UNIVERSITÀ DEGLI STUDI DI MILANO

Scuola di Dottorato in Fisica, Astrofisica e Fisica Applicata

Dipartimento di Fisica

Corso di Dottorato in Fisica, Astrofisica e Fisica Applicata

Ciclo XXX

A 4D real-time tracking device for the LHCb Upgrade II

Settore Scientifico Disciplinare FIS/01

Supervisore: Prof. Nicola NERI

Coordinatore: Prof. Francesco RAGUSA

Tesi di Dottorato di:

Marco PETRUZZO

Anno Accademico 2018/2019

Commission of the final examination:

External Referees:

Prof. Luciano RISTORI, FNAL, Chicago.
Dr. Massimiliano FERRO-LUZZI, CERN, Geneva.

External Members:

Prof. Tim GERSHON, University of Warwick.
Prof. Angelo CARBONE, Università degli Studi di Bologna.
Prof. Gianluca CAVOTO, Università degli Studi di Roma "La Sapienza".

Final examination:

May, 23 2019.
Dipartimento di Fisica, Università degli Studi di Milano,
Via Celoria 16, Milano.

MIUR subjects:

FIS/01

PACS:

07.05.Hd
29.40.Gx

To my family.

Contents

List of Figures

List of Tables

Introduction

General introduction

I started my Ph.D. at the Physics Department of Università degli Studi di Milano, in association with the Istituto Nazionale di Fisica Nucleare - Sezione di Milano, in March 2015. My research activity as Ph.D. student has been carried within the LHCb collaboration and the INFN CSN5 RETINA and TIMESPOT projects.

My contribution to the LHCb collaboration was focused on the testbeam campaign for the study and characterization of the silicon sensors for the Upgrade I of the Upstream Tracker sub-detector of the LHCb experiment. I participated to different testbeam campaigns at the CERN Super Proton Synchrotron (SPS) in 2014, 2015, 2016; I mainly worked on a custom DAQ board based on Field Programmable Gate Array (FPGA) technology for the readout of the silicon sensors that has been developed at INFN-Milano. I also took part in the data taking shifts and in the analysis of the testbeam data; during the last experience I co-directed the testbeam activity during the preparation, the set-up of the experimental system and the data taking period. Within the RETINA project I worked on the design of a device for real-time tracking of charged particles in FPGA; in particular I developed a C++ software simulation and the VHDL firmware implementation of the system in FPGA. A prototype, based on a single-sided telescope, has been successfully tested at SPS in October 2015.

The main result of my Ph.D. research activity, documented in this thesis, has been the development and implementation of a new algorithm for application to a 4D (space-time) tracking system to reconstruct the particle trajectories in real-time. The application of such a system to the Upgrade II of the LHCb experiment has been considered in this work; an application to the Beam Gas Vertex Detector has also been considered. The inclusion of the time information in the first stages of the trigger decision would help in mitigating the effects of the pile-up in the upgraded conditions of the LHC collider, allowing LHCb to exploit the larger data sample for its physics program. The proposed tracking algorithm is modular and highly parallelized, and is designed to work with large tracking devices at high rates with latency below 1 μs. The performances of the 4D fast tracking device and its firmware implementation have been tested on a custom board equipped with latest generation FPGA (Xilinx Virtex UltraScale, *XCVU095*) and high-speed serial links to sustain up to 1.6 Tbps input data bandwidth. The response of the LHCb VELO detector based on pixel sensors has been simulated assuming 30 ps time resolution. Simulated data from a sector ($\sim$ 1/64) of the detector have been provided to

the fast track finder prototype to test the system. This demonstrator prototype has to be considered as a building block of a large scale system capable to process a large tracking detector of a LHC experiment.

Thesis overview

My Ph.D. thesis is inserted in the context of the Upgrade II of the LHCb experiment, for the High Luminosity phase of LHC. In particular the thesis is focused on the development of a hardware demonstrator of a fast track finder using precise space and time information of the particle hits in the detector, *i.e.* a possible upgraded VELO detector of the LHCb experiment, able to identify and reconstruct the particle tracks in real time with low latency. This feature would allow to use the track information in the early stage of the trigger decision chain.

In the past the Silicon Vertex (SVT) of the CDF experiment at Fermilab played a very important role in the study of heavy flavour physics thanks to the possibility of fast track reconstruction with a very good quality. The selection was based on the track impact parameter and the identification of displaced vertices with respect to the position of the primary interaction. The SVT worked as part of the Level 2 of a three-level trigger system, and implemented a highly parallelized pattern-matching using a custom-made processor, the associative memory (AM). In particular the AM compares in parallel the hits from the tracking system to several precomputed patterns, providing the tracks candidates to be fitted using a simplified fitting algorithm in fast FPGAs. This made possible to reconstruct tracks with offline-like resolution and latency of about 20 μs allowing the Level 2 trigger to match the full output rate of approximately 30 KHz of the Level 1. The evolution of this concept is exploited by ATLAS in the *Fast TracKer* (FTK) device, that is being commissioned and will provide the information of the already reconstructed tracks to the High Level Trigger (HLT). The working principle is substantially unchanged while the modern electronics and a strongly pipelined architecture allow to handle a much larger number of patterns and process the more complex ATLAS events at 100 KHz rate, with an average latency below 100 μs.

The LHCb experiment is going to run at an instantaneous luminosity of 2×10^{33} cm^{-2}s^{-1} during the Run 3, from 2021 to 2023, and the collaboration is currently working on an extensive upgrade of the detector and the DAQ system in order to strengthen the physics program. In particular the tracking detectors will be completely replaced with a pixel Vertex Locator (VELO), a silicon tracking station before the magnet, the Upstream Tracker (UT), and a Scintillating Fibre downstream tracker (SciFi). The detector will be read out at 40 MHz and a full software trigger will process the full rate of inelastic collisions delivered by the LHC, providing significantly increased efficiency in hadronic final states. After the Long Shutdown 3 (LS3) the LHC conditions will be upgraded to the so-called High Luminosity phase (HL-LHC) and the luminosity at the LHCb interaction point during the Run 4, expected from 2026 to 2030, will be levelled to the same conditions of the Run 3, allowing the Upgrade I detector to continue operating without major modifications. At the end of the Run 4 the precision on some important measurements will still be limited by statistics and many parts of the detector will need to be replaced due to

the radiation damage. For these reasons the LHCb collaboration has already proposed an Upgrade II, which will operate at a luminosity of around $1 - 2 \times 10^{34}$ cm^{-2}s^{-1}, to exploit the potential of the HL-LHC during the Run 5, expected to start in 2031.

The trigger requirements at LHCb are very challenging due to the need of reading out the whole detector at 40 MHz and processing the full event information, since the current first level hardware trigger would be limited at higher luminosities for hadronic channels. During Run 2 it was demonstrated that a full software reconstruction is possible for real-time calibration and alignment and the strategy for the software and the computing of the Upgrade I are under review. The software framework is going to be revised to full profit of modern multi-core architectures, while solutions based on GPUs and *hardware accelerators* are under study. The operation of the LHCb Upgrade II will provide even more significant challenges; in fact the increased luminosity will lead to a higher event complexity and higher data volumes to process, affecting the output bandwidth to be managed by the trigger and offline system that will have to be improved to cope with the harsher environment. Even though the Upgrade II trigger will remain software based, it is not clear if the Upgrade I solutions can be adopted, even considering the expected computing power growth during the 2020s. For this reason the use of dedicated processors has to be considered as a viable solution to solve specific tasks and evaluate relevant information to be fed to the reconstruction software at the earliest trigger level. An example of these tasks could be the early reconstruction of the tracks downstream of the magnet; in fact, these are not included in the baseline trigger scheme due to the CPU time required to execute the search. Another example is the identification and reconstruction of tracks in the VELO sub-detector, and eventually in the UT; this task is computationally intensive and the use of custom processor would free a significant amount of CPU computing power. Along with pure algorithm and computing strategies, another relevant help could come by the inclusion of timing information of the hits, currently not available, that would help in suppressing the combinatoric background, reducing the negative effects of the pile-up as the incorrect track-vertex association.

Organizational note

The present thesis is organized as follows:

- Chapter 1. The LHC collider and in particular the LHCb experiment and the Upgrade I and Upgrade II are described.
- Chapter 2. An overview of the existing fast tracking devices, along with their use inside the trigger chain, is given.
- Chapter 3. The artificial retina for fast track reconstruction is described: the algorithm, the hardware implementation on a hardware prototype and the testbeam results are discussed.
- Chapter 4. An algorithm for real-time 4D track reconstruction using precise space and time information of the detector hits is presented, together with the results from the simulated response of the tracking device.

- Chapter 5. The architecture of the hardware implementation in FPGA is described.
- Chapter 6. The results from tests of the algorithm implemented on a prototype board and fed with data from a sector of the VELO sub-detector of the LHCb experiment are reported.
- Chapter 7. A possible application of a real-time tracking algorithm in FPGA is proposed for the LHC Beam Gas Vertex detector.

Chapter 1

LHCb detector and upgrades

The LHCb experiment [1] is one of the four main experiments located at the Large Hadron Collider (LHC) [2] at CERN.
In this chapter we will provide an overview of the LHCb experiment and its upgrades within the context of the LHC upgrade schedule to the High Luminosity phase (HL-LHC) [3], planned to begin after the Long Shutdown 3 (LS3, 2024-2026); in particular, the High Luminosity phase for LHCb will begin after the Long Shutdown 4 (LS4, 2030).

1.1 The Large Hadron Collider at CERN

The Large Hadron Collider is the world's largest and most powerful particle accelerator, located at CERN. The LHC is a two-ring superconducting accelerator, measuring 26.7 km in circumference. It is installed 100 m underground at the Franco-Swiss border near Geneva, in the tunnel that previously hosted the *Large Electron Positron* (LEP) collider which operated from 1989 to 2000.
The LHC is able to accelerate two counter-rotating beams of both protons and ions. The first case represents the main operation mode of the LHC experiments, investigating in particular proton-proton (pp) collisions at the center-of-mass energy of $\sqrt{s} = 13$ TeV, near the maximum design value of 14 TeV.
The LHC accelerator ring represents the last acceleration stage, after several particle accelerators. The scheme of the CERN accelerator complex, with the positions of the four LHC experiments highlighted along the LHC ring, is shown in Fig. 1.1.
At the beginning, protons are obtained by ionizing hydrogen gas with an electric discharge and are accelerated by the LINAC2 (*Linear Accelerator 2*) up to 50 MeV. The protons are then injected into the PSB (*Proton Synchrotron Booster*) in which the proton beam energy is increased up to 1.4 GeV. The PS (*Proton Synchrotron*) machine accelerates protons up to 26 GeV and the beam is injected into the SPS (*Super Proton Synchrotron*) accelerator where the proton beam reaches the energy of 450 GeV. The SPS represents the last pre-acceleration stage before the LHC: the beam is split and injected into the LHC where two separate beams travel in clockwise and anticlockwise directions and are accelerated up to 6.5 TeV. After all the acceleration stages have been performed the two beams collide in four interaction points where the main LHC experiments are located: ALICE [4], ATLAS [5], CMS [6], LHCb [1]. Each beam consists nominally of 2808 proton

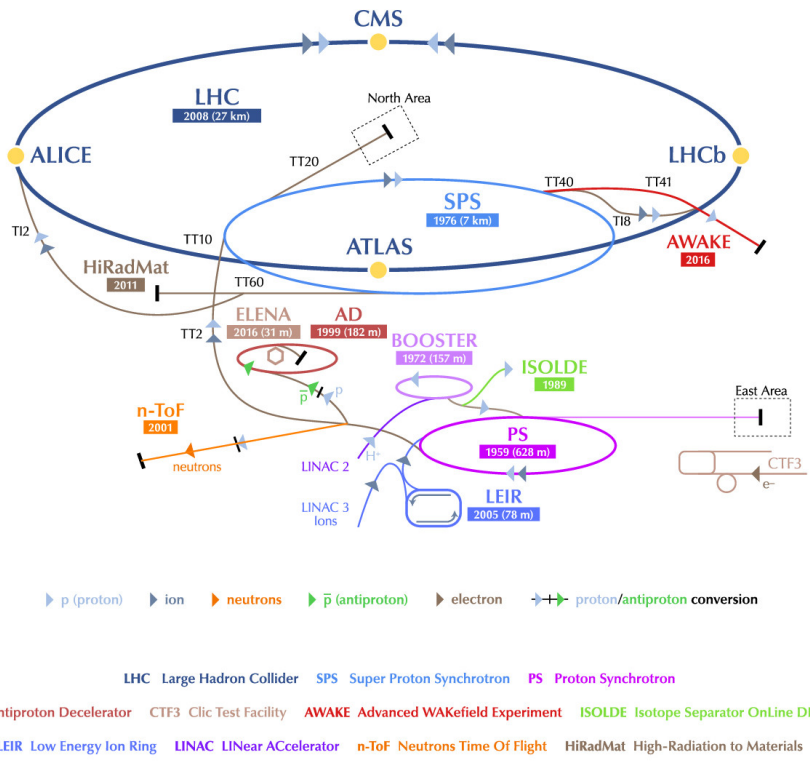

Figure 1.1: Scheme of the CERN accelerator complex showing the acceleration facilities and the four main experiments.

bunches spaced by 25 ns, resulting in the expected bunch crossing frequency of 40 MHz. Each bunch is about 7.5 cm long in the beam direction and nominally contains 1.15×10^{11} protons leading to the instantaneous luminosity of 2×10^{34} cm^{-2}s^{-1}, reached during year 2018 at ATLAS and CMS interaction points, which is a factor of two higher than the LHC design luminosity (1×10^{34} cm^{-2}s^{-1}). At the LHCb interaction point the instantaneous luminosity is reduced to 5×10^{32} cm^{-2}s^{-1} using a larger transverse beam size in order to reduce the number of interactions per bunch crossing; this is essential in terms of data quality and detector radiation damage. At the ALICE interaction point the luminosity (for pp interactions) is further reduced to 4×10^{30} cm^{-2}s^{-1}.
The main four LHC experiments are designed for different scientific programs:

- The ATLAS (***A** **T**oroidal **L**HC **A**pparatu**s***) and CMS (***C**ompact **M**uon **S**olenoid*) experiments are General Purpose Detectors (GPDs) and are designed to study collisions producing high transverse momentum particles. Their physics program is focused on the study of the Higgs Boson properties and also includes search for direct signals of New Physics beyond the Standard Model, search of Dark Matter candidates and Standard Model precision measurements. Besides sharing similar scientific goals, the ATLAS and CMS experiments differ in the apparatus design

whose main difference can be identified in the magnet systems, while both are based on a cylindrical *barrel* geometry and two symmetrical *end caps* placed along the direction of the beam axis.

- ALICE (***A** **L**arge **I**on **C**ollider **E**xperiment*) mainly profits of heavy ions collisions, in particular lead ions, and it is designed to investigate the strong interaction theory through the study of Quark-Gluon Plasma state. The detector uses both a Time Projecting Chamber and a forward spectrometer.

- The LHCb (***LHC** **b**eauty*) is a single-arm forward spectrometer experiment. It is dedicated to the indirect search of New Physics beyond the Standard Model, via precise measurement of the *CP* violation and rare decays of bottom and charm hadrons, using the vast statistics of heavy flavour hadrons produced in proton-proton (pp) collisions in the forward region. Further details will be provided in the following sections.

In addition to these four experiments, three smaller experiments dedicated to forward physics are present at LHC.

- TOTEM [7] (***TOT**al **E**lastic and diffractive cross section **M**easurement*), located at the CMS interaction point, studies the total proton-proton cross-section, elastic scattering and diffractive dissociation.

- LHCf [8] (***LHC** **f**orward*), located at the ATLAS interaction point, is used for engineering measurements for astroparticle experiments simulating cosmic rays in laboratory conditions.

- MoEDAL [9] (***Mo**nopole and **E**xotics **D**etector **A**t the **L**HC*), located at the LHCb interaction point, is dedicated to the search of the magnetic monopoles.

1.2 High Luminosity phase of LHC

The LHC was successfully commissioned in 2010 and started delivering proton-proton collisions at a centre-of-mass energy of $\sqrt{s} = 7$ TeV in 2010-2012, and at $\sqrt{s} = 8$ TeV from April 2012 until the end of the Run 1 .
The Run 2 of LHC started in June 2015 and ended in December 2018. During this period the centre-of-mass energy of pp collision has reached the value of $\sqrt{s} = 13$ TeV and an instantaneous luminosity of 2×10^{34} cm^{-2}s^{-1}, representing twice the nominal design value, was achieved. At the end of the Run 2, the ATLAS experiment collected a total integrated luminosity of 187.1 fb^{-1}, the CMS experiment 192.0 fb^{-1}, the LHCb experiment 10.1 fb^{-1} and the ALICE experiment 81.7 pb^{-1}, considering the proton and ion delivered luminosities[1].
During the LS2 the 160 MeV LINAC4 accelerator will be connected to the PSB while all the pre-accelerators and infrastructures will undergo major maintenance and consolidation.

[1] http://acc-stats.web.cern.ch/acc-stats/#lhc/.

During the Run 3 of LHC, expected to start in 2021, the centre-of-mass energy for pp collisions is expected to be increased up to $\sqrt{s} = 14$ TeV and the instantaneous luminosity is expected to stay at the late 2018 value, reaching a delivered integrated luminosity of 400 fb^{-1} by the end of the run. In particular during the Run 3 the statistical gain in running the accelerator without a luminosity increase will become marginal and if on one hand this would lead to doubling the statistics from the Run 2 to the Run 3, on the other hand the time necessary to reduce by half the statistical errors in measurement results would be ten years starting from the end of Run 2 [10]. For this reason and to fully exploit the potential of the LHC a major upgrade of the accelerator has been approved. The so-called *High Luminosity LHC* (HL-LHC) [11] will operate after the LS3 and will increase the luminosity by a factor five with respect to the current nominal design value of 10^{34} cm^{-2}s^{-1}.

The LS3 is scheduled to start in 2024 and will last 18 months. During the shutdown the LHC will undergo a major upgrade of its components, like low-β quadrupole triplets and the use of crab cavities at the interaction regions, in preparation for the HL-LHC. Moreover at that time many critical components of the accelerator will have reached the end of their lifetime and the upgrade is necessary to allow the collider to work beyond 2025.

With the Run 4 of LHC, from 2026 to 2029, the High Luminosity phase will start, making the machine able to deliver a peak luminosity of 7.5×10^{34} cm^{-2} without beam levelling, allowing an integrated luminosity of $300 - 350$ fb^{-1} per year, for an expected total integrated luminosity of 4000 fb^{-1} after twelve years of operation, in 2037, that corresponds to about ten times the integrated luminosity expected to be obtained at the end of the Run 3. In particular, these value are intended to be valid for ATLAS and CMS experiments that operate at the same level of luminosity delivered by the accelerator.

The LHC luminosity plan from 2010 to 2038 is shown in Fig. 1.2. The instantaneous and integrated luminosity are represented by red dots and solid blue line, respectively.

In the case of the LHCb experiment the instantaneous luminosity will not be increased after the Run 3 but only after the LS4, and the High Luminosity phase of LHCb will start with the beginning of the Run 5. By the end of 2037 an integrated luminosity of 300 fb^{-1} is expected, to be compared with the value of 50 fb^{-1} expected at the end of the Run 4.

The luminosity plan from 2010 to 2037 for the LHCb experiment is shown in Fig. 1.3. The LHCb experiment makes use of the beam levelling to reduce the instantaneous luminosity below the values delivered by the LHC accelerator. The instantaneous and integrated luminosity are represented by red dots and solid blue line, respectively.

The HL-LHC will rely on a number of key innovations that push accelerator technology beyond its present limits. Among these are cutting-edge 11-12 tesla superconducting magnets, compact superconducting cavities for beam rotation with ultra-precise phase control, new technology and physical processes for beam collimation and 100 metre-long high-power superconducting links with negligible energy dissipation. Details on the HL-LHC project can be found in Ref. [3].

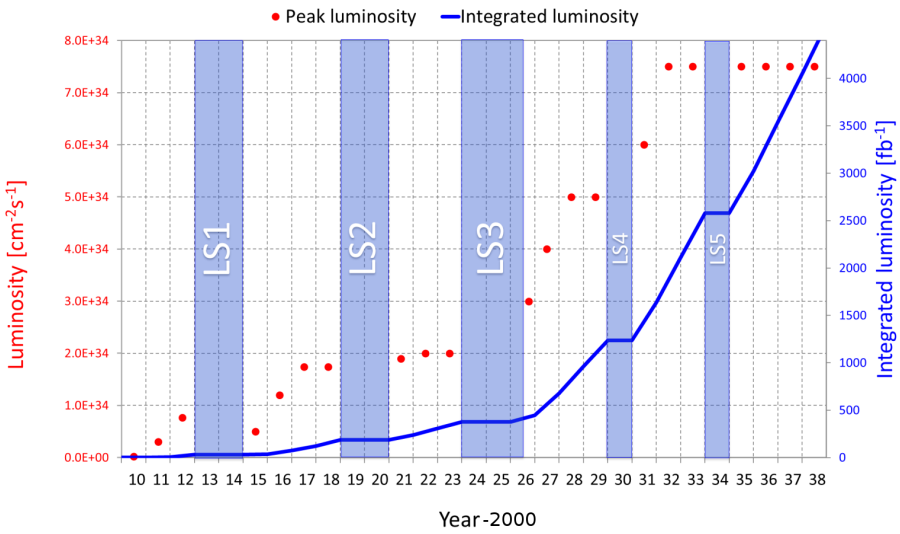

Figure 1.2: LHC luminosity plan from 2010 to 2038. The red dots represent the value of measured or predicted instantaneous luminosity. The solid blue line represents the value of measured or predicted integrated luminosity (for the ATLAS and CMS experiments).

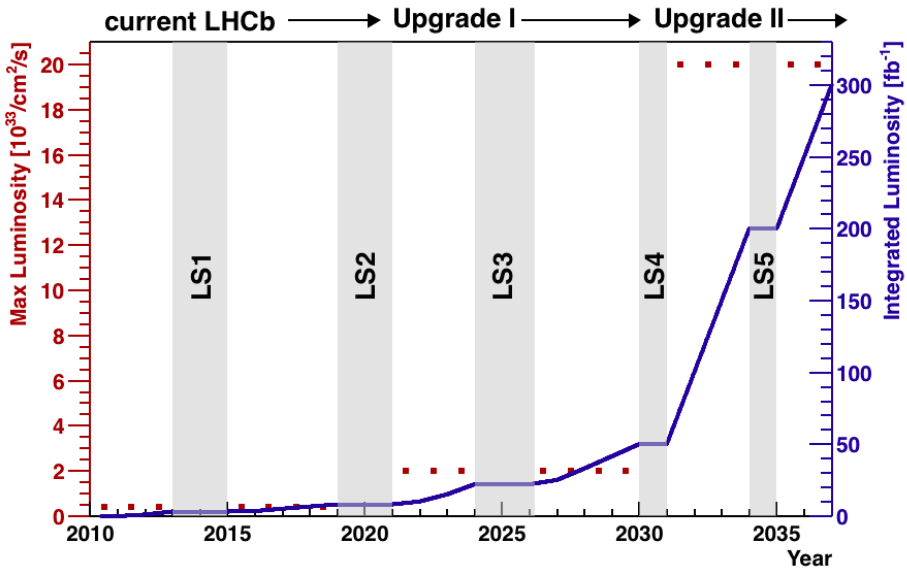

Figure 1.3: Luminosity plan from 2010 to 2037 for the LHCb experiment. Red dots represent the value of measured or predicted instantaneous luminosity. Solid blue line represents the value of measured or predicted integrated luminosity.

1.3 The LHCb detector

The LHCb detector is a single-arm forward magnetic spectrometer with a polar angle coverage from 15 mrad to 300 (250) mrad in the horizontal (vertical) plane, corresponding to a pseudorapidity (η) range of $2 < \eta < 5$ in the forward direction.
The layout of the LHCb detector and its sub-detectors is shown in Fig. 1.4. The beams travel along the direction of the z-axis and collide in the region centred around the axes origin, while the magnetic field is oriented along the vertical direction. The coordinate system of the LHCb experiment is described in Ref. [12] and can be summarized as follows:

- the origin of the coordinate system is the interaction point;
- the x-axis is horizontal, and points from the interaction point towards the outside of the LHC ring;
- the y-axis is perpendicular to the x-axis and to the beam line pointing upwards and is inclined by 3.601mrad with respect to the vertical;
- the z-axis points from the interaction point towards the LHCb detector and is aligned with the beam direction, to create a right handed Cartesian coordinate system x-y-z.

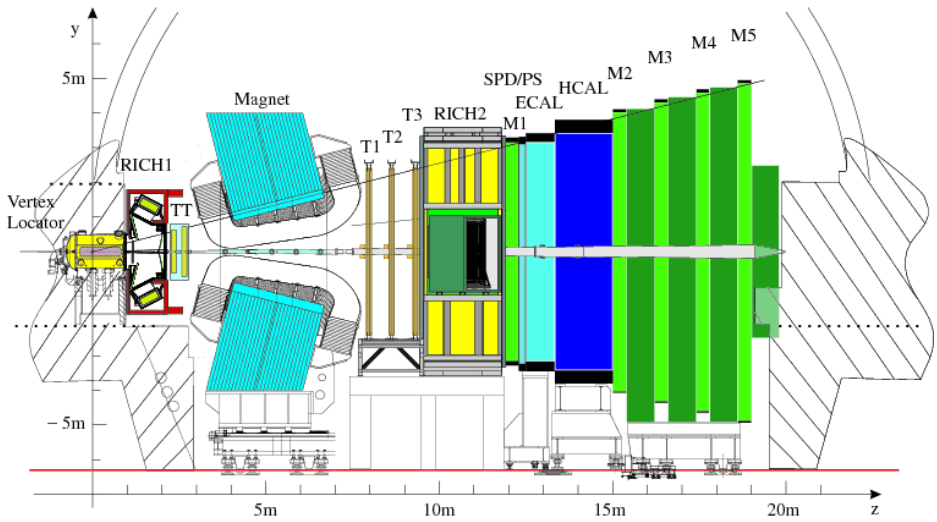

Figure 1.4: Lateral view of the LHCb detector. The interaction region is centred around the axes origin.

The peculiar geometry of the LHCb detector, compared to the typical design of the general purpose detectors at CERN, is motivated by the fact that in proton-proton collisions at the LHC energy scale the $b\bar{b}$ hadron pairs are mainly produced at small polar angles with respect to the beam direction as shown in Fig. 1.5 [13], both in the forward and backward direction, with relatively high momentum. Although the geometrical coverage of the detector corresponds only to 4% of the total solid angle, for a centre-of-mass energy

of $\sqrt{s} = 14$ TeV the probability to find either the b or the $\bar{b}$ hadron within the detector acceptance amounts to 27%, while the probability of finding the $b\bar{b}$ pair is 24%.

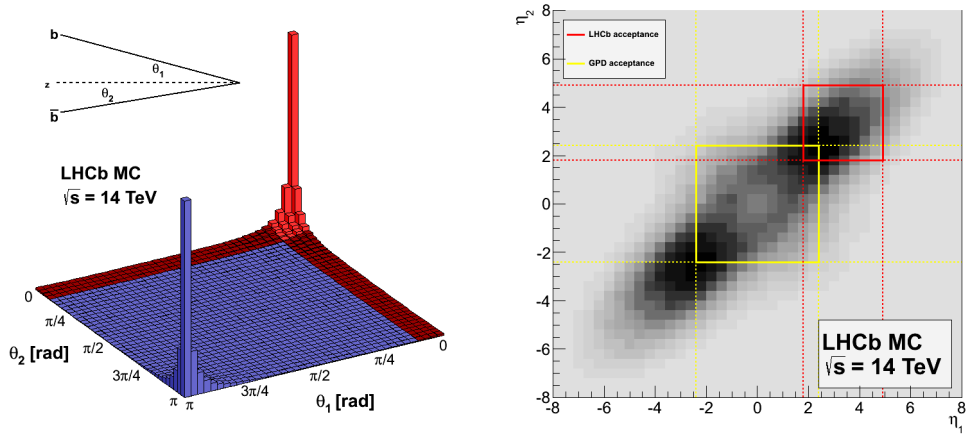

Figure 1.5: Left: Distribution of the production polar angles of the b quark and $\bar{b}$ antiquark in $b\bar{b}$ quark-pair production processes. Right: Same distribution as a function of the pseudorapidity of the b quark and $\bar{b}$ antiquark. The red square highlights the geometrical acceptance of the LHCb detector, compared to a general purpose detector, highlighted by the yellow square. Figure from Ref. [13].

1.3.1 Tracking system

The tracking system of the LHCb experiment is based on several tracking sub-detectors.

- The Vertex Locator (VELO) [14] is positioned near the interaction region and it is dedicated to the precise reconstruction of primary and displaced secondary vertices. It consists of 21 disk-shaped tracking stations based on silicon strip sensors, measuring separately the radial and azimuthal coordinates of track hits in the sensors.

- The Tracker Turicensis (TT) is a tracking detector placed upstream of the magnet. It is mainly dedicated to the matching of tracks reconstructed in the VELO detector with tracks reconstructed downstream of the magnet, in order to reduce the ghost tracks. It is also dedicated to the reconstruction of tracks produced by the decay of long-lived particles. The TT consists of four layers of silicon strip sensors arranged in *axial-stereo* configuration: the silicon strips in the first and last layers are oriented along the vertical direction, the second and third sensors are tilted by $+5°$ and $-5°$. The *axial-stereo* configuration allows to reconstruct three dimensional trajectories, with better resolution in x than in y direction.

- Three tracking stations (T1, T2, T3) are placed after the magnet; each station consists of two separate detectors based on different technologies, the Inner Tracker (IT) [15] and the Outer Tracker (OT) [16]. The IT, closer to the beam axis, is

a cross-shaped detector based on silicon strip sensors arranged in a configuration similar to that of the TT. The IT has a finer segmentation with respect to the OT but only covers the 2% of the full detector acceptance, while containing 20% of tracks produced in pp collisions. The OT is positioned around the IT to reach the full detector acceptance coverage. Each station has four layers arranged in the same configuration of each IT station. The OT is a drift gas detector based on small straw tubes aligned along the vertical direction: when a particle crosses a straw-tube, the electrons produced by the ionization of the gas drift towards the anode wire and, at the end, generate an avalanche signal which is measured. A time-to-digital converter measures the arrival time of the signal and by comparison with the LHC bunch clock it is possible to estimate the horizontal position of the hit with 200 μm space resolution.

- The dipole magnet provides an integrated field of 4 Tm in the vertical direction that allows to estimate the charged particle momenta by measuring the track trajectory curvature in the horizontal (x-z) plane.

1.3.2 Particle identification

The particle identification in the LHCb experiment is based on two ring imaging Cherenkov detectors (RICH1, RICH2) [17], a calorimeter detector [18] and five muon stations [19].
The RICH detectors are based on the emission of Cherenkov radiation and they are primarily dedicated to distinguish between kaons and pions. They identify charged particles with momenta in a range from 2 to 100 GeV/c. In particular the RICH1 sub-detector, located before the magnet, is optimized for low-momentum (< 60 GeV/c) particles and covers the full detector acceptance. The RICH2 sub-detector, located after the magnet is dedicated to high-momentum particles (> 15 GeV/c) and covers a reduced acceptance from 15 mrad to 120 (100) mrad in the horizontal (vertical) plane. The Cherenkov light emitted in the RICH detector is guided outside the LHCb acceptance using spherical and flat mirrors, then measured by a matrix of Hybrid Photon Detectors. The typical pattern of Cherenkov light is represented by cones that are detected as rings. The particle velocity is estimated from the measurement of the ring radius. This information combined with the track momentum allows to evaluate the mass of the candidate particle. The RICH1 and RICH2 sub-detectors differ in terms of radiation mediums and for the material and arrangement of the mirrors.
The calorimeter system is dedicated to identify electrons, photons and hadrons providing the measurement of their energy. The calorimeter system comprises the Electromagnetic Calorimeter (ECAL) and the Hadron Calorimeter (HCAL), both placed after the RICH2. It covers an angular acceptance range from 25 mrad to 300(250) mrad in the horizontal (vertical) plane, where the inner acceptance was driven by the particle density near the beam pipe. The ECAL is equipped with two additional sub-detectors: the Scintillating Pad Detector (SPD) and the Preshower Detector (PS).
The SPD is based on layers of scintillators. It is used to distinguish electrons from photons and pions. A 12 cm thick lead layer is inserted between the SPD and the PS. Electrons

passing the lead layer have higher probability to shower, with respect to hadrons, and are detected in the PS. In this way the PS can distinguish between electrons and hadrons.
The ECAL and the HCAL are the main components of the calorimeter system. The ECAL measures the energy of particles that interact by electromagnetic processes. It consists of alternating layers of scintillating material and lead absorbers. When a particle passes through the absorber it produces a shower originated from bremsstrahlung and pair production mechanisms; the shower interacts and deposits its energy in the scintillating material, the converted photons travel to the photomultipliers where the signal is collected and recorded.
The HCAL works in a similar fashion and measures the energy of particles that interact by strong nuclear interactions. The difference with the ECAL consists in the use of iron (instead of lead) for the absorber layers.

1.3.3 Muon system

The muon system is read out at 40 MHz and the presence of muons with high transverse momentum is a strong signature to perform hardware level trigger decisions (L0 trigger). The muon system consists of five muon stations (M1-M5) that cover the angular acceptance range from 20 (16) mrad to 306 (258) mrad in the horizontal (vertical) plane. The first station is placed between the RICH2 and the ECAL (SPD/PS), in order to improve the measurement of the transverse momentum for muons that are detected also in the other stations (M2-M5) placed after the HCAL. Stations from M2 to M5 are separated by 80 cm thick iron absorber layers, to select muons with different energies. In particular the minimum momentum to traverse all the layers is 6 GeV/c. Each station is divided in four quadrants, each of them is divided in four regions (R1-R4) arranged around the beam pipe whose dimensions follow the scale ratio (from R1 to R4) of 1:2:4:8 in order to have regions with similar occupancies. The inner region of the first station (M1-R1) is instrumented with triple-*gas electron multiplier* (triple-GEM) detectors, which better cope with the higher particle flux before the calorimeter and closer to the beam pipe, with respect to the multi-wire proportional chambers (MWPC) that are used in the rest of detector. Both the triple-GEM and the MWPC are optimized for fast signal readout and signal yield at 40 MHz, providing the detector information within 20 ns, with a time resolution smaller than 4.5 ns. In particular, only the first three stations are used for transverse momentum measurements, while the last two stations provides binary information whether the particle passed the absorber material or not. For trigger purposes a 5-hit coincidence in all the muon stations allows to identify high-momentum muons.

1.4 LHCb Upgrade I

The LHCb detector will undergo a major upgrade during the Long Shutdown 2, after the completion of Run 2 in October 2018. The Upgrade I is motivated by the necessity to extend the experiment sensitivity on crucial flavour physics observables, that is still limited by the statistics and with the current configuration would need several years of additional data taking. An increase of the instantaneous luminosity to the value 2 $\times$

10^{33} cm^{-2}s^{-1}, corresponding to a factor five more with respect to the Run 2 conditions, is foreseen. In these conditions the average number of proton-proton collisions at each bunch crossing will be equal to 7.6, leading to higher detector occupancy and radiation level. The maximum readout rate of LHCb is currently limited by the front-end electronics of many sub-detectors: this limitation also strongly motivates the need of a detector upgrade in order to read the full detector at the maximum bunch crossing rate of 40 MHz.
The lateral view of the LHCb Upgrade I detector is shown in Fig. 1.6.

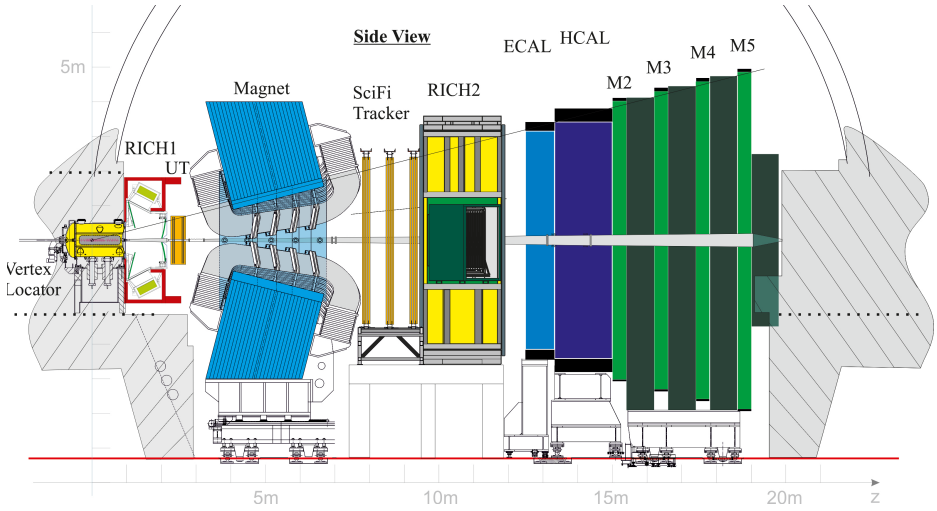

Figure 1.6: Lateral view of the LHCb Upgrade I detector.

Sub-detectors as the SPD, PS, and the M1 station of the muon detector will be removed, since they are partly obsolete and can not be operated in the upgraded conditions. The higher luminosity will lead to higher pile-up, number of tracks and irradiation of the detector. This makes necessary an increase of the granularity in order to still be able to reconstruct particles in the detector.
The most relevant modification consists in fully replacing the tracking system. The VELO is replaced with a new version [20] based on hybrid pixel sensors, instead of strip sensors. The upgraded detector is positioned closer to the beam axis in order to improve the impact parameters resolution. The sensors will have a finer granularity with respect to the current design, in order to cope with the higher particle density, primarily in the region closest to the beam. The upgraded VELO is composed of 26 detector planes perpendicular to the beam axis, arranged in two retractable halves, similar to the current design. The minimum distance from the sensitive are to the beam pipe is 5.1 mm, the pixel size is 55×55 μm^2.
The Upstream Tracker (UT) [21] replaces the TT, but maintains a similar layout. It is based on single-sided silicon strip sensors arranged in four layers in *axial-stereo* configuration. The geometrical acceptance at small polar angles is larger and new silicon sensors with improved radiation hardness and finer granularity are mounted near the beam pipe. The front-end electronics have been redesigned in order to perform the readout at 40 MHz rate.

The IT and the OT are replaced by the SciFi Tracker based on scintillating fibres that are read out by silicon photomultipliers. The layout and configuration of the SciFi Tracker is similar to the IT+OT configuration.
The current trigger system, based on both L0 hardware trigger and high level software trigger, is substituted by a full-software trigger. In fact, the L0 trigger on one hand limits the 40 MHz input rate down to 1.1 MHz, while on the other hand becomes inefficient for hadronic events at increased luminosities. The new trigger is able to process events at 40 MHz and take decisions on the basis of the full event information.
Additional information on the LHCb Upgrade I can be found in Ref. [21].

1.5 LHCb Upgrade II

As discussed in the previous sections, the LHC will undergo a major upgrade to enter the High Luminosity phase during the Run 4 . The LHCb collaboration, aware of the need to fully exploit the flavour physics opportunities of the HL-LHC, proposed a second upgrade, called LHCb Upgrade II to be installed during the LS4 in 2030.
The Expression of Interest for the LHCb Upgrade II is reported in Ref. [22]. The physics case opportunities achievable from the upgrade are reported in Ref. [23].
In the next paragraphs we will report the main contents related to the tracking system upgrade, together with the introduction of precise timing measurements.
With the increase of the instantaneous luminosity the mean number of interactions at each bunch crossing will increase up to $\mu \approx 50$, resulting in a significant increase of the total number of tracks to reconstruct per each event. Significant challenges in terms of data rates, reconstruction quality, and increased radiation damage will be faced by old sub-detectors. The collaboration has already identified some potential solutions that would allow the feasibility for the LHCb detector to work in harsher conditions and the introduction of precise timing detectors has been identified as crucial to help the event reconstruction in conditions of increased pile-up and detector occupancy. The use of precise timing will be essential in the tracking detectors, as the VELO, the Upstream, and the Downstream tracking systems. In fact, a resolution of a few tens of ps per particle, will allow charged tracks and photons to be associated with the correct interaction vertex, thereby suppressing the combinatorial background and also allowing for time-dependent *CP* violating measurements. Moreover, the experiment will also profit from the precise timing in downstream detectors for improving particle identification for low-momentum tracks. Other detector upgrades are proposed such as the replacement of the ECAL with a high granularity tungsten sampling electromagnetic calorimeter, and the instrumentation of the lateral walls of the dipole magnet to significantly increase the detector acceptance for low momentum tracks.

1.5.1 VELO sub-detector with timing at high luminosity for the Upgrade II

At the expected conditions of LHCb Upgrade II with instantaneous luminosity of 2×10^{34} cm^{-2}s^{-1}, the expected pile-up is $\mu \approx 50$, with a total number of 1500-3500 charged

particles within the detector acceptance. The quality of the track and vertex reconstruction performed by the VELO sub-detector will be primarily affected. It is demonstrated that the use of the Upgrade I VELO sub-detector in the Upgrade II environment would not be feasible, leading in particular to a consistent increase of the ghost[2] track rate from 1.6% to 40%. A modest reduction of the tracking efficiency from 99% to 96%, together with a degradation in the impact parameter resolution is expected, as a consequence of the reduced resolution on the primary vertex. Monte Carlo simulation studies for tracking performance at Upgrade II conditions are summarized in Fig. 1.7.

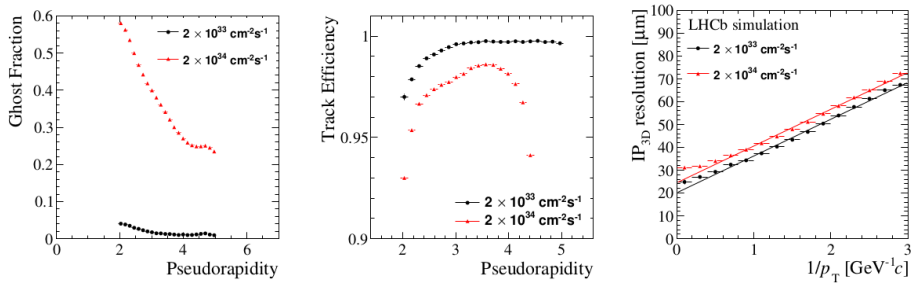

Figure 1.7: Tracking performance of the VELO Upgrade-I sub-detector, with no modifications, at Upgrade I (black points) and Upgrade II (red points) nominal luminosity conditions. Left: ghost fraction as a function of the pseudorapidity. Centre: track efficiency as a function of pseudorapidity. Right: impact parameter resolution as a function of the inverse transverse momentum. Figure from Ref. [22].

Before introducing the precise timing as a viable solution to recover the desired Upgrade I VELO tracking performance, we point out that other less innovative strategies have been considered. As an example, the performance loss can be almost entirely recovered by halving the VELO pixel pitch from the 55 μm baseline value to 27.5 μm, particularly for the innermost region of the pixel sensors, and by reducing the silicon sensor thickness from 200 μm to 100 μm. With these modifications the ghost rate ratio can be reduced to the value of 2%, keeping the tracking efficiency to the acceptable value of 96%. Another solution that would benefit the VELO track reconstruction would consist in removing or redesigning the RF foil, present in the Upgrade I VELO. The RF foil is a thin metal shield whose main purpose is to separate the VELO detector vacuum from the LHCb beam vacuum, suppress the wake-fields (produced by the charged bunch) and to reduce the interference from the bunched beams on the detector electronics. It is important to note that the pattern recognition used for this studies has been only coarsely optimized and there could be space for improving the track reconstruction with a better investigation and redesign of the track finding algorithms.
Although the proposed solutions seem encouraging, it is worth investigating the introduction of precise timing in the VELO, and in general in the other tracking detectors. In fact, even if adopting this solution the problem of the mis-association of tracks to their production vertices (PV) and decay vertices would still be present. The identification of b and c hadrons is mainly based on the reconstruction of their flight distance, in particular

[2] fake tracks reconstructed from spurious hit combinations.

on the presence of a displaced vertex. In conditions of high pile-up, it is easier to incorrectly associate a track from a secondary vertex with a primary vertex from a different pp interaction, hence reducing the reconstruction quality and degrading the time-dependent CP-asymmetry measurements. Adding the precise time information would resolve this problem by providing an additional coordinate for performing the PV association, which currently relies only on space track parameters (a track is associated with the PV with closest impact parameter value). The PV mis-association could be reduced from 20% down to 5% with a timing precision of 50-100 ps. The distribution of multiple PVs in space and time, as an example, is shown in Fig. 1.8.

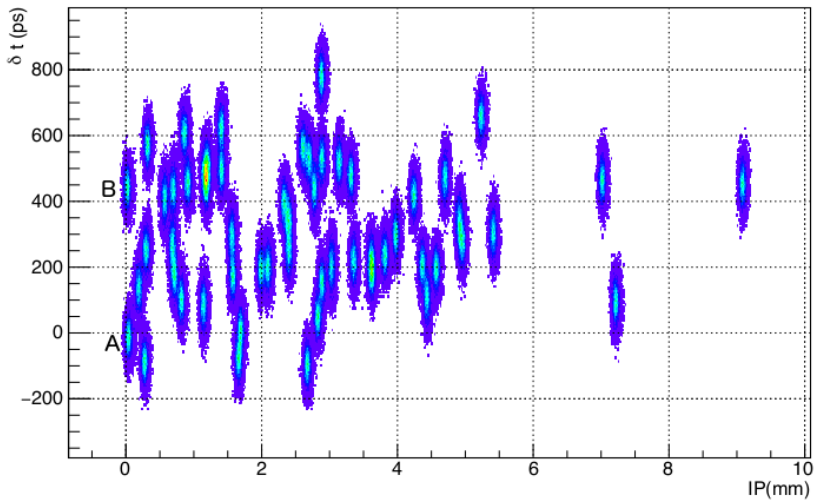

Figure 1.8: Distribution of tracks in space and time originated from different PVs. The introduction of timing helps to separate PVs with similar IP values. Figure from Ref [23].

It has been evidenced that the addition of precise timing provides benefits both to track reconstruction and to the PV association. Moreover the addition of timing to the reconstructed tracks would be helpful for other tasks of the event reconstruction, *i.e.* it could be used to improve the association between tracks reconstructed upstream and downstream of the magnet and/or the association of tracks to hits in sub-detectors not directly involved in the tracking reconstruction, as the TORCH that also features precise timing sensors. R&Ds for pixel sensors with precise timing are well motivated and the possibility to have fast-timing sensors with the characteristics required to be used in the upgraded VELO tracker has to be taken in account. In addition, the timing information of the hits can be used at an early reconstruction stage, *i.e.* in the pattern recognition, to reduce the combinatorics of hits from different pp interactions, allowing to relief the computational workload associated with the crowded events expected during the Upgrade II.

1.5.2 Timing in trigger and data processing

At Upgrade II luminosity the LHCb detector is expected to produce data at rates of $\sim$ 32 Tbps [24]. This amount of data has to be processed in real time by the trigger in order to select relevant events. The rate is expected to be reduced by at least 4-5 orders of magnitude before being stored. The projection of technological improvements indicates as feasible the task of moving this volume of data from the detector to the processing farm by the time of Upgrade II. On the other hand, the data processing will be a challenge, with pile-up of $\mu \approx 50$ and multiple heavy-flavour hadrons produced at every bunch crossing. The traditional trigger strategy based on the selection and the acquisition of bunch crossing events according to inclusive topologies, *i.e.* the presence of a displaced vertex, will no longer be able to significantly reduce the data rate. The Upgrade II data processing strategy should be based on the pile-up suppression providing data reduction by discarding detector hits that are not associated with interesting pp interactions. For this reason there is the necessity to distinguish the reconstructed objects associated with different PVs in a fast manner. The introduction of precise timing in all the sub-detectors would significantly help in this task, in particular mitigating the negative pile-up effects, *i.e.* the PV mis-association being able of separately reconstructing multiple pp events generated within the same bunch crossing. In this scenario, considered the demonstrated ability of LHCb to reconstruct full events with offline-quality in near real-time during Run 2, and considered the technology evolution in the next decade we can consider as feasible the task of triggering at high luminosity also exploiting new computing architectures based on hardware accelerators such as GPGPUs and/or FPGAs to which the IT industry is dedicating a growing interest.

Chapter 2

Fast track finding device for efficient triggering

2.1 Challenge of event reconstruction at the High Luminosity LHC

In this section an introduction to the main challenges of event reconstruction and real-time track reconstruction at 40 MHz in the environment of the High Luminosity LHC is provided.

As already introduced, one of the negative effects of the luminosity increase at HL-LHC is given by the increase of the pile-up and the track multiplicity. In the context of the event reconstruction this will result in a consistent slowdown of all the reconstruction algorithm. Moreover the pile-up increase will lead to a loss of physics performance. An already discussed example related to the tracking is the increase of the PV mis-association but effects will be visible in other contexts as the expected decrease of efficiency in the electromagnetic calorimeters, the reduced selection efficiencies for electrons and photons, for hadronic tau decays and b-jets, an expected worsening of the energy resolution for electrons, photons, taus, jets and missing energy.

A key role for profiting of the HL-LHC is to maintain the excellent efficiency in all the LHC experiments, in particular by upgrading the detectors with higher granularity devices and precise timing. Regarding the trigger and the event reconstruction, the increase of the event complexity has to be efficiently handled in order to minimally affect the processing time. Moreover the pile-up increase will lead to the increase of the ratio between interesting and uninteresting events, reducing the fraction of the events that can be discarded. In fact, the traditional trigger systems at LHC are designed to keep or discard the whole bunch-crossing event; this approach becomes unfeasible for growing luminosities. As an example, in a situation of high pile-up with an interesting signature from a single pp the best option would be to identify and store the information from products of that single interaction instead of storing all the detector information; following this approach would result in a strong data reduction but has to be based on a reliable and fast separation of pp interactions within the same bunch-crossing.

2.2 Examples of fast track finding devices

Fast track finding devices cover a fundamental role for triggering efficiently on interesting physics processes. Generally they rely on fast pattern recognition and simplified track fitting techniques.

The process of pattern recognition is typically serialized and solved by trials. A simple example of pattern recognition of straight tracks is based on the identification of a *track road* connecting one hit in the first sensor to one hit in the last sensor. Hits compatible with the track road are searched in the intermediate layers and a good track candidate is identified if a minimum number of hits are associated with the track road. On the other hand the process of track finding will fail for many track roads identified from spurious combinations of hits. This process has to be performed for different track roads, whose total number is given in first approximation by the product of hits in the first and last detector layer. In this simplified scenario the algorithm execution time is proportional to the second power of the number of hits. This feature could be problematic in trigger applications where having a fast response is crucial. For this reason parallelized approaches are generally more suitable to be applied to fast tracking devices; moreover solutions based on hardware processing are typically faster compared to software solutions.

In the following, we describe some examples of fast tracking devices that already proved being able to perform fast track reconstruction in HEP experiments with reduced latency and whose information can be included and used in trigger applications, together with proposed R&D for future applications. These solution are based on the use of Associative Memory devices or on the implementation of highly parallelized and pipelined tracking algorithms in Field Programmable Gate Arrays (FPGA).

2.2.1 Pattern recognition in Associative Memory devices

Here we describe the working principle of the pattern recognition using Associative Memories (AM) applied, for sake of simplicity, to the reconstruction of 2-dimensional straight tracks. Let's consider the example of a 2-dimensional tracking system composed of five layers. Each sensor provides the measurement of the track hit position within the layer; we consider each layer to be divided into a limited number of spatial bins defined by n_{bins}, smaller than the number of channels. In general a track crossing the detector will produce a hit in one bin per layer, as shown in Fig. 2.1.

Each bin is identified by its coordinate and we define a *pattern* as a set of bin coordinates corresponding to a possible track. In particular, since the bin size is greater than the spatial resolution there is a *one-to-many* correspondence between patterns and candidate tracks, and different tracks in a reduced region of the track parameter space can match the same pattern. If on one hand the resolution is artificially reduced, on the other hand the number of possible patterns is limited to $\sim n_{bins}^2$, while the full track resolution can be recovered in the following steps. Having a limited set of possible patterns, which can be evaluated from simulation, allows to construct the list of all the possible patterns that is called the *pattern bank*, that will be used to perform the task of pattern recognition by comparison of the measured hits in the tracking system to the list of patterns.

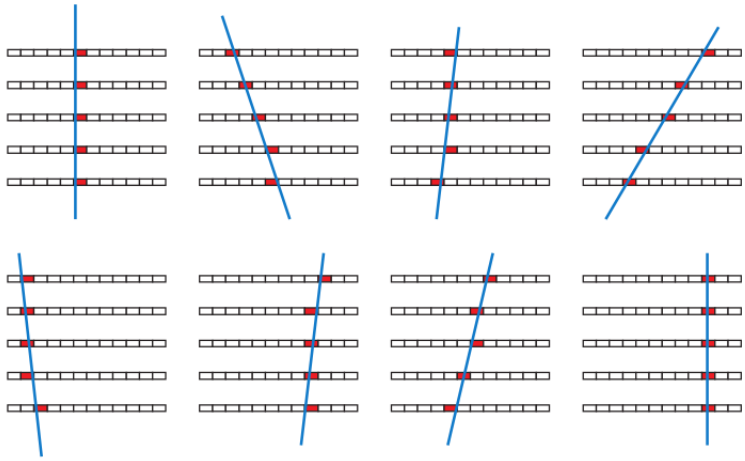

Figure 2.1: Examples of straight tracks (blue lines) matching coarse-grained patterns (red segments). Each track produces a hit in one spatial bin per layer; the number of possible patterns is proportional to the number of spatial bins in which the detector layers are divided.

The AM is a particular type of content addressable memory (CAM) in which the pattern bank is stored and can be implemented in custom ASICs. The AM receives the detector hits in input and compares them to the pre-computed patterns; in particular a pattern is matched if all the bins have been activated by at least on hit. For each matched pattern the AM provides the list of the addresses of matching hits in output.

From a practical point of view, all the measured hits are fed to multiple AM chips over which the complete pattern bank is distributed. The AM chips determine in parallel whether the received hits matched one or multiple patterns. The operation is iterated for all the hits and the pattern recognition process completes as soon as the last hit is processed.

It is important to note that the computing time is independent from the size of the pattern bank, while it is proportional to the time needed to read out and deliver the hits from the detector. The pattern and the hits provided in output from the AM are finally used to perform the track fitting in the following stage of processing, eventually discarding candidate tracks obtained from spurious combinations of hits.

As previously pointed out, the size of the detector bins has to be tuned in order limit the size of the pattern bank and optimized according to the expected trajectories evaluated from simulations. As stated before, the coarse resolution of the patterns does not allow to directly estimate the track parameters but provides a good hint of the track trajectory, since the hits lie in a limited range of coordinates; this makes possible to apply simplified fitting techniques, as the *linearized track fitting*, to evaluate the track parameters. Examples of systems using the AM are the Silicon Vertex Trigger (SVT)[25] of the Collider Detector at Fermilab (CDF)[26], and the Fast TracKer (FTK)[27] of the ATLAS experiment.

2.2.2 Silicon Vertex Trigger at CDF

Here we provide a brief description of the CDF experiment, which implemented a fast tracking device based on Associative Memories for fast track reconstruction, as part of the trigger decision chain.

CDF detector A scheme of the CDF detector is shown in Fig. 2.2.

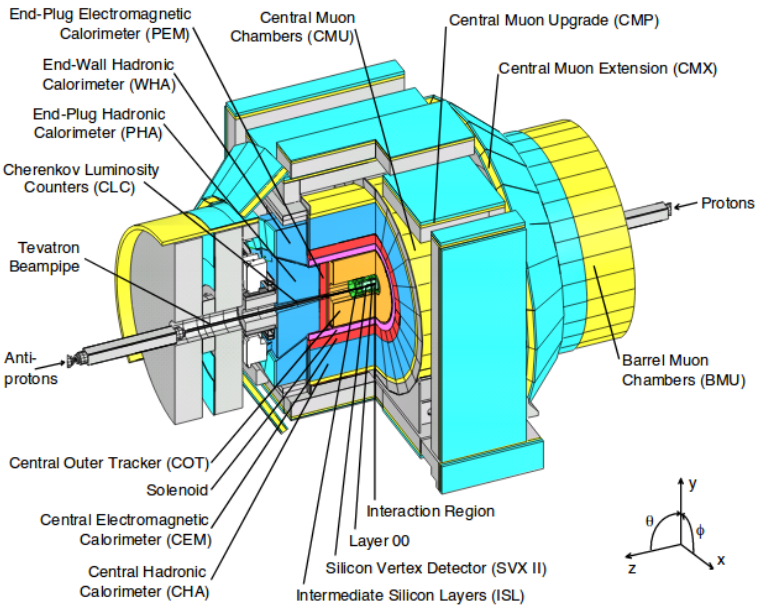

Figure 2.2: Scheme of the CDF detector.

The CDF is a cylindrical detector based on multiple layers of sub-detectors organized as follows, with increasing radius: the Silicon Tracking System and the Central Outer Tracker (COT) placed between the beam pipe and the solenoid magnet, the Electromagnetic and Hadronic Calorimeters and the Muon Detectors placed after the magnet. The solenoid magnet is located at 1.5 m from the beam line, and provides a uniform magnetic field of 1.4 T that is needed for the particles momenta measurement.
The Silicon Tracking System is composed of three concentric cylindrical sub-detectors, the Innermost Silicon Layer (L00), the Silicon Vertex Detector (SVXII) and the Intermediate Silicon Layer Detector (ISL); all of them are based on silicon strip sensors. The L00 layer is placed at a distance of 13.5 mm from the beam axis. The SVXII is based on three layers of double-sided silicon sensors to provide 3-dimensional measurement of the track hits; it is placed at a distance from the beam axis between 25 mm and 106 mm. The ISL is based on three layers of sensors at distances of 200, 220, 280 mm. The Silicon Tracking System has a total of eight layers of sensors measuring the hits position with a resolution of 10 μm and the impact parameter with 40 μm resolution, with a total pseudorapidity coverage of $|\eta| < 2$. The system covers a crucial role in the identification of b hadrons

since their typical signature is given by the presence of a displaced vertex with respect to the primary vertex of the $p\bar{p}$ collision.
The COT is dedicated to tracking at larger distances from the beam axis (from 400 mm to 1370 mm). It is a drift chamber with sensing wires in *stereo* configuration, with half of the wires oriented along the beam axis and half tilted by $\pm 2^\circ$ angle. Tracks in the COT are reconstructed by first identifying three hits compatible with a particle trajectory from the interaction point, adding further hits near the track projection and finally fitting the hits with a five parameter helix. The COT tracks are projected to evaluate the expected position of the hits in the Silicon Vertex Detector and the two best candidate hits in each layer are linked to the COT track. The final track is obtained by selecting the one with best fit by mean of a χ^2 minimization algorithm.
The Electromagnetic and Hadronic Calorimeters measure the energy of particles and jets from hadronization and they are based on alternating layers of scintillators with lead and steel absorbers, respectively. The particles interact in the absorber and produce a shower of lower energy particles that are converted into photons in the scintillators, whose light is collected by photomultiplier tubes.
The Muon Detectors represent the last layer of sub-detectors; they are based on drift chambers and arranged in multiple sub-systems covering different geometrical acceptances.

CDF Trigger System and Silicon Vertex Trigger The CDF Trigger System is based on three levels of processing. Level 1 and Level 2 are completely implemented in hardware while Level 3 is implemented in software. The CDF trigger provides a strong data reduction through fast identification of distinctive signal signatures, many of them are based on the fast track reconstruction of charged-particles in the bending plane of the spectrometer. The event rate is reduced from 2.5 MHz, corresponding to the Tevatron bunch crossing frequency, to 30 KHz at the Level 1, and down to 300 Hz at the Level 2.
The Level 1 trigger requires at least two COT tracks, reconstructed by the *eXtremely Fast Track processor* (XFT), providing 2-dimensional tracks measured in the plane transverse to the beam axis. The tracks are matched to the silicon hits from the SVXII by the Silicon Vertex Trigger (SVT) that is part of the Level 2 trigger. The Level 2 requires at least a 120 μm impact parameter, defined as the distance of a track from the primary vertex. The Level 3 provides a full software confirmation of the previous selection reducing the rate to 75 Hz.
The SVT is an example of fast tracking device able to provide reconstructed tracks with 35 μm impact parameter resolution, performing the full tracking in ~ 15 μs. The same algorithm implemented in software would require ~ 0.1 s, considering the computing power at the time of the experiment[25]. This is possible thanks to a highly parallelized and pipelined architecture, in particular using the AM and performing the track fit in FPGAs.
The SVT receives and processes the hits from the SVXII sub-detector, whose scheme is shown in Fig. 2.3.
The SVXII has a cylindrical symmetry and can be divided in 12 angular wedges, each of them can be processed independently by dedicated hardware; this feature allows a

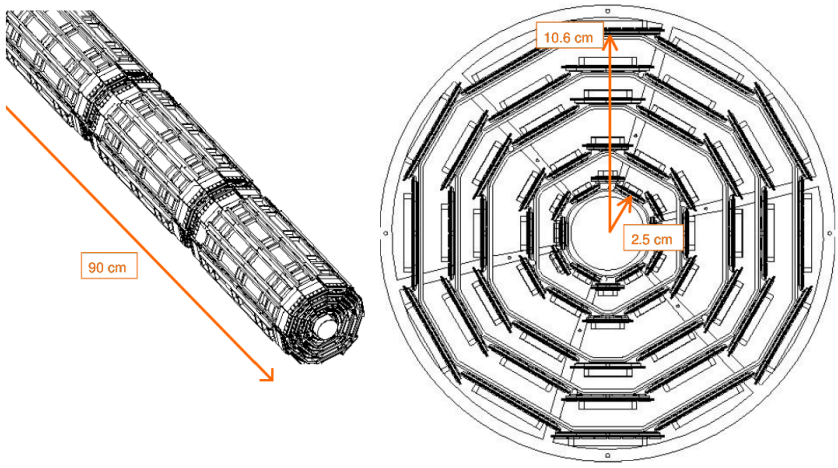

Figure 2.3: Scheme of the SVXII sub-detector.

first level of parallelization of the processing. The presence of tracks in the COT and compatible hits (based on the COT track projection to the SVXII) is required to start the reconstruction of tracks in the SVXII by the Silicon Vertex Trigger system. Each sensor layer is divided in programmable width bins, typically $250 - 700$ μm, while the COT track projection to the outer radius of the SVXII are considered as an extra (virtual) layer, with typical bin size of 3 mm. A pattern bank of 32K most probable patterns is computed offline based on Monte Carlo simulation and loaded into the AM system. The SVT compares the measured hits to the pattern bank an the AM system provides the valid patterns together with the hits of each identified candidate tracks, which are fitted using a linearized track fitting algorithm performed in FPGAs.

2.2.3 Fast Tracker at ATLAS

Here we provide a brief description of the ATLAS tracking system and the trigger system together with the description of the Fast Tracker (FTK), a fast tracking device based on Associative Memories providing the reconstructed tracks to the high level trigger.

ATLAS detector A scheme of the ATLAS detector is shown in Fig. 2.4.
The ATLAS is a cylindrical detector composed of the following sub-detectors: the Inner Detector, the Calorimeters, the Forward Detectors, the Muon System.
The Inner Detector is the main tracking system: it is based on different sub-systems and its scheme is shown in Fig. 2.5.
The Insertable B-Layer (IBL) is the first detector layer, featuring pixel sensors based on different technologies. In the *barrel* region, whose minimum and maximum radius are 31 mm and 40 mm, 3D pixel sensors are used due to the higher radiation tolerance while in the *end-cap* regions it features planar pixel sensors. The pixel size is 50×250 μm^2 for both types of sensors. The IBL is an extra detector added during LS1 to improve the track and vertex reconstruction provided by the Pixel Detector, dedicated to precise

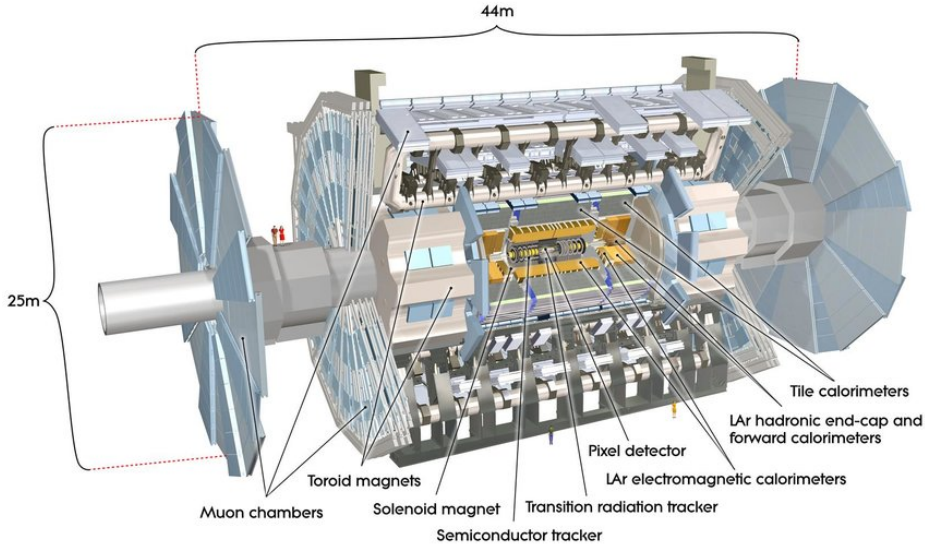

Figure 2.4: Scheme of the ATLAS detector.

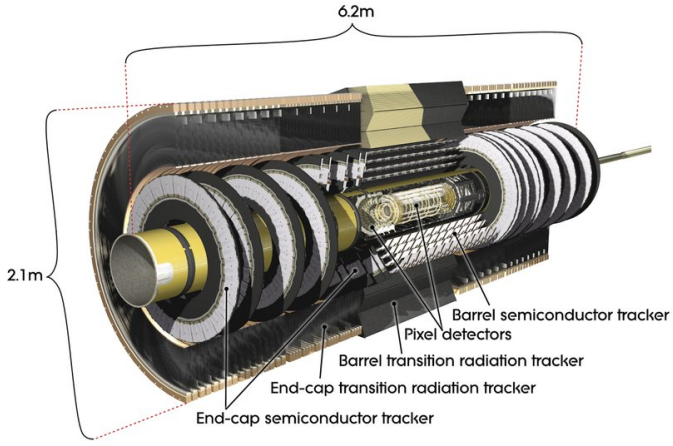

Figure 2.5: Scheme of the ATLAS Inner Detector.

vertex reconstruction. The Pixel Detector is based on silicon pixel sensors and is arranged in three concentric barrel layers at distances of 50.5 mm, 88.5 mm, 122.5 mm from the beam axis and four disks, orthogonal to the beam axis, to complete the angular coverage. The pixel size is $50 \times 400\ \mu\text{m}^2$. The Semi-Conductor Tracker detector (SCT) is positioned after the Pixel Detector and is composed of four layers at 299 mm, 371 mm, 443 mm and 514 mm from the beam axis; these layers are based on silicon strip sensors arranged in *stereo* configuration. Two groups of nine disks each, based on tapered sensors, complete the angular coverage. The Transition Radiation Tracker (TRT) is the outermost tracking

detector. It is based on straw tube detectors, combining both tracking and particle identification capabilities.
The Calorimeter system is composed of the Liquid Argon calorimeter and the Tile hadronic calorimeter, both placed after the solenoid magnet. The first features both electromagnetic and hadronic calorimeter functions either in the barrel region and in the end-cap regions. The Tile calorimeter is a hadronic calorimeter based on scintillating tiles and iron absorbers.
The Muon system is the outermost sub-detector and provides the momentum measurements for muons using the track deflection by three large superconducting magnets. Muon tracks are measured in three cylindrical layers both in barrel and forward/backward regions.

ATLAS Trigger System and Fast Tracker The ATLAS trigger system consists of two levels: a hardware based Level 1 (L1) trigger and a software based High-Level trigger (HLT). At each level uninteresting events are rejected in order to efficiently reduce the amount of data in the final storage. The L1 trigger performs the event selection based on the coarse granularity information from the calorimeter and the muon detector, reducing the event rate from 40 MHz to 100 kHz. The L1 trigger uses custom electronics to determine Regions of Interest (RoIs) to be processed in the HLT, within a decision time of 2.5 μs. The HLT applies more sophisticated selection algorithms using the full granularity detector information in either the previously identified RoIs or the full event, reducing the rate of accepted events from 100 kHz to 1 kHz, within a decision time of about 200 ms.
The FTK is a fast hardware-based tracking system designed to perform track reconstruction using the information from the Inner Detector. It is based on the use of AMs following the CDF SVT approach and experience, with some different technical features. The FTK provides the track information to the HLT, which processes the events selected by the L1 trigger. This information is used to improve many trigger selections that require the full-event tracking information, *i.e.* the identification of b-jets or decay modes with b quarks in the final state.
As in the case of the SVXII in the CDF experiment, the geometry of the Inner Detector has a cylindrical symmetry and the processing of the hits and tracks from different angular sectors is performed in parallel by dedicated hardware. In particular the Inner Detector sensors are organized in 64 overlapping *towers*, given by 4 longitudinal blocks and 16 angular wedges (22.5° plus 10° overlap).
The FTK receives the hits from the twelve layers of the Inner Detector in the barrel region; the clustered hits are sorted and distributed to the 64 processing regions. The hits from the eight inner layers are fed to the AM system and compared to a pattern bank of $\sim O\left(10^9\right)$ patterns, similarly to the case of the CDF SVT. For each identified valid pattern the full resolution of the hits is used to evaluate the track fit parameters by mean of a linearized track fit algorithm performed in FPGAs, discarding low-quality tracks based on a χ^2 cut. Finally the hits from the remaining four layers are added to the candidate track and the full track is fitted and provided to the HLT.
In the ATLAS FTK, the AM pattern recognition follows a refined approach with respect

to what previously described. Each detector layer is divided in bins, here called *Super Strips*. A Super Strip comprises $\sim 24 \times 36$ pixels or ~ 24 strips, in the Pixel Detector and in the Semiconductor Tracker, respectively. The main difference with the CDF SVT is the use of *variable resolution patterns*: this results in the possibility to change the width of the bins, allowing for more flexibility in the construction of the pattern bank. An illustration of the working principle of the variable resolution patterns is shown in Fig. 2.6. In the example the same set of tracks matches different number of patterns depending on the size of the bins in the different layers; this feature allows to optimize the number of patterns in the pattern bank, in particular dedicating more patterns to discriminate more probable track trajectories.

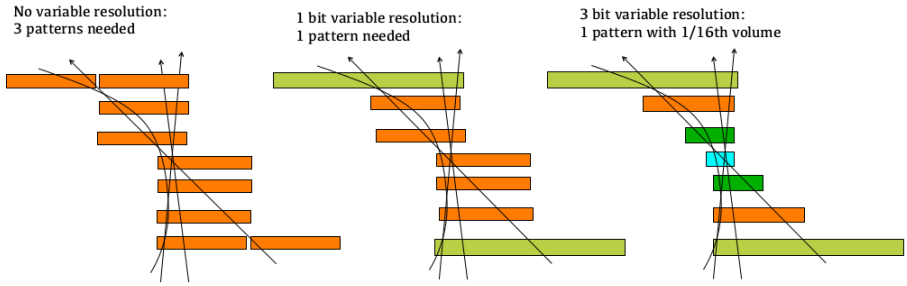

Figure 2.6: Variable resolution patterns. The same set of tracks matches a different number of patterns, depending on the size of the bins.

The ATLAS FTK is currently under commissioning in order to be fully operating during Run 3. A tower (processing one of the 64 sectors) of the final FTK has been integrated and tested, processing data from real events. Results from the operation of the FTK are reported in Ref. [28].

2.2.4 CMS Track Trigger based on FPGA

In order to cope with the harsher conditions provided by the HL-LHC, the CMS experiment is developing a new tracking detector together with a track finding algorithm implemented in FPGA, to allow to reconstruct charged particle tracks with low latency and to include the tracking information in the Level 1 (L1) trigger. Similarly to the other LHC experiments, the current L1 trigger of CMS is based on the identification of single muons, electrons and jets, whose rate is expected to exceed the current front-end capabilities at High Luminosity, and increasing the trigger thresholds would produce a loss of the physics performance. On the other hand the inclusion of the tracking information would provide several additional improvements, while reducing the L1 trigger rate and keeping high performance. It would allow to add the track isolation for leptons and photons, to improve the momentum measurement and vertex association for leptons, and to perform the vertex identification for the hadronic objects.
In the following we will give a brief description of the CMS experiment, the upgrade of the CMS Tracker and the description of the demonstrator of a fast track finding device

based on FPGA.

CMS detector A scheme of the CMS detector is shown in Fig. 2.7.

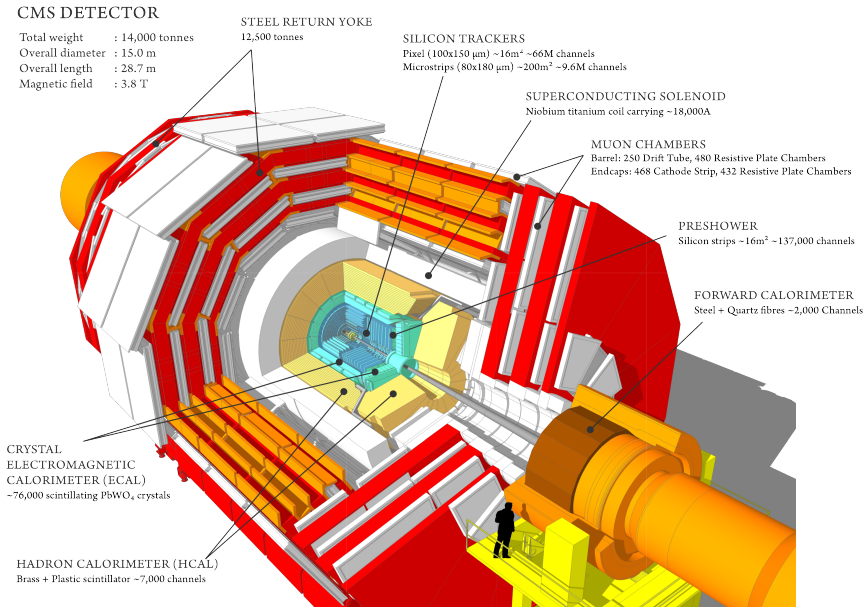

Figure 2.7: Scheme of the CMS detector.

The CMS is a barrel detector based on different sub-detectors in the following order, starting from the beam line towards the outer region: the Inner Detector, the Electronic Calorimeter and Hadronic Calorimeter are located within a superconducting solenoid magnet with internal diameter of 6 m, providing a uniform 3.8 T magnetic field; the magnetic flux outside the solenoid is returned through a yoke of three layers of steel, interleaved with the Muon Detector system; Forward Calorimeters are placed after the Muon Detector in the end-cap regions.
The Inner Detector is composed of the Inner Tracker (IT) and the Outer Tracker (OT) . The Inner Tracker is based on silicon pixel sensors with pixel size of 150×100 μm^2, with three layers covering the barrel regions at distances between 43 mm and 110 mm from the beam line, and two disk layers for each end-cap. The Outer Tracker is based on single-sided and double-sided silicon strip sensors, with pitch ranging from 83 μm to 205 μm: ten layers cover the barrel regions while twelve layers cover the end-cap regions extending the pseudorapidity acceptance to $|\eta| < 2.5$.
The Electromagnetic Calorimeter (ECAL) is based on fine granularity lead tungstate tapered crystals that induce electromagnetic cascades from photons and electrons. The scintillation light is reflected within the crystals, and collected and measured by silicon photomultipliers. In the end-cap regions, the Electromagnetic Preshower (ES) is placed before the ECAL. It is a sampling calorimeter composed of lead and silicon layers dedicated to separate high energy photons from the primary interaction from π^0 decays at

small angles. Thanks to its finer granularity, with respect to the ECAL, it is able resolve the two photons from π^0 decays individually.
The Hadronic Calorimeter (HCAL) is placed after the ECAL and is dedicated to the reconstruction of the energy and position of the QCD jets and neutral hadrons. It is based on layers of brass plates and plastic scintillator tiles. The HCAL covers the barrel and the end-cap regions, while the coverage is extended by the Forward Calorimeter (FC) to $|\eta| < 2.5$. The FC adopts the same technology and, differently from the rest of the HCAL, is placed in the very forward region after the Muon Detector.
The Muon Detector is placed outside the solenoid magnet; four layers of steel are used to return the magnetic flux providing a magnetic field of 2 T. The steel layers are interleaved with three layers of gas chamber detectors based on different technologies: Drift Tubes are used in the barrel region where the magnetic field is uniform, the neutron-induced background and the muon rate are small; Cathode Strip Chambers are used in the end-cap regions where the muon rate and background are higher; Resistive Plate Chambers are used in both regions.
The CMS trigger system consists of a Level 1 (L1) hardware trigger able to reduce the event rate from 40 MHz to 100 kHz, and the High Level Trigger (HLT) based on software that reduces the accepted rate of events down to 1 kHz.
The L1 Trigger consists of two sub-systems performing trigger decisions based on the data from the calorimeters and from the muon detectors, the Micro Global Trigger (MicroGT) combines the outputs of the calorimeter and muon triggers and provides the final decision within a latency of ~ 4 μs. The L1 trigger has been updated before the Run 2 of LHC and it features large FPGAs mounted on custom data processing boards, equipped with high speed optical links with up to 1 Tbps bandwidth per board. In fact FPGAs are well suitable for implementing sophisticated algorithm with pipelined and parallel architectures at a reduced cost compared to custom ASIC development and keeping high flexibility for the revision or the implementations of new features.
The HLT uses the complete detector information providing decisions within a latency of ~ 4 ms, by implementing simplified reconstruction algorithms with respect to the full online event reconstruction. The architecture of the HLT is based on the evaluation of sequential reconstruction and filter stages, starting from the ones that mostly reduce the number of events that will undergo higher complexity stages as the track reconstruction.

CMS Track Trigger R&D for HL-LHC During the Run 4, the expected pile-up at CMS experiment is estimated to be between 140 and 200 and an upgrade of the CMS detector is scheduled during the LS3. In particular the Inner Tracker and the Outer Tracker will be replaced, and the High Granularity Calorimeter (HGCal) will replace the end-cap ECAL. The tracking detector will feature higher granularity and more radiation hard sensors to cope with the increased pile-up and radiation damage with respect to the current detector.
The new tracker is designed with the purpose of including tracking information in the L1 trigger to keep the L1 acceptance rate below 750 kHz without reducing the sensitivity to interesting physics. In fact it is expected that even at the higher expected pile-up value of 200, the inclusion of the track reconstruction for high p_t tracks ($p_t > 3$ GeV) will allow

to keep the acceptance rate at 750 kHz, which would exceed 4 MHz without the tracking information.

One of the most innovative upgrades in the tracking system will be the use of the so-called p_t-Modules that allow for real-time estimate of the particle momentum for high-p_t tracks. In particular these are based on two closely spaced layers of silicon sensors that are read out by the same front-end ASIC that is able to identify hit doublets (*stubs*) compatible with a high-p_t track. Two types of sensors have been developed: one implementing two silicon strip layers (2S) with strip size of 50 mm $\times$ 90 μm, and the other implementing a silicon strip and a silicon macro-pixel layer (PS) with strip size of 24 mm $\times$ 100 μm and pixel size of 1.5 mm $\times$ 100 μm; the PS modules, due to their finer granularity, will be implemented in regions with higher occupancy. The use of p_t-Modules will also allow for a local data reduction in the front-end electronics, providing a pre-selection of the hits. The scheme of the proposed geometry of the upgraded CMS tracking system is shown in Fig. 2.8. In particular, it features tilted sensors in order to optimize the stub efficiency, the material budget is reduced with respect to the current design, and the pseudorapidity coverage is extended from $|\eta| < 2.4$ to $|\eta| < 4.0$, increasing the forward acceptance and mitigating pile-up effects in the forward region.

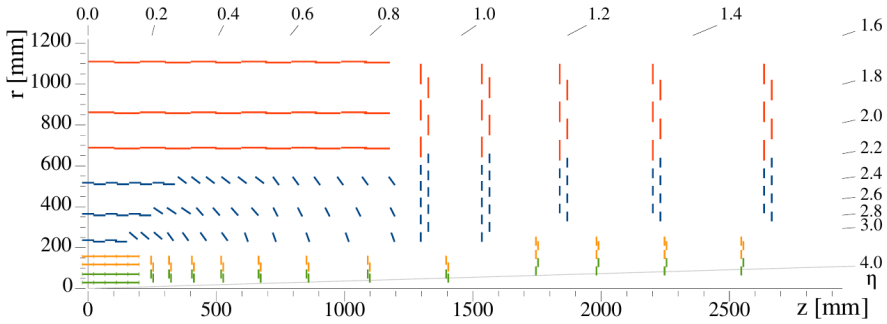

Figure 2.8: Scheme of the CMS Phase II tracking system featuring *tilted* sensors in order to optimize the stub efficiency.

Together with the new tracker, the CMS collaboration is developing a fast track finding device. Different approaches are under study based on the use of AM and FPGAs and on full FPGA-based architectures. In the following, we will present an overview of the implementations based on FPGA. Two different solutions are under investigation, one based on the *Tracklet* approach, the other based on the use of the Hough Transform.

In the following we report about the full FPGA-based solution based on the *Tracklet* approach; the information are reported from Ref. [29].

The algorithm is based on the identification track seeds formed from pairs of stubs in adjacent layers (or end cap disks), called *tracklets* in order to estimate the initial track parameters. The tracklets are then projected to the other layers (or disks) and compatible stubs are associated with the track. A track candidate is identified from at least four stubs. After the tracks have been identified a linearized χ^2 fit is performed, using the information from all the stubs belonging to the track candidate and providing the final

track helix parameters. The algorithm is massively parallelized, in particular the detector is divided in 28 angular wedges processed by independent hardware. The architecture is fully pipelined and operating at a single reference clock of 240 MHz and the processing is time multiplexed by a factor 6, which means that each board receives data from a new event once every 150 ns. Moreover it is designed to work with fixed latency that means that each processing step can perform a fixed number of operations while the processing of eventually exceeding data is truncated; nevertheless the system is designed to minimize the effect of the truncation. In the final design each detector sector will be processed by one ATCA board featuring Xilinx Virtex UltraScale+ FPGAs.

A demonstrator system based on four CPT7 [30] μTCA boards, each one featuring a Xilinx Virtex-7 FPGA and Xilinx Zynq-7000 SoC has been developed and confirmed the feasibility of the system, providing a total measured latency of 3.3 μs, compatible with the maximum required latency budget of 4 μs.

This system, together with the second approach based on the Hough Transform, represents an interesting implementation of track finding devices fully based on FPGA capable to perform the track reconstruction with high efficiency and low latency at 40 MHz rate, at the expected HL-LHC conditions.

Chapter 3

Artificial retina algorithm for 2D fast track reconstruction

In this chapter we present the artificial retina algorithm for fast track reconstruction of 2-dimensional tracks. The algorithm is inspired from neurobiology and it is based on the parallel evaluation of the response of a grid of cellular units, tuned to recognize specific track trajectories. It has been implemented in hardware using a custom acquisition board based on commercial FPGA, used for both the readout and processing of data from a 2-dimensional tracking system based on silicon strip sensors. Details about the custom data acquisition board and the silicon telescope construction are given. The testbeam results of the first real-time embedded tracking system based on the artificial retina algorithm are also reported.

3.1 Artificial retina algorithm

The artificial retina algorithm for fast track reconstruction in high energy physics has been first proposed and described in Ref. [31] and is inspired by the mammals low-level mechanism for recognizing straight edges. In particular, in the visual cortex dedicated neurons are tuned to recognize particular shapes and orientations of the objects and produce an electrical response that is proportional to the good match of the image to the feature they are tuned to recognize. In the artificial retina, cellular units distributed over the space of the track parameters are tuned to identify specific charged particles trajectories and provide a response on how well a set of hits matches a specific track hypothesis. In this section, we describe the artificial retina algorithm for a 2D tracking system and briefly discuss the possibility of its extension to 3D tracking systems.
Let's consider a two dimensional tracking system based on multiple layers of silicon strip sensors orthogonal to the z-axis as shown in Fig. 3.1. We define (x_f, z_f), and (x_l, z_l), the coordinates of the intersections of the tracks in the first and last layer, respectively. We also define the constant terms $z_\pm = (z_f \pm z_l)/2$ that depend on the geometry, and the track parameters $x_\pm = (x_f \pm x_l)/2$ used to describe the equation of a 2D track

$$x(z) = x_+ + x_-(z - z_+)/z_- \ . \tag{3.1}$$

A grid of cellular units covers the space of the track parameters (x_+, x_-). In particular each cellular unit, identified by its index (i, j) in the grid, is associated with the track

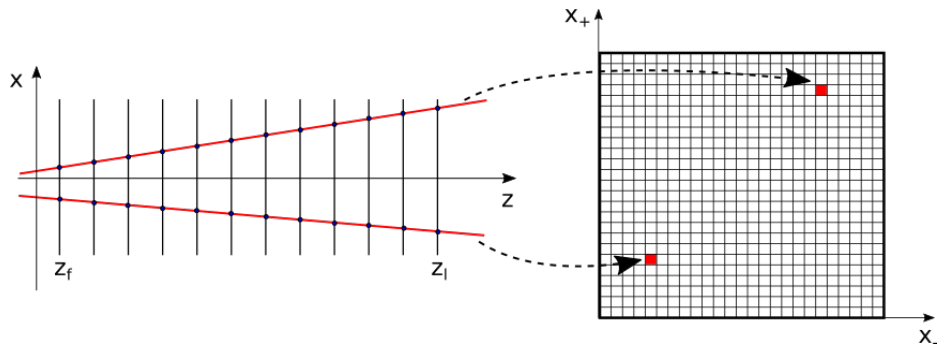

Figure 3.1: (Left) Layout of a 2-dimensional tracking detector based on multiple sensors placed along the z-axis and measuring only the x coordinate of the hits. (Right) Grid of cellular units uniformly distributed over the space of the track parameters. Each point of the grid corresponds to a track hypothesis.

parameters (x_{i+}, x_{j-}) and a set of receptors on the tracking detector, identified by the intersections of the track in the detector layers. Each cellular unit evaluates the distances s_{ijk} of its track receptors from the measured k-th hits, given by

$$s_{ijk} = x_k - x_{i+} - x_{j-} \left(z_k - z_+\right) / z_- \ , \tag{3.2}$$

and evaluates a response according to a Gaussian field, $\exp(-s_{ijk}^2/2\sigma^2)$, where σ is the width of the Gaussian to be adjusted for optimal response. The weight function W_{ij} of the (i,j)-th cell is then defined as the sum over the non-negligible contributions

$$W_{ij} = \sum_k W_{ijk}, \tag{3.3}$$

where W_{ijk} is the contribution from the k-th hit, defined as

$$W_{ijk} = \begin{cases} \exp\left(-\frac{s_{ijk}^2}{2\sigma^2}\right) & \text{if } |s_{ijk}| \leq 2\sigma, \\ 0 & \text{if } |s_{ijk}| > 2\sigma. \end{cases} \tag{3.4}$$

The evaluation of the weight function is performed in parallel by the cellular units and a candidate track is identified by a local maximum of the response in the grid; the value of the maximum response is also required to be over a certain threshold according to the minimum number of hits required to identify a track. The reconstructed track parameters $(x_+, x_-)_{\text{rec}}$ are obtained by interpolations of the weight values adjacent to the maximum, along the x_+ and x_- axes. In particular the track parameters are determined by means of a Gaussian interpolation, defined as

$$x_{-,\text{rec}} = x_{j-} + \frac{\Delta_{x_-}}{2} \frac{\ln(W_{ij-1}/W_{ij}) - \ln(W_{ij+1}/W_{ij})}{\ln(W_{ij-1}/W_{ij}) + \ln(W_{ij+1}/W_{ij})} \ , \tag{3.5}$$

$$x_{+,\text{rec}} = x_{i+} + \frac{\Delta_{x_+}}{2} \frac{\ln(W_{i-1j}/W_{ij}) - \ln(W_{i+1j}/W_{ij})}{\ln(W_{i-1j}/W_{ij}) + \ln(W_{i+1j}/W_{ij})} \ , \tag{3.6}$$

where $\Delta_{x_\pm}$ is the granularity of the grid of the track parameters.

The typical response of the artificial retina algorithm for a single track crossing a 8-layer detector in shown in Fig. 3.2. The contributions from different hits to the response, sum up in a single point of the space of the track parameters corresponding to the parameters of the tracks to be measured. A local maximum, whose value corresponds approximatively to the number of contributing hits, as expected from Eq. 3.3, can be identified and the parameters recovered by interpolation of the response.

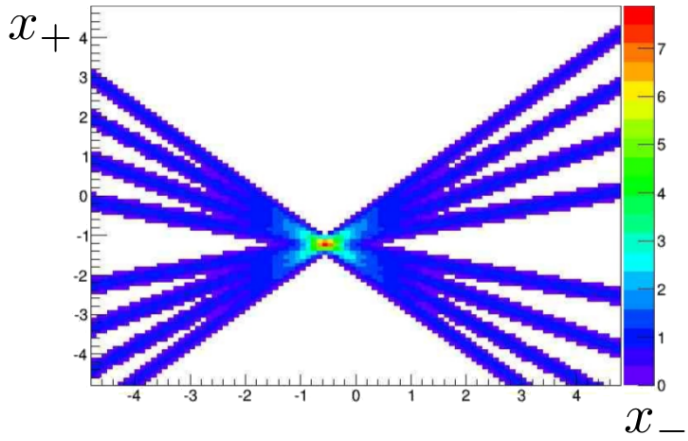

Figure 3.2: Typical response of the artificial retina to a single-track event.

Possible extension to a 3D tracking system The principle of the described algorithm could be extended to 3D tracking systems based on pixel sensors and eventually the time information of the hits could be added. Nevertheless it is worth noting that a 3-dimensional track is described by four track parameters that can be identified as (x_+, x_-, y_+, y_-) extending the definition given in Eq. 3.1 to

$$\begin{aligned} x(z) &= x_+ + x_-(z - z_+)/z_- \ , \\ y(z) &= y_+ + y_-(z - z_+)/z_- \ . \end{aligned} \tag{3.7}$$

Using this simplified extension we would need to distribute the cellular units over the 4-dimensional space of the track parameters (x_+, x_-, y_+, y_-). This would be equivalent to allocate a 2-dimensional sub-grid of cellular units distributed over the (y_+, y_-) space for each point of the grid covering the (x_+, x_-) sub-space. In order to use this approach we need to extend the hit-receptors distance from Eq. 3.2 to an Euclidean 2-dimensional distance. The rest of the algorithm would remain unchanged that means a candidate track is still identified by a local maximum of the response and the track parameters are recovered by interpolation of the weight function along the four axes.
This approach is theoretically feasible but it is worth noting that the amount of needed resources, *i.e.* the number of cellular units, would scale as the inverse fourth power of the grid step if we don't consider any geometrical constraint. As an example at the LHC the proton-proton interaction region is limited in space; this reduces the space of the track

parameters of the reconstructible tracks and allows to distribute the cellular units over a limited region of the full 4-dimensional (x_+, x_-, y_+, y_-) space.

3.2 Artificial retina architecture

Here we describe the device architecture and the firmware implementation of the artificial retina algorithm in FPGA. The system is composed of three main blocks: the *Switch* receives the data from the Data Acquisition (DAQ) and delivers to the regions of cellular units where the hits are expected to produce a non-negligible response; the *Engines* evaluate in parallel the responses associated with the cellular units; the *Track Fitter* evaluates the parameters of the candidate track by interpolation of the weight values of the identified local maxima and their neighbour cells. The generic scheme of the architecture is shown in Fig. 3.3.

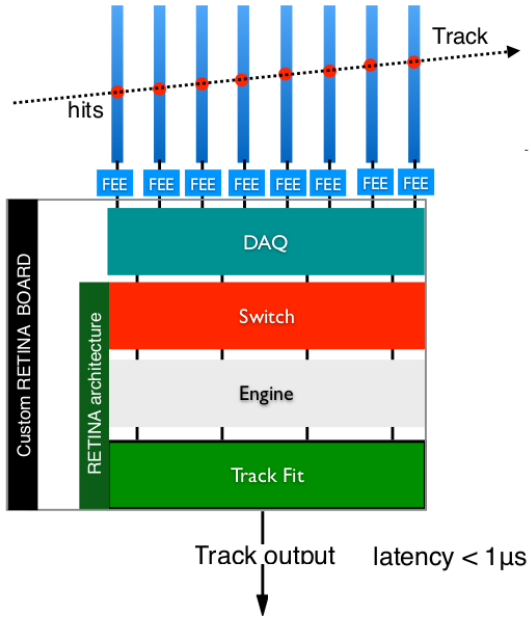

Figure 3.3: Generic scheme of the artificial retina architecture. The DAQ performs the readout of the sensors and delivers the hits to the artificial retina architecture that is divided into three main blocks: the Switch, the Engines and the Track Fitter. The device provides the reconstructed track information in output.

Switch The purpose of the Switch is to deliver the hits from the DAQ to the Engines as fast as possible. In particular the Switch distributes the hits only to the regions of cellular units expected to provide a non-negligible response according to the Eq. 3.4, optimizing the flux of data to be sent to the Engines. In order to describe the behaviour of the Switch, we define a *Group* as a set of adjacent strips of the telescope sensors, without any overlap between different Groups. In this way a hit in the detector is associated with

one and only one Group. In particular the Group information corresponds to the most significant bits (MSB) of the hit data. We also define an Engine Region as a set of Engines covering part of the space of the track parameters (x_-, x_+); even in this case there is no overlap between the Engine Regions. A hit is distributed to all the Engines in an Engine Region according to precomputed data paths evaluated offline and stored into *Look-Up Tables* (LUT) within the Switch logic. In particular, the data path is evaluated according to the Group information only. This means that a hit is distributed to all the Engine Regions where any of the hits in the Group would produce a non-negligible response in at least one Engine.
The basic unit of the Switch is the *2x2 Sorter*. It has two inputs and two outputs and is composed of a layer of *2-way Dispatchers* and a layer of *2-way Mergers* connected as in Fig. 3.4. The Switch is composed of a full-mesh network of these modules organized in such a way that hits from any input can be delivered to any of the Switch outputs.

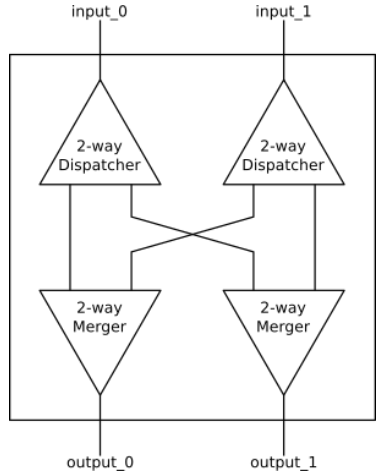

Figure 3.4: The 2x2 Sorter is composed of two 2-way Dispatchers that receive data from the inputs, and two 2-way Mergers that receive data from the 2-way Dispatchers and are connected to the output. The internal connections are organized as shown in the picture.

The 2-way Dispatcher has one input and two outputs. For an incoming hit, it compares the Group information to a LUT that contains the pre-computed data paths and provides a 2-bit word that identifies if the hit has to be forwarded to zero, one or both the outputs. The hit data is then forwarded to the corresponding outputs.
The 2-way Merger has two inputs and only one output and it is meant to receive the hit data from different 2-way Dispatchers, as shown in the 2x2 Sorter scheme in Fig. 3.4. If the 2-way Merger receives data from only one input, the data is delivered to the output; if both the inputs are active, one data is delivered and the other is stored and delivered during the next available clock period.

Hold logic in the Switch modules An *hold* logic is implemented in the Switch in order to prevent data losses when a module is busy and not able to accept data. The hold logic is based on the back propagation (with respect to the hit data flow) of signals

to communicate the status of the module. The simplest example in which the hold signal is asserted is when both the inputs of a 2-way Merger are receiving data during a clock cycle: in this case only one hit can be delivered to the output, while the other has to be stored into the 2-way Merger buffer. The module will not able to accept any data until, at least, the next clock cycle. Another general case in which the hold signal is asserted is when a module is trying to forward a data to an output port from which the hold signal is received. The use of the hold logic in the Switch ensures that none of the hit data is lost, unless the hold signal back propagates till the Switch inputs that receives data from the DAQ.

Engine An Engine corresponds to the hardware implementation of the cellular unit introduced in the algorithm description. A *FanOut* is used to connect one Switch output to all the Engines within the same Engine Region. The Engine logic is shown in Fig. 3.5.

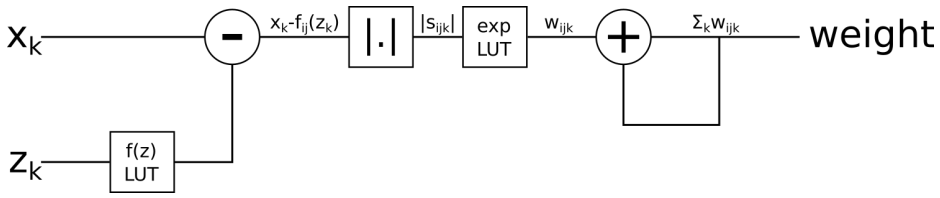

Figure 3.5: Logic scheme of the artificial retina Engine. The Engine receives the hit data, evaluates the Gaussian response based on the distance from the track receptor and sums the value to the weight function.

We recall that a cellular unit corresponds to a pre-computed track and is associated with track receptors corresponding to the intersection of the track with the sensor, at each detector plane. For each incoming hit (z_k, x_k), the evaluation of the Gaussian response is performed in different pipelined stages:

- the position of $f_{ij}(z_k)$ of the track receptor in the plane at $z = z_k$ is evaluated, according to a LUT;
- the distance from the hit to the track receptor is evaluated as $s_{ijk} = x_k - f_{ij}(z_k)$;
- the absolute value of the distance, namely $|s_{ijk}|$, is evaluated;
- a LUT provides the evaluation of the Gaussian response W_{ijk}; if the distance is greater than 2σ the evaluated response is zero;
- the response is summed to the total weight function W_{ij} of the Engine.

The Engines work in parallel and they are independent from each others during the evaluation of the Weight function. When an *End Of Event* signal is asserted from the DAQ, after the detector readout is complete, each Engine compares the values of its Weight function to a *threshold* (thr) and with the value of its first neighbour cells. If a local maximum over threshold ($W_{ij} > thr$) is identified, the coordinates of the Engine, the value of the Weight function and the value of the neighbours' response are provided

to the *Track Fitter*. Finally, the Engine resets the Weight function value and is ready to process new hits from the following readout of the tracking detector.

Track Fitter The Track Fitter receives the Weight function values from the Engines corresponding to the local maxima of the artificial retina response. A Track Fitter is connected to each Engine Region through a *FanIn* module. Two independent evaluations are performed in order reconstruct the (x_-, x_+) parameters of the candidate tracks, according to the Eq. 3.5 and the Eq. 3.6. The logarithmic terms of the interpolation equations are evaluated using a LUT, in order to improve the speed performances. After the track parameters of the identified track have been evaluated, these are sent to a PC and stored to disk or they can be used as input for further processing.

3.3 Tracking system prototype

A silicon strip telescope together with a custom DAQ board based on FPGA, in which the artificial retina algorithm has been implemented, have been developed and built at INFN Milano within the INFN RETINA project. The prototype has been tested on a particle beam at the CERN Super Proton Synchrotron (SPS) in November 2015 as a hardware demonstrator of the artificial retina capabilities. Details about the silicon strip telescope, the DAQ board and the testbeam results are given in the following.

3.3.1 Silicon strip telescope

The telescope is based on 8 planes of single-sided silicon strip sensors. The sensors can be arranged in different configurations: all with parallel strips for 2D track reconstruction or with parallel and perpendicular strips for 3D track reconstruction. We will refer to these configurations as *2D configuration* and *3D configuration*, respectively. The algorithm described in Sec. 3.1 is intended for reconstruction of 2D straight tracks, so the detector planes have been arranged to measure one track coordinate only.

Sensor module The sensors used for the telescope construction are single-sided p-in-n CMS-OB2 [32] silicon strip sensors. These sensors were originally produced by ST Microelectronics for the Outer Barrel of the CMS experiment. The silicon sensor has 512 channels with a strip of 183 μm pitch, the nominal thickness is 500 μm and the active area is 93.9×91.6 mm^2. The sensor is mounted on an aluminium frame as shown in Fig. 3.6. The aluminium plate also hosts the front-end hybrid board (*TTHybrid*) and a custom pitch adapter to connect the sensor, with a pitch of 183 μm, to the TTHybrid, with a pitch of 112 μm. The TTHybrid is the front-end board used in the LHCb experiment for the readout of the *Tracker Turicensis* (TT) sub-detector modules; the Beetle chip [33] is a custom ASIC that provides the analog readout of 128 channels at a sampling rate of 40 MHz. The TTHybrid is connected to a custom DAQ board through an adapter card via a VHDCI (*Very High Density Interconnect*) cable.

Figure 3.6: Sensor module hosting the silicon sensor, the pitch adapter and the front-end electronics.

Telescope layout at SPS A picture of the silicon strip telescope mounted at the CERN SPS during a testbeam in November 2015 is shown in Fig. 3.7.

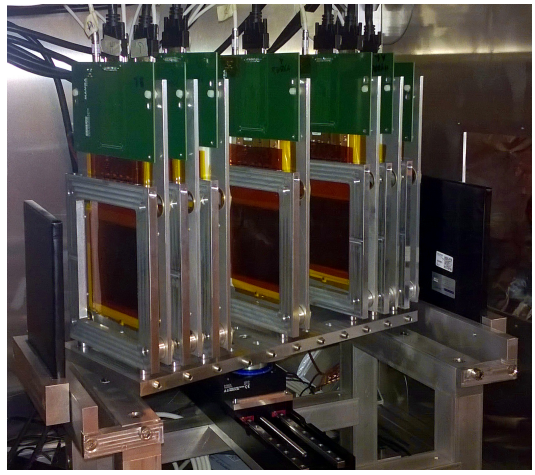

Figure 3.7: Silicon strip telescope mounted at SPS. The telescope hosts 7 planes arranged in a 2D configuration. Two scintillators provides the trigger and linear and rotation stages allow to move the system.

In this configuration the telescope hosts 7 sensors arranged as two symmetrical arms composed of 3 sensors, with one sensor in the middle. The distance between the planes is 4 cm within the arms, while the distance between the central sensors and the nearest ones is 8 cm. Two plastic scintillators, mounted before the first and after the last sensor plane, are used in coincidence to provide the trigger. A linear and a rotation stage are used to move the telescope in order to study the performance of the fast tracking device

for tracks crossing the systems in different position and at different angles. The whole system is enclosed in a light-tight box with an active cooling system.

MAMBA DAQ board The telescope is read out using a custom DAQ board called *Milano Advanced Multi Beetle Acquisition* (MAMBA) board, shown in Fig. 3.8.

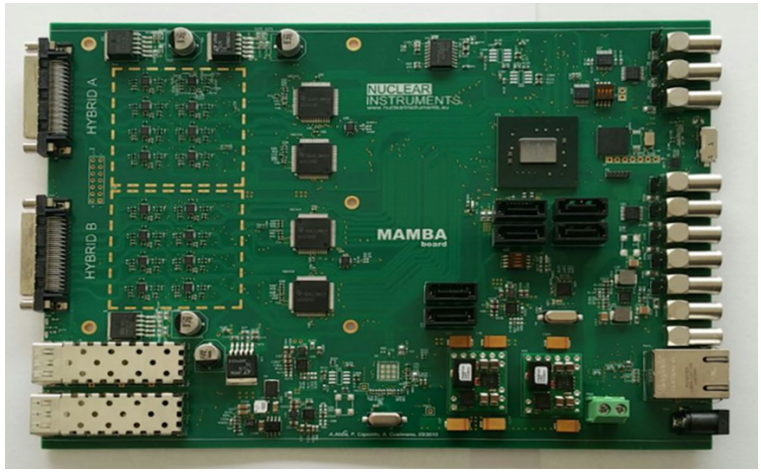

Figure 3.8: Picture of the MAMBA DAQ board for the readout of the silicon strip telescope and for the implementation of the real-time tracking algorithm.

The MAMBA board is based on a Xilinx Kintex 7 (XC7K410T) FPGA operated at 400 MHz clock and it is equipped with eight 4-channel 12-bit analog-to-digital converters (ADC) for the digitization of the analog signals from the Beetle ASICs. The Beetle ASICs are clocked at 40 MHz and configured via I2C protocol. Each one performs the readout of 128 silicon strip signals through an analog line for a total of 4096 channels. In this way one board can control and read out all the telescope planes. The DAQ board performs the internal coincidence between two transistor-transistor logic (TTL) signals from two plastic scintillators that provide the trigger to the system at a maximum rate of 280 kHz. The rate is limited by the maximum accepted trigger rate of the Beetle chips, due to the time needed to read out the strip signals from the Beetle analog pipeline. Other configurable LEMO connectors can be used for additional input/output to synchronize the board with external systems or with another MAMBA DAQ board, receiving or forwarding the clock and reset signals. A time-to-digital converter (TDC) is implemented to measure the interval between the trigger and the sampling time. The MAMBA DAQ board communicates to a PC via a USB 3.0 interface and is controlled and monitored via a dedicated graphic user interface (GUI) called MAMBA GUI.

Slow control and online monitoring system The MAMBA GUI software has been designed in Visual Basic and runs on a Windows PC and allows to perform the DAQ slow control, to store the data to disk and to process online part of the acquired data. It provides the event display and useful data quality monitor (DQM) plots, as the noise and

signal distribution, the space distribution of the hits in the planes, particularly useful at the testbeam to identify the position of the particle beam. Two CAEN SP5602 plastic scintillators, read out through Hamamatsu MPPC S10362-11-100C silicon photomultipliers, are used to provide the trigger and are controlled using the proprietary software. The telescope is enclosed in a metal light-tight box with an active cooling system. A chiller refrigerates and circulates a thermal fluid to a heat sink inside the box that cools down the air temperature. Alternatively a Peltier cell can be used to cool down only the DUT, while the telescope can work at ambient temperature. Dry air is flowed inside the light-tight box to maintain low humidity and prevent water condensation on the sensors when the box is refrigerated. The cooling is necessary for testing irradiated sensors at temperatures below 0°C and the humidity and temperature of the air in the box is monitored, as well as the temperature of the telescope and DUT aluminium frames, using four PT100 temperature sensors. The high voltage is provided independently to the DUT and to the telescope sensors using two Keithley 2410 Source Measure Units that are remotely controlled via a custom LabView VI (Virtual Instrument) to set the voltage and monitor the leakage current.

3.3.2 Hardware specific implementation of the artificial retina algorithm

The artificial retina algorithm has been implemented in FPGA on the MAMBA DAQ board. The hardware specific scheme of the artificial retina architecture is shown in Fig. 3.9. The system performs the readout of up to 8 sensor modules. A programmable

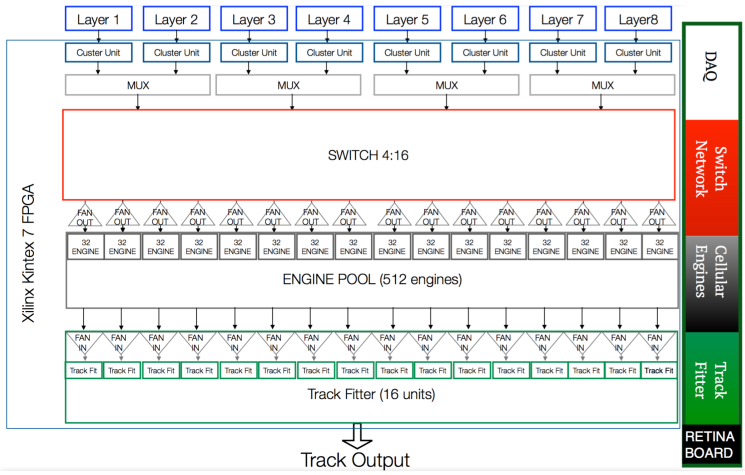

Figure 3.9: Architecture of the artificial retina hardware prototype. The device receives the hit data from the DAQ, a 4x16 Switch delivers the hit data to 16 Engine Regions with 32 Engines per region, for a total of 512 Engines. Track Fitters (16 units) receive data from the Engine Regions and are connected to the output.

threshold is applied to the strip signals to identify the hits, then a *Cluster unit* identifies clusters of adjacent hits for each sensor module. The clusters signal are formatted as

13-bit data providing the information about the index of the sensor module (3 bits) and the position of the cluster within the module (10 bits). The ADC information is discarded since it is not used in the artificial retina algorithm. Cluster data from couples of sensor modules are grouped and sent to the device through 4 lines.
The Switch has 4 inputs and 16 outputs connected to the Engine Regions. Each Engine Region hosts 32 Engines for a total of 512 Engines uniformly distributed over the space of the track parameters, within the geometrical acceptance of the telescope. The output of each Engine Region is connected to a Track Fitter for a total of 16 units. The output data of the Track Fitter units are collected through a FanIn, sent to the PC, and then written to disk. For debugging purposes the complete readout of the silicon telescope is also stored to disk.
The device operates at a clock frequency of 150 MHz, hence a clock cycle corresponds to 6.7 ns. Results are provided with a latency < 1 μs, according to number of clock cycles required to perform the track reconstruction. The amount of FPGA resources needed for the implementation of the DAQ and the tracking algorithm are 3% and 68%, respectively. The contributions to the latency and the percentage of resources needed by the three blocks of the artificial retina are reported in Table 3.1. We can observe that the major contribution to the latency is given by the Track Fitter due to the need of performing the interpolation of the weight values, defined by the Eqs. 3.5, 3.6, that includes the evaluation of logarithmic functions performed using a LUT, and division operations. In fact, the division operation can be implemented in FPGA using different techniques in order to save resources or reduce the latency. If on one hand the implementation of the Track Fitter could be revised to reduce the latency, on the other hand this would increase the resources usage. Moreover the Track Fitter, thanks to its pipelined architecture, is able to accept new data while it is processing the older ones in the pipeline, independently from its latency, which does not represent a major concern.

Module	Clock cycles	FPGA resources(%)
Switch	14	7
Engines	12	37
Track Fitter	68	24
Total	94	68

Table 3.1: Minimum latency of the tracking device response in number of clock cycles of the FPGA and percentage of logic resources allocated for each individual module.

3.3.3 Testbeam results

The full chain of the tracking system has been successfully tested using 180 GeV/c protons during a testbeam at CERN SPS. The event display of a typical single-track event is shown in Fig. 3.10.
Tracks have been reconstructed in real time using the MAMBA board tracking device. The track parameters are in good agreement with the results obtained from a simple

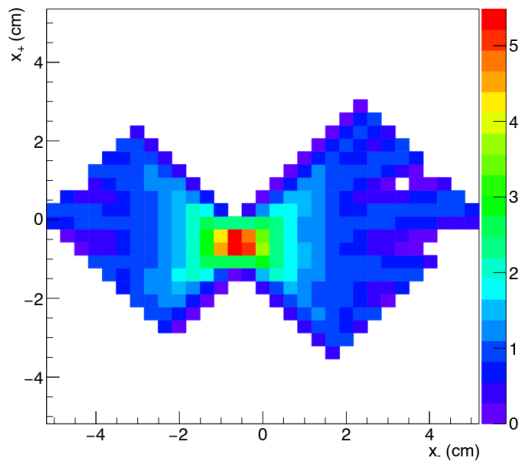

Figure 3.10: Response of the artificial retina to a single-track event at the testbeam at SPS.

χ^2-minimization algorithm. For debugging purposes the response of the device to real testbeam data has been simulated and it reproduces the results obtained from the artificial retina algorithm running in FPGA. The distribution of the track parameters determined by the MAMBA board tracking device, together with the distribution obtained by simulation of the algorithm, are shown in Fig. 3.11.

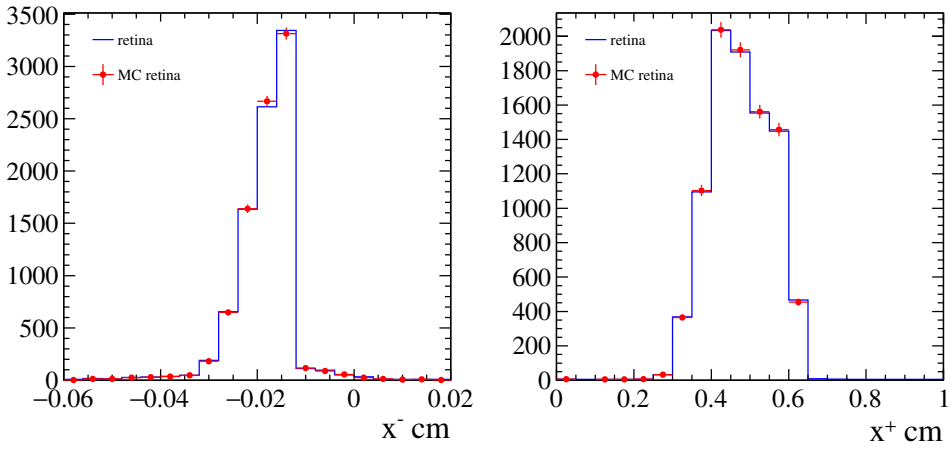

Figure 3.11: Track parameters distribution determined by the artificial retina. Testbeam data processed by the MAMBA board (retina) and verified using the simulated response (MC retina).

The resolution on the track parameters obtained by the online algorithm has been evaluated from the offline (χ^2-minimization) reconstruction. The distribution of the residuals for (x_-, x_+) are shown in Fig. 3.12 and have been fitted with a Gaussian function. The obtained widths are $\sigma_{x_-} = 12.5$ μm and $\sigma_{x_+} = 14.9$ μm .

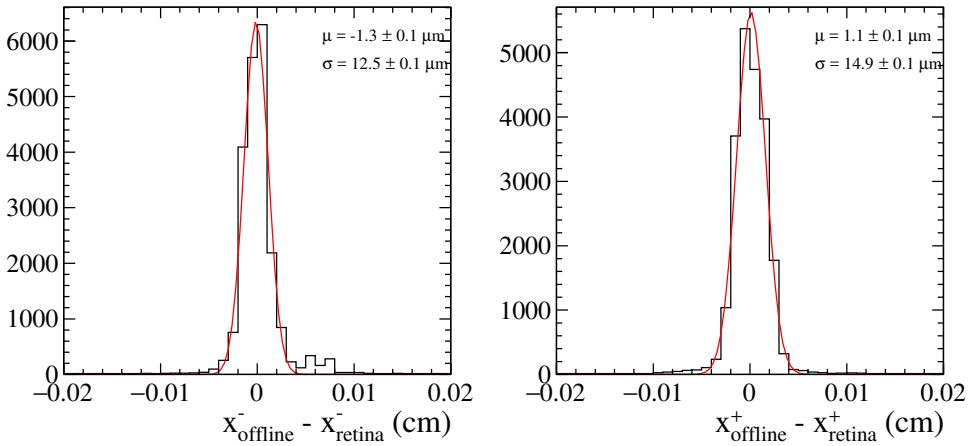

Figure 3.12: Distribution of the residuals for the track parameters evaluated using a simple χ^2 minimization algorithm (offline) and track parameters from the artificial retina algorithm

Chapter 4

4D real-time tracking algorithm

In this chapter we present the algorithm for 4D real-time track reconstruction. The algorithm is capable to perform both the pattern recognition and the reconstruction of the track parameters. It is highly parallelized and particularly suitable for implementation in FPGA with a pipelined architecture.
The version that we present is intended to reconstruct straight 3-dimensional tracks with the addition of the time information. This applies the reconstruction of tracks using a detector based on multiple layers of silicon pixel sensors providing additional precise time information of the hits, as proposed for an upgraded version of the VELO sub-detector for the LHCb Upgrade II. In general the same algorithm can be used for real-time track reconstruction of 3-dimensional tracks without considering the time information of the hits, although with lower performance.
The 4D real-time tracking algorithm that we present is new and more advanced compared to the artificial retina algorithm for 2D track reconstruction. However, there are some common features: both are based on the comparison of the measured hits to pre-computed tracks and the evaluation of the analog response of a grid of *cellular units* associated with different possible candidate tracks. More details on the algorithm are given in the following.

4.1 Algorithm description

The algorithm is intended to identify and reconstruct straight tracks using a detector based on layers of pixel sensors, providing space and time information of the hits. In this scheme a track is defined by 5 track parameters that we will consider as two primary parameters used to perform the track identification, similarly to the case of the 2D artificial retina algorithm, and three additional parameters that are reconstructed but not used in the pattern recognition.
The algorithm is based on the identification of couples of hits in adjacent sensors that we will call *stubs*. The use of the stubs is motivated by the dimension of the space of the track parameters. In fact, as we discussed in the previous section, each measured hit identifies a bundle of track. The degrees of freedom of the line bundle is given by the total number of track parameters minus the number of constraints provided by the hit. Considering the case of 4D (or 3D, without time) tracks, the measurement of single

hit identifies a bundle of track with 3 degrees of freedom (or 2, without time). A stub, instead, is more similar to a track since all the track parameters can be evaluated from the couple of hits. The number of degrees of freedom of the tracks that contains the measured stub is theoretically reduced to zero, except for the hit resolution and multiple scattering effects.
In order to define the complete algorithm we first provide the definition of the track parameters and the stubs, then we discuss how the pattern recognition is performed and the evaluation of the track parameters for the candidate tracks.

Track parameters We consider a detector based on multiple layers of pixel sensors providing space and time information of the hits. The sensors are orthogonal to the z-axis as shown in Fig. 4.1. A track is defined by 5 parameters $(x_+, x_-, y_+, y_-, t_0)$, with $(x_\pm,\ y_\pm) = ((x_f \pm x_l)/2,\ (y_f \pm y_l)/2)$ and $z_\pm = (z_f \pm z_l)/2$, where z_f, z_l are the z-coordinates of the first and last tracking planes, respectively, and (x_f, y_f) and (x_l, y_l) are the coordinates of the track at z_f, z_l. In particular (x_+, y_+) corresponds to the coordinates of the intersection of the track with a reference plane placed at z_+ and this quantity will assume a particular relevance in the following, while (x_-, y_-) are related to the tangent of the track angles.
A more classical definition of the 3D track can be given as:

$$\begin{pmatrix} x(z) \\ y(z) \end{pmatrix} = \begin{pmatrix} x_0 + m_x\ z \\ y_0 + m_y\ z \end{pmatrix} , \tag{4.1}$$

where x_0, y_0 are the coordinates at $z = 0$ and m_x, m_y are the track slopes.
The relation between the two sets of coordinates is given by the following equations:

$$\begin{pmatrix} x_- \\ y_- \end{pmatrix} = \begin{pmatrix} m_x\ z_- \\ m_y\ z_- \end{pmatrix} , \quad \begin{pmatrix} x_+ \\ y_+ \end{pmatrix} = \begin{pmatrix} x_0 + m_x\ z_+ \\ y_0 + m_y\ z_+ \end{pmatrix} . \tag{4.2}$$

Including the time coordinate to the track definition, by extension of the Eq. 4.1, we define the time of the track as the time of the particle at $z = 0$, namely t_0 according to the following equation:

$$t(z) = t_0 + \frac{\sqrt{m_x^2 + m_y^2}}{c} \times z , \tag{4.3}$$

where the particle is assumed to travel at the speed of light c.

Stub coordinates A stub is formed by any combination of hits that is compatible with a *reconstructible* track, *i.e.* at LHCb the direction of a stub must be compatible with a particle originated in the proton-proton interaction region. The stub coordinates are defined following the same notation of the track parameters, since a stub represents a track hint. Assuming (x_1, y_1, z_1, t_1) and (x_2, y_2, z_2, t_2) the coordinates of the first and second hit of a stub candidate, respectively, the stub parameters are evaluated according

to the following equations:

$$\begin{aligned} x_{-,stub} &= \frac{x_1 z_- - x_2 z_-}{z_1 - z_2} \\ x_{+,stub} &= \frac{x_1(z_+ - z_2) - x_2(z_+ - z_1)}{z_1 - z_2} \\ y_{-,stub} &= \frac{y_1 z_- - y_2 z_-}{z_1 - z_2} \\ y_{+,stub} &= \frac{y_1(z_+ - z_2) - y_2(z_+ - z_1)}{z_1 - z_2} \\ t_{0,stub} &= \frac{t_1 + t_2}{2} - \frac{z_1 + z_2}{2\ c\ \sqrt{1 + (x_-/z_-)^2 + (y_-/z_-)^2}} \end{aligned} \tag{4.4}$$

The *velocity* of the particle can be estimated as an additional stub parameter. The stub velocity is defined as $|\vec{x}_1 - \vec{x}_2|/(t_1 - t_2)$, where $\vec{x}_1 = (x_1, y_1, z_1)$, $\vec{x}_2 = (x_2, y_2, z_2)$ are the spatial coordinates of the first and second hit of a stub candidate. The velocity is used only during the stub construction and the reconstructed tracks are always assumed to be associated with particles travelling at the speed of light. In fact, if the velocity determined from the stub is not compatible with the speed of light c then the candidate stub is rejected. Other cuts based on the spatial parameters are also applied and discussed later.

Pattern recognition and cellular units The pattern recognition consists in the association of multiple stubs to a track candidate according to their projection to a reference plane. In particular, we consider the stub projections at $z = z_+$ that are defined by the couple of stub parameters (x_+, y_+).
A grid of cellular units is allocated in the sub-space of the track parameters (x_+, y_+) and is labelled by the couple of indexes (i, j). In this 4D real-time tracking algorithm a cellular unit is associated with a bundle of tracks that intersect the reference plane, placed at $z = z_+$ in the coordinates (x_{i+}, y_{j+}) defined by the cellular unit, as shown in Fig. 4.1. The other stub parameters (x_-, y_-, t_0) are free to assume any value, provided that these are compatible with the geometrical acceptance and timing cuts.
Each cellular unit, whose coordinate are defined by (x_{i+}, y_{j+}), evaluates a Gaussian response according to the distance of the cell in the reference plane from the measured stub, whose coordinates are (x_{k+}, y_{k+}). The squared distance is defined as

$$s^2_{ijk} = (x_{k+} - x_{i+})^2 + (y_{k+} - y_{j+})^2 \ , \tag{4.5}$$

and the response to a single stub is defined as

$$W_{ijk} = N_{ijk} \cdot \exp\left(-\frac{s^2_{ijk}}{2\sigma^2}\right) \ , \tag{4.6}$$

with

$$N_{ijk} = \begin{cases} 1 & \text{if } |s_{ijk}| \leq 2\Delta \\ 0 & \text{otherwise} \end{cases} \ , \tag{4.7}$$

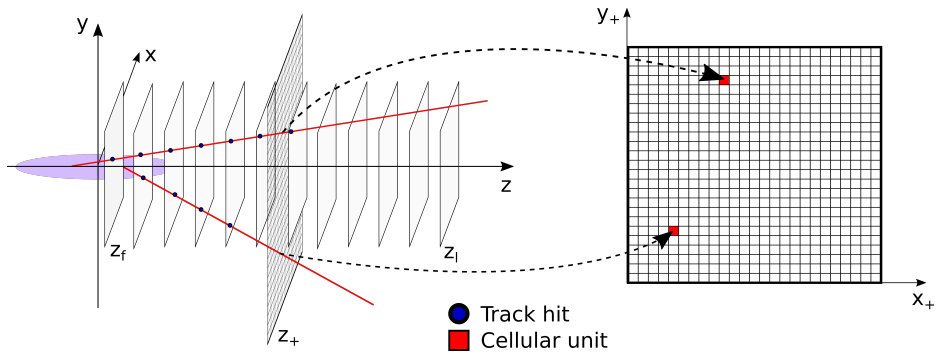

Figure 4.1: Layout of a VELO-like detector and visual representation of the grid of cellular units, distributed over a reference plane placed at $z = z_+$.

where Δ, σ are parameters to be adjusted for optimal response. In particular Δ is about the size of the grid step in the simplest case of a grid of uniformly distributed cellular units, and the value of σ is comparable to Δ. The total response (weight function) of each cellular unit is evaluated as the average of the contributions from the measured stubs. A contribution is considered negligible, hence not included in the average, if $N_{ijk} = 0$. The weight function is then defined by the following equation:

$$W_{ij} = \frac{1}{N_{ij}} \sum_k W_{ijk}, \tag{4.8}$$

where

$$N_{ij} = \sum_k N_{ijk} \ . \tag{4.9}$$

The weight function of the cellular units is shown in Fig. 4.2 for an event with 1200 generated tracks.
It is worth noting that the weight function only depends on the x_+, y_+ quantities from the measured stubs. Three additional quantities are evaluated by each cellular units from the (x_{-k}, y_{-k}, t_{0k}) stub parameters.

$$x_{-ij} = \frac{1}{N_{ij}} \textstyle\sum_k x_{-ijk} \ , \tag{4.10}$$

$$y_{-ij} = \frac{1}{N_{ij}} \textstyle\sum_k y_{-ijk} \ , \tag{4.11}$$

$$t_{0ij} = \frac{1}{N_{ij}} \textstyle\sum_k t_{0ijk} \ , \tag{4.12}$$

where

$$x_{-ijk} = N_{ijk} \cdot x_{-k} \ , \tag{4.13}$$

$$y_{-ijk} = N_{ijk} \cdot y_{-k} \ , \tag{4.14}$$

$$t_{0ijk} = N_{ijk} \cdot t_{0k} \ , \tag{4.15}$$

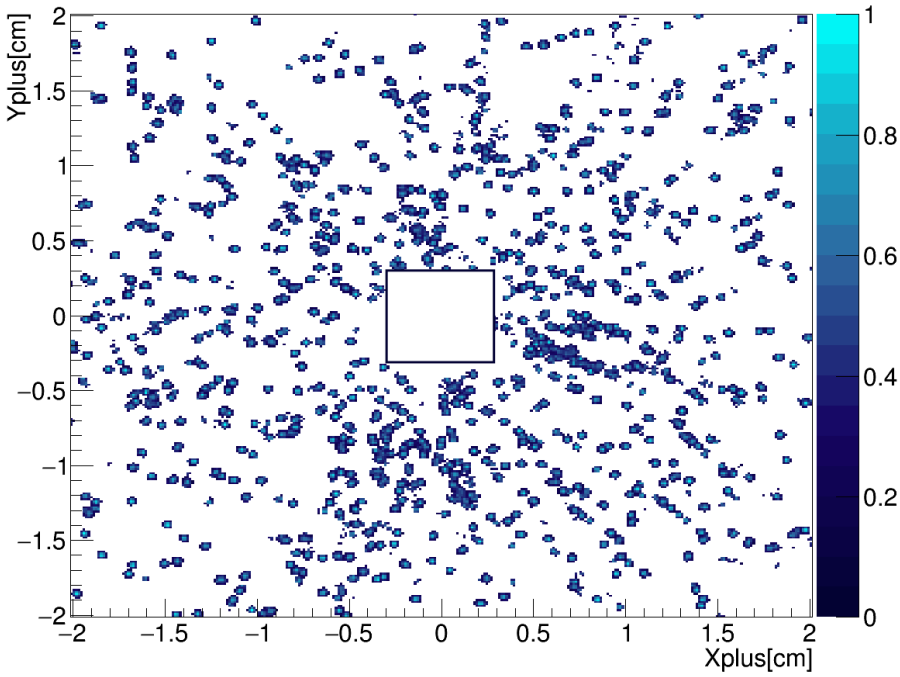

Figure 4.2: Typical response of the grid of cellular units for an event with 1200 generated tracks and 90,000 cellular units. The candidate tracks are identified by the cellular units whose response is maximum with respect to their neighbours.

and x_{-k}, y_{-k}, t_{0k} are the (x_-, y_-, t_0) values of the k-th stub.
The evaluation of the weight function can be performed in parallel since it only depends on the inputs from the measured stubs and a track is identified by a local maximum of the weight function. The function N_{ij} for the local maxima is required to be greater than a certain threshold. In fact the value N_{ij} corresponds to the number of stubs that belong to the candidate track, in the simplest case in which stubs from different tracks excite different cellular units. In this simulation the minimum number of stubs required to identify a track has been set to 2.

Track fitting For each candidate track the reconstructed track parameters $(x_+, y_+)_{\rm rec}$ are obtained by interpolations of the weight values adjacent to the maximum along the x_+, y_+ axes. In particular these are determined by means of a Gaussian interpolation, defined as

$$x_{+,\rm rec} = x_{+ij} + \frac{\Delta_{x_+}}{2} \frac{\ln(W_{i-1j}/W_{ij}) - \ln(W_{i+1j}/W_{ij})}{\ln(W_{i-1j}/W_{ij}) + \ln(W_{i+1j}/W_{ij})} \ , \tag{4.16}$$

$$y_{+,\rm rec} = y_{+ij} + \frac{\Delta_{y_+}}{2} \frac{\ln(W_{ij-1}/W_{ij}) - \ln(W_{ij+1}/W_{ij})}{\ln(W_{ij-1}/W_{ij}) + \ln(W_{ij+1}/W_{ij})} \ , \tag{4.17}$$

assuming $\Delta_{x_+} = (x_{i+1j} - x_{ij}) = (x_{ij} - x_{i-1j})$, $\Delta_{y_+} = (y_{ij+1} - y_{ij}) = (y_{ij} - y_{ij-1})$, where (i, j) are the indexes associated with the local maximum in the grid. The remaining

track parameters are evaluated as

$$x_{-,rec} = x_{-ij} \ , \quad y_{-,rec} = y_{-ij} \ , \quad t_{0,rec} = t_{0ij} \ , \tag{4.18}$$

according to the Eqs. (4.10), (4.11), (4.12).

4.2 Simulated response of the 4D tracking algorithm

In this section we present the results of the high-level simulations of a software-implemented version of the 4D tracking algorithm. The purpose of this simulation is to evaluate the performance of the algorithm applied to a tracking system similar to the VELO Upgrade detector providing both space and time information of the hits.
The performances are estimated in terms of the quality of the reconstructed track parameters, compared to the parameters of the generated tracks, and in terms of the reconstruction efficiency and purity of the identified tracks.
A comparison between the presented 4D real-time tracking algorithm and the performances of the same algorithm without including the time information, but still based on the use of the stubs, is provided.
In the software simulation the evaluation of the response is not parallelized and data flow to the cellular units is not modelled nor simulated. The reconstruction is performed on a *event-by-event* basis. For each event a certain number of tracks is generated, the hits are measured and the stubs are identified by a loop over all the couples of hits in adjacent planes, then spatial and time cuts are applied. When all the stubs have been processed the weight function is evaluated for all the cellular units and the tracks are identified and fitted.

Tracking system layout and track generation In the simulations we considered a tracking system composed of 12 planes of silicon pixel detectors positioned along the z axis. The layout is similar to part of the VELO Upgrade of the LHCb detector. The positions of the sensors are reported in Table 4.1. Each pixel sensor has been modelled as a 66×66 mm^2 square with a square "hole" of 5×5 mm^2 around the beam pipe. The pixel size is 55×55 μm^2 and the thickness is 200 μm. The timing information of the track hits has been modelled by evaluating the time of passage of the tracks in the sensors, then a Gaussian smearing has been applied, with a Gaussian width $\sigma_{t,hit}$ corresponding to 30 ps if not specified.

Sensor	0	1	2	3	4	5	6	7	8	9	10	11
z [mm]	1	26	51	76	101	126	151	176	201	226	251	276

Table 4.1: Positions of the sensors along the beam axis.

In the LHCb the typical r.m.s. of the longitudinal size of the colliding proton bunches is 7.5 cm and the r.m.s. of the transverse size is few μm. In this scenario the proton-proton interactions are distributed over a region of about 10 cm. In particular we simulate an interaction region Gaussian distributed along the beam axis with $\sigma_z = 5$ cm. To take into

account displaced vertices from secondary interactions the interaction region is Gaussian distributed also along the x, y axes with a width $\sigma_{x,y} = 500$ μm. The distribution in time is Gaussian with a width $\sigma_{t,lumi} = \sigma_{z,lumi}/c = 167$ ps. Tracks are generated in the forward region with a uniform distribution of the polar angle θ_z (w.r.t. to the z axis) in the range $[0.035, 0.270]$, while the distribution of the azimuthal angle $\phi_{x,y}$ is uniform in the range $[0, 2\pi]$.

Cellular units distribution and stub cuts The response of the 4D real-time tracking algorithm has been simulated using a grid of about 360,000 cellular units uniformly distributed over a reference plane (x_+, y_+) placed at $z = z_+ = 138.5$ mm in the range $[-40 \text{ mm}, 40 \text{ mm}] \times [-40 \text{ mm}, 40 \text{ mm}]$. The region of track parameters in the range $[-3 \text{ mm}, 3 \text{ mm}] \times [-3 \text{ mm}, 3 \text{ mm}]$ has not been considered in the reconstruction since it is outside the acceptance of the simulated detector. The granularity of the grid in (x_+, y_+) is 133 μm for both the directions, and depends on the number of cellular units in the grid. Different cuts have been applied to reject misidentified stub candidates that are not compatible with a generated track. In this way most of the stubs obtained by linking two points belonging to different tracks are removed. Selection cuts have been applied based both on the spatial parameters of the stub and on the difference in time between the two measured hits. The geometrical acceptance cuts are reported below:

$$\sqrt{(x_0^2 + y_0^2)_{stub}} \quad < 15 \text{ mm} , \tag{4.19}$$

$$\sqrt{(m_x^2 + m_y^2)_{stub}} \quad < 0.27 , \tag{4.20}$$

$$|d_0|_{stub} \quad < 1.25 \text{ mm} , \tag{4.21}$$

$$|z_0|_{stub} \quad < 150 \text{ mm} . \tag{4.22}$$

where d_0 is defined as the (signed) distance of closest approach of the stub projection line to the z axis, and z_0 is the position along the z-axis where d_0 is evaluated. The variables d_0 and z_0 are defined as follows:

$$d_0 = \quad (x_0\ m_y - y_0\ m_x)/\sqrt{m_x\ m_x + m_y\ m_y} , \tag{4.23}$$

$$z_0 = \ - \ (x_0\ m_x + y_0\ m_y)/\left(m_x\ m_x + m_y\ m_y\right) . \tag{4.24}$$

In order to apply a cut based on the time of the hits, the velocity is calculated as $\vec{v}_{2,1} = (\vec{x}_2 - \vec{x}_1)/(t_2 - t_1)$, where $(\vec{x}_2 - \vec{x}_2)$ is the spatial distance between the first and the second hit of the stub, $(t_2 - t_1)$ is the difference in time and $\Delta z_{2,1} = (z_2 - z_1)$ is the distance between the planes. The following cut is applied, according to the resolution on the time of the hit σ_t :

$$\left| \Delta z_{stub} \left(\frac{1}{v_{stub}} - \frac{1}{c} \right) \right| < 4\sqrt{2}\sigma_t . \tag{4.25}$$

Results Simulations of events with 1200 generated tracks in the acceptance have been performed. The algorithm response has been simulated in two cases, using the information on the time of the hits with a resolution of 30 ps and without using the time information. Events with only one track have been simulated as benchmark for evaluating the quality

of the reconstruction. The resolution on the track parameters are evaluated fitting the residual distribution of the reconstructed minus the generated parameter with the sum of two Gaussian functions. The value of the resolution is defined as the *root mean square* of the fitting function.

The obtained resolution on the x_-, y_- track parameters are $\sigma_{x_-,y_-} = 46$ μm, the resolution on x_+, y_+ are $\sigma_{x_+,y_+} = 27$ μm and the resolution on the time of the track (at $z = 0$) is $\sigma_{t0} = 11.4$ ps .

In presence of a high number of tracks different contributions can decrease the quality of the reconstruction and increase the number of ghost tracks. In particular, the pattern recognition, which is the identification of local maxima, is performed only in a subspace of the track parameters; in this sense tracks that have similar values of (x_+, y_+) or misidentified stubs can produce an excitation of the same cellular units. The worsening of the resolution is in general due to the contamination of the weight function near the local maximum corresponding to an identified track. This effect depends on the granularity of the grid and it is reduced when increasing the number of cellular units. The contributions given by the misidentified stubs can be mitigated using the time information in the fast tracking algorithm.

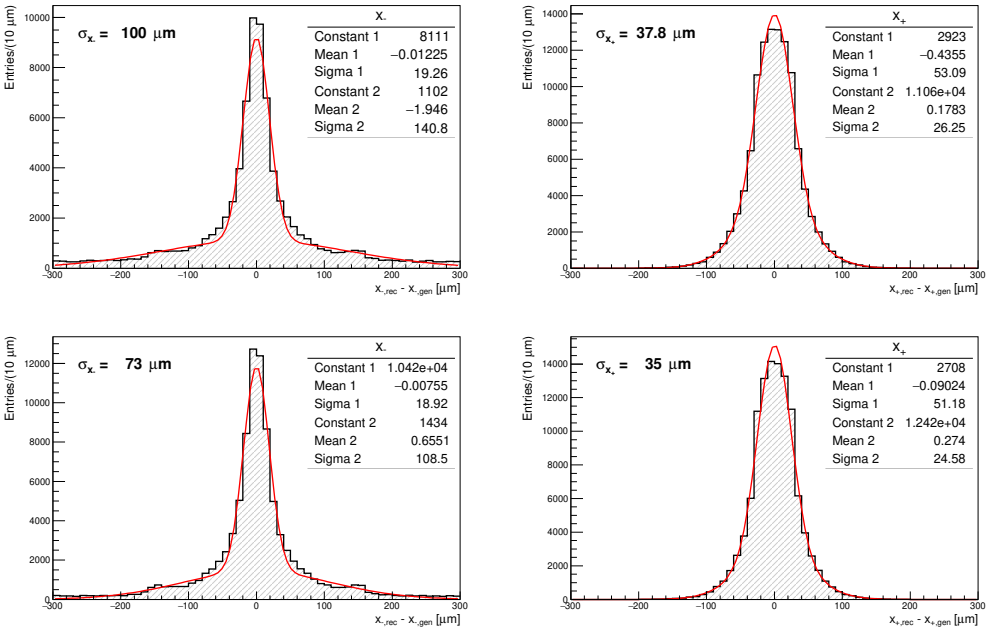

Figure 4.3: Residual distribution for reconstructed minus generated x_- and x_+ track parameters for simulated events with 1200 generated tracks in the acceptance with no time information (top) and using the information of th hits with 30 ps resolution (bottom). The resolution improves when using the time information and changes from $\sigma_{x_-} = 100$ μm and $\sigma_{x_+} = 37.8$ μm to $\sigma_{x_-} = 73.0$ μm and $\sigma_{x_+} = 35.0$ μm . Similar results are obtained for σ_{y_-} and σ_{y_+} .

The residual distributions of reconstructed minus generated x_- and x_+ track parameters

for events with 1200 generated tracks in the acceptance are shown in Fig. 4.3. The resolutions obtained without using the time information of the hit are $\sigma_{x_-} = 100$ μm and $\sigma_{x_+} = 37.8$ μm. When including the time information of the hit the tracking performance improves and the resolution on the track parameters becomes $\sigma_{x_-} = 73.0$ μm and $\sigma_{x_+} = 35.0$ μm.
The residual distribution for the reconstructed minus generated time of the track is shown in Fig. 4.4. The simulated resolution on the time of the hit is 30 ps and the obtained resolution on the time of the track is $\sigma_{t0} = 29.1$ ps when the time cut defined in Eq. 4.25 is applied in the stub construction. The system provides a good determination of the time of the track and this information can be used at later stages of the event reconstruction to distinguish among particles coming from vertices close in space but originated from proton-proton interactions occurring at different times. The previous result is compared to the case in which the time of the track is evaluated according to the Eq. 4.18 but the cuts based on the velocity of the particle are not applied. In this case the resolution on the time of the track is $\sigma_{t0} = 45.2$ ps.

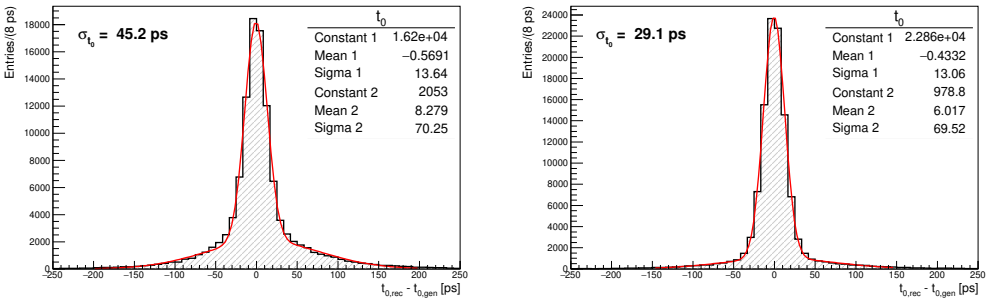

Figure 4.4: Residual distribution for reconstructed minus generated time of the track for simulated events with 1200 generated tracks in the acceptance without using the time cut in the stub construction (top) and applying the cut (bottom). The simulated resolution on the time of the hit is 30 ps in both cases. The resolution improves in the latter case, from $\sigma_{t_0} = 45.2$ ps to $\sigma_{t_0} = 29.1$ ps .

Resolution	Without timing cuts	With timing cuts
σ_{x_-} (μm)	100.0	73.0
σ_{y_-} (μm)	99.2	72.4
σ_{x_+} (μm)	37.8	35.0
σ_{y_+} (μm)	38.2	35.2
σ_{t_0} (ps)	45.2	29.1

Table 4.2: Resolution on x_+, x_-, t_0 track parameters, obtained without applying the timing cuts during and applying the timing cuts.

The resolutions obtained on x_+, x_-, t_0 track parameters are summarized in Table 4.2. The reconstruction efficiency ϵ_{rec} is defined as the ratio between the number of recon-

structed tracks and the number of generated tracks that can be reconstructed using the 4D tracking algorithm. To evaluate the efficiency, for each generated track it is required to find a reconstructed track whose distance in the (x_+, y_+) is less than 200 μm. The reconstructed track is also required not to be a ghost track. A track is defined as well reconstructed (not a ghost track) if more than 70% of the contributions to the cellular unit identified as maximum comes from a real track. In both cases, with and without the time information of the hit, the efficiency is $\epsilon_{rec} = 98.5\%$. The rate of ghost tracks is instead reduced when using the time information. The reconstruction purity $\mathcal{P}_{rec}$ is defined as the ratio between the number of well reconstructed tracks and the total number of reconstructed tracks. The purity improves when using the time information, going from $\mathcal{P}_{rec} = 60\%$ to $\mathcal{P}_{rec} = 82\%$. The reconstruction purity as a function of the number of stubs that contributed to the weight function of the engine associated with an identified track is shown in Fig. 4.5. In both cases the purity increases with the increasing of the number of stubs, that is correlated to the number of hits that belong to each track.

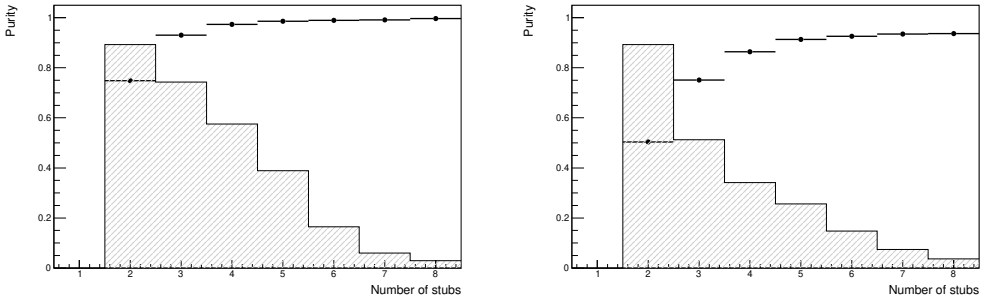

Figure 4.5: Purity of the reconstruction as a function of the number of stubs contributing to the identified track, using the time information with 30 ps resolution (left) and without using the time information (right). The distributions of number of stubs are superimposed as shaded histograms.

The results presented in the following, if not specified, refer to simulations that includes the time information of the hit, with 30 ps resolution.
The reconstruction efficiency as a function of the track parameters (x_+, y_+) and as a function of the variable $r_+ = \left(x_+^2 + y_+^2\right)^{1/2}$ is shown in Fig. 4.6. It can be seen that there is a small inefficiency for low values of r_+, where the distribution of the tracks is maximum as represented by the shaded histogram superimposed to the efficiency plot.
The efficiency as a function of the pseudo-rapidity $\eta = -\ln(\tan(\theta/2))$, the azimuthal angle ϕ, d_0 and z_0, where θ is the polar angle with respect to the z axis, is shown in Fig. 4.7. The efficiency decreases for high values of the pseudo-rapidity η, corresponding to tracks with lower values of the polar angle θ: this reflects the inefficiency shown for low values of r_+. The efficiency does not show a dependence on the ϕ and d_0 track parameters as expected, while it decreases for increasing values of z_0 reflecting the fact that tracks with higher z_0 intersect a lower number of planes.
The reconstruction purity as a function of the resolution on the time of the hit is shown in Fig. 4.8. The purity decreases for increasing values of the time resolution and the results

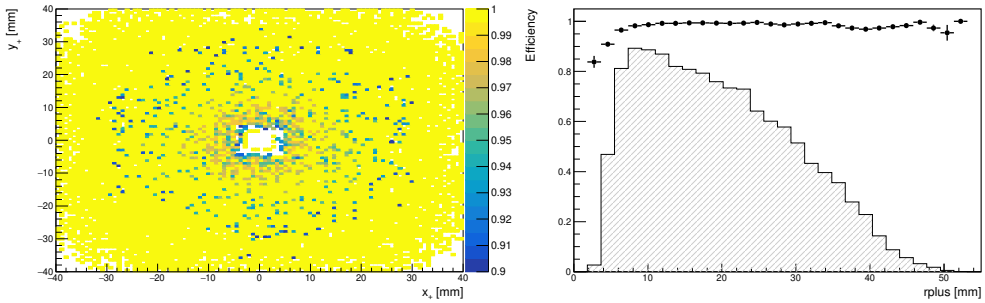

Figure 4.6: Efficiency as a function of (x_+, y_+) (left) and $r_+ = \left(x_+^2 + y_+^2\right)^{1/2}$ (right); the shaded histograms represents the r_+ distribution of the tracks.

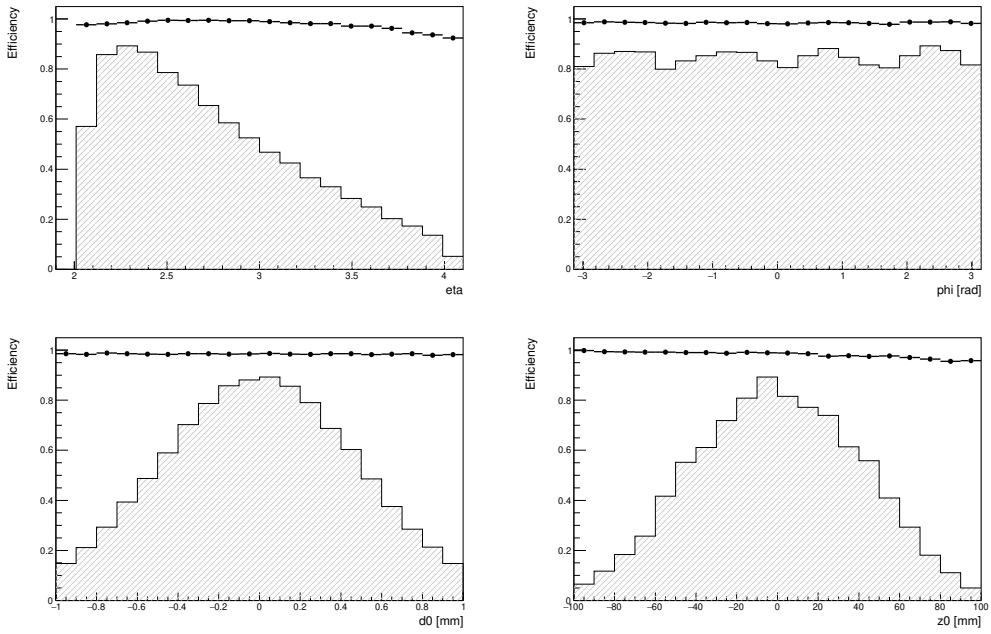

Figure 4.7: Efficiency as a function of the pseudo-rapidity η (top-left), the azimuthal angle ϕ (top-right), d_0 (bottom-left), z_0 (bottom-right). The distributions of the track parameters are superimposed as shaded histograms.

obtained assuming 150 ps resolution are comparable with the results obtained without using the time information. In fact if the value of the time resolution is comparable to the dimension of the beam (in time, $\sigma_{t,beam} = 167$ ps), the time cut defined in Eq. 4.25 becomes inefficient. Nevertheless it should be noted that the information on the time of the track is still accessible and can be used in the next level of processing, *i.e.* in the vertex reconstruction.

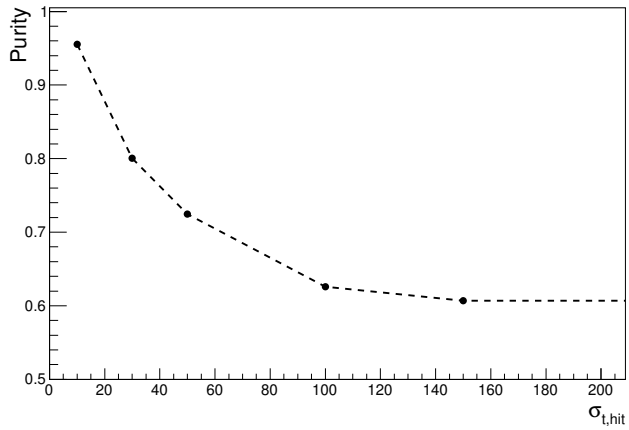

Figure 4.8: Reconstruction purity as a function of hit time resolution. The lower limit is represented by the case in which the time information is not used.

Chapter 5

Device architecture and implementation in FPGA

In this chapter we describe the implementation of the proposed algorithm for fast track reconstruction in FPGA. The system has been implemented using Xilinx Vivado Design Suite[1] that has been used for both synthesis and simulation. The hardware description language (HDL) that has been used is VHDL[2] for all the fast tracking algorithm implementation. In some cases the hardware description has been provided using Verilog language, in particular for the instantiation of some Xilinx LogiCORE IPs.
Here we first give an overview of the architecture of the proposed fast track finding system and then provide the description of all the modules that are used inside the main blocks.

General architecture The architecture of the proposed fast tracking device is divided in different major modules.

- The Stub Constructor module receives the data from the DAQ, identifies and formats the stubs, then forwards them to the following module.

- The Switch module receives the stub data and delivers each one to a unique output according to the address evaluated by the Stub Constructor module. Each output of the Switch is connected to an Engine Region that is associated with a region of the space of the track parameters.

- The Engine Region receives the data from the Switch and delivers them to a number of Engine modules that evaluate the response to the stimuli provided by the incoming stubs. The Engines correspond to the cellular units described in Chapter 4 and identify the candidate tracks and evaluate the track parameters. A FanIn collects the outputs of the Engines and forwards them to the Engine Region output.

- One or multiple FanIn modules receive the reconstructed track data from the Engine Regions and provide them to the output ports of the whole system. The FanIns represent the last modules of the proposed fast track finding system.

[1] Vivado Design Suite; https://www.xilinx.com/products/design-tools/vivado.html
[2] *Very High Speed Integrated Circuits* Hardware Description Language

5.1 Description of the implemented modules

In this section we give a description of the modules that have been implemented and used for the implementation of the 4D real-time tracking algorithm. In particular all the modules are optimized to reduce the minimum latency of the system.

5.1.1 VHDL package: algorithmtools

A VHDL package called algorithmtools has been developed and used in this project. The package allows to define constants, custom types and functions to be shared and used within multiple design units in order to enhance the readability and maintainability of the code. In particular all the modules that we will describe in the following sections make use of the definitions provided by the algorithmtools package and the majority of them is transparent with respect to changes of the custom types definitions. Simple examples of values defined in algorithmtools are the data size, the dimension of the Switch, the number of Engines in each Engine Region. Others definitions provided in the package are the hit, stub, track and hold custom types, that we describe with more detail in the following.

Hit data format A record has been used to define the hit data format as a custom data type. In VHDL the records are similar to the structures in C programming. The r_hitdata type is used to describe the hit data format and it is defined as follows:

```
type r_hitdata is record
  dv        : std_logic;
  bco       : std_logic_vector(hit_bco_size-1 downto 0);
  x_address : std_logic_vector(hit_x_address_size-1 downto 0);
  y_address : std_logic_vector(hit_y_address_size-1 downto 0);
  t0        : std_logic_vector(hit_t0_size-1 downto 0);
  eoe       : std_logic;
  padding   : std_logic_vector(hit_padding_size-1 downto 0);
end record r_hitdata;
```

In particular, dv is the *Data Valid* flag, bco is the bunch crossing timing information, where 1 unit represents 25 ns, x_address and y_address form the address of the data, namely the position of the hit in one tracking plane, t0 represents the timing information with finer resolution within the 25 ns bunch-crossing interval, eoe is the *End Of Event* flag that is used to identify when all the hit data from one detector have been read, padding is a *null padding* that is filled with zero bits.
hit_bco_size, hit_x_address_size, hit_y_address_size, hit_t0_size, hit_padding_size are constant integers and define the size of the items in the r_hitdata record. The data size of the record is defined by the r_hitdata_size constant value and the hit_padding_size is defined accordingly.
The auxiliary constant r_hitdata_zero is defined as an r_hitdata with all the signals set to zero. Arrays and matrices of r_hitdata are often used in the implementation and

dedicated types are defined as `r_hitdata_array` and `r_hitdata_matrix` respectively. The definitions are given by:

```
constant r_hitdata_zero : r_hitdata := (
  dv        => '0',
  bco       => ( others => '0' ),
  x_address => ( others => '0' ),
  y_address => ( others => '0' ),
  t0        => ( others => '0' ),
  eoe       => '0',
  padding   => ( others => '0' )
);

type r_hitdata_array is ARRAY( natural range <> ) of r_hitdata ;
type r_hitdata_matrix is ARRAY( natural range <>, natural range <> ) of r_hitdata ;
```

Stub data format Similarly to the case of the hit data format, the `r_stubdata` record defines the stub data format

```
type r_stubdata is record
  dv        : std_logic;
  bco       : std_logic_vector(stub_bco_size-1 downto 0);
  detindex  : std_logic_vector(stub_detindex_size-1 downto 0);
  x_address : std_logic_vector(stub_x_address_size-1 downto 0);
  y_address : std_logic_vector(stub_y_address_size-1 downto 0);
  xminus    : std_logic_vector(stub_xminus_size-1 downto 0);
  yminus    : std_logic_vector(stub_yminus_size-1 downto 0);
  t0        : std_logic_vector(stub_t0_size-1 downto 0);
  eoe       : std_logic;
  conf      : std_logic;
  padding   : std_logic_vector(stub_padding_size-1 downto 0);
end record r_stubdata;
```

In particular, `dv` is the *Data Valid* flag, `bco` is the bunch crossing timing information, where 1 unit represents 25 ns, `detindex` is the index of the couple of detector planes in which the stub has been identified, `x_address` and `y_address` form the address of the data, and are used in the Switch module to evaluate the data path to the Engines, and represent the position of the stub projection to the tracking reference plane. The `xminus`, `yminus`, `t0` fields represent the (x_-, y_-, t_0) values of the stub, as defined in the Eq. 4.4, `eoe` is the *End of Event* flag that is used to identify when all the stub data from the Stub Constructor have been read and delivered to the Switch module, `conf` is the *Configuration* flag that is used for the initial configuration of the Engines, `padding` is the *null padding*. Similarly to the previous case, the size of the record items are defined by integer constant values.

The auxiliary constant `r_stubdata_zero` is defined as an `r_stubdata` with all the signals set to zero and arrays and matrices of `r_stubdata` are defined by two custom types `r_stubdata_array` and `r_stubdata_matrix`.

Track data format The `r_trackdata` record is used to contain the information of the identified and reconstructed tracks, evaluated and provided by the Engines. The record is defined as follows:

```
type r_trackdata is record
  dv          : std_logic;
  bco         : std_logic_vector(track_bco_size-1 downto 0);
  x_address   : std_logic_vector(track_x_address_size-1 downto 0);
  y_address   : std_logic_vector(track_y_address_size-1 downto 0);
  counter_sum : std_logic_vector(track_counter_sum_size-1 downto 0);
  xplus_sum   : std_logic_vector(track_xplus_sum_size-1 downto 0);
  yplus_sum   : std_logic_vector(track_yplus_sum_size-1 downto 0);
  xminus_sum  : std_logic_vector(track_xminus_sum_size-1 downto 0);
  yminus_sum  : std_logic_vector(track_yminus_sum_size-1 downto 0);
  t0_sum      : std_logic_vector(track_t0_sum_size-1 downto 0);
  padding     : std_logic_vector(track_padding_size-1 downto 0);
end record r_trackdata;
```

In particular, `dv` is the *Data Valid* flag, `bco` is the bunch crossing timing information, where 1 unit represents 25 ns, `x_address` and `y_address` form the data address, in particular these value are used to store the indexes of the Engine (in the instantiated grid of Engines) that identified the track, `counter_sum` represents the number of stubs identified as part of the reconstructed track, `xplus_sum`, `yplus_sum`, `xminus_sum`, `xminus_sum` and `t0_sum` represent the sums of the stub parameters from the stubs that contributed to the track, `padding` is the *null padding*.
Even in this case, the size of the record items are defined by integer constant values.
The auxiliary constant `r_trackdata_zero` is defined as an `r_trackdata` with all the signals set to zero and arrays and matrices of `r_trackdata` are defined by two custom types `r_trackdata_array` and `r_trackdata_matrix`.

Hold data format The *hold* signals are used to back propagate (with respect to the flow of the stub data) the information about the state of the modules receiving the hit, stub or track information. In particular a module asserts the hold signal when it is not able to accept an incoming data. More information about the hold logic will be given in the following section.
A `record` has been defined for the hold signals, together with the `r_hold_zero` constant, `r_hold_array` and `r_hold_matrix` data types. In this case the `r_hold` record contains only a `std_logic` signal, according to the definitions given in the following:

```
type r_hold is record
  value : std_logic;
end record r_hold;
```

5.1.2 Ring Buffer

The Hit, Stub and Track Ring Buffers are basic components that are used inside different other modules, *i.e.* the Stub Ring Buffer are present inside the submodules used to build the Stub Switch, as the Stub N-way Dispatchers and N-way Mergers. The Hit, Stub and Track Ring Buffer are defined in the same way and only differs for the type of data provided in input and output. For this reason we will generically refer to any of these module as Ring Buffer, unless specifically required. In general a Ring Buffer is used at each input of the Switch sub-modules.
Here we describe the Ring Buffer implementation using the Hit Ring Buffer as an example; the entity is called `hit_ring_buffer` and it is described as follows:

```
entity hit_ring_buffer is
generic (
  buffer_size : integer := 2
);
port (
  clk       : in  std_logic;
  reset     : in  std_logic;
  w_en      : in  std_logic;
  r_en      : in  std_logic;
  data_in   : in  r_hitdata;
  data_out  : out r_hitdata;
  full      : out std_logic;
  not_empty : out std_logic
);
end hit_ring_buffer;
```

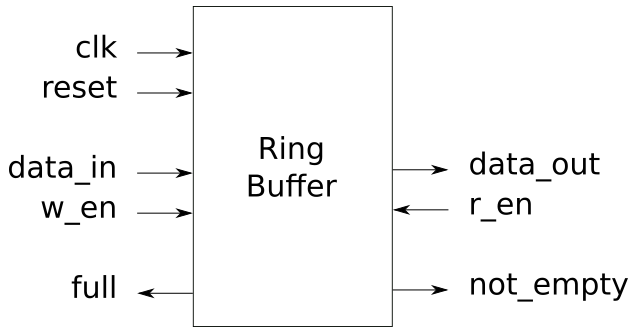

Figure 5.1: Scheme of the Ring Buffer ports. The direction of the arrows highlights input and output ports.

The scheme of the ports is described in Fig. 5.1. The module includes a buffer whose depth is defined by the parameter `buffer_size`, defined by the `generic` integer and provided when instantiating the Ring Buffer module. The `generic` value of `buffer_size` shown in the code and in general all the `generic` values in the entity declarations are

intended as default ones, to be used when a specific value is not passed during the entity instantiation.
The behaviour of the module is described as follows: when the *write enable* signal `w_en` is high the incoming `r_data` is read from `data_in` and stored in the buffer at the position defined by the `write_index` internal signal, then the `write_index` is increased. The `data_out` port always outputs the content of the buffer at the position defined by the `read_index` internal signal. When the *read enable* signal `r_en` is high the buffer data at `read_index` position is deleted and the index is incremented. The `full` and `not_empty` flags provide information about the status of the buffer, in particular the `full` signal is connected to the module that is providing data to the buffer and it is asserted when the Ring Buffer is not able to accept data. The signal is deasserted as soon as a data is read and a position in the buffer becomes available for writing a new one.
The module operates at a reference clock speed given by the `clk` signal. At each rising edge of the clock the module updates all the output and internal signals. The `reset` signal is provided asynchronously and is used to clear the content of the buffer and reinitialize the `read_index` and `write_index`.
The Ring Buffer, in fact, acts as a First Word Fall Through (FWFT) FIFO[3] buffer. The FWFT FIFO differs from a Standard FIFO in the way the output data is accessed: in a Standard FIFO the read enable signals is asserted and after one clock period the FIFO output is available at the corresponding port, in the FWFT FIFO the first available data immediately appears at the output port without having to strobe the read enable signal, that is used as an *acknowledge* signal to clear the data content in the FIFO after it has been read.
The Ring Buffers can be substituted by FWFT FIFOs generated using the Xilinx FIFO Generator IPcore, that allows either to generate FIFOs using the FPGA sparse logic or using dedicated Block RAMs (BRAM) present in the FPGA. The main reason to usually prefer the custom implemented Ring Buffers is that it is more flexible to changes in the implementation, *i.e.* changes of the generic constant `buffer_size` or the input/output data size, that would require the FIFO IPcore to be regenerated. Moreover, when implementing FIFOs based on the use of BRAMs we should consider that the minimum BRAM size is 18 Kb or 36 Kb, that is too much if compared to a typical data size of 64 bits and buffer size of 2, so the use of BRAMs would end in a waste of resources since if a BRAM is only partially used the rest of the memory is not available for other purposes.

5.1.3 Single Clock FIFO FWFT Wrapper

Similarly to the previous case three modules are defined to be used with hit, stub or track data. This module is a *wrapper* that includes a FWFT FIFO based on a 18 Kb or 36 Kb BRAM (depending on the data size). The entity of the Hit Single Clock FIFO FWFT Wrapper is described as follows:

```
entity hit_single_clock_fifo_fwft_wrapper is
port (
```

[3] First In First Out

```
  clk       : in  std_logic;
  reset     : in  std_logic;
  w_en      : in  std_logic;
  r_en      : in  std_logic;
  data_in   : in  r_hitdata;
  data_out  : out r_hitdata;
  full      : out std_logic;
  not_empty : out std_logic
);
end hit_single_clock_fifo_fwft_wrapper;
```

The module has exactly the same structure of the Ring Buffer and the two modules can be considered as interchangeable, provided the fact the latter does not make us of generic value to declare the buffer depth.
The wrapper includes also the logic for the type conversion from `r_hitdata` to `std_logic_vector` and vice versa, in order to write and read the FIFO that accepts and provides `std_logic_vector` input and output data. The type conversion functions for hits, stubs and tracks are included in the `algorithmtools` package. The depth of the FIFO is chosen in order to be 512 in both the case of `r_hitdata_size` = 32 or `r_hitdata_size` = 64 ; in the first case one 18 Kb BRAM is used, while in the second case one 36 Kb BRAM is used. It has to be noted that 512×32 bits = 16 Kb, that corresponds to the user available memory of a 18 Kb BRAM, being 2 Kb reserved for parity check; similarly the user available memory of a 36 Kb BRAM is 32 Kb.
The Single Clock FIFO FWFT Wrapper modules are used as moderately large input buffers at the Stub Constructor inputs and at the Stub Switch inputs to absorb fluctuations in the data flow between main components of the hardware architecture.

5.1.4 N-way Dispatcher

The N-way Dispatcher is a module with one input and multiple outputs and two versions have been implemented for the hit and stub data, respectively. It receives a data from the only input and evaluates to which ports it has to be forwarded according to `x_address` and `y_address` content of the received data. The N-way Dispatcher can also be used as a FanOut according to the value of the `mode` generic value; in this case the data is always forwarded to all the output ports. The entity of the Hit N-way Dispatcher is called `hit_n_waydispatcher` and it is described as follows:

```
entity hit_n_waydispatcher is
generic (
  buffer_size     : integer := 2;
  x_outsize       : integer := 2;
  y_outsize       : integer := 2;
  x_address_index : integer := 0;
  y_address_index : integer := 0;
  x_level         : integer := 0;
```

```
  y_level         : integer := 0;
  x_overlap_level : integer := 0;
  y_overlap_level : integer := 0;
  mode            : integer := 1   -- mode: 0 -> fanout, 1 -> dispatcher
) ;
port (
  clk      : in std_logic;
  reset    : in std_logic;
  data_in  : in  r_hitdata_matrix( 0 downto 0, 0 downto 0 );  --hard coded dimension
  hold_in  : out r_hold_matrix( 0 downto 0, 0 downto 0 );     --hard coded dimension
  data_out : out r_hitdata_matrix( x_outsize - 1 downto 0, y_outsize - 1 downto 0 );
  hold_out : in  r_hold_matrix( x_outsize - 1 downto 0, y_outsize - 1 downto 0 )
);

constant x_insize : integer := 1;
constant y_insize : integer := 1;

constant x_outsize_nbits : integer := integer(ceil( log2( real( x_outsize ) ) ) );
constant y_outsize_nbits : integer := integer(ceil( log2( real( y_outsize ) ) ) );

constant x_address_index_vector : std_logic_vector(hit_x_address_size-1 downto 0)
  := std_logic_vector(to_unsigned(x_address_index,hit_x_address_size));
constant y_address_index_vector : std_logic_vector(hit_y_address_size-1 downto 0)
   := std_logic_vector(to_unsigned(y_address_index,hit_y_address_size));

end hit_n_waydispatcher;
```

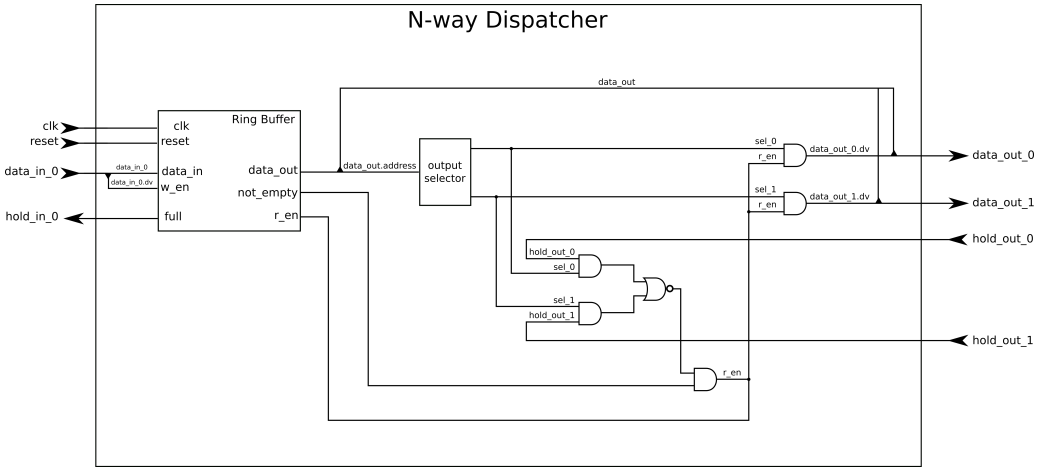

Figure 5.2: Scheme of the N-way Dispatcher with N = 2 outputs.

The scheme of an N-way Dispatcher with 2 output ports is shown in Fig. 5.2.

Many generic values are used to set relevant parameters when instantiating the entity. The meaning of these values will be discussed together with the description of the behaviour of the module.

The ports associated with the one and only input are `data_in` and `hold_in`, receiving and providing a `r_hitdata_matrix` and a `r_hold_matrix`, respectively. The dimensions of the matrices are hard-coded and equal to one; the constant values `x_insize` and `y_insize` are defined for better code readability. The choice of using matrices even for the case of one input is motivated to keep the same structure for inputs and outputs in this and other structures and modules described in this work. Similarly the ports associated with the outputs, `data_out` and `hold_out` are arranged in matrices whose dimensions are defined by the generic values `x_outsize` and `y_outsize`. Moreover, it has to be noted that according to the entity description `hold_in` is actually an output port and similarly `hold_out` is an input port, despite the naming convention. In order to avoid misunderstandings we will use *_in* and *_out* suffixes according to the direction of the data flow and we remind that the hold signals always propagate in the opposite direction (back propagation); this justifies the non obvious naming convention for the hold signals.

Now we describe the behaviour of the N-way Dispatcher module: a Ring Buffer, whose depth is defined by `buffer_size` is instantiated and `data_in` and `hold_in` are directly connected to the data input and the full flag of the buffer respectively, the `w_en` is given by the `dv` of the incoming `r_hitdata` (or `r_stubdata`); this means that every valid data at the input of the N-way Dispatcher is written into the buffer, if the buffer is not full.

When used in *fanout* mode, the N-Way Dispatcher performs the *AND* of all the signals from `hold_out`: if the result is negative the data from the Ring Buffer is forwarded to all the output ports, otherwise the output signal is set to `r_hitdata_zero` (or `r_stubdata_zero`) on all the output ports. If the Ring Buffer is empty the data output is set to zero by default. When used in *dispatcher* mode, the N-Way Dispatcher processes the data from the Ring Buffer (if not empty) and selects the output ports to which the data from the buffer should be forwarded, according to the content of `x_address` and `y_address`: if none of the selected ports is receiving the hold value (formally, if the hold value is equal to `r_hold_zero` for all the selected output ports) the data is forwarded to those port, while the output data to the remaining port is set to `r_hitdata_zero` (or `r_stubdata_zero`); if any of the selected ports is receiving a hold signal, all the output signals are set to zero. In both cases the read enable signal of the Ring Buffer is asserted only if the data has been forwarded.

The instantiated Ring Buffer receives the `clk` signal and operates synchronously with the input clock, while all the evaluations performed by the other logic of the N-way Dispatcher are performed asynchronously. Nevertheless, since all the input signals are synchronous with the input clock, the output signals result to be synchronous too; in particular at the rising edge of the `clk` signal, the `hold_out` signals (from the modules that follow the N-way Dispatcher) and the signals from the internal Ring Buffer are updated, then the output data are updated and the results remain unchanged until the input changes at the next rising edge of the clock signal. The `reset` signal is provided asynchronously and is used only by the Ring Buffer since the N-way Dispatcher has no internal signals to reset.

Output selection mechanism in *dispatcher* mode We previously stated that the selection of the active output ports is based on `x_address` and `y_address` data content that are processed as independent from each others. In this paragraph we provide more detail on the selection rules; in particular we will describe only how the first index of the output matrix is selected, based on the `x_address` value, while the second index is selected using the same rules applied to the `y_address` data content.
The `x_address` is split in different parts according to the values of `x_level`, `x_overlap_level` and `x_outsize_nbits`.

- the (x_level+x_outsize_nbits)-th bit of the data from the Ring Buffer is compared to the LSB[4] of the `x_address_index` generic values (converted from integer). In particular the `x_address_index` represents the position of the Dispatcher in a network of different modules. The result of the comparison is stored in the signal `x_is_from_border`;
- if `x_is_from_border` is '0', the `x_address` content of the data is stored in the signal `x_temp_address`, otherwise the *bitwise NOT* is applied and the value stored to the `x_temp_address` signal;
- `x_central_output_index_from_address` is evaluated by conversion of `x_temp_address( x_level+x_outsize_nbits-1 downto x_level )` to an integer value;
- the signal `x_rest_of_address` is built from `x_temp_address( x_level-1 downto x_overlap_level )`;
- the `x_left_edge` signal is evaluated by application of the *AND* operator to the result of the *bitwise NOT* applied to `x_rest_of_address`;
- the `x_right_edge` signal is evaluated by application of the *AND* operator to `x_rest_of_address`. it is worth noting that `x_left_edge` and `x_right_edge` can not be simultaneously equal to '1';
- all the useful quantity have now been evaluated; the "x" index of the selected output ports are
  `x_central_output_index_from_address`,
  `x_central_output_index_from_address - 1` if `x_left_edge` = '1',
  `x_central_output_index_from_address + 1` if `x_right_edge` = '1'.

All these considerations apply also to the selection of the output based on the `y_address` content. The result of merging the two selection processes is that one data can be forwarded to one, two or four output ports.
The N-way Dispatcher is, *de facto*, an intelligent 1-to-N demultiplexer, in which the selection of the output ports is based on the data from the input and the hold signals from the output ports. In order to save resources, only the *Data Valid* value of the output signals is evaluated in the selection process while all the other items within the Hit (or

[4] least significant bit

Stub) data record are directly forwarded from the Ring Buffer to all the output ports; this scheme allows to reduce the number of look-up tables needed to synthesize the module, moreover it does not affect the logic behaviour since any module receiving from the N-way Dispatcher will accept the data only if the *Data Valid* is equal to '1'.

5.1.5 N-way Merger

The N-way Merger is a module with multiple inputs that has the task of sorting the incoming data and forwarding them to the only output. It has been implemented for hit, stub and track data without differences. The entity of the Hit N-way Merger is called `hit_n_waymerger` and it is described as follows:

```
entity hit_n_waymerger is
generic (
  buffer_size     : integer := 2;
  x_insize        : integer := 4; -- x_insize must be a power of 2
  y_insize        : integer := 4; -- y_insize must be a power of 2
  x_address_index : integer := 0;
  y_address_index : integer := 0;
  x_level         : integer := 0;
  y_level         : integer := 0;
  mode            : integer := 0 -- mode: 0->normal mode; 1->fanout
);
port (
  clk      : in std_logic;
  reset    : in std_logic;
  data_in  : in  r_hitdata_matrix( x_insize-1 downto 0, y_insize-1 downto 0 );
  hold_in  : out r_hold_matrix( x_insize-1 downto 0, y_insize-1 downto 0 );
  data_out : out r_hitdata_matrix( 0 downto 0, 0 downto 0 ); -- hard coded dimension
  hold_out : in  r_hold_matrix( 0 downto 0, 0 downto 0 )     -- hard coded dimension
);

constant x_insize_nbits : integer := integer( ceil ( log2( real ( x_insize ) ) ) ) ;
constant y_insize_nbits : integer := integer( ceil ( log2( real ( y_insize ) ) ) ) ;

constant x_outsize: integer := 1;
constant y_outsize: integer := 1;

end hit_n_waymerger;
```

The scheme of an N-way Dispatcher with 2 input ports is shown in Fig. 5.3.
The `x_insize` and `y_insize` values define the dimension of the `data_in` and `hold_in` matrices; both the values must be a power of two in this implementation. The `x_address_index` and `y_address_index` are defined as for the N-way Dispatcher and represent the position of the module within a network.

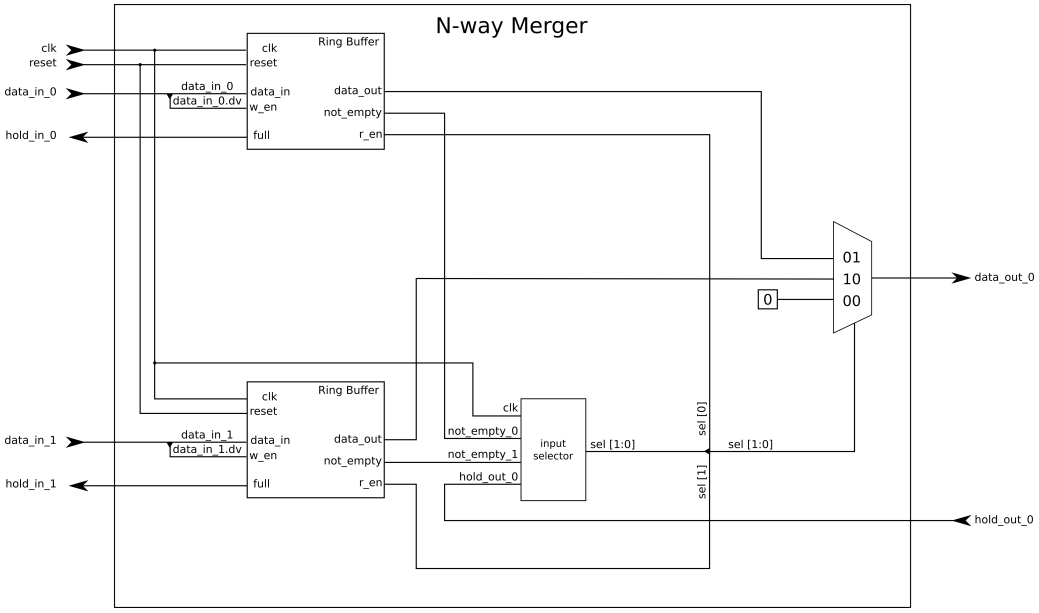

Figure 5.3: Scheme of the N-way Merger with N = 2 inputs.

The N-way Merger can work in two different modes, according to generic value `mode`; in particular if mode = 0 the `x_address` and `y_address` content of the output data is forwarded from the selected input data, while if mode = 1, that we will call *FanIn* mode, part of the `x_address` and `y_address` values are overwritten with the information of the index from which the data has been forwarded. The latter mode is used to keep track of the path of the data within a network of N-way Merger.
A Ring Buffer, whose depth is defined by `buffer_size`, is instantiated for each input of the N-way Merger. Similarly to the case of the N-way Dispatcher, the `data_in` matrix is connected to the data input port of the matrix of Ring Buffers, while the `hold_in` matrix is connected to the full flags of the buffers; the `w_en` are represented by the *Data Valid* values of the input data and every valid data is written into the corresponding buffer, if not full. The `r_en` is provided by the rest of the internal logic, that evaluates from which input the data has to be read and then forwards the data to the output port.
Before giving further details about the N-way Merger implementation a general description of its behaviour is given: if the `hold_out` is high or no data are available from the Ring Buffers, nothing is forwarded to the output; for data not flagged as *End Of Event*, if one or multiple data are available the one which first entered the N-way Merger is forwarded to the output; in case of an *End Of Event* signal received from one input, the stream of data on that line is paused until the *End Of Event* signals are received from all the other inputs and then a unique *End Of Event* is provided at the output.

Input selection mechanism It is important to note that the selection mechanism of the input from which the data has to be read treats the input indexes as a 1-dimensional array, instead of a matrix. This is done by simply unrolling the matrix whose dimensions

are given by `x_insize` and `x_insize`, to an array which dimension is given by x_insize × y_insize.
This is necessary because the selection process is based on a pyramidal network of *Merger Selectors* that perform a binary selection of the input.
The scheme of the N-way Merger with 2, 4 and 8 inputs is shown in Fig. 5.4.

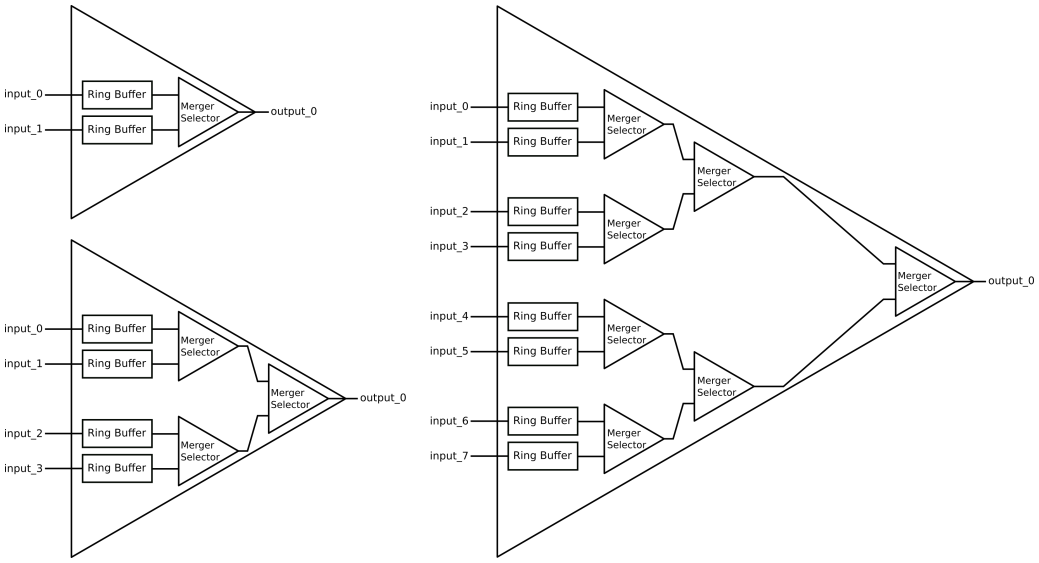

Figure 5.4: Structure of the N-way merger with N = 2, 4, 8 inputs. A Ring Buffer is instantiated for each input, then a network of $N-1$ interconnected Merger Selectors. The output of the last Merger Selector is connected to the only output of the N-way Merger.

The N-way Merger can be considered as an intelligent N-to-1 multiplexer that selects the input signal to be forwarded, on the basis of the input signals themselves. In particular the N-way Merger is composed of $m = \log_2(N)$ layers of Merger Selectors that perform the selection between two inputs. Each layer has 2^{m-1-i} Merger Selectors with $i \in [0, m-1]$ for a total of $N-1$ submodules.
The logic of the Merger Selector is shown in Fig. 5.5 and can be summarized as follows: if no data is available from the inputs or the `dv_out` is low, nothing is forwarded; if one or multiple data are available, one data is forwarded and the corresponding `r_en` is asserted, while the other is deasserted. The selection of the input from which the data has to be forwarded is based on these rules, considering data not flagged as *End Of Event*: if only one input is active, then it is selected; if both the inputs are active, the input that was not active during the previous clock cycle is selected.
It is worth noting that all the signals in the Merger Selector are evaluated asynchronously, except for the `prev_path` value that contains the information about the active input index during the previous clock cycle. The asynchronous evaluation allows the N-way Merger to promptly evaluate which input has to be activated and to forward the corresponding data from the Ring Buffer to the N-way Merger output, making its latency independent from the number of inputs. It is also important to point out that storing the `prev_path`

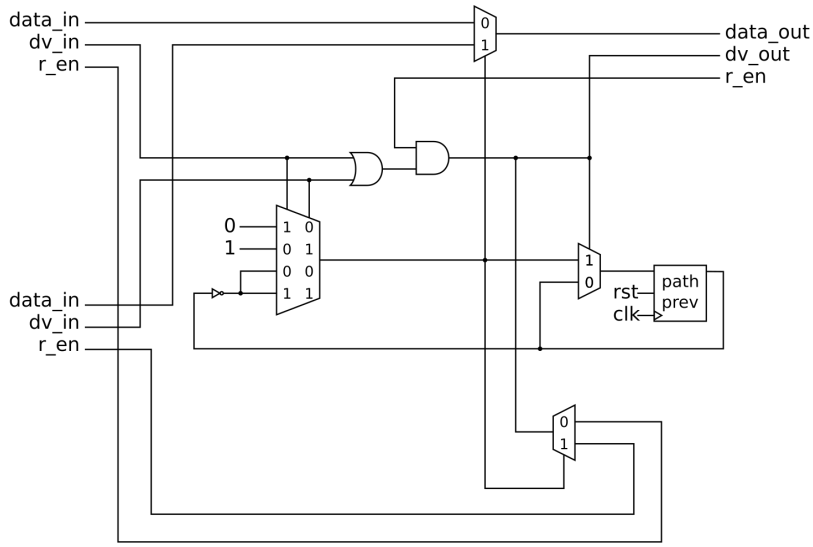

Figure 5.5: Internal logic of the Merger Selector used in the module is used in the N-way Merger. The module acts as a 2-to-1 multiplexer in which the selection of the data to be forwarded is performed according to the data themselves.

signal value inside all the Merger Selectors plays a key role in queueing the data, allowing the first data that was available at the input to be first forwarded to the output.

5.1.6 NxN Sorter

The NxN Sorter is a module with multiple inputs and multiple outputs composed of a layer of N-way Dispatchers and a layer of N-way Mergers. Despite its name, the number of inputs and outputs don't have to be necessarily identical.
It has been implemented for hit and stub data. The entity of the Hit NxN Sorter is called `hit_nxn_sorter` and it is described as follows:

```
entity hit_nxn_sorter is
generic (
  dispatcher_buffer_size : integer := 2;
  merger_buffer_size     : integer := 2;
  x_insize         : integer := 2;
  y_insize         : integer := 2;
  x_outsize        : integer := 2;
  y_outsize        : integer := 2;
  x_address_index  : integer := 0;
  y_address_index  : integer := 0;
  x_level          : integer := 0;
  y_level          : integer := 0
```

```
);
port (
  clk      : in STD_LOGIC;
  reset    : in STD_LOGIC;
  data_in  : in  r_hitdata_matrix( x_insize - 1 downto 0, y_insize - 1 downto 0 );
  hold_in  : out r_hold_matrix( x_insize - 1 downto 0, y_insize - 1 downto 0 );
  data_out : out r_hitdata_matrix( x_outsize - 1 downto 0, y_outsize - 1 downto 0 );
  hold_out : in  r_hold_matrix( x_outsize - 1 downto 0, y_outsize - 1 downto 0 )
);

constant x_insize_nbits  : integer := integer(ceil(log2(real( x_insize ))));
constant y_insize_nbits  : integer := integer(ceil(log2(real( y_insize ))));

constant x_outsize_nbits : integer := integer(ceil(log2(real( x_outsize ))));
constant y_outsize_nbits : integer := integer(ceil(log2(real( y_outsize ))));

end hit_nxn_sorter;
```

The NxN Sorter has no internal logic, except for the connections between its inputs and the layer of N-way Dispatchers, the internal connections to the N-way Mergers, and the connections to the output.
The generic values `dispatcher_buffer_size` and `merger_buffer_size` represent the values to be passed to the dispatchers and mergers, respectively, as their generic value `buffer_size`. All the other generic values are directly passed to the internal components.
A grid of x_insize × y_insize N-way Dispatchers forms the first layer of the NxN Sorter. The N-way Dispatchers receive the data directly from the inputs of the NxN Sorter. The dimensions of each N-way Dispatcher are *x_outsize* and *x_outsize*.
A grid of x_outsize × y_outsize N-way Mergers forms the second layer of the NxN Sorter. The N-way Mergers provide the data directly to the outputs of the NxN Sorter. The dimensions of each N-way Merger are *x_insize* and *x_insize*.
The connections between the two layers are arranged in order to connect each input to all the outputs of the NxN Sorter. In particular the (i_x-th,i_y-th) output of the (j_x-th,j_y-th) N-way Dispatcher is connected to the (j_x-th,j_y-th) input of the (i_x-th,i_y-th) N-way Merger.
The scheme of the NxN Sorter is shown in Fig. 5.6 for N = 2, 4. In order to keep the visual description of the module as simple as possible, we considered a one dimensional graphical representation of the inputs and the outputs.
An example of NxN Sorter with non equal number of inputs and outputs is shown in Fig. 5.7.

5.1.7 NxN Switch

The Switch is a module with multiple inputs and outputs composed of a network of NxN Sorters, N-way Mergers, N-way Dispatchers arranged to connect all the input to all the

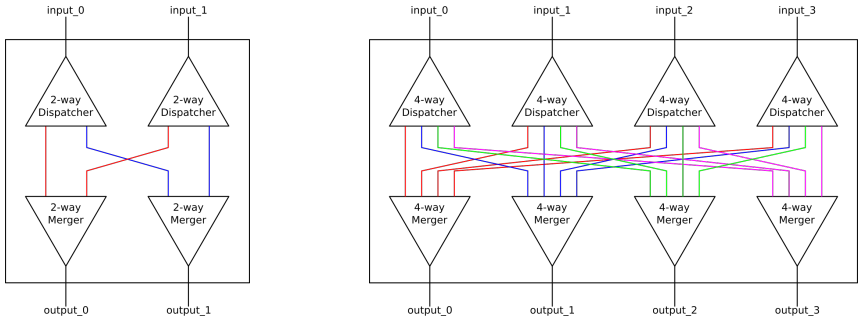

Figure 5.6: Structure of an NxN Sorter with 2 inputs, 2 outputs (left) and a with 4 inputs, 4 outputs (right). The data flow is intended from top to bottom. A row of N-way Dispatchers deliver the data from the inputs to a row of N-way Mergers connected to the outputs, according to the data address. Connections associated with different addresses are highlighted using different colours.

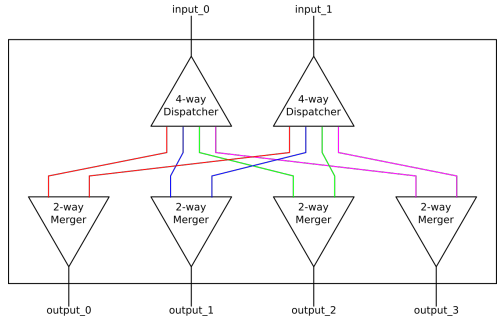

Figure 5.7: Structure of an NxN Sorter with 2 inputs, 4 outputs. The data flow is intended from top to bottom. Two 4-way Dispatchers deliver the data from the inputs to four 2-way Mergers connected to the outputs, according to the data address. Connections associated with different addresses are highlighted using different colours.

outputs and it delivers the data to the proper output according to their `region_address` content.
The NxN Switch is implemented for Hit and Stub data; the entity is called `hit_nxn_switch` and it is described as follows:

```
entity hit_nxn_switch is
generic (
  dispatcher_buffer_size : integer := 2;
  merger_buffer_size     : integer := 2;
  x_sorter_size : integer := 2;
  y_sorter_size : integer := 2;
  x_insize : integer := 4;
  y_insize : integer := 4;
  x_outsize : integer := 4;
  y_outsize : integer := 4;
```

```
  x_address_index : integer := 0;
  y_address_index : integer := 0;
  x_level : integer := 0;
  y_level : integer := 0
);
port (
  clk      : in std_logic;
  reset    : in std_logic;
  data_in  : in  r_hitdata_matrix( x_insize - 1 downto 0, y_insize - 1 downto 0 );
  hold_in  : out r_hold_matrix( x_insize - 1 downto 0, y_insize - 1 downto 0 );
  data_out : out r_hitdata_matrix( x_outsize - 1 downto 0, y_outsize - 1 downto 0 );
  hold_out : in  r_hold_matrix( x_outsize - 1 downto 0, y_outsize - 1 downto 0 )
);

constant x_switch_size : integer :=
    integer(realmin(real(x_insize),real(x_outsize)));
constant y_switch_size : integer :=
    integer(realmin(real(y_insize),real(y_outsize)));

constant x_sorter_size_nbits : integer :=
    integer(ceil(log2(real( x_sorter_size ))));
constant y_sorter_size_nbits : integer :=
    integer(ceil(log2(real( y_sorter_size ))));

constant x_switch_size_nbits : integer :=
    integer(ceil(log2(real( x_switch_size ))));
constant y_switch_size_nbits : integer :=
    integer(ceil(log2(real( y_switch_size ))));

constant x_outsize_nbits : integer := integer(ceil(log2(real( x_outsize ))));
constant y_outsize_nbits : integer := integer(ceil(log2(real( y_outsize ))));

end hit_nxn_switch;
```

First we will describe the topology of Switches with equal number of inputs and outputs, then the general case will be discussed.
A *square* Switch is built using a network of NxN Sorters and we will describe a Switch that is composed of NxN Sorters with N=2, that we will call 2x2 Sorters, since it is easier to visualize. In this case the simplest and smallest Switch is the 2x2 Switch that is exactly a 2x2 Sorter. In order to construct a 4x4 Switch two layers of 2x2 Sorters are used, while to construct a 8x8 Switch three layers of 2x2 Sorters are used, arranged as in Fig. 5.8.
In general, to construct an NxN Switch using MxM Sorters, $\log_M N$ layers of N/M MxM Sorters are used[5] for a total of $N/M \ \log_M N$ MxM Sorters.

[5]Both $\log_M N$ and N/M must be integer numbers.

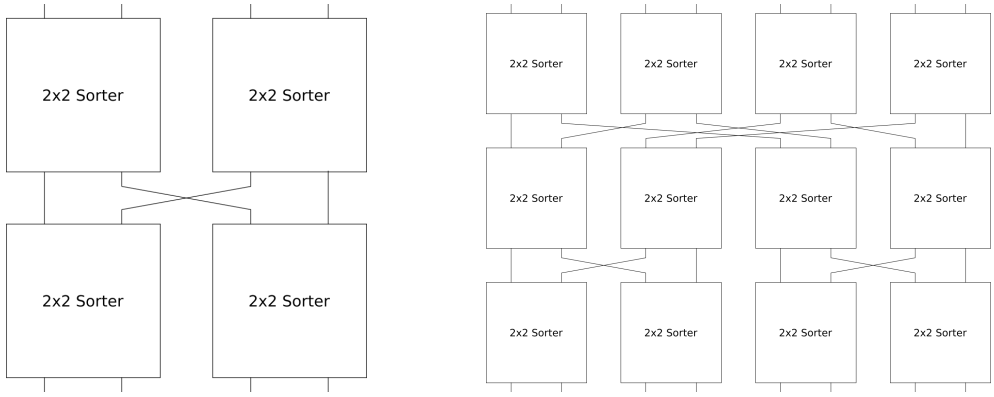

Figure 5.8: Structure of a 4x4 Switch (left) and a 8x8 Switch (right). The NxN Switch is composed of a network of Sorters; 2x2 Sorters are used in this example.

The internal connections between the 2x2 Sorters are arranged in such a way that the first layer of 2x2 Sorters is directly connected to the Switch inputs and dispatches the data to the first or the second half of 2x2 Sorters in the second layer. The first half of 2x2 Sorters of the second layer dispatches the data to the first and second quarter of Sorters in the third layers while the second half of 2x2 Sorters of the second layers is connected to the third and fourth quarter of Sorters. This structure is repeated until the last layer of Sorters, whose output are connected to the Switch outputs.

Alternative description of the Switch The structure of the NxN Switch can be described in a simpler way as a layer of $N/2$ 2x2 Sorters followed by a layer of two (N/2)x(N/2) Switches, that we will call sub-Switches: the first output of each 2x2 Sorter is connected to an input of the first sub-Switch, the second output of each 2x2 Sorter is connected to an input of the second sub-Switch. As an example, the scheme of the 8x8 Switch is shown in Fig. 5.9.

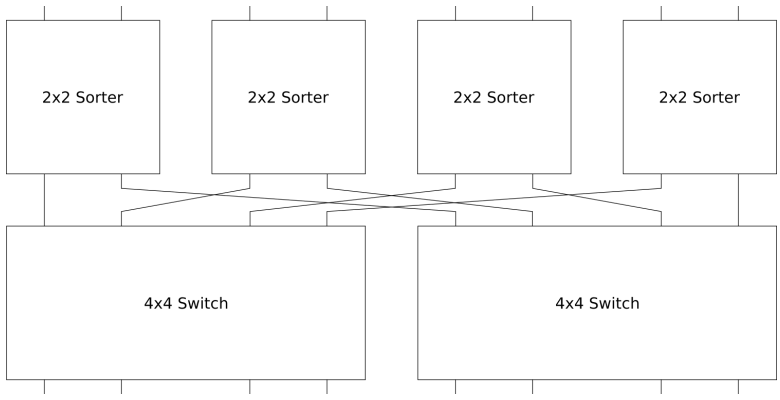

Figure 5.9: Recursive structure of a 8x8 Switch, built using four 2-way Sorters and two 4x4 sub-Switches.

Also in this case, the NxN Switch can be built using MxM Sorters, with a first layer of N/M MxM Sorters connected to a layer of M (N/M)x(N/M) sub-Switches.
Using this descriptive approach, the scheme of the internal connections is very simplified: the i-th output the j-th Sorter is connected to the j-th input of the i-th sub-Switch.

Extension to non degenerate case In the previous examples, we illustrated the scheme of connections in the case of 2x2 Sorters. We have to note that a 2x2 Sorter has 2 inputs that could be the case of *x_n_inputs* = 2 and *y_n_inputs* = 1, so the `data_in` and `hold_in` matrices degenerated to 1-dimensional arrays; the same consideration is valid for the number of outputs of the Sorter. In general in the Switch implementation we always used 4x4 Sorters, in which *x_n_inputs* = *y_n_inputs* = 2, and *x_n_outputs* = *y_n_outputs* = 2. All the previous considerations are still valid if we consider the *non degenerate* case by adding an extra dimension (namely the *y dimension*) to the example shown in the previous description, paying attention to satisfy the following equation that defines the number of layers needed in the Switch, that is log(x_n_inputs/x_sorter_size) = log(y_n_inputs/y_sorter_size).

Non-square Switch Switches with different number of inputs and outputs can be constructed. When x_n_outputs > x_n_inputs and y_n_outputs > y_n_inputs , the Switch is constructed by connecting N-way Dispatchers to the outputs of a *square* Switch in order to fit the requested number of *x_n_outputs* and *y_n_outputs*. In the opposite case N-way Mergers are instantiated before a *square* Switch.
Two examples of non-square Switches are shown in Fig. 5.10. In the first example the number of outputs is greater than the number of inputs and a layer of N-way dispatchers is used; vice versa, in the second example a layer of N-way Mergers is used.

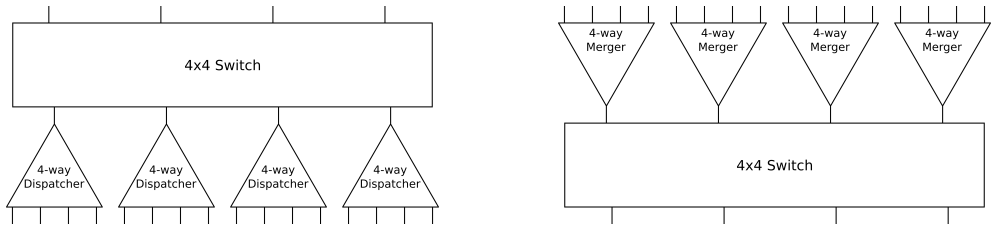

Figure 5.10: Structure of a 4x16 Switch (left) and a 16x4 Switch (right). The non-square Switch is composed of a square Switch connected to a layer of N-way Mergers or N-way Dispatchers, depending on the number of inputs and outputs.

5.1.8 Stub Maker

The Stub Maker is the basic module of the Stub Constructor, that we will discuss later, and multiple Stub Maker modules are instantiated for each couple of detector planes in order to identify the stubs from couples of hits that are compatible with a particle track from the beam interaction region. The entity is called `stub_maker` and it is described as follows:

```
entity stub_maker is
generic (
  bufferdata_depth : integer := 8;
  x_outsize : integer := 1;
  y_outsize : integer := 1
) ;
port (
  clk       : in std_logic;
  reset     : in std_logic;
  data_in   : in  r_hitdata_array( 2-1 downto 0 ); -- hard coded dimension
  hold_in   : out r_hold_array( 2-1 downto 0 );    -- hard coded dimension
  data_out  : out r_stubdata_matrix(x_outsize-1 downto 0, y_outsize-1 downto 0);
  hold_out  : in  r_hold_matrix(x_outsize-1 downto 0, y_outsize-1 downto 0)
);

constant insize : integer := 2; -- hard coded dimension

end stub_maker;
```

The module has two inputs, receiving hit data from two compatible regions of pixels in adjacent planes. The definition of the regions of pixels is based on the geometry of the tracking detectors and the interaction region and is obtained from the simulation.
For each input a buffer (not a Ring Buffer), whose dimension is given by the generic parameter `bufferdata_depth`, is instantiated. Hit data are temporarily stored in both the buffers until the *End Of Event* signals are received (one for each input port), then a process to evaluate the combinatorics between hits from one and the adjacent detector starts. All the combinations between the hits in the first and second detector regions are evaluated but, in principle, not all are compatible with the trajectory of a particle from the beam interaction region; each candidate stub is checked and, if valid, provided to one of the output ports.
The number of output ports is defined by the generic parameters `x_outsize` and `y_outsize`, that formally are the dimensions of the `r_stubdata_matrix` provided in output. The simplest case is represented by `x_outsize` = 1 and `y_outsize` = 1, in which all the hit combinations are evaluated in a serialized double loop: the first hit in the first buffer is selected and a candidate stub is formed by combining it with one hit of the second buffer; when all the hits in the second buffers have been checked the loop starts over considering the second hit in the first buffer and all the combinations with hits of the second buffer; the process continues until all the combinations have been checked.
After the *End Of Event* signal has been received at each input, the corresponding `hold_in` signal is raised; the hold signals stay high until the combinatoric process complete, then the buffers are reset and the Stub Maker is able to process hits from the following event.
It is important to note that due to the fixed depth of the buffer, if the number of hit data on one input port exceeds the dimension defined by `bufferdata_depth`, the extra ones are discarded and lost, without raising the `hold_in` signals. In fact, if on one hand raising the hold signal would prevent data to be lost, on the other hand this would prevent also

the *End Of Event* signal to reach the Stub Maker that would end in a stuck state, not being able to process data any more.
The general behaviour of the Stub Maker has been described. A simplified scheme of the module is shown in Fig. 5.11, in which an example case with `bufferdata_depth` = 8, `x_outsize` = `y_outsize` = 2 is represented. The incoming data flow to the buffers until the *End Of Event* signals are received; in this example each buffer is divided in two halves, in particular the even and the odd positions of each buffer are processed independently; four blocks evaluate in parallel the hit combinations and provide the stub data on a dedicated output port.

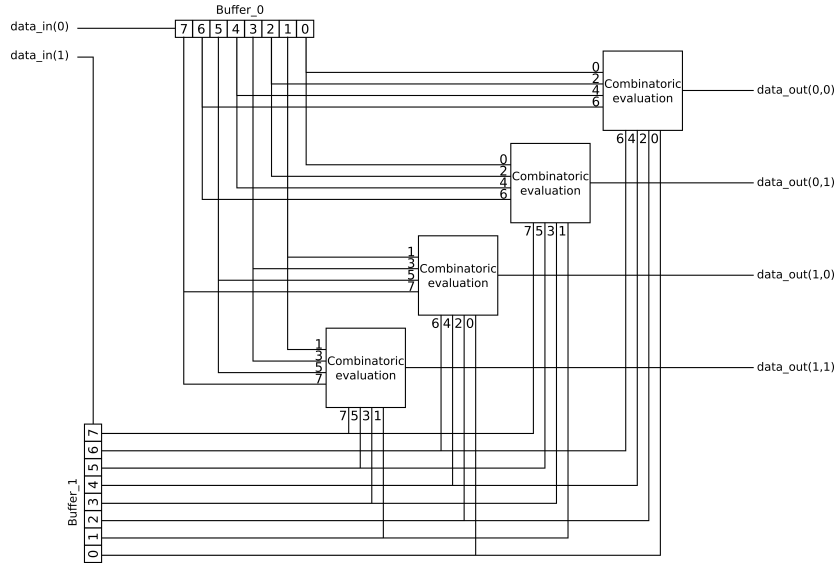

Figure 5.11: Schematic view of the Stub Maker with *bufferdata_depth* = 8, *x_outsize* = *y_outsize* = 2. The incoming data are stored into the buffers, then four blocks evaluate in parallel the possible combinations of hit data from the two buffers. In this example each block processes only half of the hit data from each buffer and provide the stub data on a dedicated output port.

Detailed description of the Stub Maker logic The behavioural logic of the Stub Maker is implemented using a finite-state machine with three states that we called *FSM_START*, *FSM_WRITING*, *FSM_READING*. In this paragraph we describe the logic in more detail, through the description of its states.

- *FSM_START*: all the internal signals are initialized, the content of the buffers is cleared, then the state machine moves to the next state *FSM_WRITING*. The `hold_in` signal is high for both the inputs, since the buffers are not able to accept data while resetting.

- *FSM_WRITING*: the `hold_in` is released and the incoming data are written into the buffers. The `hold_in` is never raised during this phase, even if the buffers are full and the extra data are lost. For each input a latch signal `eoe` is raised when the

End Of Event signal is received and the corresponding `hold_in` signal is raised. When both the `eoe` signals are high, the finite-state machine moves to the following state FSM_READING.

- *FSM_READING*: the `hold_in` signals stay high for both the inputs. The first buffer is divided in *x_outsize* parts, the second buffer is divided in *y_outsize* parts and a total of x_outsize × y_outsize blocks evaluate in parallel the combinations of hits from the different parts of the buffers. In particular the (i,j)-th block process the hit data stored in the first and second buffer at the positions that satisfy the following rules: buffer_0_i *mod* x_outsize $= i$ and buffer_1_j *mod* y_outsize $= j$. The choice to slice the buffers in this way, instead of dividing the buffers in fixed parts is motivated by the fact that the buffers are not always completely filled and this would lead to an unbalanced number of data in the different buffer parts. Using this method the maximum difference in occupancy of the different buffer parts is one.
  Each block evaluates one combination per clock cycle and provides it in output, if the corresponding `hold_out` signal is not high, the it moves to the next candidate combination. This process is achieved through a double loop on the buffer positions associated with that block. When one block has finished the loop, a latch signal `reading_complete` is raised.
  When all the blocks have finished the finite state machine moves to the initial state *FSM_START*.

5.1.9 Stub Constructor

The Stub Constructor is one of the main modules on which the full architecture is based and one module is instantiated for each couple of detector planes; in particular the Stub Constructor is composed of two Hit Switches and a pool of Stub Makers. Each Hit Switch receives the data from one detector plane and delivers them to the Stub Makers according to pre-computed paths obtained from simulation.
The entity is called `stub_constructor` and it is described as follows:

```
entity stub_constructor is
generic (
  bufferdata_depth       : integer := 8;
  dispatcher_buffer_size : integer := 2;
  merger_buffer_size     : integer := 2;
  fanin_buffer_size      : integer := 1;
  x_sorter_size          : integer := 2;
  y_sorter_size          : integer := 2;
  x_stub_maker_size      : integer := 2;
  y_stub_maker_size      : integer := 2;
  x_insize               : integer := 2;
  y_insize               : integer := 2;
  x_outsize              : integer := 4;
  y_outsize              : integer := 4;
```

```
  x_address_index        : integer := 0;
  y_address_index        : integer := 0;
  x_level                : integer := 0;
  y_level                : integer := 0
) ;
port (
  clk          : in std_logic;
  reset        : in std_logic;
  data_det0_in : in  r_hitdata_matrix(x_insize-1 downto 0, y_insize-1 downto 0);
  hold_det0_in : out r_hold_matrix(x_insize-1 downto 0, y_insize-1 downto 0);
  data_det1_in : in  r_hitdata_matrix(x_insize-1 downto 0, y_insize-1 downto 0);
  hold_det1_in : out r_hold_matrix(x_insize-1 downto 0, y_insize-1 downto 0);
  data_out     : out r_stubdata_matrix(x_outsize-1 downto 0, y_outsize-1 downto 0);
  hold_out     : in  r_hold_matrix(x_outsize-1 downto 0, y_outsize-1 downto 0)
);
end stub_constructor;
```

The module has two groups of inputs, associated with the first and second detector planes in which the stub search will be performed; the number of inputs for each detector depends on the number of lines used to carry out the hit data and it is set through the generic parameters `x_insize` and `y_insize`. The total number of outputs is set through the generic parameters `x_outsize` and `y_outsize`, while the number of instantiated `Stub Maker` modules is evaluated from the ratio between the total number of output lines of the Stub Constructor and the number of output lines of each Stub Maker. All the other generic parameters are used for the configuration of the internal sub-modules.
A simplified scheme of the module is shown in Fig. 5.12, in which only one line from each detector plane to the corresponding Hit Switch is shown and Stub Makers with only one output are represented.
Similarly to other modules described in the previous subsection, the Stub Constructor is based only on the logic provided by the internal interconnected sub-modules. In particular the scheme of the internal connections is the following: the inputs associated with the first (second) detector plane are directly connected to the corresponding Hit Switch inputs; each output of the first (second) Hit Switch is connected to the first (second) input of each Stub Maker; the outputs of the Stub Makers are directly connected to the output ports of the Stub Constructor.

5.1.10 Engine Region

The Engine Region is a container of Engines that cover a region of the space of the track parameters. The entity is called `engine_region` and it is described as follows:

```
entity engine_region is
generic (
x_address_index : integer range 0 to 2**x_region_address_size-1 := 0;
y_address_index : integer range 0 to 2**y_region_address_size-1 := 0;
```

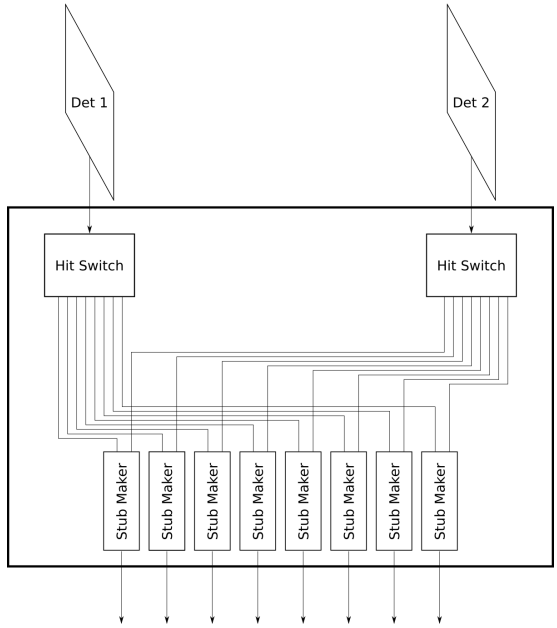

Figure 5.12: Scheme of the Stub Constructor, composed of a Hit Switch for each detector plane and a pool of Stub Makers.

```
x_n_engines     : integer := 2**x_engine_address_size;
y_n_engines     : integer := 2**y_engine_address_size
) ;
port (
clk       : in std_logic;
reset     : in std_logic;
data_in   : in  r_stubdata_matrix(x_n_engines-1 downto 0, y_n_engines-1 downto 0);
hold_in   : out r_hold_matrix(x_n_engines-1 downto 0, y_n_engines-1 downto 0);
data_out  : out r_trackdata_matrix(x_n_engines-1 downto 0, y_n_engines-1 downto 0);
hold_out  : in  r_hold_matrix(x_n_engines-1 downto 0, y_n_engines-1 downto 0)
);
end engine_region;
```

The input of an Engine Region is represented by an `r_stub_data_matrix` and the output is represented by a `r_trackdata_matrix`, whose dimensions are given by the generic parameters `x_n_engines` and `y_n_engines`.

In fact, an Engine Region is a wrapper for a grid of Engine modules that are connected one by one to the input and outputs of the regions. Multiple Engine Regions are connected to the Stub Switch output, while the output of the Engine Regions are connected to the Track FanIn, that collects the results of the Engines.

5.1.11 Engine

The Engine is the module that mimics the behaviour of a cellular unit and its first neighbour cells, placed along the x_+ axis and the y_+ axis. Each Engine has an unique address inside the Engine Region and its (x_+, y_+) coordinates depends on the position of the Engine Region within the region of the track parameters, and on the position of the Engine within the Engine Region.
The entity is called `engine` and it is described by:

```
entity engine is
generic (
  x_address_index : integer range 0 to 2**x_engine_address_size-1 := 0;
  y_address_index : integer range 0 to 2**y_engine_address_size-1 := 0
) ;
port (
  clk      : in std_logic;
  reset    : in std_logic;
  data_in  : in r_stubdata;
  hold_in  : out r_hold;
  data_out : out r_trackdata;
  hold_out : in  r_hold
);

end engine;
```

The Engine receives an `r_stubdata` in input and provides an `r_trackdata` in output, corresponding to a track data, when a candidate track has been identified by the Engine logic.
The behaviour of the Engine is slightly different from the one described in the algorithm section for practical and technical reasons. In the algorithm description we said that a stub data is accepted by the cellular unit if the distance in the (x_+, y_+) reference plane is less than a certain value, according to the Eq. 4.7. Nevertheless it has to be noted that the stub data path is already evaluated in the Switch, so the stub data are already delivered to the proper Engines and there is no need to check again if the distances of the stub projection to the Engine is within a fixed tolerance. For this reason a stub data is accepted by default by the Engine that corresponds to its destination address. Moreover we recall that the N-way Dispatchers, hence the Switch, are programmed to deliver the data to one or multiple output, hence multiple Engines, according to the LSB values of the data address. In particular a data is sent to adjacent Engines if the stub projection falls halfway between the Engines. In practice if we consider that the Engines are distributed over a two dimensional grid, a stub data is distributed to multiple Engines if the stub projection lies near the border of the cells in the grid.
A simplified schematic view of the Engine is shown in Fig. 5.13.
In the following we describe the logic of the Engine:

- a stub data is received at the input port `data_in` and a *total* data counter is incremented;

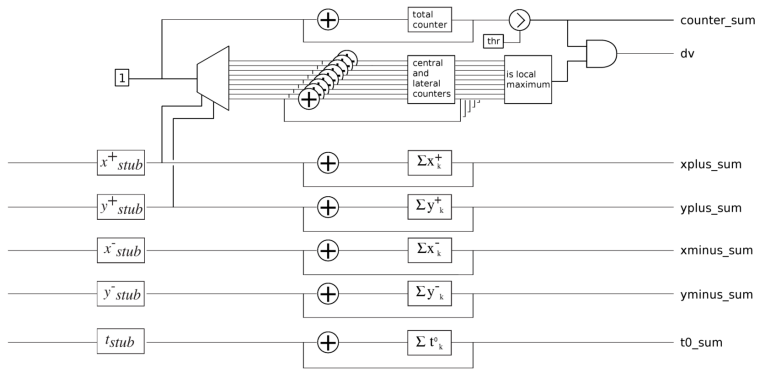

Figure 5.13: Behavioural logic of the Engine. The Engine receives the stub data, accumulates the $(x_+, y_+, x_-, y_-, t_0)$ values, increments the proper counters.

- an Engine corresponds to one central and eight lateral cells; the `x_address` and `y_address` contents of the incoming data are compared to the `x_address_index` and `y_address_index` values of the Engines, respectively. The result of the comparison identifies in which cell the stub projection lies. A counter is instantiated for each cell and the one corresponding to the activated one is incremented;

- a `record` is defined within the Engine to store the accumulated $(x_+, y_+, x_-, y_-, t_0)$ values of the incoming stubs. In particular, for each incoming data the x_+ value is summed to the x_+ values of the previous data, and similarly for the other track parameters. It has to be noted that the (x_-, y_-, t_0) values where already defined in the `r_stubdata` record, while the (x_+, y_+) values are retrieved from the `x_address` and `y_address` contents, respectively. In particular the x_+ is obtained from the LSBs of the `x_address`, that identifies the coordinate of the stub projection within the Engine acceptance area. The same consideration is valid for the y_+ coordinate evaluation;

- the identification of the activated cell and the sum of the stub contributions are performed in parallel during one clock cycle. The process is repeated for any incoming stub data during the following clock cycles.

- when an incoming data flagged as *End Of Event* is received, the sum process stops and the counter values are checked to evaluate if a track has been identified. In particular the counter corresponding to the central cell is required to be greater then zero and the total data counter is required to greater than or equal to a fixed threshold, that is set to 2 in order to recognize tracks with at least two identified stubs (4 hits) in the tracking detector. Moreover the central cell counter value is compared to the counter of four lateral cells, along the x_+ and y_+ axes; the central counter is required to be grater than or equal to any of the lateral cell counters that means the central cell of the Engine has been identified as a local maximum. If both the conditions are satisfied the results of the sums of the stub parameters and the

value of the total counter are provided together with other relevant information, as output from the Engine, formatted as a `r_trackdata`. The track parameters will be evaluated as the average of the stub parameters that contributed to the identified track as the result of the division of the $(x_+, y_+, x_-, y_-, t_0)$ contents of the track data by the total counter. We prefer to not evaluate the division within the Engine logic since it would be resource and/or time consuming, depending on implementation of the division algorithm in FPGA. The complete evaluation of the track parameters could be included after the Track FanIns, allowing for resources sharing, instead of having to replicate the division logic for all the Engines. The other possibility, considered as the baseline strategy is to carry out the track information in the previously described format and perform the final track parameters evaluation in the following stage of process, *i.e.* in a PC receiving and processing the track data.

Another technical difference from the description of the Engine that has been provided in the algorithm section and its practical implementation is evident from the architecture description; in fact, the evaluation of the *weight* response of the cells to the stub projections is not performed on the basis of a Gaussian function and it is substituted by the evaluation of the data counters for the central and lateral cells, that corresponds to the substitution of the Gaussian shape with a binary response based on the distance from the cells. This choice is motivated by the necessity to reduce the logic needed for the Engine implementation, in particular for the Gaussian function evaluation and for the track fitting results described by Eqs. 4.16 and 4.17, that would require extra logic as Look-Up Tables to perform the evaluation of the logarithmic functions.

Nevertheless it should be noted that the choice of the Gaussian response is motivated by the need of having a near-zero response to stubs whose projections are far from the position of the cell in the tracking reference plane, but the Switch already provides this kind of suppression, since the data are delivered only to the Engines with expected non zero response.

5.2 Hold logic details, latency and throughput of the implemented modules

In the previous section we stated that all the modules have been implemented putting particular attention on minimizing the latency. The latency is defined as the time required by the logic to complete a computation. In particular the latency of a module can be defined as the time interval between the arrival of the data at the input(s) and the processed data appearing at the output(s). We can observe that all the basic components of the architecture act as synchronous circuits, and in particular they all share the same clock signal. For this reason we can measure the latency in terms of clock cycles: this makes the measurement independent from the value of the system clock. As an example we will see that the minimum latency of a Ring Buffer is equal to one clock cycle, that is equivalent to 2.5 ns if we consider a system clock frequency of 400 MHz, while it would be 3.125 ns using a system clock frequency of 320 MHz.

We recall that a hold logic has been implemented, based on the back propagation of signals that provide information on the status of the module, in particular on its ability to accept a data. According to this we have to note that the latency of each module will be influenced by the status of the module itself and the hold signal coming from the following module, that has to receive the processed data. For this reason we define the minimum latency of a module, as the minimum achievable latency by the module when no hold signals are received and all the data can be processed without introducing any pause in the data flow. In particular the simplest case where a hold signal could be generated is represented by the N-way Merger, when multiple data are available at the different inputs, but only one data for clock cycle can be forwarded to the only output port. In this case, if other data access the N-way Merger and the buffers completely fill, the hold signal is generated and back propagated at the input ports. We also have to note that when a module raises the hold signals, it does not necessarily generates a pause in the data flow. In fact, the module that is receiving a hold signal could have no data available at its output and so its behaviour is not affected by the received hold signal. In the opposite case, the data is kept until the hold signal is deasserted; if, during this period, the buffers fill up, the module raises the hold signal that is back-propagated to the previous module.
Together with the concept of latency, we define the throughput of a module as the processing rate or, equivalently, the data transfer rate inside the module. Again this is related to the clock frequency and its value will be provided in terms of number of data (that can be `r_hitdata`, `r_stubdata` or `r_trackdata`) that can be processed during one clock cycle. In particular the throughput of a module also depends on the hold signals and the maximum throughput is achieved when no hold signals is received.

Ring Buffer latency and throughput According to the description of the Ring Buffer and the Single Clock FIFO FWFT Wrapper, these modules behave exactly in the same way and any Ring Buffer can be substituted by a Single Clock FIFO FWFT Wrapper in the implementation, and vice versa. For this reason any consideration that is valid for the Ring Buffer can be applied also to the Single Clock FIFO FWFT Wrapper module.
The minimum latency of the Ring Buffer is one clock cycle. The simplest case is represented by a data accessing an empty Ring Buffer. The data is processed at the rising edge of the reference clock and stored in one of the buffer positions. At the next clock cycle the data is already available at the output. If another data accesses the Ring Buffer during this clock cycle, it is stored in the following buffer position and it will be available at the output during the following clock cycle. If no hold signal is received at the `hold_out` port, no pauses are introduced in the data flow and the occupancy of the buffer remains limited to one, in this simple example. On the other hand, if a hold signal is received and the Ring Buffer is still receiving data, these are stored in the available buffer positions until the buffer is full: only in this case the `hold_in` signal is raised and the module will stay in the same state until the `hold_out` is released and the Ring Buffer can restart to output the data and free the buffer positions that will be able to accept further incoming data.
We recall that the role of the Ring Buffer (and the Single Clock FIFO FWFT Wrapper)

is to absorb the fluctuations of the data flow in the modules in which that Ring Buffer is used, as the N-way Dispatcher or the N-way Merger. In particular the maximum data throughput of the Ring Buffer is defined as 1 data per clock cycle and, considering that the maximum input data rate is also 1 data per clock cycle, the Ring Buffer is available to process a continuous stream of data. This makes it possible to introduce the Ring Buffer in a pipeline.

The simulated response (*testbench*) of the Ring Buffer is shown in Fig. 5.14. A series of 16 data with is provided to the `data_in` port. The left marker highlights the start of the input sequence.

In this example the `bco` value is used as a counter in order to distinguish the data, and it is incremented at each clock cycle. Each data is processed by the Ring Buffer and provided at the `data_out` port after one clock cycle, assuming that no hold signal is received. The right marker highlights the end of the output sequence, one clock after the last input signal has been received.

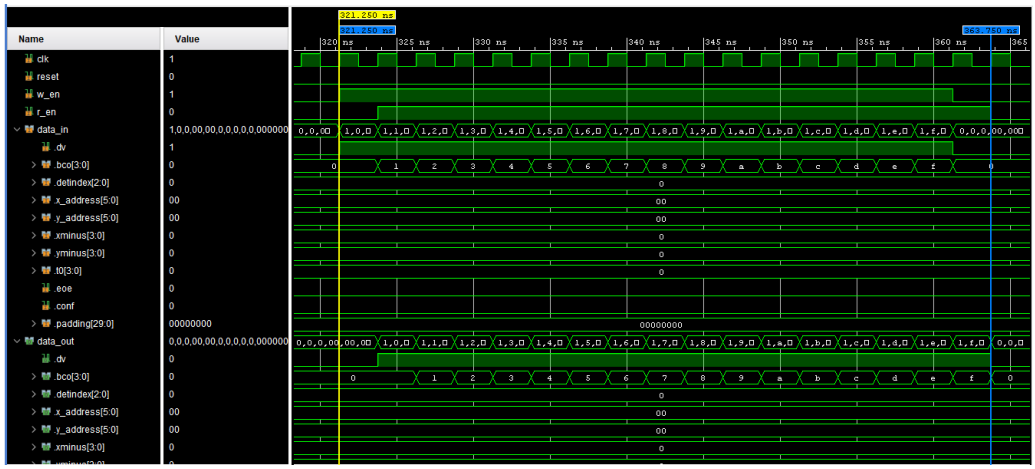

Figure 5.14: Testbench of the Ring Buffer. A sequence of 16 data is provided to the data input port. Each data appears to the data output port after one clock cycle.

N-way Dispatcher latency and throughput The N-way Dispatcher is composed of a Ring Buffer that works synchronously with the input reference clock and some additional logic to select the output port(s) to which the input data has to be delivered; in particular the additional logic is asynchronous and its input is represented by the internal Ring Buffer's. In this scheme the N-way Dispatcher can be considered as a synchronous circuit even if it is a mix of synchronous and asynchronous logic; in fact if we assume a data accessing an empty N-way Dispatcher[6] the data is stored into the Ring Buffer and will be available at its output during the next clock cycle; then the asynchronous logic of the N-way Dispatcher acts as demultiplexer forwarding the data to one ore multiple output ports. If the Ring Buffer was not empty, the data at the input of the N-way Dispatcher

[6] an N-way Dispatcher whose internal Ring Buffer is empty

is forwarded to the buffer, while one data from the buffer is processed and forwarded.
It is important to note that the asynchronous logic introduces an additional delay between when the Ring Buffer data is available and when a stable output is achieved. This delay is not taken in account in the evaluation of the minimum latency since it is typically lower than a clock cycle period if the clock frequency is below a reasonable value of 400 MHz. The maximum achievable clock frequency depends, in practice, on the particular implementation of the module in hardware performed by the synthesizing tool, Xilinx Vivado in our case. Moreover, the output of an N-way Dispatcher is typically connected to some synchronous logic, as the Ring Buffer instantiated in another N-way Dispatcher, or an N-way Merger, that processes the data at the rising edge of the reference clock; this justifies how a small delay (with respect to the clock period) can be neglected in the evaluation.
With the previous assumptions, the minimum latency of the N-way Dispatcher is equivalent to the Ring Buffer value. The maximum throughput is equivalent to the maximum input data rate that is defined by 1 data per clock cycle.
The effective latency depends on the status of the `hold_out` port signals, in particular on the signals associated with the selected outputs. This means that the N-way Dispatcher can work without introducing pauses even if some output ports are receiving a hold signal while the module is sending data to other available outputs.
The simulated response of the N-way Dispatcher is shown in Fig. 5.15. In this example an N-way Dispatcher with `x_outsize`=`y_outsize`=2 has been considered, for a total of $2 * 2 = 4$ output ports. The dimension of the `x_address_size` and `y_address_size` has been set to four, with `level` = 3. This means the destination output of the data is given by the most significant bit of the data addresses, *i.e.* data with `x_address` in range $[8 * i, 8 * (i + 1) - 1]$ and `y_address` in range $[8 * j, 8 * (j + 1) - 1]$ are forwarded to the (i,j)-th output of the N-way Dispatcher, and one or multiple neighbour outputs, with one bit overlap.
A series of $16*16 = 256$ data is provided at the `data_in` port. In particular the `y_address` value is incremented at each clock cycle in the range [0,15] while the `x_address` value is incremented every 16 clock cycles, in the range [0,15].
In the top picture the testbench is shown in the case that no hold signal is provided from the `hold_out` ports. Only the `dv` values of the output data are shown for a better visualization. It can be seen that each data appears at the proper output port(s) after one clock cycle. In particular some data reach multiple output ports, *i.e.* data with `x_address`=7, `x_address`=8 are forwarded the (i,j)-th output ports with both i=0 and i=1. The same consideration is valid for the output evaluation based on the `y_address` value.
The left marker highlights the start of the input data sequence, while the right marker highlights the end of the output data sequence, after 256+1 clock cycles, where 256 is the length of the sequence, while 1 clock cycle is given by the N-way Dispatcher minimum latency discussed before.
In the bottom picture the testbench is shown in the case in which the hold signal is provided to each of the `hold_out` ports with 25% probability. In this testbench the same sequence of data is provided, but the data is accepted only if there is no hold signal at

the `hold_in` port. The markers are placed in the same position of the previous case.

It can be seen, by comparison with the top picture, that the data are not always output. This is due to the presence of the hold signals and causes the filling of the Ring Buffer. When the buffer is full, the `hold_in` signal is raised and the input data is not accepted, even if still present and valid at the input port. In fact it can be seen that about one half of the data has been processed at the time highlighted by the right marker, but no data is lost during the process, thanks to the hold logic.

Figure 5.15: Testbench of the N-Way Dispatcher. In the top picture a sequence of 256 data is provided to the *data_in* port and no hold is provided to the *hold_out* ports. Each data appears at the output port(s) after one clock cycle delay. In the bottom picture the same sequence of data is provide while the hold signals at the *hold_out* ports is randomly generated with 25% probability. The data are not output when the *hold_out* are present, causing the filling of the internal Ring Buffer and consequent raising of the *hold_in* signal, that prevents the N-way Dispatcher accepting the input data, that are not incremented until the *hold_in* signal is released.

N-way Merger latency and throughput The N-way Merger is composed of multiple Ring Buffers, in particular one for each input port, and the asynchronous logic that selects the Ring Buffer from which the data has to be read and delivered to the only output. As in the previous case the N-way Merger can be considered as a synchronous circuit, from a global point of view, if the asynchronous logic introduces a negligible delay compared to the reference clock period. As we stated before, this module can be considered as a multiplexer; since only one output port is present, it means that only one data can be read and forwarded from one of the Ring Buffers. In terms of throughput this means that the maximum achievable value is 1 data per clock cycle; the maximum input data rate, instead, is given by N data per clock cycles, where N identifies the total number of inputs. This means, in terms of latency, that even if no hold signal is present at the `hold_out` port, the minimum latency is guaranteed only for one of the N (or less) data accessing the N-way Merger. This is valid for data that do not represent an *End Of Event* signal, in which case the stream of data from one input is paused until all the other *End Of Event* signals are received from the other ports. In fact the N-way Merger is the first module in which a hold signal on `hold_in` can be generated, even if no `hold_out` signal is received
In the following we identify a couple of example cases to show how a hold signal can be generated:

- considering to start from an empty N-way Merger (all buffers empty), if the input rate is equal to or less than 1 data per clock cycles, at most one data will be available in one of the buffers; in this situation the only available data from the Ring Buffers will be selected and forwarded to the output without increasing the occupancy of the buffers, resulting in no one filling up nor providing the hold signals on `hold_in` port;

- if the input rate is more than 1 data per clock, that is higher than the maximum throughput, only 1 data can be forwarded while the others will be stored in the corresponding buffers. If this situation is repeated one or multiple Ring Buffers will eventually get full and not able to receive any more data and the hold signal will be generated and raised on the `hold_in` port. In this case, if all the buffers are full and the output port not receiving any hold, a continuous stream of data is guaranteed with the maximum data throughput, while the internal generation of the `hold_in` signals allows a levelling of the total input data rate down the maximum data throughput;

- the last case is represented by the back propagation of the `hold_out` signal to one or multiple inputs. This results in the output data rate to be zero, the Ring Buffers independently receive the data from the inputs until they completely fills and then they provide the hold signal on `hold_in`.

While the first and last case can occur even with other described modules, like the N-way Dispatcher, the second example case is typical of the N-way Merger. For this reason when an N-way Merger is used in a network of different modules, as could be the case of the Switch, it has to be guaranteed that the input data rate is, on average, less than the

maximum data throughput, or output data rate, while fluctuations around the average are absorbed by the buffers.

Higher average input data rates will eventually produce a bottleneck in the data flow that would end in the back propagation of the hold signal through all the network until the very first input ports.

The simulated response of the N-way Merger is shown in Fig. 5.16. In particular an N-way Merger with $2 * 2 = 4$ inputs has been simulated. The left marker highlights the start of the input data sequence, while the middle and right markers highlights the start and end of the output data sequence. For the input data port, only the `dv` and `counter` are shown, for sake of better visualization. The `counter` has been defined as an alias and is not present in the stub data record definition. For the output data port, the `dv`, `x_address`, `y_address` and the `counter` values are shown; in particular the `x_address` and `y_address` are evaluated by the N-way Merger and represent the indexes of the input port from which the data has been received and forwarded.

In this example the depth of the Ring Buffer is equal to two and a sequence of four data has been provided at each data input port, while no hold signal is present at the output port. Starting from the left marker, we can see that the first four data marked with *bco* $= 0$ are accepted. During the next clock cycle one of these data is output and other four data, with *bco* $= 1$ are accepted; at this point some of the buffers completely filled and will raise the corresponding `hold_in` signals during the following clock cycle and some data at the inputs will not be accepted by the N-way Merger. Without hold signal from the `hold_out` port, the output data sequence is represented by a continuous stream of data; in particular it is important to note that the output data are ordered with respect to the *bco* value, that means that data that first entered the N-way Merger are output first.

This example highlights how the hold signal is internally generated in the N-way Merger, even if there is no hold signal back propagated from the following module to the `hold_out` port. It can also be seen that the first data is provided at the output after one clock cycle latency.

NxN Sorter latency and throughput In the previous paragraph we provided the values of the minimum latency and maximum throughput for the N-way Merger and N-way Dispatcher, that are the basic modules of the NxN Sorter and the NxN Switch. We recall that an NxN Sorter is always composed of two layers of sub-modules: one layer of N-way Dispatchers receives the data from the inputs, one layer of N-way Mergers receives the data from the first layer and provides the data to the output ports of the NxN Sorter, as shown in Figs. 5.6 and 5.7.

Due to this structure, each incoming data has to pass through one N-way Dispatcher and one N-way Merger; the minimum latency can be evaluated as the sum of the minimum latencies of the sub-modules, then it is quantified in 2 clock cycles. The maximum data throughput is proportional to the number of output ports and it is equivalent to N data per clock cycles, where N is the number of output ports (non necessarily equal to the number of input ports), as shown in Fig. 5.7.

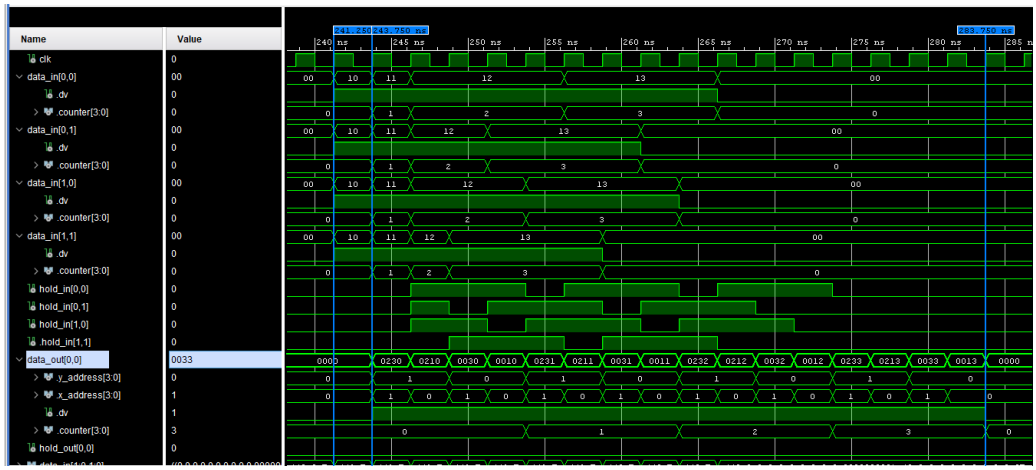

Figure 5.16: Testbench of the N-Way Merger. A sequence of four data is provided at each of the four inputs of the N-way Merger and no hold signal is present at the *hold_out* port. The left marker highlights the start of the input data sequence. The middle marker highlights the start of the output data sequences, represented by a continuous stream of data, that lasts for 16 clock cycles until the right marker. The minimum latency is measured in one clock cycle for the first data, while the other data are stored into the internal Ring Buffers of the N-way Merger until they are full. If a buffer is full the corresponding *hold_in* signal is raised and the *data_in* is not accepted.

NxN Switch latency and throughput According to its description, the NxN Switch is a network of NxN Sorters that form the so-called *square* Switch as shown in Fig. 5.8 and an additional layer of N-way Dispatchers if the number of outputs is higher than the number of inputs, or a layer of N-way Mergers in the opposite case, as shown in Fig. 5.10. As in the case of the NxN Sorter, the minimum latency is evaluated as the sum of the minimum latencies of the different layers, since every data has to pass through all of them flowing from any of the input, to any of the output ports. From the description of the Switch, we recall that a *square* Switch with N inputs and N outputs, made of Sorters with M inputs and M outputs is composed of $\log_M N$ layers, then its minimum latency is given by $2 * \log_M N$. The minimum latency for a non-square Switch is given by $2 * \log_M N + 1$, where N in this case is the minimum value between the number of inputs and number of outputs of the Switch.

Stub Maker latency and throughput In the previously described modules, like the Switch and its components, each evaluation depends only on the data that is being processed, and all the data are processed in a pipeline without any correlation between the previous or following data flowing through each module. In these cases all the evaluations can be performed in parallel and the data flow is self adjusted according to the data content and the status of the modules in the network, represented by the hold signals.
The Stub Maker is the first example where a pause in the data flow is inserted on purpose. In fact, the Stub Maker has to evaluate the combinatoric of two lists of hit data: first, the

hit data have to be stored in two separate buffers, then the combination are processed, filtered and provided through one or multiple ports.
In the case of the Stub Maker we can not define the latency for a single hit data and we will define the latency as the difference in time between the first hit data entering the Stub Maker, and the last processed stub data exiting the module. In particular we described the Stub Maker through a finite-state machine with three states *FSM_START*, *FSM_WRITING*, *FSM_READING* and we will define the latency according to the time duration of these three states.

- The finite-state machine *FSM_START* state only lasts one clock cycle; during this state the Stub Maker does not accept any data, so we will not take in account its duration in the total latency evaluation

- The *FSM_WRITING* state duration depends on how the hit data are received from the data acquisition system. Assuming that these are provided as two continuous streams followed by the *End Of Event* signals, and defining $N_{hits,1}$ and $N_{hits,2}$ the number of hits from the first and second detector planes, then the minimum duration of the *FSM_WRITING* state is evaluated as the maximum between $(N_{hits,1}+1)$ and $(N_{hits,2}+1)$

- The *FSM_READING* state duration depends either on the number of hits to be combined and the number of parts in which the two buffers are divided, that corresponds to the dimensions of the output data matrix. In particular the first buffer is divided into the `x_outsize` parts, the second buffer is divided into `y_outsize` parts. The duration of the *FSM_READING* state is evaluated from the maximum number of combinations that have to be checked in each sub process associated with a particular output; two additional clock cycles are needed to check that all the subprocess have completed the combinatoric process and to generate and output the extra stub data flagged as an *End Of Event*. The duration can be written as $\text{floor}((N_{hits,1} + \text{x\_outsize} - 1)/\text{x\_outsize}) \times \text{floor}((N_{hits,2} + \text{y\_outsize} - 1)/\text{y\_outsize})$.

The total minimum latency is given by the sum of the latencies produced during the *FSM_WRITING* and *FSM_READING* states. In particular, excluding the case in which a Stub Maker does not receive any hit data, then the minimum value that we can obtain is given by the case in which $N_{hits,1} = 1$ and $N_{hits,2} = 1$, that leads to a total minimum latency of (2)+(1+2) = 5 .
In general the value depends on the variables described before, in particular the number of hits provided to each Stub Maker depends on the detector occupancy and on the number of dedicated Stub Maker modules for each couple of detectors, and the number of outputs declared in each Stub Maker.
The maximum throughput of the Stub Maker is evaluated in the special case in which all of the hits combinations, namely the candidate stubs, are properly filtered by the application of the geometrical and timing cuts, that means that no fake stubs are identified. In this case the maximum throughput is given by the ratio between the number of identified stubs and the total processing time, measured in units of clock periods.

The simulated response of the Stub Maker is shown in Fig. 5.17. In this example a stream of five and four hit data, followed by the *End Of Event* signals, has been provided to the first and second input of the Stub Maker, respectively. For purposes of better data visualization only part of the hit input data and stub output data is shown. In particular the `counter` is an incremental index to identify the hits; the same is valid for `counter_det0` and `counter_det1` that represent the indexes of the hits from the first and second input, respectively[7] .
The left marker highlights the start of the data input sequence: the finite-state machine is in the *FSM_WRITING* state, and the `hold_in` at each input port signal is low, until the *End Of Event* is received. When both the *End Of Event* signals have been received, the state changes to *FSM_READING* and the Stub Maker starts to output the data. In the top picture the testbench of a Stub Maker with one output port is shown: the state-machine stays in the *FSM_READING* state for $5*4+1 = 21$ clock cycles, necessary to evaluate all the combinations and generate the *End Of Event* signal. In the bottom picture the same input data are provided to a Stub Maker with $2 * 2 = 4$ outputs: in this case the state-machine stays in the *FSM_READING* for a reduced amount of time, $3*2+1 = 6+1$ clock cycles in this example, since the evaluation of the hit combinations is performed in parallel by four sub processes. When all the hit combinations have been evaluated the state-machine transitions to the *FSM_START* starts, resets all the internal signals and transitions again to the *FSM_WRITING* state; the right marker highlights the end of the process.

Engine latency and throughput The Engine is the second and last module where pauses in the data flow are inserted on purpose. Similarly to the case of the Stub Maker, the Engine has to accumulate data until an *End Of Event* stub data is received, then the candidate track, if any, is provided to the output. Even in this case we consider the minimum latency of one Engine as the difference in time between the first stub data entering the Engine and the track data being provided at the output of the engine. In particular the minimum number of stubs that an engine should receive in order to identify a candidate track is given by the `r_threshold` value, defined within the `algorithmtools` package. The value is typically set to 2, in order to identify tracks with at least 4 hits in the tracking device.
In general the minimum latency of the Engine can be evaluated from the number of stub data that the Engine receives, that we will call N_{stubs}. For each received stub data, the Engine takes one clock cycle to sum the stub parameters to the parameters of the previously received stubs; conversely when the *End Of Event* signal is received there is no need to evaluate any sum and the `data_out` signal is immediately updated with the value of the candidate track. Then the total minimum latency is simply given by N_{stubs}, provided the simple assumption of a continuous stream of stub data followed by the *End Of Event* signal. Without considering any special case the latency is given, instead, by the difference in time between the first and last received data.
The throughput of the Engine is given by the ratio between the number of identified tracks and the processing time. This depends on many factors since the Engine is basically the

[7] The *counter*, *counter_det0*, *counter_det1* are not defined in the hit and stub data record definition.

Figure 5.17: Testbench of the Stub Maker. In this example five and four hit data are provided to the Stub Maker inputs, respectively. In the top picture a Stub Maker with one outputs has been tested, while in the bottom picture a Stub Maker with four outputs has been considered. Starting from the left marker the Stub Maker receives the hit data from the inputs until both the *End Of Event* signals have been received, then it starts evaluating the hit combinations and provides the stubs in output through one or four ports. When all the stubs have been output, the Stub Makers reset all the internal signals, releases the *hold_in* signals and is able to process new data.

last processing unit of all the algorithm. Similarly to the Stub Maker case, we could define the maximum throughput of the Engine, in units of tracks over clock cycles, as the inverse ratio of its minimum latency. In general this is an over optimistic value, since according to the number of tracks per event and in particular to the number of identified stubs, and the number of Engines instantiated to cover the space of the track parameters, it is expected to have a certain number of Engines that don't identify any track for a specific event. In particular for a well reconstructed event we expect this value to be

$1 - N_{tracks}/N_{Engines}$.
The simulated response of the Engine is shown in Fig. 5.18. The left marker highlights the start of the input data sequence, represented by a stream of six stub data followed by the *End Of Event* signal, while no hold signal is received by the Engine. In this case, for each received stub data the Engine updates the internal values and never provides the `hold_in` signal. One clock after the *End Of Event* signal is received, the Engine outputs the track data, if a candidate track has been identified.

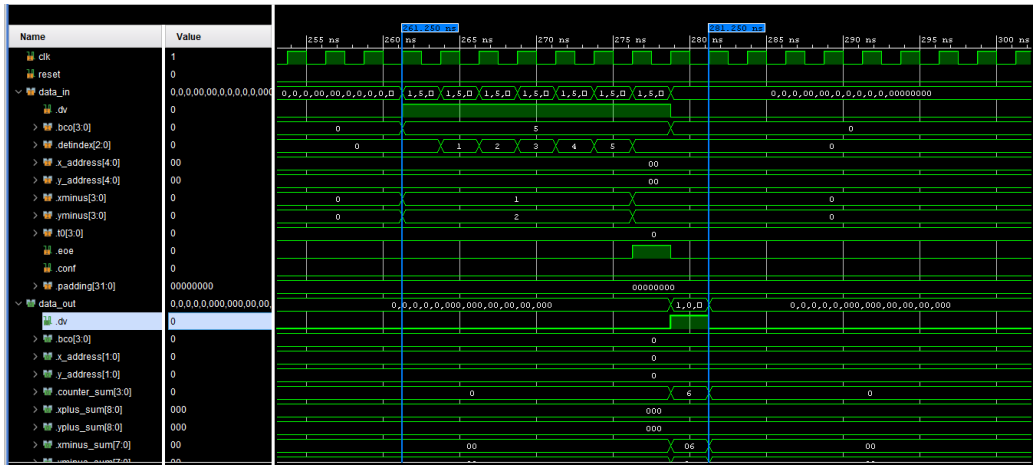

Figure 5.18: Testbench of the Engine. Six stub data are provided *data_in* port, followed by the *End Of Event* signal. The Engine outputs the track data one clock cycle after receiving the *End Of Event*. The left and right marker represent the start of the input sequence and the end of the processing, respectively.

5.3 Overview of the complete architecture

Here we give an overview of the architecture of the 4D real-time tracking algorithm implementation. All the major and basic modules have been described in the previous section and the complete architecture is based on the combination of the Stub Constructors, Stub Switches, Engine Regions and FanIns, that are connected together following a *vertical* approach.
The scheme of the full architecture of the tracking system is shown in Fig. 5.19.
We can identify the major modules that are instantiated in rows. In particular the Stub Constructors receive data from the detectors, the Switches receive data from the Stub Constructors and so on. In the beginning of the chapter we referred to the major blocks as single units, while we can note that the architecture is not *monolithic* and the processing is parallelized over multiple independent blocks; a clear example can be identified in the presence of multiple Stub Switches, each one connected only to a group of Engine Regions.
In the following the architecture scheme is briefly described. The detectors are grouped into couples and each couple is connected to a Stub Constructor; each Stub Constructor

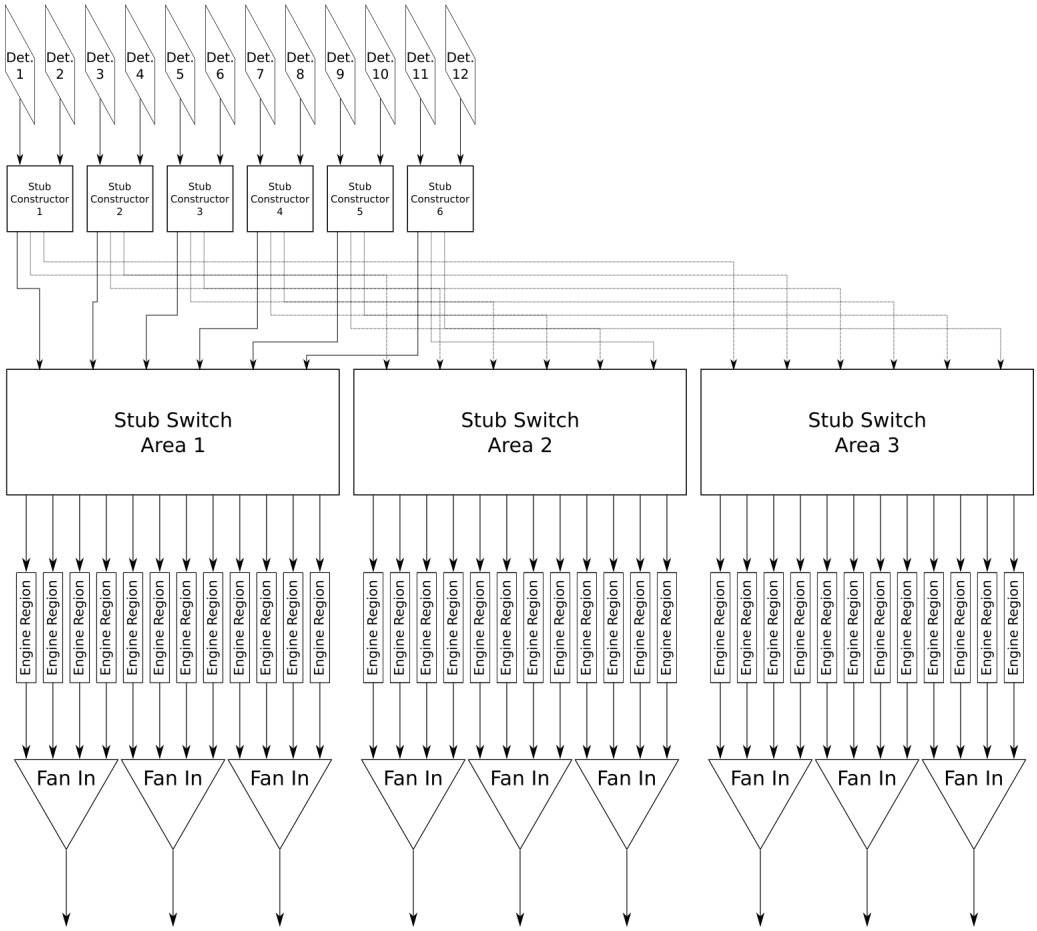

Figure 5.19: Architecture of the 4D real-time tracking system.

evaluates and provides the Stub data to multiple Stub Switches. In particular each Stub Switch receives Stubs with track parameters belonging to non-overlapping region of the space of the main track parameters, hence the Switches works completely in parallel without any kind of lateral communication. This allows to implement these modules into different FPGAs. Each Stub Switch distributes the Stub data to a set of Engine Regions, that host the Engines; even in this case no lateral communication is implemented between the Engine Regions, nor between the Engines within the region. The Engines process the Stub data and provide the candidate tracks as output. The Track data are then collected via multiple FanIns and provided to the next stages of processing.

Discussion on the architecture parallelization As evidenced in Fig. 5.19 the architecture is based on different interconnected modules that are always connected following the direction of the data (and hold logic) flow and the same is valid within all the submodules described in the previous section. According to this structure, modules that lie

on the same row are independent from each other and can operate in parallel resulting in a high level of parallelism of the complete algorithm architecture.
It is worth noting that the possibility to parallelize the track reconstruction is given primarily by the fact that the charged particles production at colliders, as in the case of LHC, is invariant with respect to rotations around the beam axis. For this reason the LHC detectors ALICE, ATLAS, CMS are based on a cylindrical layout. Conversely LHCb does not have a cylindrical symmetry and, moreover, the particle trajectories after the magnet are not invariant under rotation around the beam axis; nevertheless it should be noted that before the LHCb magnet, in particular within the VELO the charged particle distribution shows the cylindrical production symmetry.
According to these considerations the problem of the track reconstruction can be approached dividing the space of the track parameters in non-overlapping ϕ-regions, where ϕ represents the azimuthal angle with respect to the beam axis, and running the algorithm for the different regions in parallel. Similarly, the space of the track parameters can be divided in non-overlapping regions with respect to other variables, even if there is no invariance in the track generation. Referring to the proposed architecture, this means that Engines belonging to different Engine Regions identify tracks with parameters belonging to different regions of the space of the track parameters.
So far we stated that the parallel nature of the algorithm reflects in the parallel structure of the architecture that allows for no lateral communication between the same level modules. Even though there are no *horizontal* connections we should observe that it is not true that the behaviour of a logic module is not affected by the status of its neighbour modules, as one could naively think. In fact, this is true only in situations in which the hold logic signal is not back-propagating. In the opposite case the presence of a hold signal affects the data distribution by introducing a pause in the data flow: a simple example could be the case in which a hold signal is received at one output of a FanOut[8] and the data coming at the input are buffered instead of being forwarded to the N outputs. In this case, even if the hold signal comes from only one port, none of the modules connected to the FanOut will receive any data until the hold signal is released. On the other hand it is always true that no data can be transferred between same level modules.

[8] refer to the N-way Dispatcher module description

Chapter 6

Results on a prototype device

The algorithm described in the previous chapters has been implemented in FPGA and tested both in simulation and running on hardware. In particular the system has been implemented on a Xilinx Virtex UltraScale FPGA mounted on a gFEX prototype v2 board. In this chapter we will describe the prototype board together with its main characteristics, then we will provide the obtained implementation results.

6.1 gFEX prototype v2 board

The Global Feature Extractor (gFEX) is part of the Level-1 online trigger system of the Phase-I Upgrade of the ATLAS experiment, designed to help maintain the ATLAS Level-1 trigger acceptance rate at increased LHC luminosity. The gFEX is designed to extract information from the entire calorimeter and to identify patterns of energy associated with hadronic decays of high momentum Higgs, W and Z bosons, top quarks, and exotic particles in real time at the 40 MHz LHC bunch crossing rate. The role of the gFEX is described in Ref. [34] and it is not part of this thesis.
The design of the gFEX is based on a custom ATCA board equipped with multiple FPGAs. The board has reached the fourth generation, that represents the final design for production.
In this section we will describe the features of the gFEX prototype v2 board and in particular the features of the one used for the implementation and test of the proposed 4D real-time tracking algorithm. In fact the board that has been purchased at INFN - Milano slightly differs from the original design in terms of number and model of the FPGAs, hence in the number of high-speed serial links for communication with other systems and FPGA-to-FPGA internal communication.
The fully assembled gFEX prototype v2 board and its floor plan are shown in Fig. 6.1.
The gFEX prototype v2 board (from now on, only *gFEX board*) is equipped with three Virtex UltraScale FPGAs (*XCVU095*) for data processing, we will refer to these as *Processor FPGA A, B, C*, and one Xilinx Zynq FPGA (*XC7Z045*) for control and monitoring. The four FPGAs communicate between each others via different groups of parallel data buses, implemented using GPIO (*general purpose input/output*) links running in DDR (*double data rate*) at 560 MHz clock rate for a total data rate of 1.12 Gbps for each data line. 96 DDR lines connect the Processor A with the Processor B, 72 DDR lines connect

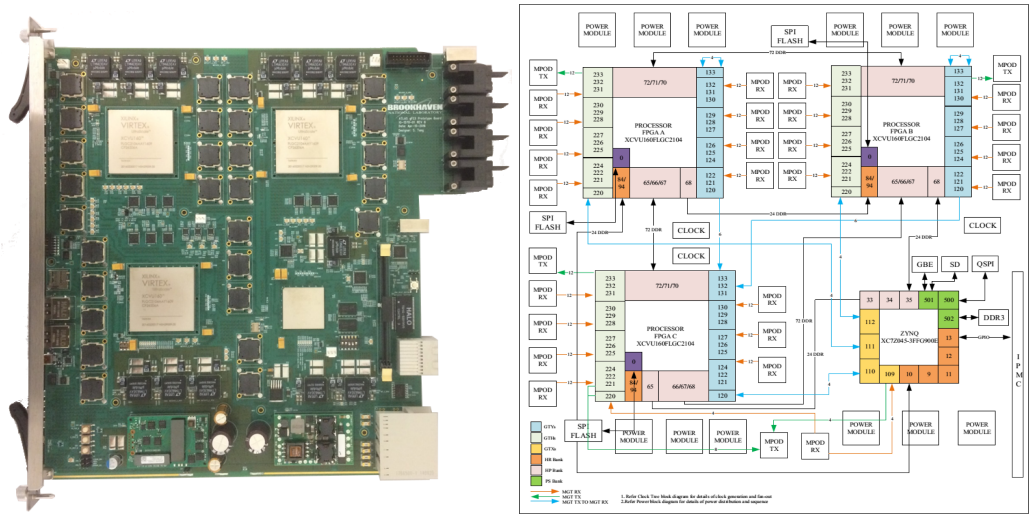

Figure 6.1: Picture (left) and floor plan (right) of the fully assembled gFEX prototype v2 board.

the Processor A and Processor C, 72 DDR lines connect the Processor B and Processor C. The Zynq FPGA has 23 DDR lines connected to each of the Processor A,B,C FPGAs. The parallel data buses can be operated both for transmitting or receiving data from/to the different chips.

Different kinds of MGTs (*Multi-Gigabit Transceiver*) are present in the design depending on the FPGA model: GTX on the Zynq FPGA and GTH, GTY on the Virtex UltraScale FPGAs. Most of the MGT links are connected to MiniPOD receivers and transmitters for communication via optical fibres and each link can run at 12.8 Gbps. In particular the Processor A and B have 96 RX (receiver) and 12 TX (transmitter) optical links, the Processor C has 92 RX and 20 TX optical links, while the Zynq FPGA has 4 RX and 4 TX links. A small group of MGT links is used for on-board communication between the four FPGAs and can run at 25.6 Gbps. The on board MGT links are distributed as follows: the Zynq FPGA communicates with 4 RX and 4 TX links with each of the Processor A, B, C FPGAs, while the Processor A and B communicates to the Processor C via 4 TX lines each.

It is worth noting that the gFEX board is highly unbalanced in terms of ratio between input and output high speed serial links connections. In fact, since the board is designed to be a part of the trigger system, it is expected to receive all the data from the detector and select only the useful events, with a significant data reduction.

The gFEX prototype v2 board has been fully characterized at Brookhaven National Laboratory (BNL). The board description together with the results of the characterization, as the BER (*Bit Error Ratio*) for the MGT links and the test of the communication over the parallel data buses and the power consumption can be found in Ref. [35].

gFEX board at INFN - Milano A custom version of the described gFEX prototype v2 board has been purchased by INFN - Milano by both the LHCb and ATLAS groups. This board slightly differs from the previously described design, in particular it hosts the same model of Xilinx Zynq FPGA (XC7Z045), while only two Processor FPGAs are present, namely the Processor A and B. Moreover the model of the Processor FPGAs is different, XCVU095 instead of XCVU160. A comparison table between the Xilinx Virtex UltraScale XCVU095 and XCVU160 models is shown in Fig. 6.2; the product tables of Zynq-7000 Family and Virtex UltraScale FPGAs can be found in Refs. [36], [37].

	XCVU095	XCVU160
System Logic Cells (K)	1,176	2,027
DSP Slices	768	1,560
Memory (Mb)	60.8	115.2
GTH 16.3 Gb/s Transceivers	32	52
GTY 30.5 Gb/s Transceivers	32	52
I/O PIns	832	702

Figure 6.2: Comparison summary of the Xilinx Virtex UltraScale XCVU095 and XCVU160 resources. The XCVU095 has, in general, less resources with respect to the second and most powerful model.

The XCVU095 features less resources for the logic implementation; moreover it has a reduced number of MGT transceivers and this affects mostly the internal communication capabilities between the Processor A,B (C is not present) and the Zynq FPGA.
A picture of the gFEX board available at INFN - Milano is shown in Fig. 6.3. By comparison with Fig. 6.1 it is easy to note the absence of the Processor C FPGA, together with all the MiniPODs originally connected to its MGT transceivers. It is also worth noting that that some MiniPODs (one for Processor A, one for Processor B) are missing since they where originally connected to MGT transceivers that are not present in the XCVU095 FPGA; for the same reason, some of the MiniPODs are not fully connected while the on board MGT connections for internal communication are missing. All the other features of the gFEX board are untouched, as for the on-board chip-to-chip communication via the parallel data buses implemented using DDR links.
In the following a summary of the input/output communication capabilities for the custom gFEX board available at INFN - Milano: 96 DDR lines between Processor A and Processor B, 23 DDR lines between Processor A and Zynq FPGA, 23 DDR lines between Processor B and Zynq FPGA, no on-board chip-to-chip communication via MGT links, 64 MGT RX lines and 4 TX lines connected to the optical fibres via the MiniPODs for Processor A, 64 MGT RX lines and 8 TX lines for Processor B, 4 MGT RX and 4 MGT TX lines for Zynq FPGA.
With this design, if we consider a simplified scheme in which we want to implement data communication between Processor A and Processor B, the maximum achievable data

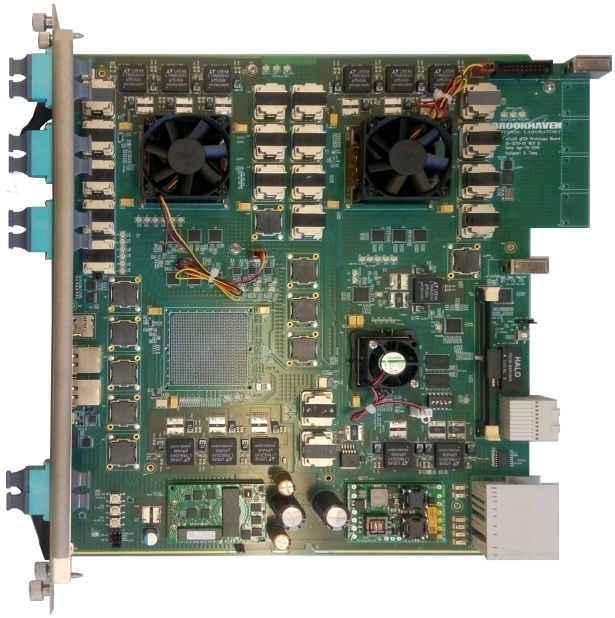

Figure 6.3: Picture of the gFEX v2 prototype board available at INFN - Milano. This board is equipped with only two Xilinx Virtex UltraScale FPGAs (*XCVU095*) and one Xilinx Zynq FPGA (*XC7Z045*).

transfer via serial links is obtained connecting the 8 MGT TX lines of the Processor B to 8 MGT RX lines of the Processor A, obtaining a total of 102.4 Gbps unidirectional data transfer rate. This value is similar to the maximum unidirectional data transfer achievable via parallel data bus, equal to 107.52 Gbps, considering 96 DDR lines running at 560 MHz; the latter value could be different since the DDR links can be used in RX or TX mode, then many configurations are possible, *i.e.* half of the DDR lines set for communication from Processor A to B and half of DDR lines used in the opposite direction. Moreover we should consider that part of the DDR lines are dedicated clock lines used for data synchronization. Regarding the MGT links the maximum achievable data transfer rate depends on the communication protocol, *i.e.* if the *8b10b encoding* is used, each MGT lines can transfer data at the maximum rate of 10.24 Gpbs $= 8/10 * 12.8$ Gpbs, since part of the transferred data are not available to the user, being used to perform the data encoding and decoding.

6.2 Test of optical MGT links

Before implementing any kind of protocol for chip-to-chip and external communication via MGT links, the good quality of the links has been proved performing a Bit Error Ratio test using the IBERT (*Integrated Bit Error Ratio Tester*) LogiCore IP from Xilinx design suite. The tool allows to evaluate and monitor the quality of the links implemented using the GTX/GTH/GTY transceivers. In particular it is based on the generation of

specific data patterns that are transmitted by the TX transceiver to an RX transceiver that receives and compares the data to the expected pattern.
The setup of the gFEX board for the test of the MGT links is shown in Fig. 6.4. In

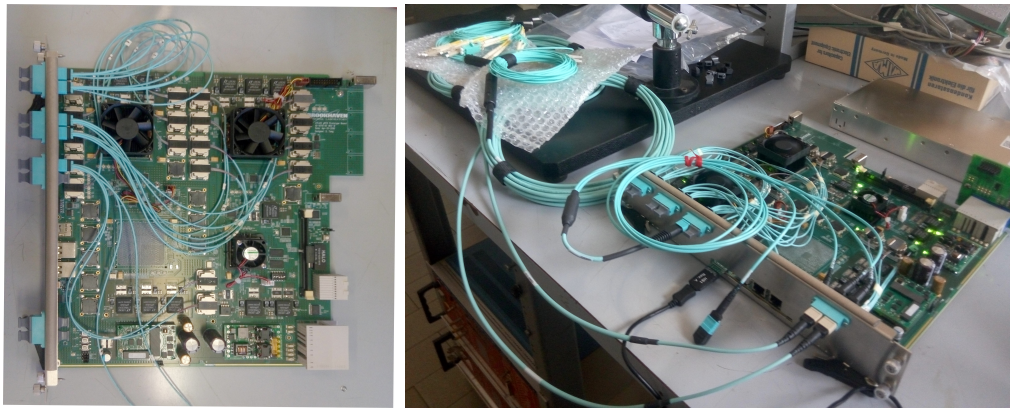

Figure 6.4: gFEX setup for the test of MGT links. (Left) All the MiniPODs are connected to the front panel using MTP-48 patch cables. (Right) Eight TX lines from the Processor B are connected to 8 RX lines of the same FPGA; the optical fibres can be accessed singularly by using different patch cables.

the left picture the connections from the MiniPOD connectors to the MTP-48 couplers, mounted on the front panel (left) are highlighted, in particular each MTP-48 coupler hosts 48 optical fibres connected to 4 MiniPODs, each one having 12 optical fibres; 2 MTP-48 patch cables are connected to the Processor A, 2 are connected to the Processor B and 1 is connected to the Zynq FPGA. Moreover we remind that the MTP-48 connectors are not fully populated. In the right picture a particular setup is shown: the gFEX board is powered using a *Vadatech VT000-1000* ATCA power supply [38] (right part of the picture) and only some links from/to the Processor B are connected resulting in the connection of the 8 TX lines to 8 RX lines of the same FPGA. In particular *MTP-48 to MTP-12* patch cables are used to split the fibres into groups of MTP-12 connectors and *MTP-12 to LC* patch cables are used to access and connect the fibres one by one. Despite this scheme of connections seems tricky it allows to save space on the front panel by use of MTP-48 connectors while allowing to test the fibres singularly with different configuration, while the position of the fibres and how they are associated with the MGT links in the firmware implementation has been mapped to simplify the access to the different groups of MGT links for debugging purposes.

Link speed test with IBERT The BER test has been performed to confirm the quality of the links. Using the IBERT tool it is possible to produce the eye diagram and obtain the BER value for the link under test. The link speed has been set to 12.8 Gbps and different configurations have been tested, in particular the connections between the Processor A and B (and vice versa) and the connection between different MGT links on the same FPGA; in fact since the Processor A (B) has only 4 GTH (8 GTY) TX lines it was not possible to test all the RX lines at the same time.

The test reproduced the results from Ref. [35], by giving no errors in the IBERT test. An example of the eye diagram obtained using the IBERT tool is shown in Fig. 6.5. In this example the link under test connects an MGT TX transceiver to an RX transceiver on a different group of MGT links in the same FPGA. The total length of the fibre is about 20 meters.

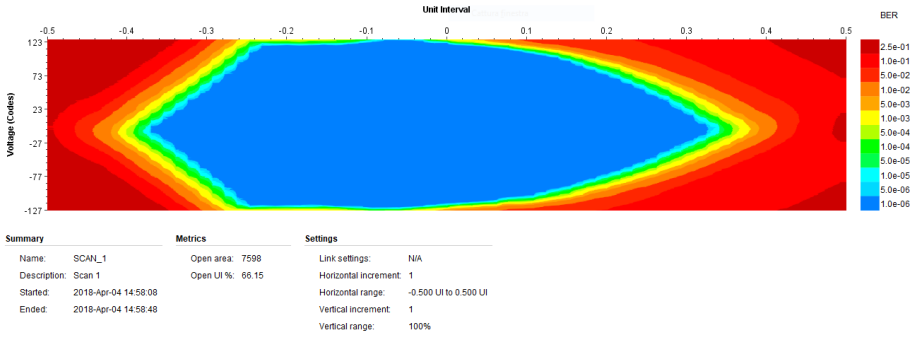

Figure 6.5: Example of eye diagram obtained using the Xilinx IBERT LogiCORE IP. In this test an MGT transceiver is connected to a second MGT transceiver on the same FPGA.

6.3 Implementation of user data communication over MGT links

Starting from the Xilinx *UltraScale FPGAs Transceivers Wizard* LogiCORE IP example design, a simple communication protocol based on *8b10b encoding* has been implemented and first tested using pseudorandom binary sequence (PRBS) data, transmitted over an optical fibre and received on the other end and compared to the expected pattern by a simple data checker.

The TX 8b10b encoder and the RX 8b10b decoder are built-in features of the GTH and GTY LogiCORE IPs. The *8b10b encoding* is an industry standard encoding in which for each byte (8 bits) two additional bits are generated resulting in a total of 10 bits transmitted for each 8 bits provided by the user logic; similarly on the receiver side 10 bits are received and an 8 bit word is decoded and then provided to the user logic. In particular the additional bits are generated in such a way to achieve DC-balance and bounded disparity, the RX decoder checks for errors in the data by comparison of the extra bits to the expected bits evaluated from the received data. In general, different status flags are provided by the RX decoder and when the *8b10b encoding* is enabled some of them are used to address errors in the received data: in the implemented design the *running disparity error* is monitored together with a flag that indicates if the received byte is a valid character with respect to the 8b10b encoding table; if any of this error is raised the data is discarded. More information on the Xilinx *UltraScale FPGAs Transceivers Wizard* LogiCORE IP can be found in Ref. [39].

The design has been customized to transmit 32-bits user data at 320 MHz reference clock

for an effective link speed of 10.24 Gbps per each GTH/GTY line. This is in agreement with the previously declared value of 12.8 Gbps that refers to the total data rate obtained from the link characterization with the IBERT tool; the maximum value could be achieved by disabling the *8b10b encoding*.
In the design implementation, the transceivers are instantiated in groups of four lines, that share the same reference clock and are part of the same FPGA bank; each group, including both four TX and four RX lines is referred to as *Quad transceiver*. A wrapper has been designed to include the GTH or the GTY Quad transceivers and allows to instantiate up to eight modules for a maximum of 32 lines of the same kind. The GTH and GTY transceivers are customized in the same way and are interchangeable within the wrapper. All the lines can be accessed singularly and the wrapper includes a dual clock *First Word Fall Through* FIFO for each line. In particular the choice of using dual clock FIFO is motivated by the necessity to write data that are generated by the user logic at a reference clock; the same is valid for data received from transceiver lines at the reference clock of 320 MHz and have to be provided to the user logic, *i.e.* the tracking algorithm, running at different clock speed.

Test with PRBS data A PRBS (*pseudorandom binary sequence*) is a sequence of 0 and 1 bits generated with a deterministic algorithm and represents a typical test pattern for evaluating the quality of serial links. In fact the IBERT LogiCORE IP is itself based on the generation and reconstruction of PRBS data. The Xilinx *UltraScale FPGAs Transceivers Wizard* example design includes the logic for the generation of PRBS data on the transmitter side and the checking of the received data and these features have been kept in the customized design, in particular the data generated has been moved before the FIFO on the TX side and after the FIFO in the RX side: this mean that PRBS data are generated, provided to the FIFO, then transmitted over the optical fibre; on the other side the received data pass through the FIFO and are checked to be consistent with the expected pattern.
The test with PRBS data has been performed for groups of four (eight) optical fibres that represents the maximum number of available TX lines on Processor A (B).
The test consists in the following steps:

- a special sequence of data is sent through the link in order to align the transmitter and the receiver, in fact since the data are serialized these are received as a continuous stream of bits and a reference has to be provided to identify the first and last bit of the transmitted word(s). The RX transceiver module includes the logic to identify special characters within the received data and aligns itself in order to provide the correct output. This process has to be performed even when using the transceivers with user provided data;

- the RX transceiver evaluates and provides a signal that indicates the status of the link, in particular if the received data are correctly aligned, the status signal is provided to a Virtual Input/Output LogiCORE IP (VIO) that can be monitored on a PC within the Xilinx Vivado Hardware Manager software;

- when the link is up and correctly running the generation of PRBS data is started manually. The receiver logic checks if the received data are consistent with a PRBS sequence and latches a status signal if the result is negative at least once: the status signal is monitored through the VIO and stays high until manually reset.

The communication has been tested both connecting TX and RX within the same FPGA and connecting different FPGAs, proving no errors in the received data.

Test with user generated data The communication over optical fibres has been tested also with user generated data. A data generator has been designed to provide a known sequence of data to the TX transceivers. The TX transceivers have been connected to the RX ones, where a data checker module compares the received data to the known pattern. The data checker provides a status flag that is read using a VIO LogiCore IP on a PC: if the pattern comparison fails at least once, the status flag indicates an error and the value stays unchanged until the sequence is reset from the VIO. Also in this case the communication has been tested and validated both connecting TX and RX within the same FPGA and from one FPGA to the second FPGA of the gFEX board.

6.4 Implementation and test of the Stub Switch and Engines

In this section we describe the test performed on a simplified implementation of the algorithm comprising the Stub Switch and Engines. The Stub Constructor is not fully implemented but it is based on the Hit Switch, that follows the same behaviour of the Stub Switch, and on the Stub Makers, whose performances in terms of latency and throughput have been demonstrated.
The implemented system consists of a 64×64 Stub Switch and 16 Engine Regions, for a total number of 1024 Engines. The engines distribution in the space of the main track parameters has been optimized in order to achieve uniform occupancy of the Engine Regions, and of the Engines within the region. To do this we decided to perform the pattern recognition, hence the distribution of the Engines, in the space of the track parameters (r_+, ϕ_+), the radial and azimuthal coordinates of the track intersection at the reference plane, instead of (x_+, y_+). This space has been divided in 8×8 sectors according to the distribution of the track parameters obtained by the simulation; in particular the distribution of the tracks with respect to the ϕ_+ is flat as expected due to the cylindrical symmetry of the particle production in pp collisions. The distribution with respect to r_+ depends on multiple factors, as the polar distribution of the tracks and the vertex distribution in pp collisions; this leads to a non uniform occupancy of the reference plane at which the pattern recognition is performed. Each r_+ sector covers a different range in the r_+ sub-space in such a way that the distribution of the Stubs data to the Engines is balanced and as uniform as possible. It is worth noting that the distribution of the Engines over a non uniform grid in the reference plane can be also modelled with a change of coordinates making the occupancy uniform over the normalized space.

Resources occupancy It is estimated from the simulation of one sector that 1024 Engines are needed to keep high efficiency and purity, in order to reproduce the results from the high level simulation of the algorithm, shown in Section 4.2. The number of inputs/outputs of the Stub Switch has been chosen to be equal to the number of RX transceivers (64) available on the Xilinx XCVU095 FPGAs (Processor A and Processor B) on the gFEX board. The number of Engines is limited by the resources needed for their implementation, in particular the implementation of the system with the Stub Switch, the pool of Engines, and the FanIns for data collection, takes the 60% of the FPGA resources, while the rest is kept for including the Stub Constructor as soon as it will be fully implemented.
From this point we focus on the details of the described sub-system. In this scheme the 4D real-time tracking algorithm can be implemented using a total of 64 sectors with 1024 per each, implemented in different FPGAs. Recalling the design of the gFEX board, this means that one board can be used to process 1/32 of the full tracking system. Moreover, in the system there is space for technical optimizations.

Test architecture The proposed test of the system has been performed with a standalone architecture fully implemented in a single FPGA (Processor A/B). In fact the input/output communication has already been tested and proved to work.
The architecture for the test of the Stub Switch and Engines is shown in Fig. 6.6.
The test architecture is composed of the Data Generator, the Device Under Test, the Data Checker; all the system works at a reference clock of 320 MHz.
The Data Generator comprises a read-only memory (ROM) in which the Stub Data evaluated from simulation are stored. At each clock cycle the ROM is read out providing one data per input to the Stub Switch, if the Switch itself is not back propagating any of the hold signals (in Fig 6.6 the *Hold* signal is highlighted with only a single line for a better view). If at least one hold signal is active, the ROM is not read, in order not to lose any data, and the address of the value to be read is not incremented. The Hold Checker, together with additional internal logic of the Data Generator, controls the Input data ROM and monitors the number of received hold signals, compared to the number of times in which the Detector Under Test was able to receive data. Two counters, *hold counter* and *tot counter*, are associated with these values and can be read out from a PC through a VIO LogiCore IP.
The Device Under Test features the Stub Switch, Engines and FanIns.
The Data Checker comprises a ROM in which the expected Track Data evaluated from simulation are stored and a Data Comparator that reads data from both the ROM and the Device Under Test; every time the Data Comparator receives a valid Track Data from the Engines, the data is compared to the value stored in the ROM. The internal logic of the Data Checker controls the readout of the ROM and provides a status flag that indicates if the comparison provided negative result. The status flag is read out through a VIO LogiCore IP.

Test results Using the previously described test architecture we obtained the following results.

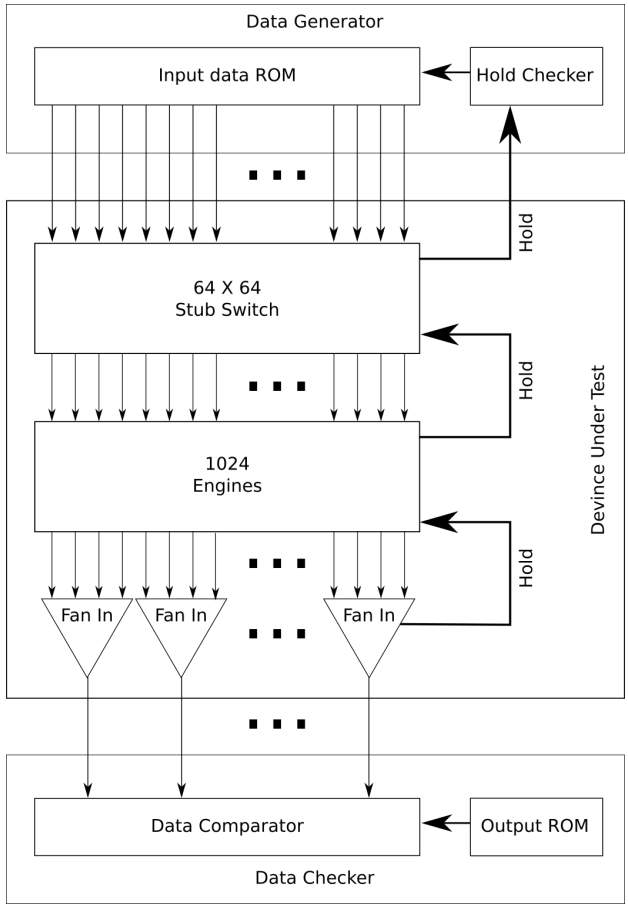

Figure 6.6: Architecture test for the 4D real-time tracking system, based on the Stub Switch and Engines. Input data are read from a ROM and fed to the Device Under Test, output data are compared to the expected value written in a second ROM.

First it has been checked if the evaluations performed in the hardware implemented algorithm reproduce the expected results. The Stub Switch has been tested in a standalone test and then the Device Under Test described in the previous paragraph has been tested, including the Engines and the FanIns. This kind of test has been performed first with single track events, for debugging purposes, in order to have a reduced data flux within the Stub Switch and exclude possible effects related to an incorrect hold logic management within the Switch; this test provided a positive result with exact match between the output data processed by the Device Under Test and the expected values, stored in the Data Checker with the system running at a reference clock of 320 MHz. The same test has been reproduced simulating events with full events in 1/64 of the tracking system, in particular in the central region of the space of the track parameters (r_+, ϕ_+). Considering 64 bits for each Stub Data, and considering that the Data Generator has to feed data to 64 Stub Switch inputs, a 4 MB Block Ram has been used to implement the Input Data

ROM with a depth equal to 1024. In particular the ROM contains the data associated with the already identified and filtered Stubs that point to the region of the space of parameters covered by the Engines; in these conditions, the ROM was loaded with data associated with 398 simulated events and the hardware implemented algorithm provided the expected output values.
The second result obtained is the estimation of the maximum throughput of the system. This measurement, even if limited by the low statistics due to the ROM size, is achieved as follows. The reference clock of the test architecture is set at a value of 320 MHz. The Data Generator provides the Stub data associated with 398 events, stored in a ROM with a depth of 1024 that is continuously read (in a loop) if no hold signal comes from the Device Under Test, this means that the Data Generator is able to provide a maximum event rate of $398/1024 \times 320$ MHz $= 124.375$ MHz; the Hold Checker manages the readout of the Input Data ROM and starts to increment the *hold counter* and the *tot counter* as soon as the first hold signal is received from the Stub Switch: the *tot counter* is incremented at each clock cycle, while the *hold counter* is incremented when the Stub Switch is providing at least one hold signal, that prevents the ROM to be read out. The ratio between the *hold counter* and the *tot counter* is 0.67 and together with the maximum event rate evaluated before provides an estimation of the maximum throughput of the Device Under Test that is evaluated as: throughput $=$ Max.evt.rate $\times (1 - hold\_counter/tot\_counter) = 40.9$ MHz. It is worth noting that this result depends on the reference clock of the system, that if increased to 400 MHz would push the maximum throughput up to 51.1 MHz.

Chapter 7

Possible application of real-time tracking algorithm to the Beam Gas Vertex Detector

In this chapter we will provide a description of the Beam Gas Vertex Detector, together with the results of a study on the feasibility of the application of a real-time tracking algorithm for event reconstruction, implemented in FPGA and based on an architecture similar to the one described in the previous chapters.

7.1 Beam Gas Vertex Detector

The Beam Gas Vertex Detector (BGV) [40] is a beam monitor system that is being developed as part of the High Luminosity LHC project. The measurement of the beam profile is obtained from the vertex reconstruction of particle tracks produced in proton-gas interactions. The BGV aims at providing a non-invasive measurement of the transverse beam size with less than 5% error within an integration time of 5 minutes. A prototype demonstrator has been commissioned and installed at LHC Point 4, on the Beam 2 ring in 2016. The detector consists of three main parts, as shown in Fig. 7.1 (from left to right): the Gas Target, the Tracking Detector composed of four tracking station based on SciFi detectors, the Hardware Trigger system composed of three scintillator stations.

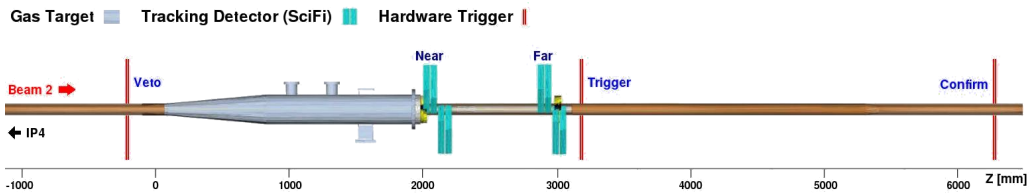

Figure 7.1: Layout of the Beam Gas Vertex Detector. From left to right: Gas Target, Tracking Detector, Hardware Trigger system.

The Gas Target system is a 2-meters aluminium vacuum tank designed to minimize the impact on the LHC beam. It is composed of a conical tube followed by a cylindrical section. Protons interact with neon gas at low pressure and multiple tracks are produced in the forward direction. Different optimizations include a reduced thickness of the tank exit window (down to 1.15 mm) to minimize the particles' multiple scattering, and a

reduced size of the beam pipe diameter at the end of the conical section and near the tracking stations; the beam pipe reduction allows to minimize the distance of the sensors from the interaction region and acts as a gas flow restriction allowing to reach $\approx 1 \times 10^{-7}$ mbar gas pressure in the tank, while keeping the pressure in the adjacent beam pipes at nominal LHC vacuum value (1×10^{-11} mbar).
The tracking detector is composed of two stations (*near* and *far* stations). The near station is placed just after the exit window of the Gas Target tank, the second station is placed at about 1 meter distance. Each station includes two pairs of SciFi sensor modules placed above and under the beam pipe. Each pair of modules is arranged to provide the 3-dimensional measurement of the track hits, with the first orthogonal to the second one. Each SciFi module contains two sensitive sides with 4 (*near* station) or 5 (*far* station) layers of scintillating fibres of 250 μm diameter. The readout of the scintillating fibres is performed using silicon photomultipliers for a total of 1024 channels for each module side, using four Beetle ASICs.
The Trigger system consists in three stations of 300×300 mm^2, 1 cm thick, plastic scintillators above and below the beam pipe: the first station, located before the Gas Target system is used to veto interactions occurring before the gas tank, the second and third station are located after the tracking system and provide the trigger based on the coincidence signal from the two planes.
First results from the 2016 and 2017 collected data are reported in Ref. [41], first results from the 2018 collected data are reported in Ref. [40] and are not discussed here.

7.2 Proposed real-time reconstruction algorithm in FPGA

We will describe an algorithm for 3-dimensional pattern recognition and track reconstruction based on a stub identification approach, similarly to the one described for 4-dimensional track reconstruction using pixel sensors. In particular in the BGV, SciFi sensors provides only 2-dimensional measurements of the track hits, and the information from pairs of orthogonal layers have to be merged in order to provide a 3-dimensional measure of the trajectories.

Track parameters A 3-dimensional track is defined by 4 track parameters (x_+, y_+, m_x, m_y), with $(x_+,\ y_+) = ((x_f + x_l)/2,\ (y_f + y_l)/2)$ and $z_\pm = (z_f + z_l)/2$, where z_f, z_l are the z-coordinates of the first and last tracking planes, respectively and (x_f, y_f) and (x_l, y_l) are the coordinates of the track at z_f, z_l. In particular (x_+, y_+) are the coordinates of the intersection of the track at a reference plane placed at $z = z_+$; (m_x, m_y) are the slopes of the track along the x, y directions as defined in Eq. 4.1, described in Sec. 4.1.

Tracking system layout The detector is composed of a total of 16 SciFi module sides, which we will simply refer to as *sensors* in the following. Each sensor can be modelled as strip sensor with dimensions of 261.25 mm and 340.00 mm in the direction orthogonal and parallel to the strips, respectively.

an 97.9×97.9 mm^2 cut-out is present to accommodate the beam pipe. Sensors are organized in groups of four, as shown in Fig. 7.2: the first pair measures the x coordinate

of the hit, in the local coordinate system highlighted in the picture. One sensor is rotated by 2° with respect to the other (*stereo* configuration). Similarly the second pair of sensors measures the y coordinate. Both the near and far station comprise two groups of four sensors above and below the beam pipe, symmetrical to each other. We will refer to the sensors placed above and below the beam pipe as *top* and *bottom* sensors, respectively.

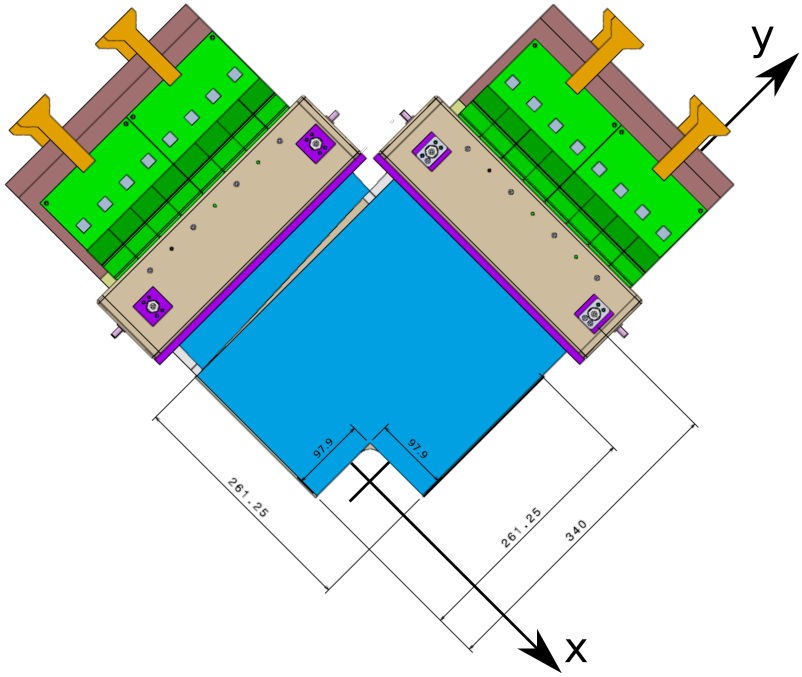

Figure 7.2: Arrangement of a group of four BGV sensors, organized as two pairs of orthogonal sensors, in stereo configuration.

The positions of the BGV sensors along the beam line are reported in Table 7.1. The sensors 0-7 are located above the beam line while sensors 8-15 are located below. For the purposes of the algorithm description we will refer to the tracking systems as two separate tracking systems formed by the two groups of sensors. We will refer to the sensors within the group of four as *x, u, v, y sensors*; in particular the x and v sensors are orthogonal to each others, the u sensor is rotated by 2° with respect to the u sensor, the v sensor is rotated by 2° with respect to the v sensor. Each tracking system is composed of 8 layers arranged according to the following scheme: *x-u-y-v-x-u-y-v*. The tracking algorithm is performed separately on the two tracking systems and will be described for just one of them and applied in the same way to the second.

Pattern recognition The pattern recognition algorithm is based on the identification of tracks with a hit in each tracking plane. It is based on the identification of pairs of hits in adjacent planes, called *halfstubs*, then on the identification of pairs of *halfstubs* that will form a *stub* and finally on the identification of tracks as pairs of *stubs*.

Sensor	0	1	2	3	4	5	6	7
z [mm]	1995.40	2042.16	2045.32	2092.08	2978.40	3025.16	3028.32	3075.08
Sensor	8	9	10	11	12	13	14	15
z [mm]	2095.24	2142.00	2145.16	2191.92	3078.24	3125.00	3128.16	3174.92

Table 7.1: Longitudinal positions of the *bottom* and *top* sensors in the BGV

In particular, according to the layout of the tracking system, a halfstub is formed by hits in sensors that are arranged in *stereo* configuration. The use of the stereo configuration allows to retrieve a 3-dimensional information of the pair of hits, with better precision on one of the two coordinate in the plane transverse to the z-axis. As an example, a halfstub obtained from a pair of x-u sensors, will provide a precise measurement of the x coordinate; vice versa for a halfstub obtained from a pair of v-y sensors. The 3-dimensional coordinates of a halfstub are defined as follows: the z coordinate is defined as the position in between the z coordinates of the two planes; the x and y coordinates are obtained from the coordinate of the line that minimizes the 3-dimensional distance of the hit strips from the line itself, in particular it is assumed that the origin of the line is positioned at (0,0,1000 mm). This assumption is motivated by the fact that the interaction region is approximately defined by 0 mm $< z <$ 2000 mm. The identification of the halfstubs can be performed in parallel over all the pairs of sensors: for each pair of sensors a set of geometrical cuts is evaluated from Monte Carlo simulation and applied to prevent processing pair of hits that are not compatible with a track from the interaction region; in particular the cuts consist in the application of a threshold to remove hit strips whose distance is higher with respect to the maximum value evaluated from simulation. The identification of the halfstubs from the measured hits, and the identification of the stubs from pairs of halfstubs is graphically described in Fig. 7.3.

The following step of the pattern recognition is based on the identification of *stubs* starting from the previously evaluated halfstubs. As we previously observed the use of the *stereo* configuration allows to evaluate the 3-dimensional coordinate of the halfstub, even if with different precisions. A stub is obtained from a pair of halfstubs whose coordinates are compatible within a certain distance range, evaluated from simulation. Each stub is then formed by a x-u halfstub and a v-y halfstub, hence a full track trajectory can be evaluated from the four 2-dimensional measured hits in four adjacent plane: the stub parameters are defined in the same way as the track parameters (x_+, y_+, m_x, m_y) and are evaluated as the parameters of the track the minimizes the 3-dimensional distance of the 2-dimensional hits from the line itself, similarly to the case of the halfstub parameters. Although, it has to be noted that no assumption on the line trajectory is done in this case. Even in this case the identification of the stubs can be performed in parallel over all the groups of four x-u-v-y sensors. After a candidate stub has been identified and its parameters evaluated, additional cuts are applied to discard stubs that are not compatible with particles from the interaction region, *i.e.* the d_0 and z_v parameters, evaluated as the distance of closest approach of the track to the beam line and the longitudinal coordinate of the track where this distance is minimum, respectively, are evaluated and required to be compatible with the distribution obtained from simulation.

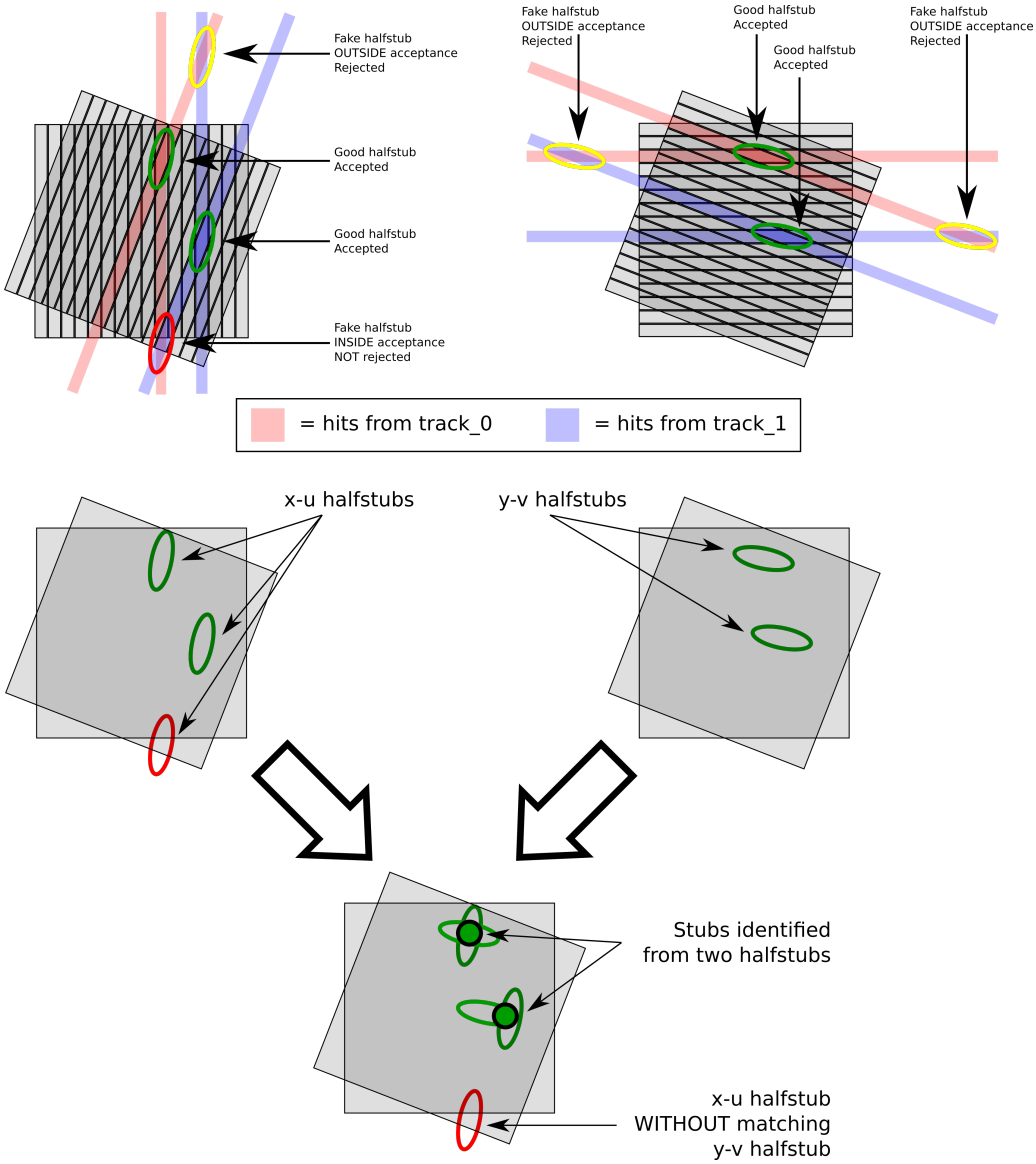

Figure 7.3: Example of identification of halfstubs and stubs in the BGV. Hits from two tracks, track_0 (red) and track_1 (blue) are considered. The halfstub identification process is performed in parallel on $x-u$ and $y-v$ sensor pairs. The halfstubs whose coordinates are outside the sensitive area are rejected. The stubs are identified from pairs of halfstubs with compatible coordinates.

The last step of the pattern recognition is the identification of compatible pairs of stubs, that we will refer to as *superstubs*. A superstub is formed by associating two stubs, one in the *near* station and one in the *far* station; in particular we point out that the association of the two stubs is performed on the basis of their (x_+, y_+) coordinates, only: stubs belonging to the same superstub need to have similar values of the main

track parameters, while the other two parameters (m_x, m_y) are not used in the pattern recognition but only to evaluate the complete track parameters of the superstub. It is worth noting that a superstub actually represents a candidate tracks, since it is formed by eight hits (in the *bottom* or *top* tracking system).
The identification of the superstubs from the measured stubs is graphically described in Fig. 7.4.

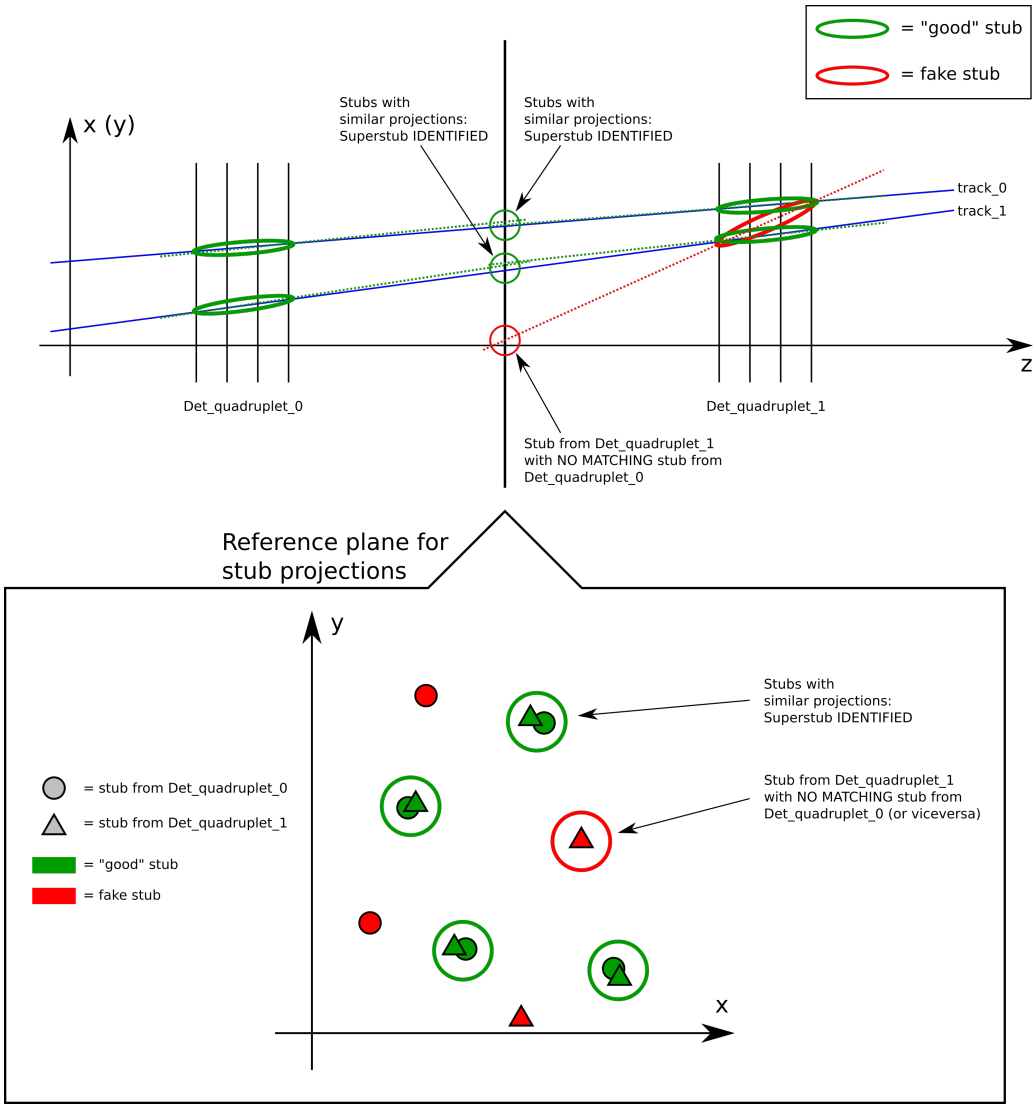

Figure 7.4: Example of identification of superstubs in the BGV. The stubs identified in the *near* and *far* station are projected to a reference plane. A superstub is identified from pairs of stubs with similar x, y coordinates.

The process of superstubs identification is performed in parallel for the *bottom* and *top* tracking systems. Once a superstub has been identified, the track parameters are obtained

as the parameters of a straight line connecting the barycentres of the first and second stub. Further cuts are applied to refine the evaluation and discard superstubs based on the line-to-hit distances of the track trajectory from the measured hits.

Algorithm architecture The proposed architecture for the algorithm implementation is similar to the one proposed in Section 5.3 for the 4D real-time tracking system with some modifications. In general it is possible to use part of the described modules to build the architecture, whose scheme is shown in Fig. 7.5 for a sub-system composed of the *top* sensors. The described architecture is replicated for a second sub-system composed of the *bottom* sensors.

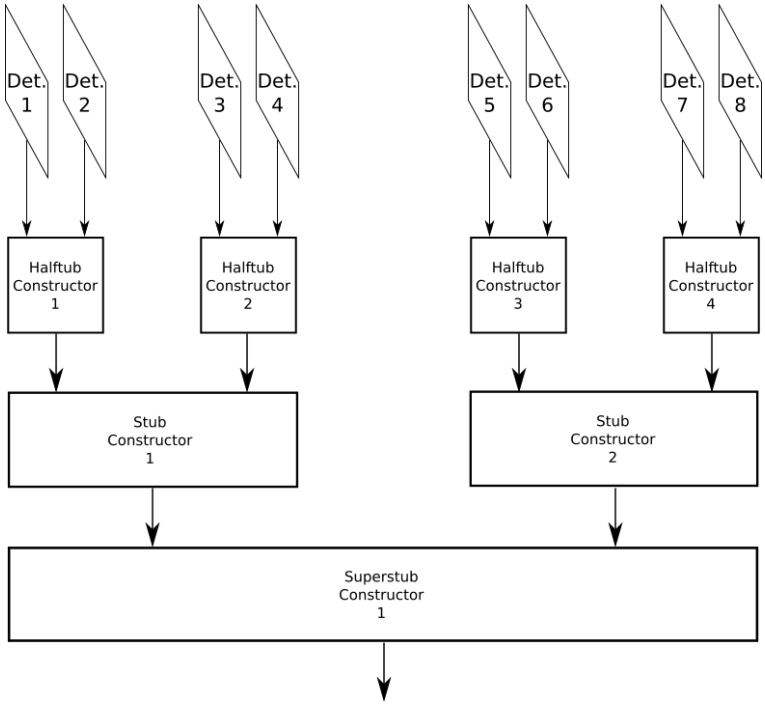

Figure 7.5: Architecture of the proposed algorithm for track reconstruction in BGV for half of the tracking system (*bottom* or *top*).

For each pair of sensors a *Halfstub Constructor* is instantiated. Its architecture follows from the architecture of the *Stub Constructor* described in Chapter 5. The difference in the use of the terminology has to be noted and is motivated as follows: in both cases a *stubs* provides the minimum information to identify a track without any extra assumption, even if with a reduced number of hits. Since pixel sensors (without timing) provide 3-dimensional information, all the four parameters of a straight 3-dimensional tracks can be estimated from the combination of two points; strip sensors, instead, provide only 2-dimensional information, so the combination of two hits is not enough to estimate the four track parameters, while the combination of four 2-dimensional hits allows to estimate

all the track parameters[1].
The architecture of the Halfstub Constructor is based on the distribution of the hits from the first and the second sensor (within a pair) to a pool of Halfstub Makers, that process the combination of hits and provide the halfstubs in output. Each sensor is divided in a certain number of regions in such a way that each region is equally populated, according to the distribution of the hits obtained from simulated events. One Halfstub Maker is associated with each region of the first sensor, while a Hit Switch delivers the hits from the second sensor to a finite set of Halfstub Makers, according to the map of compatible pairs of regions, that has been previously evaluated from Monte Carlo simulations. The Halfstub Makers operate in parallel, each one processing just a part of all the hits combinations: this allows to speed up the computation time and to simplify the combinatoric problem. In fact, the hit distribution in the Switch already reduces the number of combinations, by delivering the hits only to the regions where valid halfstubs are expected to be found.
The architecture of the Stub Constructor is replicated from the previous one. In this case the Stub Constructor receives the halfstubs from the Halfstub Constructor and provides Stubs in output. The space of the halfstub coordinates for each detector pair is divided into an equally populated grid. The map of compatible regions is previously evaluated and used to configure the Switches inside the Stub Constructor: halfstubs from the first and second sensor doublets (within each group of four) are delivered to the appropriate regions according to the map; the Stub Constructor is composed of a pool of Stub Makers, each one processing one region. The Stub Makers operate in parallel and evaluate and filter the stubs from halfstubs combinations.
Regarding the identification of superstubs, the same strategy is applied. In this case a pool Superstub Makers is associated with a grid of regions in the (r_+, ϕ_+) space, designed to be equally populated. This choice of parameters is motivated by the fact that the distribution of the tracks with respect to r_+ is almost independent from the distribution the tracks with respect to ϕ_+: in fact the track generation is independent with respect to rotations around the z-axis; nevertheless the detector is not cylindrically symmetric. The Superstub Makers operate in parallel, as in the previous cases, and provide the superstubs (tracks) in output.
It is evident from the description that the proposed architecture is intrinsically parallel. On one hand the simplest level of parallelization is obtained by separating the processing over independent modules, *i.e.* the halfstub search is performed for each pair of sensors independently from each other. The second level of parallelization is obtained within the single Halfstub/Stub/Superstub constructors, in which the data are delivered in parallel to the regions with expected non-zero response.

Simulation results The algorithm has been simulated using 10000 events of the beam-gas interactions in the Gas Target system. Events with only one track have been used to perform the training of the algorithm, *i.e.* for evaluating the compatible regions on the sensors. Moreover single track events are used to provide a benchmark of the reconstruction quality in order to compare the resolutions obtained from simulation of realistic

[1] from a pure geometrical point of view only three hits are needed.

events.

Results are obtained from high level simulation of the algorithm that is intended to provide information on the algorithm reconstruction quality and it is not fully optimized. In this simulation 32 Halfstub Makers have been instantiated for each Halfstub Constructor, 8×8 Stub Makers for each Stub Constructor, 8×16 Superstub Makers for each Superstub Constructor.

The resolution on the track parameters is evaluated from the fit of the distribution of the reconstructed parameters minus the parameter of the simulated track. The distributions have been fitted with a Gaussian function or the sum of two Gaussian functions. The resolution is defined as the *root mean square* of the fitting function. The resolution on (x_+, y_+) parameters for single track events is $\sigma_{x_+,y_+} = 0.050$ mm, while the resolution on (m_x, m_y) parameters is $\sigma_{m_x,m_y} = 0.0001$.

The residual distributions for reconstructed minus generated (x_+, y_+, m_x, m_y) track parameters, obtained from simulation of beam-gas interactions in the BGV are shown in Fig. 7.6. The resolutions obtained from the Gaussian fit of the distributions are similar to the values obtained from single track events and are $\sigma_{x_+} = 0.056$ mm, $\sigma_{y_+} = 0.056$ mm, $\sigma_{m_x} = 0.0001$, $\sigma_{m_y} = 0.0001$.

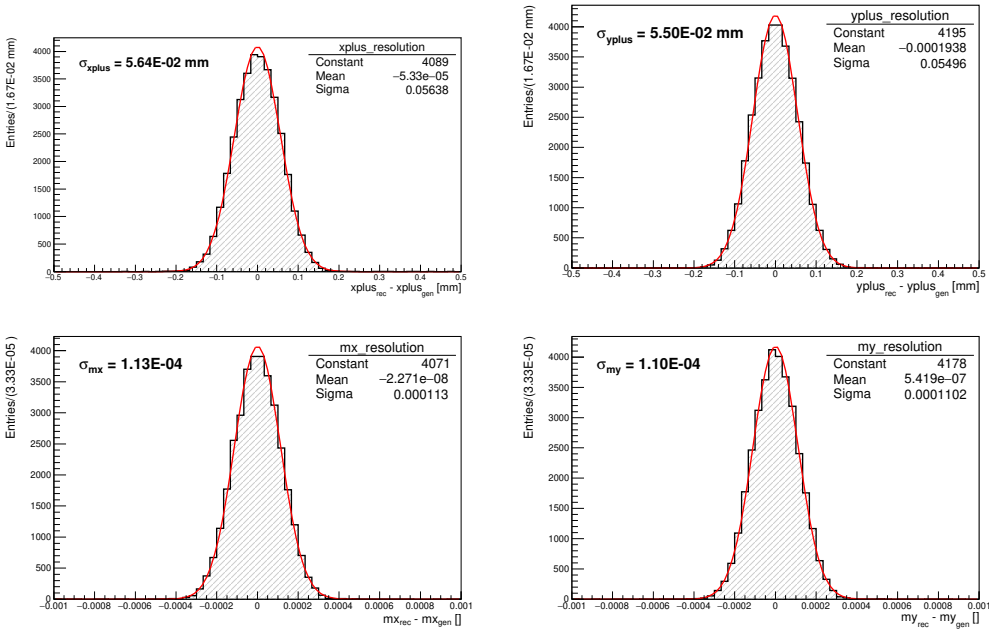

Figure 7.6: Residual distributions for reconstructed minus generated (x_+, y_+, m_x, m_y) parameters for simulated beam-gas events in the BGV. The resolution are: $\sigma_{x_+} = 0.056$ mm, $\sigma_{y_+} = 0.056$ mm, $\sigma_{m_x} = 0.0001$, $\sigma_{m_y} = 0.0001$.

The residual distributions for (d_0, z_0) parameters are shown in Fig. 7.7. These are defined as the distance of closest approach of the track to the z-axis and the z coordinate where the distance is minimum, evaluated according to Eq. 4.23. The resolutions obtained on

d_0, z_0 track parameters are $\sigma_{d_0} = 0.213$ mm, $\sigma_{z_0} = 6.17$ mm.

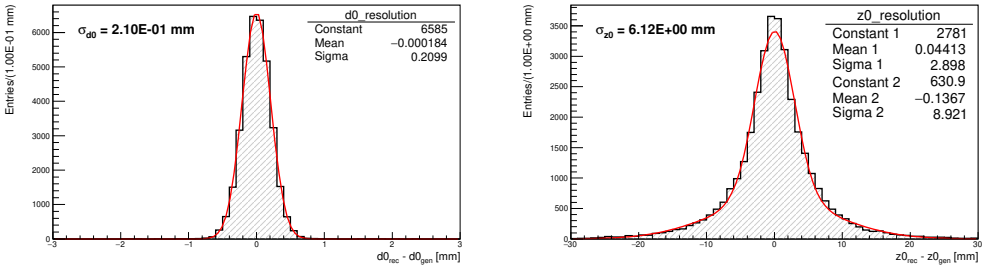

Figure 7.7: Residual distributions for reconstructed minus generated (d_0, z_0) parameters for simulated beam-gas events in the BGV. The resolution are: $\sigma_{d_0} = 0.213$ mm, $\sigma_{z_0} = 6.17$ mm.

The reconstruction efficiency ϵ_{rec} is defined as ratio between the number of reconstructed tracks and the number of reconstructible tracks. In order to evaluate the efficiency, for each generate track it is required to find a reconstructed track within a maximum distance of $2\sqrt{2}$ mm in the (x_+, y_+) plane. The estimated efficiency is $\epsilon_{rec} = 97\%$.
The reconstruction efficiency as a function of the r_+, ϕ_+, d_0, z_0 track parameters is shown in Fig. 7.8.

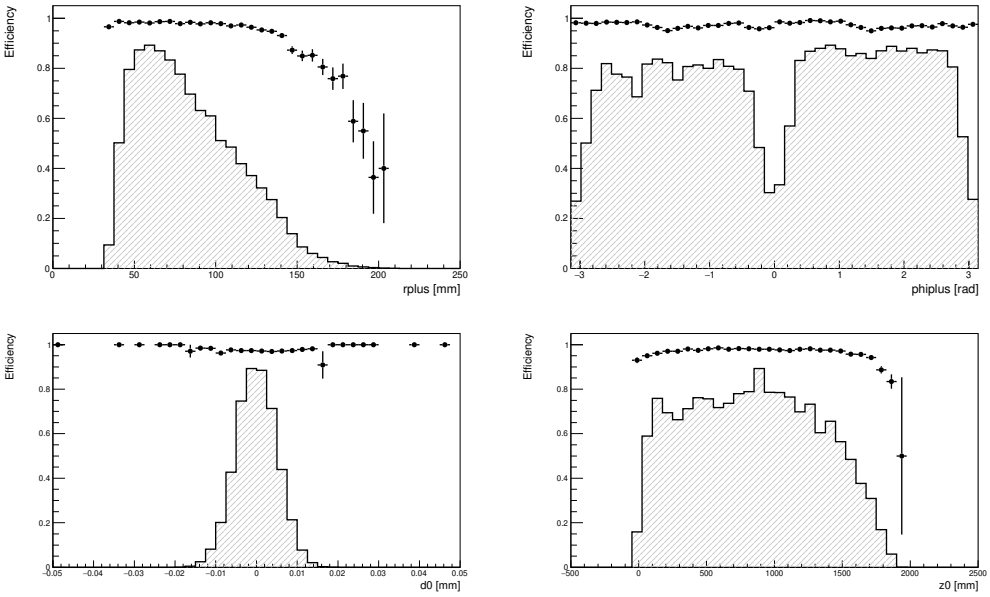

Figure 7.8: Reconstruction efficiency as a function of r_+, ϕ_+, d_0, z_0. The dashed histograms show the distribution of the generated track parameters.

The reconstruction efficiency as a function of the number of reconstructible tracks per event and as a function of the number of track hits in the whole detector is shown in Fig. 7.9.

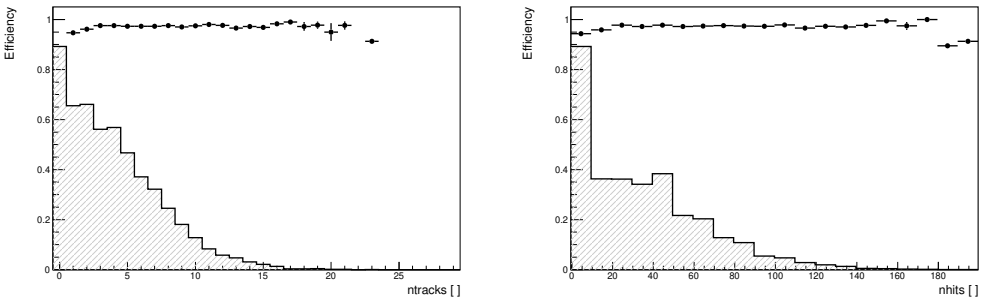

Figure 7.9: Reconstruction efficiency as a function of the number of tracks and hits in the BGV. The dashed histogram show the distribution of the number of reconstructible track and hits in the BGV.

The efficiency plots show no dependence of the reconstruction efficiency form the different track and event parameters.

Discussion on hardware implementation In the previous paragraph the results from high level simulation have been shown, providing good reconstruction quality in terms of track parameters resolution and in terms of efficiency.
The high-level simulation does not provide direct evidence of the feasibility of the hardware implementation, which has to be discussed. Nevertheless, information about the expected throughput and latency can be extracted from the simulation at intermediate steps. In this paragraph we will discuss the expected performances of the processing system, and in particular we will refer to the configuration proposed before: 32 Halfstub Makers for each Halfstub Constructor, 8×8 Stub Makers for each Stub Constructor, 8×16 Superstub Makers for each Superstub Constructor.
The distribution of the maximum occupancy of the Halfstub Makers (on an event-by-event basis) for each Halfstub Constructor, associated with the eight pairs of sensors, is shown in Fig. 7.10. The distributions show the maximum number of candidate halfstubs to be processed in each event (dots), namely the product of the number of hits from the first and second sensor within the pair, and the number of halfstubs provided in output after the application of the geometrical cuts (solid line).
We point out that the maximum number of hit combinations can be used to estimate the computation time needed for the identification of the halfstubs in a single event: in fact, in presence of multiple Halfstub Makers, the computation time can be approximated to the time needed by the slowest one. From the plots we can observe that the most occupied modules are associated with the pairs of sensors in the *far* stations, with an average number (worst case) of ~ 3.9 candidate combinations to process, with standard deviation of ~ 2.7. Recalling the discussion on the latency and throughput of the *Stub Maker* in Chapter 5, one candidate halfstub can be processed per each clock cycle, in a pipelined architecture; this means that if we assume an event rate of 40 MHz and the algorithm processing data at a reference clock of 320 MHz we can expect no loss of data and a full throughput of identified and filtered halfstubs.

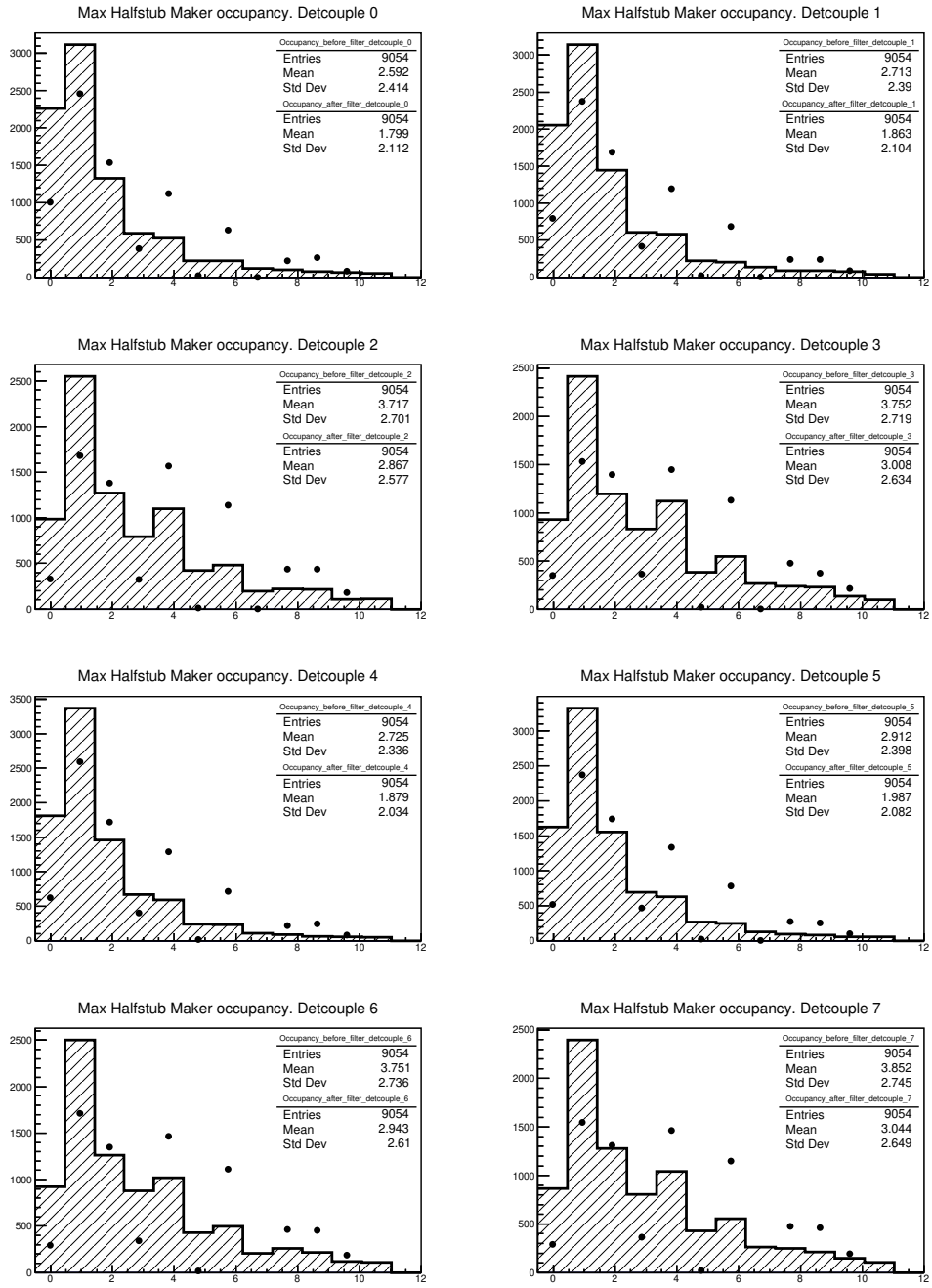

Figure 7.10: Distribution of the maximum occupancy per event of the Halfstub Makers for the eight detector pairs: number of hits combinations to process (dots) and number of filtered halfstubs (solid line).

The distribution of the maximum occupancy of the Stub Makers (on an event-by-event basis) for each group of four tracking sensors is shown in Fig. 7.11. Similarly to the previous plots, the distributions show the maximum total number of stubs to be processed in an event (dots) and the number of identified stubs, filtered by the application of geometrical cuts (solid line).

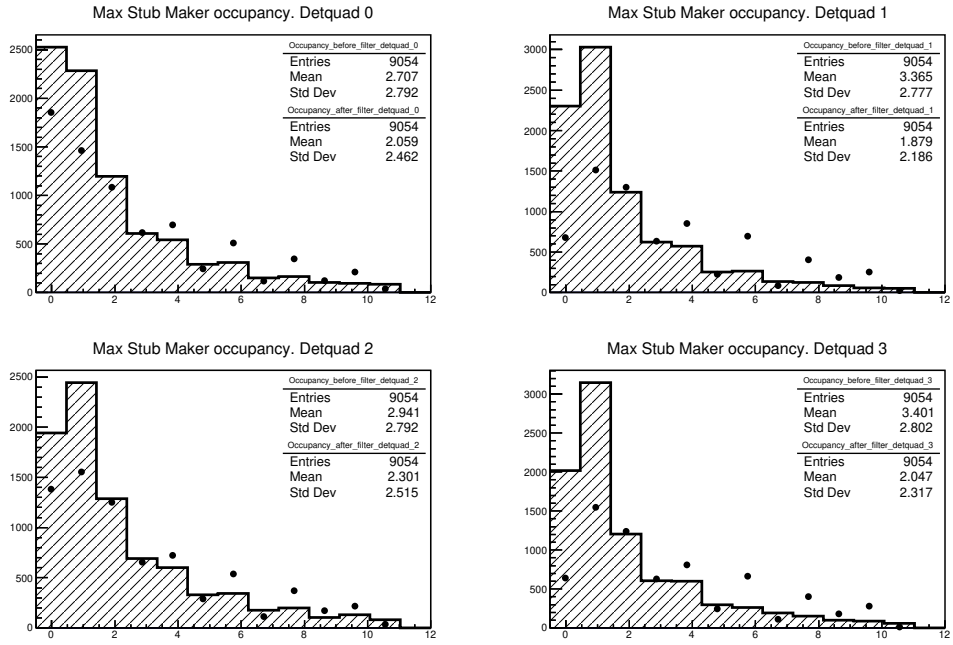

Figure 7.11: Distribution of the maximum occupancy per event of the Stub Makers for the four detector quadruplets: number of halfstubs combinations to process (dots) and the number of filtered stubs (line).

The most occupied Stub Makers corresponds to the quadruplets of sensors in the *far* stations. The worst case value of occupancy corresponds to an average of ~ 3.4, with standard deviation of ~ 2.8. Provided the previous assumptions on the reference clock for the running algorithm we can expect on average no loss of data and full data throughput of identified and filtered stubs.
The distribution of the maximum occupancy of the Superstub Makers for the *bottom* and *top* tracking systems is shown in Fig. 7.12. The distribution of the maximum value of number of superstubs to be processed in an event (dots) and the corresponding number of identified superstubs after the geometrical cuts (solid line) is shown.
The obtained value of occupancy corresponds to ~ 2.0 (~ 2.3) with standard deviation of ~ 2.6 (~ 2.6) for the Superstub Makers associated with the *bottom* (*top*) tracking system. This means that an average of 2 clock cycles are needed to process the superstubs of each event, while the proposed system is expected to be able to process eight combinations, at a reference clock of 320 MHz, compared to the bunch collision rate of 40 MHz, before new data are available from the processing of the following event.

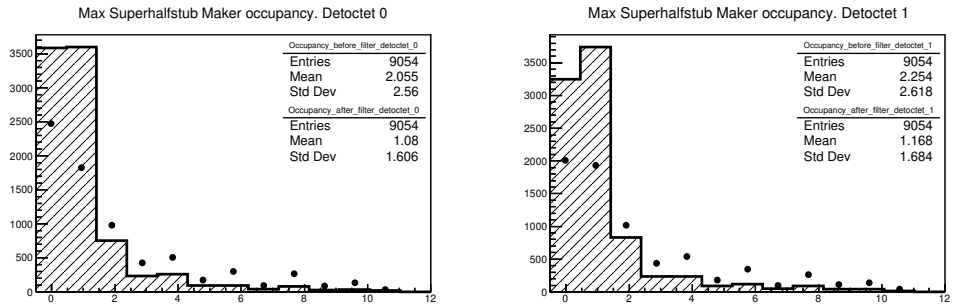

Figure 7.12: Distribution of the maximum occupancy per event of the Superstub Makers for the *bottom* and *top* tracking systems: number of stubs combinations to process (dots) and number of filtered superstubs (line).

It is worth that the current trigger rate for the BGV, is at the order of 100 KHz, with bunches of 10^{11} protons and neon pressure of $\approx 1 \times 10^{-7}$ mbar [42]. These conditions are very relaxed with respect to the maximum rate of 40 MHz used in the previous considerations. In this sense the quantity of resources for reconstructing the BGV tracks in real-time could be revised, *i.e.* reducing the number Halfstub/Stub/Superstub Makers or by resource sharing in the FPGA implementation: this would lead to higher computation times while reducing the hardware resources.

Conclusions In this section an algorithm for real-time track reconstruction has been proposed. The algorithm is based on the identification of hit doublets in adjacent sensors, then on the identification of pairs of doublets and finally on the identification of track candidates from quadruplets of hits in the *near* and *far* tracking stations of the BGV.
The architecture is highly parallelized and the high level simulation of the system shows that it is able to provide a good reconstruction quality with high efficiency. The feasibility of the hardware implementation in FPGA has been briefly discussed; in fact the architecture is similar to the one described for the implementation of the 4D real-time reconstruction algorithm and the implementation of the major logic modules can be reused with minor modifications. At first approximation, there is no evidence of possible bottlenecks in the reconstruction chain even at 40 MHz event rate. Moreover the conditions of the BGV beam-gas interactions are quite relaxed.
The proposed study can not be intended as a complete study since the system has not been implemented and tested in hardware. Nevertheless it is intended as a starting point for a possible application of a real-time tracking algorithm in the future BGV detector, since after having demonstrated its capabilities, it is going to be upgraded to operate at HL-LHC.

Summary

In this thesis we presented a fast-tracking device for real-time track reconstruction and its implementation in FPGA. The device has been designed to be applied to a possible Upgrade II of the VELO detector of the LHCb experiment, capable to provide precise space and time information of the detector hits for a 4D reconstruction of the tracks. The LHCb collaboration identified the addition of timing information as a solution to cope with the effects of the luminosity increase and to allow the experiment to keep producing high quality measurements.
The proposed tracking algorithm is based on the early parallel identification of hit doublets in adjacent sensor planes, *stubs*, and on the identification of candidate tracks from groups of stubs with similar parameters. The algorithm has been characterized in terms of track reconstruction quality and efficiency using a high-level standalone simulation of a VELO-like system and provided high efficiency and good resolution on the track parameters. In particular the results from the simulations show a reduction of the ghost track ratio with respect to the case in which the timing information is not included in the pattern recognition, while keeping high efficiency.
The tracking algorithm has been implemented in FPGA and described using VHDL language. All the modules used for the firmware implementation have been designed from scratch with a particular focus on the minimization of the system latency, achieved by the implementation of a highly parallelized and pipelined architecture and a *hold* logic for the optimization of the data flow. The parallelized nature of the algorithm and the modularity of the system make it suitable for implementation on independent FPGAs, and scalable to large detectors.
The system has been tested in hardware using the gFEX prototype board featuring multiple Xilinx Virtex UltraScale FPGAs and a Xilinx Zynq SoC FPGA, together with high-speed optical links with up to 1.6 Tbps input bandwidth. The hardware test of a sector corresponding to 1/64 of the VELO-like detector reproduced the results obtained from low-level simulation of the implemented algorithm. The test has been performed by generating, processing and analyzing the elaborated data with an *ad-hoc* architecture implemented in a single FPGA. From these results we can assume that the full system can be processed using 32 gFEX boards, equipped with two FPGAs. Moreover, more powerful

FPGAs with respect to the one that we used are already available on the market at the time of this writing. The system has proved to be able to process the simulated data at event rates of 40 MHz within a latency of O(1) μs, which proves the feasibility of this approach for a fast tracking device for a future LHCb Upgrade.
Other topics covered in this thesis include the implementation of the Artificial Retina algorithm for real-time track reconstruction applied to a 2-dimensional tracking system based on silicon strip sensors. This work has been propaedeutic to the design of the 4D real-time tracking device; in fact on one hand the two systems implement similar technical features, while on the other hand all the implemented modules for the 4D real-time tracking system have been completely redesigned. The Artificial Retina algorithm has been implemented in FPGA on a custom DAQ board (MAMBA board) and has been tested and validated with real data from a testbeam and represents a demonstration of the Artificial Retina approach, even if with relaxed conditions and lower rates with respect to collisions at LHC.
Finally, we provided a feasibility study for a real-time reconstruction algorithm implemented in FPGA to be applied to the Beam Gas Vertex detector for beam profile measurements at LHC. In this study, an architecture similar to the one proposed for the 4D real-time tracking algorithm has been addressed for the parallel identification of hit doublets, quadruplets and finally tracks in the BGV detector. This system has not been implemented in FPGA but it is based on the same modules implemented for the 4D tracking system with minor modifications. This study represents a starting point for discussing a possible upgrade of the reconstruction strategy of BGV events, now based on software.

Bibliography

[1] A. Augusto Alves, Jr. et al. The LHCb Detector at the LHC. *JINST*, 3:S08005, 2008.

[2] Lyndon Evans and Philip Bryant. LHC Machine. *JINST*, 3:S08001, 2008.

[3] Apollinari G., Béjar Alonso I., Brüning O., Fessia P., Lamont M., Rossi L., and Tavian L. *High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report V. 0.1.* CERN Yellow Reports: Monographs. CERN, Geneva, 2017.

[4] K. Aamodt et al. The ALICE experiment at the CERN LHC. *JINST*, 3:S08002, 2008.

[5] G. Aad et al. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008.

[6] S. Chatrchyan et al. The CMS Experiment at the CERN LHC. *JINST*, 3:S08004, 2008.

[7] G. Anelli et al. The TOTEM experiment at the CERN Large Hadron Collider. *JINST*, 3:S08007, 2008.

[8] LHCf Collaboration, O. Adriani, L. Bonechi, M. Bongi, G. Castellini, R. D'Alessandro, D. A. Faus, K. Fukui, M. Grandi, M. Haguenauer, Y. Itow, K. Kasahara, D. Macina, T. Mase, K. Masuda, Y. Matsubara, H. Menjo, M. Mizuishi, Y. Muraki, P. Papini, A. L. Perrot, S. Ricciarini, T. Sako, Y. Shimizu, K. Taki, T. Tamura, S. Torii, A. Tricomi, W. C. Turner, J. Velasco, A. Viciani, H. Watanabe, and K. Yoshida. The LHCf detector at the CERN Large Hadron Collider. *Journal of Instrumentation*, 3:S08006, August 2008.

[9] James Pinfold. The moedal experiment at the lhc. *EPJ Web of Conferences*, 145:12002, 01 2017.

[10] Burkhard Schmidt. The high-luminosity upgrade of the LHC: Physics and technology challenges for the accelerator and the experiments. *Journal of Physics: Conference Series*, 706:022002, apr 2016.

[11] G. Apollinari, O. Brüning, T. Nakamoto, and Lucio Rossi. High Luminosity Large Hadron Collider HL-LHC. *CERN Yellow Report*, (5):1–19, 2015.

[12] R. Lindner.

[13] Christian Elsässer. $\bar{b}b$ production angle plots. [Online; accessed 5-November-2018].

[14] LHCb VELO TDR: Vertex locator. Technical design report. 2001.

[15] LHCb: Inner tracker technical design report. 2002.

[16] LHCb: Outer tracker technical design report. 2001.

[17] LHCb: RICH technical design report. 2000.

[18] LHCb calorimeters: Technical design report. 2000.

[19] LHCb muon system technical design report. 2001.

[20] LHCb Collaboration. LHCb VELO Upgrade Technical Design Report. Technical Report CERN-LHCC-2013-021. LHCB-TDR-013, Nov 2013.

[21] LHCb Collaboration. LHCb Tracker Upgrade Technical Design Report. 2014.

[22] Roel Aaij et al. Expression of Interest for a Phase-II LHCb Upgrade: Opportunities in flavour physics, and beyond, in the HL-LHC era. Technical Report CERN-LHCC-2017-003, CERN, Geneva, Feb 2017.

[23] Roel Aaij et al. Physics case for an LHCb Upgrade II - Opportunities in flavour physics, and beyond, in the HL-LHC era. 2018.

[24] Guoming Liu and Niko Neufeld. DAQ Architecture for the LHCb Upgrade. *J. Phys. Conf. Ser.*, 513:012027, 2014.

[25] B. Ashmanskas, A. Barchiesi, A. Bardi, M. Bari, M. Baumgart, S. Belforte, J. Berryhill, M. Bogdan, R. Carosi, A. Cerri, G. Chlachidze, R. Culbertson, M. Dell'Orso, S. Donati, I. Fiori, H. Frisch, S. Galeotti, P. Giannetti, V. Glagolev, A. Leger, Y. Liu, T. Maruyama, E. Meschi, L. Moneta, F. Morsani, T. Nakaya, G. Punzi, M. Rescigno, L. Ristori, H. Sanders, S. Sarkar, A. Semenov, M. Shochet, T. Speer, F. Spinella, H. Vataga, X. Wu, U. K. Yang, L. Zanello, A. M. Zanetti, and CDF-II Collaboration. The CDF Silicon Vertex Trigger. *Nuclear Instruments and Methods in Physics Research A*, 518:532–536, February 2004.

[26] Bill Ashmanskas et al. The CDF silicon vertex trigger. *Nucl. Instrum. Meth.*, A518:532–536, 2004.

[27] M Shochet, L Tompkins, V Cavaliere, P Giannetti, A Annovi, and G Volpi. Fast TracKer (FTK) Technical Design Report. Technical Report CERN-LHCC-2013-007. ATLAS-TDR-021, Jun 2013. ATLAS Fast Tracker Technical Design Report.

[28] Nicolo Vladi Biesuz and ATLAS Collaboration. First performance measurements of the Fast Tracker Real Time Processor at ATLAS. Technical Report ATL-DAQ-PROC-2018-028, CERN, Geneva, Oct 2018.

[29] M. Trovato. Track Trigger at the High Luminosity LHC, 2018.

[30] A. Svetek, M. Blake, M. Cepeda Hermida, S. Dasu, L. Dodd, R. Fobes, B. Gomber, T. Gorski, Z. Guo, P. Klabbers, A. Levine, I. Ojalvo, T. Ruggles, N. Smith, W.H. Smith, J. Tikalsky, M. Vicente, and N. Woods. The calorimeter trigger processor card: the next generation of high speed algorithmic data processing at cms. *Journal of Instrumentation*, 11(02):C02011, 2016.

[31] L. Ristori. An artificial retina for fast track finding. *Nucl. Instrum. Meth.*, A453:425–429, 2000.

[32] Laura Borrello, Alberto Messineo, Ettore Focardi, and Anna Macchiolo. Sensor Design for the CMS Silicon Strip Tracker. 2003.

[33] S Löchner and M Schmelling. The Beetle Reference Manual - chip version 1.3, 1.4 and 1.5. Technical Report LHCb-2005-105. CERN-LHCb-2005-105, CERN, Geneva, Nov 2006.

[34] Michael Begel, William Buttinger, Hucheng Chen, Hal Evans, Walter Hopkins, Vasiliki Kouskoura, Sabine Lammers, Francesco Lanni, Stephanie Majewski, David Miller, Sam Silverstein, Nikolai Sinev, Giordon Stark, David Strom, Helio Takai, Fukun Tang, and Shaochun Tang. Atlas note atlas level-1 calorimeter trigger upgrade global feature extractor prototype module (l1calo gfex) technical specifications, 11 2014.

[35] Shaochun Tang, Michael Begel, Hucheng Chen, Kai Chen, Francesco Lanni, Helio Takai, and Weihao Wu. The development of Global Feature eXtractor (gFEX) - the ATLAS calorimeter Level 1 trigger for ATLAS at High Luminosity LHC. Technical Report ATL-DAQ-PROC-2017-035, CERN, Geneva, Oct 2017.

[36] Xilinx. UltraScale FPGA. Product Tables and Product Selection Guide, 2018. [Online; accessed 8-October-2018].

[37] Xilinx. Zynq-7000 All Programmable SoC Family. Product Tables and Product Selection Guide, 2018. [Online; accessed 8-October-2018].

[38] vadatech. vadatech ATCA Moudle Power VT000 Datasheet, 2018. [Online; accessed 8-October-2018].

[39] Xilinx. UltraScale FPGAs Transceivers Wizard v1.7 - Product Guide, 2018. [Online; accessed 10-October-2018].

[40] A. Alexopoulos et al. Noninvasive LHC transverse beam size measurement using inelastic beam-gas interactions. *Phys. Rev. Accel. Beams*, 22(4):042801, 2019.

[41] Sotiris Vlachos et al. The LHC Beam Gas Vertex Detector - a Non-Invasive Profile Monitor for High Energy Machines. In *Proceedings, 6th International Beam Instrumentation Conference, IBIC2017*, page WE3AB1, 2018.

[42] BGV Team. Measuring beam size with the BGV - Results from the demonstrator in Run2.

List of Publications

N. Neri *et al.*, "Testbeam results of the first real-time embedded tracking system with artificial retina",
Nucl. Instrum. Meth. A **845** (2017) 607. doi:10.1016/j.nima.2016.05.129

N. Neri, A. Cardini, R. Calabrese, M. Fiorini, E. Luppi, U. Marconi and M. Petruzzo, "4D fast tracking for experiments at high luminosity LHC",
JINST **11** (2016) no.11, C11040. doi:10.1088/1748-0221/11/11/C11040

N. Neri *et al.*,
"First results of a detector embedded real-time tracking system with artificial retina",
doi:10.1109/NSSMIC.2015.7581772

A. Abba *et al.*,
"An "artificial retina" processor for track reconstruction at the full LHC crossing rate",
Nucl. Instrum. Meth. A **824** (2016) 260. doi:10.1016/j.nima.2015.10.048

A. Abba *et al.*,
"Real time tracking with a silicon telescope prototype using the "artificial retina" algorithm",
Nucl. Instrum. Meth. A **824** (2016) 343. doi:10.1016/j.nima.2015.11.064

A. Abba *et al.*,
"Testbeam studies of pre-prototype silicon strip sensors for the LHCb UT upgrade project",
Nucl. Instrum. Meth. A **806** (2016) 244 doi:10.1016/j.nima.2015.10.031 [arXiv:1506.00229 [physics.ins-det]].

N. Neri *et al.*,
"First results of the silicon telescope using an 'artificial retina' for fast track finding,"
ANIMMA (2015) pp. 1-4, doi: 10.1109/ANIMMA.2015.7465644

N. Neri *et al.*,
"First prototype of a silicon tracker using an 'artificial retina' for fast track finding",
PoS TIPP **2014** (2014) 199 [arXiv:1409.3466 [physics.ins-det]].

A. Abba *et al.*,
"A specialized track processor for the LHCb upgrade",
LHCb-PUB-2014-026

A. Abba *et al.*,
"A Retina-Based Cosmic Ray Telescope",
Proc. of the 2014 IEEE Real-Time Conference, May 26-30, 2014, Nara, Japan.

A. Abba *et al.*,
"First prototype of a silicon tracker using an "artificial retina" for fast track finding",
Proc. of TIPP 2014, 2-6 June, 2014, Amsterdam, The Netherlands.