® software. As an output of the compilation, a bitstream code is created that can be loaded into the FPGA to configure all required ALMs and interconnect paths. The behavior of the firmware on the actual hardware can then be tested.

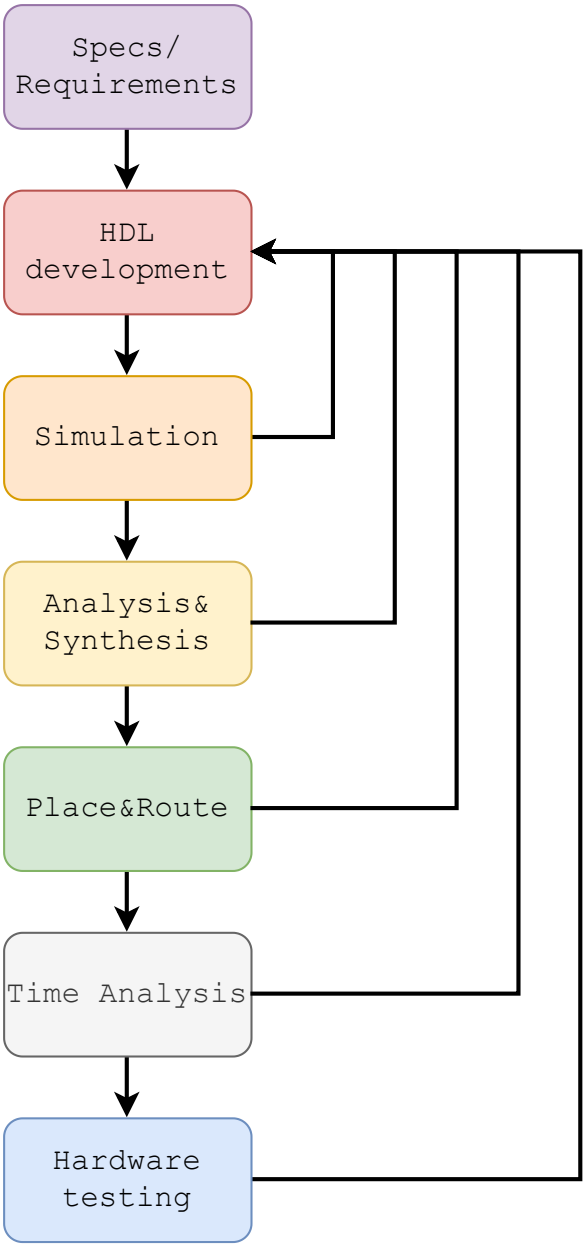

Figure A.4: Firmware development workflow.

Appendix B

Firmware timing violations

Timing analysis is one of the key steps in FPGA firmware design, as described in Appendix A, and one of the main differences with respect to CPU or GPU programming [103]. Two timing–related constraints usually need to be fulfilled by a firmware design: throughput and latency. Throughput is the average rate at which valid output data are delivered per clock cycle, whereas latency is the amount of time required to output valid data after the input arrives. At the bare minimum, timing usually refers to the time it takes a signal to propagate from one flip–flop, through some combinational logic[1], to the next flip–flop, as shown in Fig. B.1. The key concept behind timing analysis is that flip–flops and combinatorial logic do not have a zero–time response:

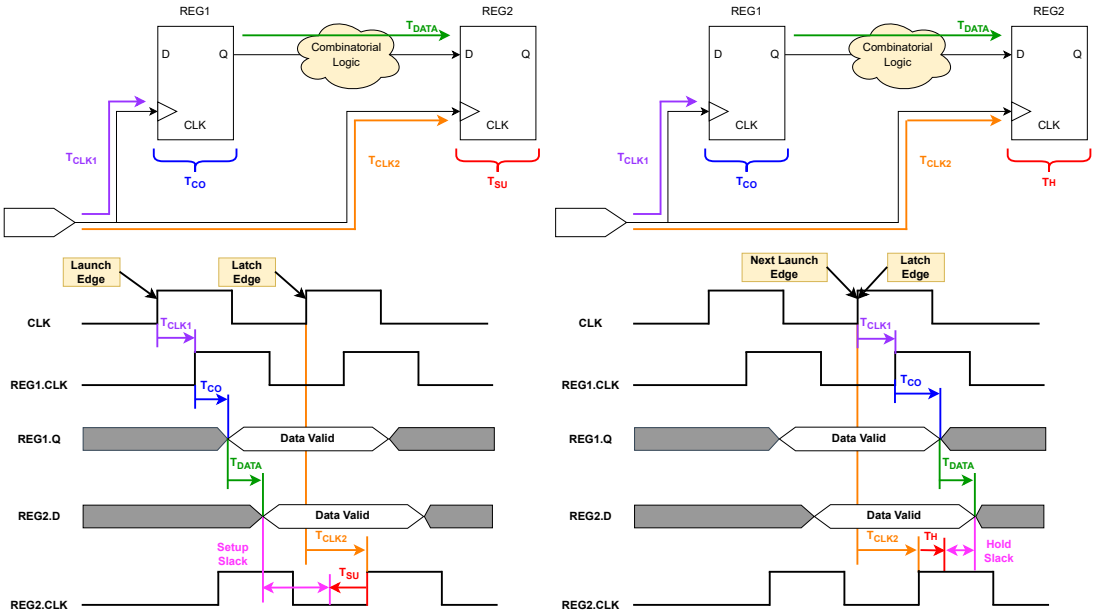

Figure B.1: Block and timing diagrams showing (left) setup slack and (right) hold slack. T_{CO} is the clock to out time defined as the time it takes for a signal to propagate out of a flip–flop after a clock edge. T_{SU} is the setup time defined as the minimum time required for data to be valid before the latch edge for successful latching. T_H is the hold time defined as the minimum time required for data to be valid after the latch edge for successful latching.

[1]Here combinatorial logic stays for operations done on data without storing them.

once receiving a launch edge[2] a flip–flop needs a certain amount of time to output its content, this time is known as clock to out time T_{CO}. In the same way, combinatorial logic requires some time to output the result value, given the input, mainly due to the switching time of transistors. Moreover, data to be stored in a flip–flop need to be valid some time before (setup time T_{SU}) and to remain valid some time after (hold time T_H) the latch edge[3]. These considerations need to be taken into account when designing a FPGA firmware in order to have stable and reliable signal propagation within the FPGA chip.

The left timing diagram of Fig. B.1 shows the clock and data propagation between two flip–flops running at the same clock speed, with emphasis on the setup time. The clock signal requires T_{CLK1} time to propagate from the source to the REG1 source flip–flop. After a T_{CO} time, data stored in the flip–flop are sent to the output and received by the REG2 destination flip–flop after T_{DATA} time required to perform the combinatorial logic operations. As described above, data out of REG1 need to be valid a T_{SU} time before the arrival of the latch edge, taking into account the T_{CLK2} time needed for the clock propagation from the source to REG2. This leads to a time window known as setup slack defined as the difference between the time data are required to arrive at REG2 and the time they actually arrive. The setup slack needs to be a positive value to guarantee correct signal handling. If the setup slack is negative, there is a so–called timing violation. The same type of argument can be made for the hold slack (right diagram of Fig. B.1), since data need to remain valid at least T_H time after the arrival of the latch edge. The setup slack depends on the clock frequency, whereas the hold slack does not.

Given the design and the user–specified timing constraints, such as the clock frequencies, the firmware compiler tries to place the components inside the chip and to route signals between them so that timings are correct and no timing violation is present. If the firmware is not well designed or optimized enough to match the clock frequency constraint, the compiler reports the failing paths for which the setup or the hold slack are not positive, providing also tools to understand the cause of the timing violation [104]. An example of the compiler timing violation report is shown in Fig. B.2. Fixing timing violations can be a hard job, especially if done during advanced firmware development stages. It might require rewriting entire parts of the code, for example, moving from combinatorial to staged logic or introducing pipelining [105, 106].

Slow 900mV 100C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	TELL40_0|i_pll_velo_250|iopll_0|outclk0	-0.541	-219.797
2	TELL40_0|i_pll_velo_350|iopll_0|outclk0	-0.193	-0.297
3	pcie_top|pcie_0|\gen_pcie:qsys_pcie|pcie|coreclkout	-0.157	-1.626
4	pcie_top|pcie_1|\gen_pcie:qsys_pcie|pcie|coreclkout	-0.128	-0.152

Slow 900mV 100C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	lli_inst|\multiLink_gen_loop:2:multiLink..._a10_0|g_xcvr_native_insts[2]|rx_pma_clk	-0.153	-0.153
2	lli_inst|\multiLink_gen_loop:3:multiLink..._a10_0|g_xcvr_native_insts[2]|rx_pma_clk	-0.049	-0.049

Figure B.2: Quartus® report on (top) setup and (bottom) hold timing violations. There is one entry for each clock source showing the worst case slack value and the total negative slack (TNS) as the sum of the timing violations of all failing paths running at that clock.

[2]The launch edge is defined as the clock edge that activates or launches the source register.
[3]The latch edge is defined as the clock edge that latches the data into the destination register.

Appendix C

Bias in PV reconstruction

During the comparison studies between CPU and FPGA clustering algorithms on the cluster and track reconstruction performance (see Sect. 6), a bias on the reconstructed value of the primary vertex z position (z_{PV}) was observed. The bias is limited to the low–efficiency and low–resolution region $-60 < z_{\mathrm{PV}} < 0\,\mathrm{mm}$, as shown in the left plot of Fig. C.1. The size of the bias was about $-3\,\mu\mathrm{m}$, corresponding to about 4% of the z_{PV} resolution (see Fig. 6.28).

The origin of the bias was initially investigated in terms of search patterns and cluster sizes. As discussed in Sect. 3, clusters from non–isolated SPs are reconstructed using a pattern matching search algorithm with a cluster maximum size limited to 3×3 pixels. However, it was found that the bias was not due to this characteristic of the clustering algorithm. Moreover, it was also present when considering only clusters from isolated SPs that are not affected by the cluster maximum size limitation. The origin of the bias was traced back to the first version of the cluster coordinate rounding procedure. Cluster coordinates are encoded in the FPGA output, as shown in Fig. 4.5, with two fractional binary digits, corresponding to a resolution of one fourth of a pixel, both for x and y coordinates. In the first firmware and software implementations, the coordinate values were truncated, instead of being rounded, as discussed in Sect. 3.4, causing a bias on the cluster positions. The issue was then fixed and the bias is no longer present, as shown in the right plot of Fig. C.1. This is just one example of the improvements on the physics reconstruction quality obtained by studying in detail the algorithm behavior under different conditions and fine tuning its parameters.

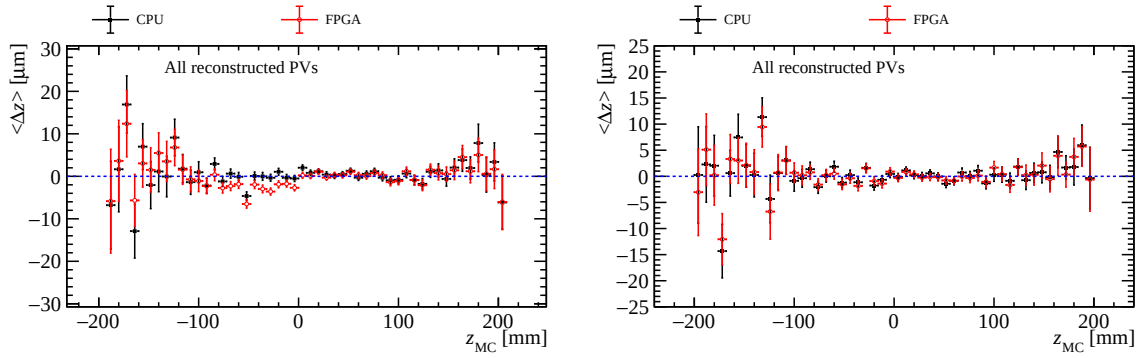

Figure C.1: Comparison of the mean of the Gaussian fits to the residuals of the primary vertex position between the CPU and FPGA clustering algorithms for all clusters, (left) with truncated cluster coordinates and (right) with rounded cluster coordinates to the closest one–forth step.

References

[1] ATLAS collaboration, G. Aad *et al.*, *Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC*, Phys. Lett. B **716** (2012) 1.

[2] CMS collaboration, S. Chatrchyan *et al.*, *Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC*, Phys. Lett. B **716** (2012) 30.

[3] LHCb collaboration, R. Aaij *et al.*, *Observation of D^0–$\overline{D}^0$ oscillations*, Phys. Rev. Lett. **110** (2013) 101802.

[4] LHCb collaboration, R. Aaij *et al.*, *Observation of CP violation in charm decays*, Phys. Rev. Lett. **122** (2019) 211803.

[5] LHCb collaboration, R. Aaij *et al.*, *Measurement of the $B^0_s \to \mu^+\mu^-$ decay properties and search for the $B^0 \to \mu^+\mu^-$ and $B^0_s \to \mu^+\mu^-\gamma$ decays*, Phys. Rev. D **105** (2022) 012010.

[6] LHCb collaboration, R. Aaij *et al.*, *Measurement of CP violation in $B^0 \to J/\psi K^0_S$ and $B^0 \to \psi(2S)K^0_S$ decays*, JHEP **11** (2017) 170.

[7] LHCb collaboration, R. Aaij *et al.*, *Precision measurement of the B^0_s-$\bar{B}^0_s$ oscillation frequency with the decay $B^0_s \to D^-_s\pi^+$*, New J. Phys. **15** (2013) 053021.

[8] LHCb collaboration, R. Aaij *et al.*, *First observation of a doubly charged tetraquark candidate and its neutral partner*, arXiv:2212.02716, to appear in Phys. Rev. Lett.

[9] LHCb collaboration, R. Aaij *et al.*, *Observation of a $J/\psi\Lambda$ resonance consistent with a strange pentaquark candidate in $B^- \to J/\psi\Lambda\overline{p}$ decays*, arXiv:2210.10346, to be published in Phys. Rev. Lett.

[10] LHCb collaboration, R. Aaij *et al.*, *Test of lepton universality in beauty-quark decays*, Nat. Phys. **18** (2022) 277.

[11] LHCb collaboration, R. Aaij *et al.*, *Measurement of the CKM angle γ from a combination of LHCb results*, JHEP **12** (2016) 087.

[12] B. Odom *et al.*, *New measurement of the electron magnetic moment using a one-electron quantum cyclotron*, Phys. Rev. Lett. **97** (2006) 030801.

[13] S. L. Glashow *et al.*, *Weak interactions with lepton-hadron symmetry*, Phys. Rev. D **2** (1970) 1285.

[14] UA1 collaboration, G. Arnison *et al.*, *Experimental observation of isolated large transverse energy electrons with associated missing energy at s=540 GeV*, Phys. Lett. B **122** (1983) 103.

[15] UA1 collaboration, G. Arnison *et al.*, *Experimental observation of lepton pairs of invariant mass around 95 GeV/c^2 at the CERN SPS collider*, Phys. Lett. B **126** (1983) 398.

[16] N. Cabibbo, *Unitary symmetry and leptonic decays*, Phys. Rev. Lett. **10** (1963) 531.

[17] M. Kobayashi *et al.*, *CP-Violation in the Renormalizable Theory of Weak Interaction*, Prog. Theor. Phys. **49** (1973) 652.

[18] B. Pontecorvo, *Inverse beta processes and nonconservation of lepton charge*, Zh. Eksp. Teor. Fiz. **34** (1957) 247.

[19] Z. Maki *et al.*, *Remarks on the Unified Model of Elementary Particles*, Prog. Theor. Phys. **28** (1962) 870.

[20] J. N. Butler *et al.*, *Report of the 2021 U.S. Community Study on the Future of Particle Physics (Snowmass 2021)*, FERMILAB-CONF-23-008, 2021.

[21] LHCb collaboration, R. Aaij *et al.*, *First evidence for the decay $B^0_s \to \mu^+\mu^-$*, Phys. Rev. Lett. **110** (2013) 021801.

[22] LHCb collaboration, R. Aaij *et al.*, *Measurement of the $B^0_s \to \mu^+\mu^-$ Branching Fraction and Search for $B^0 \to \mu^+\mu^-$ Decays at the LHCb Experiment*, Phys. Rev. Lett. **111** (2013) 101805.

[23] CMS and LHCb collaborations, V. Khachatryan *et al.*, *Observation of the rare $B^0_s \to \mu^+\mu^-$ decay from the combined analysis of CMS and LHCb data*, Nature **522** (2015) 68.

[24] LHCb collaboration, R. Aaij *et al.*, *Measurement of the $B^0_s \to \mu^+\mu^-$ branching fraction and effective lifetime and search for $B^0 \to \mu^+\mu^-$ decays*, Phys. Rev. Lett. **118** (2017) 191801.

[25] LHCb collaboration, R. Aaij *et al.*, *Measurement of lepton universality parameters in $B^+ \to K^+\ell^+\ell^-$ and $B^0 \to K^{*0}\ell^+\ell^-$ decays*, arXiv:2212.09153, submitted to Phys. Rev. D.

[26] LHCb collaboration, R. Aaij *et al.*, *Test of lepton universality in $b \to s\ell^+\ell^-$ decays*, arXiv:2212.09152, submitted to Phys. Rev. Lett.

[27] LHCb collaboration, R. Aaij *et al.*, *LHCb Framework TDR for the LHCb Upgrade II Opportunities in flavour physics, and beyond, in the HL-LHC era*, CERN-LHCC-2021-012, 2022.

[28] A. Cerri, *Trigger upgrades at LHC experiments*, Talk at 10^{th} Edition of the Large Hadron Collider Physics Conference, 2022.

[29] A. Andreazza *et al.*, *What Next: White Paper of the INFN-CSN1*, vol. 60, Istituto Nazionale di Fisica Nucleare, 2015. Available at http://www.lnf.infn.it/sis/frascatiseries/Volume60/Volume60.pdf.

[30] *OneAPI main site*, https://www.oneapi.io/.

[31] A. Cabrera *et al.*, *Design and Analysis of CXL Performance Models for Tightly-Coupled Heterogeneous Computing*, in *Proceedings of the 1st International Workshop on Extreme Heterogeneity Solutions*, (New York, NY, USA), 2022.

[32] Y. Sun *et al.*, *Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices*, arXiv:2303.15375.

[33] C. Wang *et al.*, *CXL over Ethernet: A Novel FPGA-based Memory Disaggregation Design in Data Centers*, arXiv:2302.08055.

[34] ALICE collaboration, P. Buncic *et al.*, *Technical Design Report for the Upgrade of the Online-Offline Computing System*, CERN-LHCC-2015-006, 2015.

[35] ATLAS collaboration, M. Shochet *et al.*, *Fast TracKer (FTK) Technical Design Report*, CERN-LHCC-2013-007, 2013.

[36] ATLAS collaboration, A. Cerri, *L1Track: A fast Level 1 track trigger for the ATLAS high luminosity upgrade*, Nucl. Instrum. Meth. A **824** (2016) 263.

[37] E. Bartz *et al.*, *FPGA-based tracking for the CMS Level-1 trigger using the tracklet algorithm*, JINST **15** (2020) P06024.

[38] CMS collaboration, S. Acharya *et al.*, *The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger*, CERN-LHCC-2021-007, 2021.

[39] CMS collaboration, S. Acharya *et al.*, *The Phase-2 Upgrade of the CMS L1 Trigger Interim Technical Design Report*, CERN-LHCC-2017-013, 2017.

[40] A. Bocci *et al.*, *Heterogeneous reconstruction of tracks and primary vertices with the CMS pixel tracker*, Front. Big Data **3** (2020) 601728.

[41] R. Aaij *et al.*, *Allen: A high-level trigger on GPUs for LHCb*, Comput. Soft. Big Sci. **4** (2020) 7.

[42] LHCb collaboration, R. Aaij *et al.*, *LHCb Upgrade GPU High Level Trigger Technical Design Report*, CERN-LHCC-2020-006, 2020.

[43] C. Fitzpatrick, *RTA Upgrade II summary*, Talk at 5^{th} Workshop on LHCb Upgrade II, 2021.

[44] R. Cenci *et al.*, *Development of a High-Throughput Tracking Processor on FPGA Boards*, in *Proc. Topical Workshop on Electronics for Particle Physics (TWEPP 2017)*, (Santa Cruz, CA, USA), 2017.

[45] F. Lazzari, *Improving charm CPV measurements with real-time data reconstruction*, PhD thesis, Università degli Studi di Siena, 2022, https://cds.cern.ch/record/2813167.

[46] D. Ninci, *Ricostruzione di traccia in tempo reale su FPGA ad LHC*, Master's thesis, Università di Pisa, 2014. https://cds.cern.ch/record/2262626.

[47] A. Piucci, *Reconstruction of tracks in real time in the high luminosity environment at LHC*, Master's thesis, Università di Pisa, 2014. https://cds.cern.ch/record/2746996.

[48] A. Di Luca, *Real-time reconstruction of tracks in the Scintillating Fibre Tracker of the LHCb Upgrade*, Master's thesis, Università di Pisa, 2018. https://cds.cern.ch/record/2649477.

[49] G. Tuci, *Searching for confirmation of charm CP violation in K^0_S final states at LHCb*, PhD thesis, Università di Pisa, 2020, ttps://cds.cern.ch/record/2765102.

[50] LHCb collaboration, R. Aaij *et al.*, *Expression of Interest for a Phase-II LHCb Upgrade: Opportunities in flavour physics, and beyond, in the HL-LHC era*, CERN-LHCC-2017-003, 2017.

[51] LHCb collaboration, A. A. Alves Jr. *et al.*, *The LHCb detector at the LHC*, JINST **3** (2008) S08005.

[52] LHCb collaboration, R. Aaij *et al.*, *LHCb detector performance*, Int. J. Mod. Phys. **A30** (2015) 1530022.

[53] LHCb collaboration, R. Aaij *et al.*, *Implications of LHCb measurements and future prospects*, Eur. Phys. J. **C73** (2013) 2373.

[54] LHCb collaboration, R. Aaij *et al.*, *The LHCb trigger and its performance in 2011*, JINST **8** (2013) P04022.

[55] LHCb collaboration, R. Aaij *et al.*, *Absolute luminosity measurements with the LHCb detector at the LHC*, JINST **7** (2012) P01010.

[56] LHCb collaboration, K. C. Akiba *et al.*, *The HeRSCheL detector: high-rapidity shower counters for LHCb*, JINST **13** (2018) P04017.

[57] LHCb collaboration, R. Aaij *et al.*, *Letter of Intent for the LHCb Upgrade*, CERN-LHCC-2011-001, 2011.

[58] LHCb collaboration, R. Aaij *et al.*, *Framework TDR for the LHCb Upgrade: Technical Design Report*, CERN-LHCC-2012-007, 2012.

[59] LHCb collaboration, R. Aaij *et al.*, *The LHCb Upgrade I*, LHCB-DP-2022-002-002, In preparation for submission to JINST.

[60] LHCb collaboration, R. Aaij *et al.*, *LHCb VELO Upgrade Technical Design Report*, CERN-LHCC-2013-021, 2013.

[61] LHCb collaboration, R. Aaij *et al.*, *LHCb Tracker Upgrade Technical Design Report*, CERN-LHCC-2014-001, 2014.

[62] LHCb collaboration, R. Aaij *et al.*, *LHCb PID Upgrade Technical Design Report*, CERN-LHCC-2013-022, 2013.

[63] T. Poikela *et al.*, *VeloPix: the pixel ASIC for the LHCb upgrade*, JINST **10** (2015) C01057.

[64] T. Poikela *et al.*, *Timepix3: a 65K channel hybrid pixel readout chip with simultaneous ToA/ToT and sparse readout*, JINST **9** (2014) C05013.

[65] LHCb collaboration, L. Eklund *et al.*, *The VELO optical and power board*, LHCb-PUB-2021-012, 2021.

[66] J. P. Cachemiche *et al.*, *The PCIe-based readout system for the LHCb experiment*, JINST **11** (2016) P02013.

[67] K. Hennessy *et al.*, *Readout Firmware of the Vertex Locator for LHCb Run 3 and Beyond*, IEEE Trans. Nucl. Sci. **68** (2021) 2472.

[68] J. Barbosa *et al.*, *Front-end electronics control and monitoring for the LHCb Upgrade*, EPJ Web Conf. **214** (2019) 01002.

[69] R. Jacobsson *et al.*, *The final LHCb readout supervisor "ODIN"*, in *Proc. 8th Workshop on Electronics for LHC Experiments*, (Colmar, France), 2002.

[70] F. Alessio *et al.*, *The readout supervisor firmware for controlling the upgraded LHCb detector and readout system*, arXiv:1806.08626.

[71] LHCb collaboration, R. Aaij *et al.*, *LHCb Trigger and Online Upgrade Technical Design Report*, CERN-LHCC-2014-016, 2014.

[72] LHCb collaboration, R. Aaij *et al.*, *Computing Model of the Upgrade LHCb experiment*, CERN-LHCC-2018-014, 2018.

[73] LHCb collaboration, R. Aaij *et al.*, *LHCb Upgrade Software and Computing*, CERN-LHCC-2018-007, 2018.

[74] A. Hennequin *et al.*, *SparseCCL: Connected Components Labeling and Analysis for sparse images*, in *DASIP 2019 - The Conference on Design and Architectures for Signal and Image Processing*, (Montréal, Canada), 2019.

[75] D. H. Cámpora Pérez, *Optimization of high-throughput real-time processes in physics reconstruction*, PhD thesis, Universidad de Sevilla, 2019, https://cds.cern.ch/record/2718278.

[76] G. Bassi *et al.*, *FPGA implemention of a fast 2D clustering algorithm (VHDL language)*, 2019. doi: 10.15161/oar.it/23524.

[77] LHCb collaboration, S. Miglioranzi *et al.*, *The LHCb Simulation Application, Gauss: Design, Evolution and Experience*, CERN-LHCb-PROC-2011-006, 2011.

[78] L. Giambastiani, *A 2D FPGA-based clustering algorithm for the LHCb silicon pixel detector running at 30 MHz*, Master's thesis, Università di Pisa, 2020. https://cds.cern.ch/record/2725831.

[79] G. Bassi *et al.*, *A real-time FPGA-based cluster finding algorithm for LHCb silicon pixel detector*, EPJ Web Conf. **251** (2021) 04016.

[80] G. Barrand *et al.*, *GAUDI — A software architecture and framework for building HEP data processing applications*, Comput. Phys. Commun. **140** (2001) 45.

[81] C.-L. Sotiropoulou *et al.*, *A Multi-Core FPGA-Based 2D-Clustering Implementation for Real-Time Image Processing*, IEEE Trans. Nucl. Sci. **6** (2014) 3599.

[82] F. Spagnolo *et al.*, *An Efficient Connected Component Labeling Architecture for Embedded Systems*, Journal of Low Power Electronics and Applications **8** (2018) 7.

[83] M. J. Klaiber *et al.*, *A high-throughput FPGA architecture for parallel connected components analysis based on label reuse*, in *2013 International Conference on Field-Programmable Technology (FPT)*, (Kyoto, Japan), 2013.

[84] LHCb collaboration, R. Aaij *et al.*, *A Comparison of CPU and GPU Implementations for the LHCb Experiment Run 3 Trigger*, Comput. Soft. Big Sci. **6** (2021) 1.

[85] LHCb collaboration, R. Aaij *et al.*, *Tracking Definitions and Conventions for Run 3 and Beyond*, CERN-LHCb-PUB-2021-005, 2021.

[86] Wikipedia, *Wilson score interval*, 2022. Available at `https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval#Wilson_score_interval`.

[87] R. E. Kalman, *A New Approach to Linear Filtering and Prediction Problems*, J. Basic Eng. **82** (1960) 35.

[88] L. Granado Cardoso *et al.*, *LHCb MiniDAQ control system*, EPJ Web Conf. **214** (2019) 01005.

[89] C. Gaspar *et al.*, *DIM, a portable, light weight package for information publishing, data transfer and inter-process communication*, Comput. Phys. Commun. **140** (2001) 102.

[90] M. Adinolfi *et al.*, *LHCb data quality monitoring*, J. Phys. Conf. Ser. **898** (2017) 092027.

[91] R. Aaij *et al.*, *Evolution of the energy efficiency of LHCb's real-time processing*, EPJ Web Conf. **251** (2021) 04009.

[92] M. Domke *et al.*, *Commissioning of the beam conditions monitor of the LHCb experiment at CERN*, in *2008 IEEE Nucl. Sci. Symp. Conf. Rec.*, (Dresden, Germany), 2008.

[93] LHCb collaboration, R. Aaij *et al.*, *LHCb PLUME: Probe for LUminosity MEasurement*, CERN-LHCC-2021-002, 2021.

[94] S. van der Meer, *Calibration of the effective beam height in the ISR*, CERN-ISR-PO-68-31, 1968.

[95] P. Grafström *et al.*, *Luminosity determination at proton colliders*, Prog. Part. Nucl. Phys. **81** (2015) 97.

[96] V. Balagura, *Van der Meer scan luminosity measurement and beam–beam correction*, Eur. Phys. J. C **81** (2021) 26.

[97] D. Passaro, *Real-time luminosity and detector monitoring using FPGAs at LHCb experiment*, Master's thesis, Università di Pisa, 2022. `https://cds.cern.ch/record/2842603`.

[98] F. Follin *et al.*, *Implementation and experience with luminosity levelling with offset beam*, in *ICFA Mini-Workshop on Beam-Beam Effects in Hadron Colliders*, (Geneva, Switzerland), 183–187, 2014.

[99] LHCb collaboration, R. Aaij *et al.*, *Mass plots with early Run 3 data*, LHCB-FIGURE-2023-002, 2023.

[100] G. Bassi *et al.*, *A FPGA-Based Architecture for Real-Time Cluster Finding in the LHCb Silicon Pixel Detector*, IEEE Trans. Nucl. Sci. **70** (2023) 1189.

[101] Intel®, *Intel® Arria® 10 Device Overview*, https://www.intel.com/content/www/us/en/docs/programmable/683332/current/device-overview.html, 2022.

[102] Intel®, *Intel® Arria® 10 Core Fabric and General Purpose I/Os Handbook*, https://www.intel.com/content/www/us/en/docs/programmable/683461/current/logic-array-blocks-and-adaptive-logic-05488.html, 2023.

[103] L. Semiconductor, *Timing Closure*, https://www.latticesemi.com/~/media/LatticeSemi/Documents/UserManuals/RZ/Timing_Closure_Document.pdf, 2013.

[104] Intel®, *Timing analyzer*, https://www.intel.com/content/www/us/en/docs/programmable/683243/21-3/timing-analysis-introduction.html, 2021.

[105] Intel®, *Design Optimization*, https://www.intel.com/content/www/us/en/docs/programmable/683230/18-1/design-optimization-overview.html, 2018.

[106] Intel®, *Timing Closure Methodology for Advanced FPGA Designs*, https://www.intel.com/content/www/us/en/docs/programmable/683145/21-3/an-584-timing-closure-methodology-for.html, 2021.