Studies on FPGA-Based Solutions for the ATLAS Phase II Trigger and Data Acquisition System

Axiotis, Konstantinos

How to cite

AXIOTIS, Konstantinos. Studies on FPGA-Based Solutions for the ATLAS Phase II Trigger and Data Acquisition System. Doctoral Thesis, 2025. doi: 10.13097/archive-ouverte/unige:183583

This publication URL: https://archive-ouverte.unige.ch/unige:183583
Publication DOI: 10.13097/archive-ouverte/unige:183583

UNIVERSITÉ DE GENÈVE
Département de physique nucléaire
et corpusculaire (DPNC)

FACULTÉ DES SCIENCES
Professeure Anna Sfyrla

Studies on FPGA-Based Solutions for the ATLAS Phase II Trigger and Data Acquisition System

THÈSE

présentée à la Faculté des sciences de l'Université de Genève
pour obtenir le grade de Docteur ès sciences, mention physique

Konstantinos Axiotis

de Thessaloniki (Grèce)

Thèse N° 5887

GENÈVE
2025

UNIVERSITÉ
DE GENÈVE

FACULTÉ DES SCIENCES

DOCTORAT ÈS SCIENCES, MENTION PHYSIQUE

Thèse de Monsieur Konstantinos AXIOTIS

intitulée :

**«Studies on FPGA-Based Solutions
for the ATLAS Phase II
Trigger and Data Acquisition System»**

La Faculté des sciences, sur le préavis de

Madame A. SFYRLA, professeure associée et directrice de thèse
Département de physique nucléaire & corpusculaire

Monsieur S. SCHRAMM, professeur assistant
Département de physique nucléaire & corpusculaire

Monsieur B. PETERSEN, docteur
CERN, Genève

Monsieur A. UPEGUI POSADA, professeur associé
HEPIA, Genève

autorise l'impression de la présente thèse, sans exprimer d'opinion sur les propositions qui y sont énoncées.

Genève, le 17 février 2025

Thèse - 5887 -

La Doyenne

N.B. - La thèse doit porter la déclaration précédente et remplir les conditions énumérées dans les "Informations relatives aux thèses de doctorat à l'Université de Genève".

Abstract

The nature of dark matter, the imbalance between matter and antimatter, and the properties of the Higgs boson are among the fundamental questions in high-energy physics studied by experiments like ATLAS at the LHC. The high collision rates and massive data volumes produced by the ATLAS experiment require an efficient real-time processing system, the *Trigger and Data Acquisition* (TDAQ) system. This system is tasked with identifying valuable information within a large background of data and ensuring its storage. To meet these challenges in the *High-Luminosity LHC* (HL-LHC) era, the ATLAS experiment is undergoing a Phase II upgrade.

This thesis focuses on developing FPGA-based low-latency and high-throughput solutions for various components of the Phase II TDAQ upgrade. The projects presented include the firmware development for the *Central Trigger Processor* (CTP) readout and data buffering feasibility studies; firmware development for the *Pattern Recognition Mezzanine* (PRM) and hardware validation of the system; exploration of FPGA accelerator capabilities in a data center architecture; and the development and validation of the Vitis accelerator backend for HLS4ML to automate the generation of HLS4ML projects targeting CPU-FPGA accelerator setups, including its application with an ML-based algorithm.

The work contributes to the ATLAS experiment's goal of exploring fundamental physics phenomena, including the Standard Model and potential new physics. These developments enhance the efficiency and capability of the TDAQ system, preparing it to handle the increasing data rates and complexities of future upgrades.

Résumé

La nature de la matière noire, le déséquilibre entre la matière et l'antimatière et les propriétés du boson de Higgs font partie des questions fondamentales de la physique des hautes énergies étudiées par des expériences comme ATLAS au LHC. Les taux de collision élevés et les volumes de données considérables produits par l'expérience ATLAS nécessitent un système de traitement en temps réel efficace, le système *Trigger and Data Acquisition* (TDAQ). Ce système est chargé d'identifier les informations utiles dans un vaste ensemble de données et d'en assurer le stockage. Pour relever ces défis à l'ère du *High-Luminosity LHC* (HL-LHC), l'expérience ATLAS fait l'objet d'une mise à niveau de Phase II.

Cette thèse se concentre sur le développement de solutions à faible latence et à haut débit basées sur des FPGA pour divers composants de la mise à niveau du TDAQ de la phase II. Les projets présentés comprennent le développement de microprogrammes pour le *Central Trigger Processor* (CTP) et les études de faisabilité de la mise en mémoire tampon des données ; le développement de microprogrammes pour le *Pattern Recognition Mezzanine* (PRM) et la validation matérielle du système; l'exploration des capacités de l'accélérateur FPGA dans une architecture de centre de données; et le développement et la validation du backend de l'accélérateur Vitis pour HLS4ML afin d'automatiser la génération de projets HLS4ML ciblant des configurations d'accélérateur CPU-FPGA, y compris son application avec un algorithme basé sur le ML.

Ces travaux contribuent à l'objectif de l'expérience ATLAS d'explorer les phénomènes de physique fondamentale, y compris le modèle standard et la nouvelle physique potentielle. Ces développements améliorent l'efficacité et la capacité du système TDAQ, le préparant à gérer les débits de données croissants et les complexités des futures mises à jour.

Acknowledgments

I would like to express my heartfelt gratitude to my supervisor, Anna Sfyrla for the opportunity to start this PhD and invaluable guidance and support throughout the years. I also could not have completed this pursuit without Brian Petersen, who co-supervised me during the first years of my PhD.

Furthermore, I am extremely grateful to all my collaborators who worked alongside me on the same projects. The PRM developers team, Alessandra Camplani, Sebastian Dittmeier, and Joachim Zinßer. The University's engineers Quentin Berthet and Stéphane Débieux, and the CERN ESE group's physicists and engineers Thilo Pauly, Stefan Haas and Anna Kulinska.

I am also grateful to my colleagues at the university of Geneva. Namely, the postdocs Claire Antel and Pantelis Kontaxakis, and the PhD students Lucas Bezio, Stefano Franchellucci, Edward Galantay, Noshin Tarannum, and Ondřej Theiner.

I would like to thank my friends for their constant support and encouragement throughout this journey and last but not least, I want to express my heartfelt appreciation to my family, whose love and support has been by my side every step of the way.

Contents

Chapter 1

Introduction

1.1 Thesis Overview

The *Large Hadron Collider* (LHC) at CERN is one of the most advanced scientific instruments ever constructed, which provides the opportunity to explore fundamental questions in particle physics. The data rate produced by the proton-proton collisions during its operation, require state-of-the-art technology and robust processing system designs that select and store the information produced by the LHC experiments. This thesis addresses several aspects of these challenges. It focuses on the development, optimization of FPGA based *Trigger and Data Acquisition* (TDAQ) systems for the Phase II upgrade of the ATLAS experiment for the Run 4 phase of the LHC operations. The goal of this work is to bridge multiple projects on different parts of the ATLAS trigger, each contributing to the goal of enabling efficient and reliable data handling under the demanding conditions of high-energy physics experiments.

The central problem addressed in this thesis is how to design and implement scalable, efficient systems capable of processing and filtering the vast amounts of data generated by the ATLAS detector. The implementations need to comply to the requirements of low latency while maintaining high data throughput. The projects that are presented include firmware development for FPGAs, and performance testing. While each project tackles a distinct technical problem, they collectively contribute to a unified objective: optimizing the TDAQ system to meet the requirements of Run 4 and beyond.

The thesis begins by outlining the design and architecture of the TDAQ system, with an overview of the technical advancements described in subsequent chapters. It focuses on the development and testing of firmware for FPGA-based systems, which

are critical for handling real-time data processing tasks. This includes the design and verification of modules responsible for data buffering, data encoding, sorting, and hardware verification, with an emphasis on achieving low-latency performance. Subsequently, it discusses the role of heterogeneous computing systems, leveraging both CPUs and FPGA accelerators, showcasing how these technologies can be used in the overall system.

A significant part of the thesis includes simulation studies to verify the functionality of the applications, as well as hardware tests to evaluate their performance. The test results are analyzed, highlighting the efficiency, scalability, and robustness of the developed systems. The results show how these systems are efficient, scalable, and reliable, supporting the larger goals of the ATLAS experiment by improving its ability to analyze data and demonstrate how the upgrades can contribute to advancing the physics program.

1.2 Personal contributions

The work presented in this thesis is the result of contributions from numerous collaborators focusing on research and development on custom and of the shelf FPGA applications for the ATLAS trigger. While the projects presented in this thesis are built on a collective effort, the author's individual contributions are summarized below:

Firmware and Readout Studies of the CTP

The author contributed to the development and validation of the *Central Trigger Processor* (CTP) readout firmware. This work included investigating the feasibility of the design to ensure that the memory buffers could handle the required data rates efficiently. Rigorous testing was conducted to validate the performance and robustness of the readout system, establishing its viability for use in production environments. These contributions are detailed in chapter 5.3.

PRM Firmware Development and Validation

The author played a key role in the firmware development for the Pattern Recognition Module (PRM). Specifically, the author designed and implemented critical components, including the Data Generator, Output Comparator, and SSID Encoder. The

development was followed by validation tests on hardware platforms to confirm functionality and performance under realistic conditions, ensuring the firmware's readiness for deployment. These contributions are detailed in chapters 6.2 and 6.3.

Data Transfer and Accelerator Development

The work described in Chapter 7.2 highlights the author's contributions to data transfer tests, which involved configuring data center setups using off-the-shelf FPGA accelerators. These tests were instrumental in evaluating the feasibility and efficiency of data transfer mechanisms under different configurations.

Development of Vitis Accelerator Backend for HLS4ML

A significant contribution was made in developing the Vitis accelerator backend for the *High Level Synthesis for Machine Learning* (HLS4ML) framework. This involved designing and optimizing the backend to enable efficient deployment of machine learning workloads, leveraging FPGA technology to accelerate inference tasks. This work is presented in Chapter 8.2 and 8.3.

Chapter 2

The ATLAS experiment at LHC

2.1 The High-Luminosity Large Hadron Collider

2.2 The Large Hadron Collider

The *Large Hadron Collider* (LHC) [1] is the world's largest and highest energy particle accelerator. Its tunnel was constructed by the *European Organization of Nuclear Research* (CERN) approximately 100 meters under the Swiss - French border near Geneva. It comprises a 27 kilometer ring of superconducting magnets and accelerating structures designed to boost two proton or heavy ion beams to high energy and directing them in opposite directions before colliding them in the center of the four main experiments of the LHC. The proton bunches are spaced 25 ns apart, resulting in collisions every 25 ns. The objective of the LHC experiments is the study of known phenomena in detail, as well as to search for phenomena that can lead to physics beyond the Standard Model.

The Standard Model

The Standard Model [2] of particle physics is a well established physics theory that describes how the elementary particles and three out of the four fundamental forces are related to each other. The building blocks of all the matter are elementary particles that are subject to the four fundamental forces.

The fundamental particles are divided into two main categories, the quarks and the leptons. There are six quarks and six leptons that are further subdivided into three groups, the generations. The first generation includes the lightest and most stable

particles, while the third generation consists of the heavier and less stable particles. The “up” quark and the “down” quark are first generation particles and comprise part of the stable matter in the universe. The “charm” and “strange” quarks belong to the second generation, while the “top” and “bottom” quarks third generation particles. The leptons are also divided into three generations. The electron and the electron neutrino are first generation particles, the muon and the muon neutrino are second generation particles, and the tau and the tau neutrinos are third generation particles. Unlike the electrons, the muons and the taus, the neutrinos have a very small mass and they are electrically neutral.

The Standard Model describes three fundamental forces. The strong force and the weak force have a short range, and they are effective at a subatomic level. In contrast, the gravitational and the electromagnetic forces have infinite range. The carriers of three out of the four fundamental forces are particles called bosons. The boson that carries the strong force is the gluon, the boson that carries the electromagnetic force is the photon, and the bosons carrying the weak force are the “W” and “Z” bosons. Although a boson for the gravitational force has not been measured, it is expected that there is a boson that carries the force, the graviton.

As of 2012 there was no observation of the last particle that is part of the Standard Model, which is the Higgs boson. The Higgs boson is an elementary particle in the Standard Model of particle physics, and it is a manifestation of the Higgs field. It was first predicted by physicist Peter Higgs and others in the 1960s. The LHC experiments confirmed in 2012 its existence [3]. The discovery of the Higgs boson was a landmark achievement because it confirmed the existence of the Higgs field, which plays a crucial role in explaining why particles have mass.

The Standard Model is a well established theory, however, there are phenomena that it fails to describe. As already mentioned, it includes only three out of the four fundamental forces. A few other weaknesses of the Standard Model is that there is no description of the dark matter, and it fails to explain what happened to the antimatter after the big bang.

The mission of the LHC experiments

The mission of the LHC experiments is to study the subatomic world to understand in greater detail what is already established and describe the phenomena that are yet to be explained. The existence of Supersymmetry [4], the nature of dark matter, the

imbalance between matter and antimatter, and the properties of the Higgs field, which is crucial to understanding how particles acquire mass, are a few of the questions that the LHC experiments are trying to answer.

The accelerated beams reach nearly the speed of light and collide in the middle of the two main LHC experiments: ATLAS and CMS. These collisions achieve energies of 13.6 TeV that recreate the conditions that resemble those that existed just after the Big Bang. This allows scientists to study phenomena that occur at extremely high energies, providing insight into the fundamental particles and forces described by the Standard Model and beyond.

The cross-section measurements, which expresses the likelihood of different processes to happen in proton-proton collisions, is one of the main objectives of the studies done at the LHC experiments. The Standard Model cross-section plot in figure 2.1 summarizes the probabilities for producing particles like W and Z bosons, Higgs bosons, top quarks, photons and jets. The cross-section plot shows that some production processes are more common than others. Comparing these measurements with theoretical prediction helps to confirm or challenge the Standard Model.

The experiments also study the extreme environments created in these collisions. For example, ALICE examines the quark-gluon plasma, a state of matter thought to have existed shortly after the Big Bang, while the LCHb experiment specializes in the research of the slight differences between matter and antimatter by studying the behavior of b quarks [5]. This helps scientists understand how the universe evolved [6].

2.3 High-Luminosity upgrade

The LHC will undergo an upgrade to increase its luminosity by a factor of five beyond its initial design, resulting in the increase of the integrated luminosity by a factor of ten. By doing so, the experiments will enlarge their data sample by one order of magnitude compared to the LHC program [8]. The upcoming *High Luminosity LHC* (LH-LHC) will introduce a novel machine setup, incorporating innovative technologies, each presenting remarkable technological challenges. As a consequence of the increment of the data produced by the HL-LHC, the ATLAS experiment will upgrade its detectors by introducing upgraded instrumentation and technologies to handle the challenging radiation levels as well as higher data rates [9].

Fig.2.2 shows the LHC and HL-LHC project timeline. Following the *first Long*

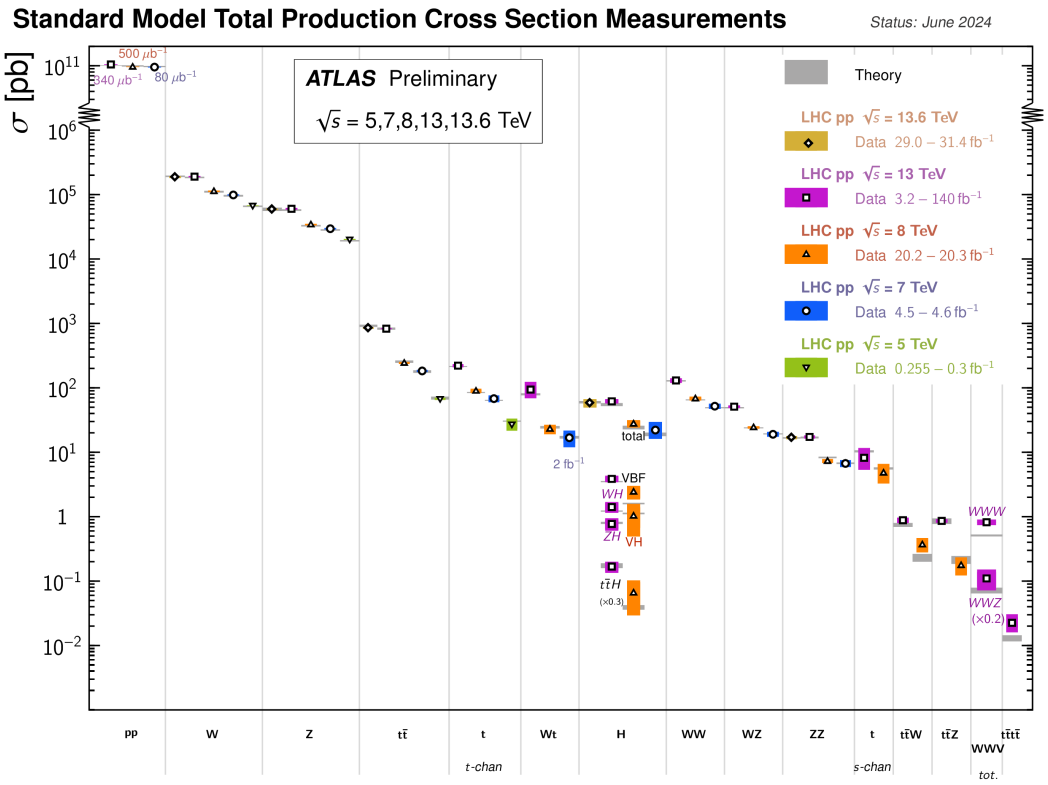


Figure 2.1: Standard Model cross-section measurements plot [7]

Shutdown (LS1) (Phase 0) the LHC operated at a center-of-mass energy of $\sqrt{s}$ = 13 TeV throughout its second run (Run-2). By the second half of 2017, it boasted an instantaneous luminosity exceeding L = 2.0 x 10^{34} cm^{-2} s^{-1}, along with an average pile-up of approximately 38 minimum bias collisions per bunch crossing. By the conclusion of Run-2, the LHC had provided ATLAS and CMS with a total integrated luminosity of 156 fb^{-1}.

In 2019 after the second period of operation that lasted more than three years started the *second long shutdown* (LS2) introducing the Phase I upgrade. LS2 introduced the LHC *Injectors Upgrade* (LIU) for the *third run* (Run-3) and to prepare the injectors for providing beams to the HL-LHC. In addition to the LIU, full maintenance of the LHC equipment was performed, consolidating part of the machine while ATLAS was upgraded such that it can record up to approximately 300 fb^{-1}. Run-3 was scheduled to follow the LS2 in the spring of 2021. However, the COVID-19 pandemic forced the rescheduling of the start of Run-3 for July 2022.

The upgrades for the HL-LHC, denoted as "Phase II upgrade" will be implemented during the *third long shutdown* (LS3). During the HL-LHC era the luminosity

will reach 3000 fb^{-1} thus achieving the full potential utilization of the LHC and its experiments. The operation of HL-LHC is scheduled to start in 2030.

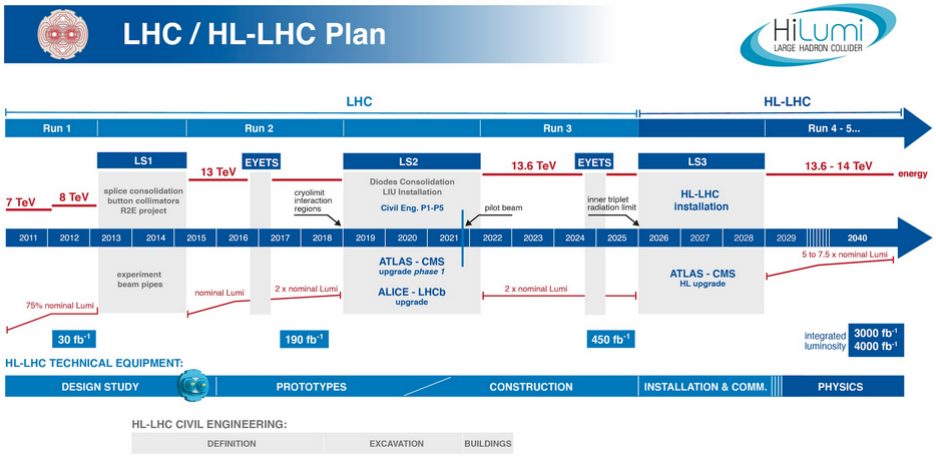


Figure 2.2: LHC and HL-LHC timeline [10]

2.4 The ATLAS detector

The ATLAS experiment studies the elementary particles that constitute our universe and the fundamental forces between them. The experiment has a rich physics program. It includes precision measurements of the standard model and searches for new particles. Over 3000 scientists from 38 countries are involved. The ATLAS experiment is one of the largest scientific projects in the history of mankind.

The ATLAS experiment is a general-purpose detector. The detector was designed to meet the high multiplicity and high rate of bunch crossing required by the LHC. The ATLAS layout uses the traditional onion-shell structure seen in collider experiments. It is about 45 meters long, over 25 meters high, and weighs around 7000 tons. The central region of the ATLAS detector is called the barrel due to its shape. Additional detectors are placed on both sides to provide end-cap coverage. This ensures that all directions around the interaction point are covered as much as possible. A toroidal magnetic system is part of the detector, designed to bend the trajectory of charged particles to measure their momentum. Additionally, the inner part of the detector includes a solenoid magnet, which provides a uniform magnetic field for the inner tracking system to precisely measure particle tracks.

The detector is divided into five main sub-detectors. These are the inner detector (ID), the electromagnetic liquid argon calorimeter (LAr), the hadronic calorimeter

(Tile), the muon spectrometer (MS), and the forward detectors (Lucid and BCM). Each sub-detector has a different task. The ID provides high-resolution tracking and vertex reconstruction to help identify long-lived particles. The calorimeter system measures the energy of electrons, photons, and jets. It also identifies photons, electrons, muons, and taus. The calorimeters provide near-hermetic coverage, allowing for good measurement of total and missing energy. The MS, the most external part of the ATLAS detector, triggers, identifies, and measures the momentum of muons. The forward detectors measure luminosity.

For each bunch crossing, all subsystems measure the outcome of the proton-proton collision event. The data acquisition system handles the data flow at an input rate of 40 MHz. The data must be moved reliably among the multi-level trigger system. This system decides which events to store for physics analysis and which to reject and lose forever, a process detailed further in Chapter 3. This selection procedure is called triggering. The trigger is necessary due to the extremely high volume of data and the limited offline analysis capabilities. These factors create a downstream bottleneck on data output. The trigger and data acquisition (TDAQ) system, whose implementation and challenges are discussed in Chapter 3, reduces the bunch crossing rate from 40 MHz to about 1 kHz. With an average event size of 1.7 MB, these rates correspond to a data bandwidth reduction from about 68 TB/s to about 1.5 GB/s.

Coordinate system

The coordinate system is defined by the x, y and z axes. The x-axis extends from the interaction point toward the ring's center, the y-axis points vertically upward, and the z-axis aligns with the beam's direction.

The angle ϕ measures the azimuthal angle around the beam axis and ranges from $-\pi$ to π. The polar angle θ is the angle relative to the positive z-axis, and can take values within the interval [0, π] and is typically expressed in terms of pseudorapidity, given by $\eta = log(tan(\pi/2))$. Pseudorapidity is particularly useful in collider experiments since the particle density from a collision remains roughly constant per unit of η across different values of η and $\Delta\eta$ remains invariant under boosts along the z-axis.

The angular separation between two objects is quantified by ΔR and is defined as the square root of the sum of the squares of the differences in azimuthal angle and pseudorapidity $\Delta R = \sqrt{(\Delta\phi^2 + \Delta\eta^2)}$. The momentum in the plane perpendicular

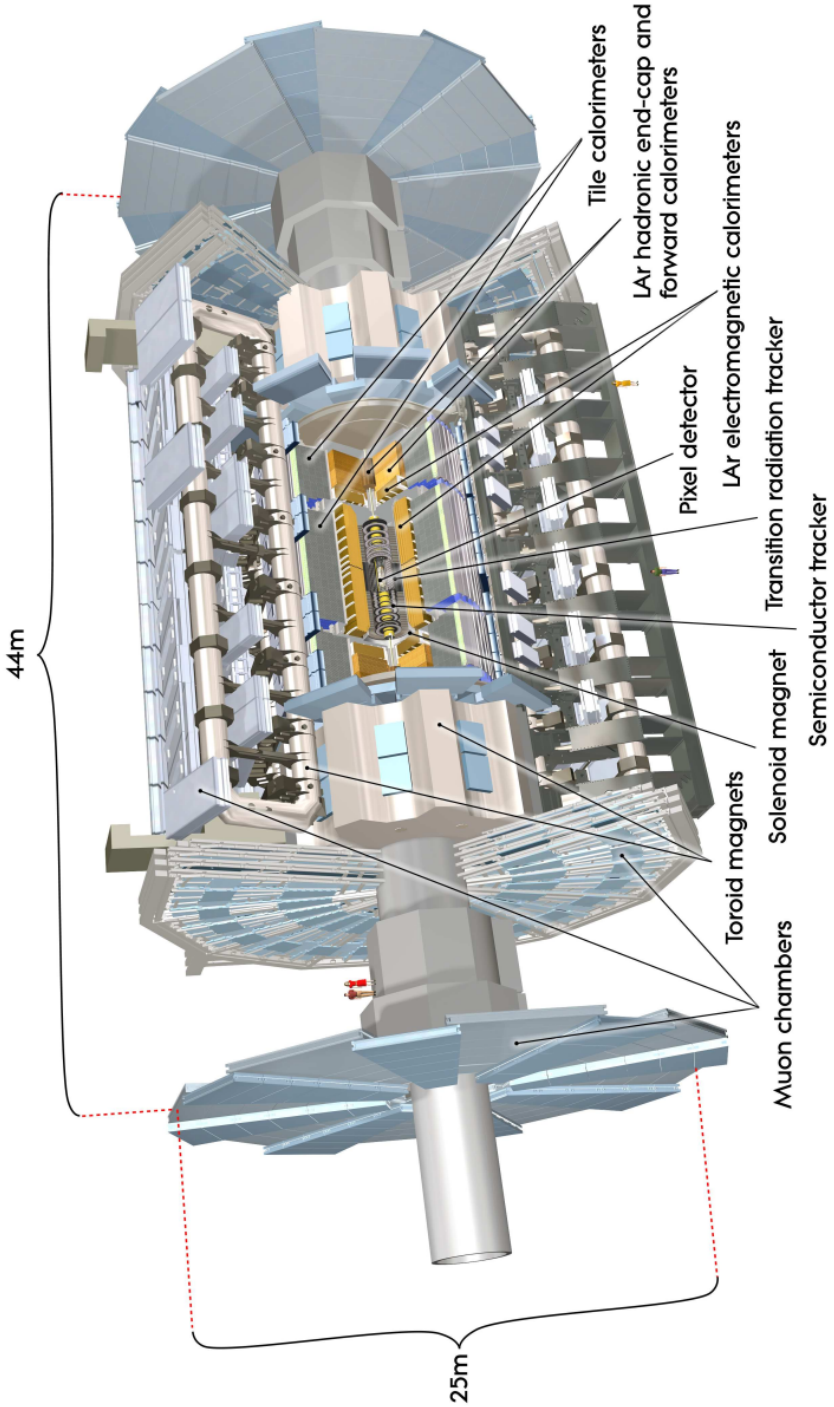


Figure 2.3: The ATLAS detector schematic view [11]

to the beam axis is referred to as transverse momentum and is calculated as $p_T = p\sin(\theta)$.

The magnet system

The curvature of a charged particle's path as it passes through the detector is used to calculate its momentum. The tracking detectors are positioned in a magnetic field that bends the trajectories of charged particles in order to do this. The magnet system of the ATLAS detector is made up of four sizable superconducting magnets. Figure 2.4 shows a schematic view of the ATLAS magnet system.

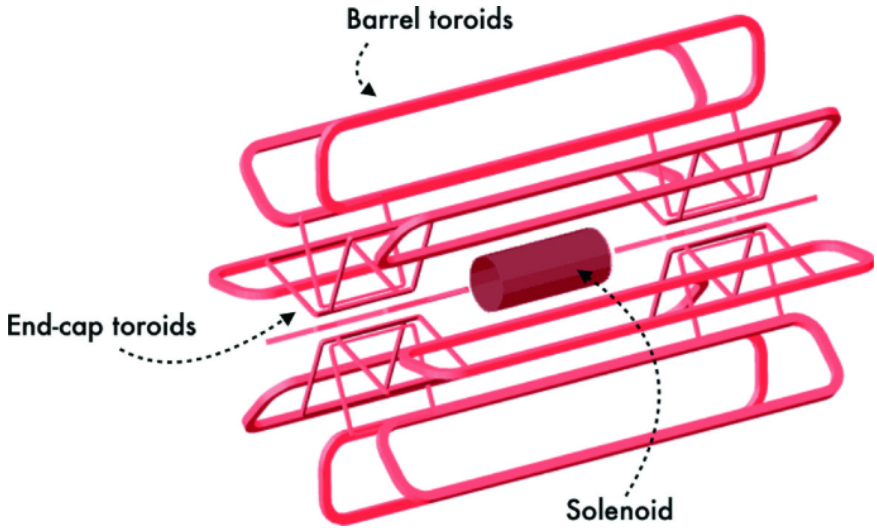


Figure 2.4: The ATLAS magnet system

A solenoid, a barrel toroid, and two end-cap toroids make up this magnet system. The solenoid produces a 2 T axial magnetic field oriented parallel to the beam axis. It is 5.3 meters long and 2.4 meters in diameter, and it encloses the ID cavity. A magnetic field of 4 T is produced by the two end-cap toroids and the barrel toroid. The end-cap toroids have an axial length of 5 meters and an outer diameter of 10 meters, whereas the barrel toroid is 25 meters long and 20 meters wide.

Inner detector

The Inner Detector (ID) is the part of the detector responsible for tracking charged particles and pinpoint the locations of primary and secondary vertices. It comprises three distinct sub-detectors. The innermost part of the ID is the pixel detector. This

sub-detector is engineered to reconstruct vertex positions. The middle layer is the semiconductor tracker (SCT). It delivers high-resolution points close to the interaction region. The final component of the ID is the transition radiation tracker (TRT). The TRT ensures high transverse momentum p_T resolution for tracks with significant p_T. Collectively, the ID covers a pseudorapidity range of $|\eta| < 2.5$.

The pixel detector includes three layers in the barrel and three disks in each end-cap. To achieve the necessary granularity, each pixel measures 50×400 μm^2, and contributes to a total active area of about 1.7 m². These pixel sensors are mounted on staves, which are positioned within a carbon fiber frame using five mounting points. As accurate alignment of the pixel sensors is crucial for achieving the required vertex location precision the entire assembly is placed within the SCT with an accuracy of 100 μm.

During the first long shutdown of the LHC from 2013 to 2015, an additional layer known as the insertable B-layer (IBL) was installed at a radius of 3.3 cm between the beamline and the ID. This cylindrical layer, measuring 3.5 m in total length, contains an active sensitive area optimized for detecting particles in key regions. The smaller pixel size, 25×100 μm^2 for the L0 barrel and 50×50 μm^2 for the L0 rings, enhances the precision and resolution of particle tracking, contributing to improved efficiency in secondary vertex reconstruction, which is vital for identifying jets originating from b-quarks and other critical processes. Furthermore, the IBL improves overall tracking by providing an extra measurement point.

At larger radii, pixel sensors are no longer needed to maintain the required granularity, so SCT strip sensors are employed. There are eight strip layers that are crossed by each track. In both the barrel and endcap regions, the detector uses stereo strips angled at ± 20 mrad to measure both coordinates. This configuration ensures precise tracking and measurement of particle trajectories, with one set of strips in each layer parallel to the beam direction, measuring $R - \phi$. In the end cap region there is a set of strips running radially, as well as, a set of stereo strips at an angle of 40 mrad. Each SCT strip is 12 cm in length with an average pitch of 80 μm.

The TRT is made up of drift tubes with a diameter of 4 mm. These drift tubes are called straws. The straws are 144 cm long in the barrel and 37 cm long in the end-cap. Transition radiation fibers, which generate photons when an electron passes through, are positioned between the straws. To detect these transition radiation photons efficiently, the tubes are filled with a gas mixture that includes Xenon, which has a high photon absorption efficiency. Their stability is ensured by carbon fibers,

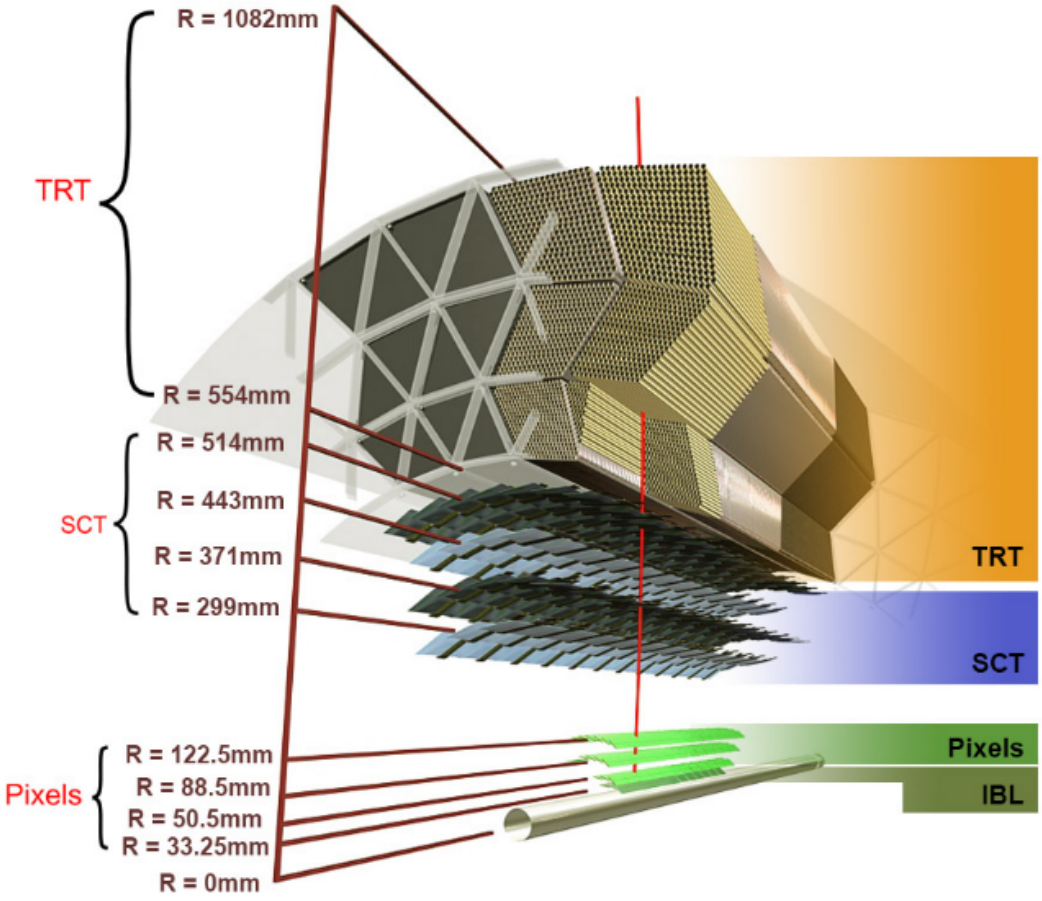


Figure 2.5: ATLAS Inner Detector [11]

and their anodes are made of gold-plated tungsten wires. The tubes are filled with a gas mixture, and a high voltage between the tube and wire. The high voltage causes an avalanche when an ionizing particle traverses the tube, allowing the signal to be collected and read out.

The TRT has two primary functions. Firstly, it measures the momentum of charged particles, and due to its large radius, it provides the most precise measurements among the ID sub-detectors. To achieve this, the particle's position is only measured in the bending plane and must be combined with the 3D measurements from the pixel and SCT detectors. The second function of the TRT is the discrimination between electrons and charged pions. This is achieved by using the transition radiation emitted by electrons as they pass through the TRT. Overall, the TRT provides coverage for the region of $|\eta| < 2$.

The new Inner Tracker for Phase II

The requirement of the current inner detector was to operate for 10 years, at a constant instantaneous luminosity of $\mathscr{L} = 1.0 \times 10^{34} cm^{-2} s-1$, 14 TeV center-of mass energy, 25 ns between beam crossings and an average pile-up of 23 proton-proton interactions per crossing. Furthermore, the front-end electronics were designed for a single hardware trigger signal maxing at a rate of 100 kHz.

During the HL-LHC era, an average of 200 proton-proton interactions per beam crossing is expected for the ultimate leveled luminosity. The new tracker will have to adapt to these new requirements with an extended role for tracking information anticipated in the final design. Some of the limitations of the existing ID that make the upgrade necessary are shown below:

Radiation Damage: The existing Pixel detector has radiation-hard electronics that are able to handle luminosity of 400 fb^{-1} while the SCT was designed to withstand integrated luminosity of 700 fb^{-1}. These specifications are not expected to be sufficient for operation for the HL-LHC due to radiation damage. Furthermore, the intrinsic hit efficiencies drop below the limits required for efficient pattern recognition and leakage currents from the detectors will surpass the limits of the power supplies.

Bandwidth Saturation: Limitations in the buffering of the links between the on-detector pixel electronics and the readout driver will result to inefficiencies when 0.2 to 0.4 hits per double column per 25 ns bunch crossing is exceeded. This limitation is not uniform across all layers, as the innermost layers, which experience the highest hit rates due to their proximity to the interaction point, are most affected. In addition, the same limitation applies in the SCT at about the same value of instantaneous luminosity.

Limitations from Detector Occupancy: The increase of the proton-proton interactions will result to additional hits. In order to keep the efficiency of the pattern recognition and the track finding, it is required to increase the detector's granularity.

The new inner detector architecture integrates precise central tracking despite encountering an average of 200 pile-up events, while also extending tracking capabilities to a pseudorapidity of 4, which is critical for reducing pile-up effects in forward physics analyses and maintaining reliable performance. The ITk system consists of two main components: the Strip and the Pixel Detector. The layout is such that the Strip Detector encircles the Pixel Detector. The Strip Detector is made up of four barrel layers and six end-cap disks with a petal design, both featuring double modules with a small stereo angle to enhance z(R) resolution in the barrel and end-caps,

respectively. This Strip Detector covers a pseudorapidity range of $|\eta| < 2.7$ and is supplemented by a 5-layer Pixel Detector that extends the coverage to $|\eta| < 4$. The Pixel and Strip Detectors are separated by a Pixel Support Tube (PST). Given the intense radiation environment anticipated in the HL-LHC, the two innermost layers of the Pixel Detector are designed to be replaceable. These inner pixel layers are separated from the outer three layers by an Inner Support Tube (IST), which allows for their replacement. Together, the Strip and Pixel Detectors provide a total of 13 hits within $|\eta|$ <2.7, except in the barrel/end-cap transition area of the Strip Detector, where the hit count is reduced to 11. The Pixel Detector is engineered to ensure a minimum of 13 hits from the end of the strip coverage up to a pseudorapidity of $|\eta| = 4$ [14].

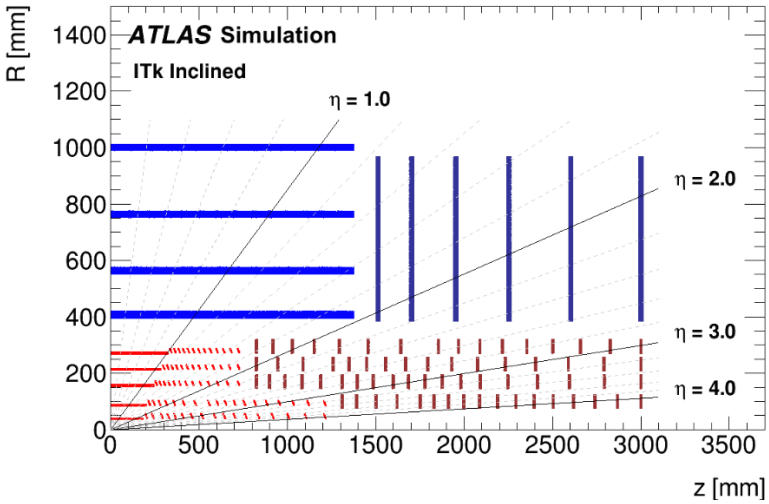


Figure 2.6: Layout of the ITk for the HL-LHC phase of ATLAS [14]

Calorimeter system

The ATLAS calorimeter system is depicted in Fig. 2.7. It is composed by both electromagnetic and hadronic calorimeters. It is designed to deliver precise energy measurements for photons, electrons, jets, and missing transverse energy (E_T^{miss}). It provides complete ϕ coverage around the beam axis within the pseudorapidity range $|\eta| < 4.9$. Additionally, the calorimeter data, in combination with information from other detector systems, is utilized to differentiate between electrons and photons up to $|\eta| < 2.5$, a limit set by the Inner Detector (ID).

The Electromagnetic Calorimeter

The electromagnetic (EM) calorimeter employs a lead-liquid Argon (LAr) configuration. Its design includes lead absorber plates that are arranged with kapton electrodes in an accordion pattern to ensure full and uniform azimuthal coverage. LAr functions serves as active medium. The barrel section of the EM calorimeter covers $|\eta| < 1.475$, and the end-cap section spans $1.375 < |\eta| < 3.2$. The barrel is divided into two identical halves separated by a 4 mm gap. The end-cap sections are housed in shared cryostats. The EM calorimeter's readout is segmented into separate towers in both η and ϕ. Each tower is further divided into three layers along the radius (R). The innermost layer operates with the finest granularity to guarantee the accurate detection of the start of the shower. The overall thickness of the EM calorimeter is at least 22–24 times the mean radiation length (X_0).

The Hadronic Calorimeter

The *Hadronic End-cap Calorimeter* (HEC) is a sampling calorimeter that uses liquid Argon and copper plates. It covers the pseudorapidity ranging $1.5 < |\eta| < 3.2$ and is composed of two wheels, where each wheel is divided longitudinally into two layers. The cells within the HEC have dimensions of $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ in the inner layers and 0.2×0.2 in the outer layers.

The hadronic barrel calorimeter, known as TileCAL, is constructed with steel absorber plates interspersed with scintillator plates that act as the active material. TileCAL features three radial layers with increasing coarseness in granularity as the radius increases, and it comprises 64 modules along the azimuthal direction. The total thickness of this detector is equivalent to 9.7 times the nuclear interaction length. TileCAL coverage in the barrel region is $|\eta| < 1$ while in the extended barrel region it ranges $0.8 < |\eta| < 1.7$.

The Forward Calorimeter

The *Forward Calorimeter* (FCAL) is located within the same cryostat as the end-cap LAr calorimeters. It is essential for providing the necessary E_T^{miss} measurements by extending coverage to $|\eta| < 4.9$. It is positioned approximately 1.2 meters behind the other end-cap calorimeters, and it is designed to reduce neutron background. The FCAL is divided into three sections. All three sections use LAr as the active material. The first section uses copper as the absorber for electromagnetic measurements,

while the other two sections employ tungsten as the absorber, to form the hadronic part of the FCAL. To prevent ion buildup in the active regions of this high-radiation environment, the FCAL is designed in a way that allows very narrow gaps that can be constrained down to 0.25 mm.

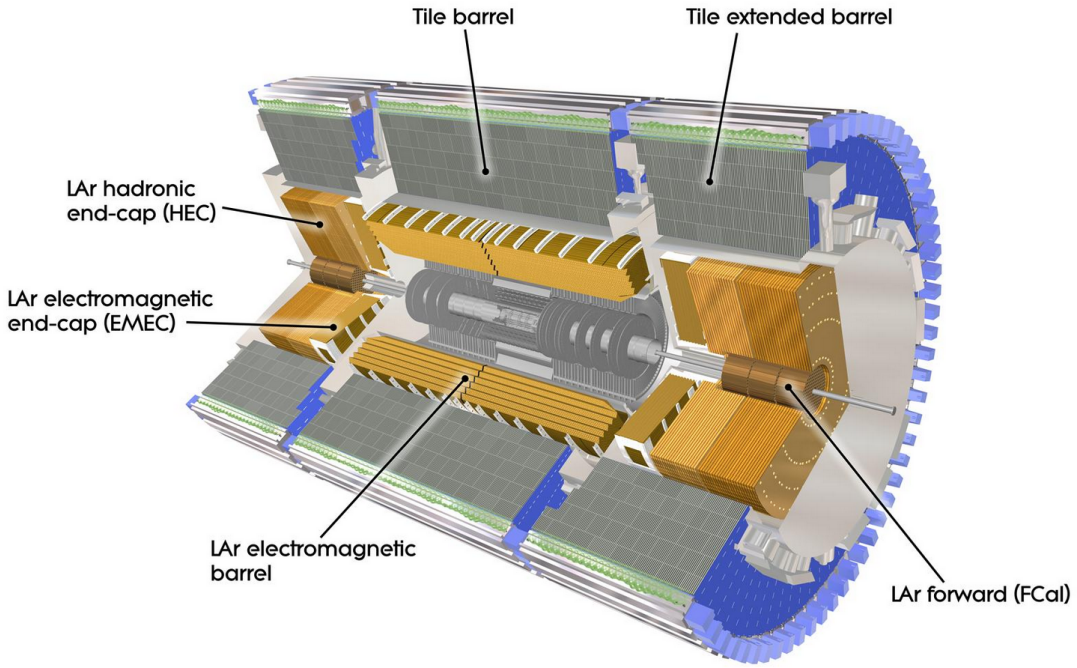


Figure 2.7: ATLAS Calorimeter [16]

The Phase II calorimeter detectors

For Run 4 while the detector components of the LAr calorimeter will remain unchanged, the readout will be upgraded with optimized supercells. The supercells were introduced during Run 3. They combine signals from multiple calorimeter cells to improve granularity and resolution for trigger purposes. In Run 4 they will provide full granularity data to the trigger system, making use of a new pre-processor, the *LAr Signal Processor* (LASP). An on-detector digitization will be used for the readout, while a new backend pre-processor will be introduced to increase the bandwidth and the buffering capabilities and satisfy the Phase II requirements.

Muon spectrometer

The outermost layers of ATLAS are constructed with the Muon Spectrometer (MS), as depicted in Fig. 2.8. This spectrometer employs four distinct technologies. Mon-

itored drift tubes (MDTs) are responsible for precision measurements in the bending plane (η direction) for both the barrel and most of the end-cap region, spanning 0 $< |\eta| <$ 2.7.

In the innermost sections of the end-cap, the MDTs would be overwhelmed by the high radiation levels and occupancy. Hence, small-strip Thin Gap Chambers (sTGCs) take over to ensure precise tracking in these regions. Thin gap chambers (TGCs) and resistive plate chambers (RPCs) are used for generating fast signals that assist with triggering and identifying bunch crossings. They also measure the second coordinate, perpendicular to the bending direction.

Large air-core toroidal magnets create the magnetic field in the barrel ($|\eta| < 1.4$). The end-cap regions ($1.6 < |\eta| < 2.7$) feature end-cap toroidal magnets, while the transition zone ($1.4 < |\eta| < 1.6$) uses a combination of the barrel and end-cap toroidal fields to provide sufficient bending power for the muons.

The muon deflection in the magnetic field is detected through hits in three layers of precision drift tube chambers for $|\eta| < 2$. In the region of $2.0 < |\eta| <$2.7, two layers of MDTs work in tandem with one layer of CSCs. Three layers of RPCs in the barrel region ($|\eta| < 1.05$) and three layers of TGCs in the end-cap regions (1.05 $< |\eta| <$ 2.4) are responsible for the Level-1 muon trigger.

Geometric coverage in the barrel is somewhat limited due to an opening around η = 0, allowing space for the Inner Detector (ID) services, calorimeters, the structural supports of ATLAS, and two small elevators located at the bottom of the spectrometer.

MDT tubes are made of aluminum and filled with Ar/CO2 gas, which becomes ionized as muons pass through. The electrons from this ionization drift toward a 50 μm thick tungsten/rhenium wire. A high voltage between the wire and the tube wall induces an avalanche near the wire, amplifying the signal.

The MDTs measure the closest approach of the muon track to the wire by calculating the drift time of the electrons. This drift time, which reflects the time taken by ionized electrons to reach the wire, is used to determine the radial distance of the muon from the wire. Since the maximum drift time (700 ns) is significantly longer than the bunch crossing interval, RPC/TGC trigger chambers play a key role in bunch crossing identification. They determine the angular position and time-stamp each bunch crossing.

An MDT chamber (shown in Fig. 2.9) contains two multi-layers separated by a spacer, which provides mounting points for high-voltage supplies, readout electronics, and B-field sensors. Each multi-layer holds three tube layers for the outer and

middle detector shells and four tube layers for the inner shell. The chamber also includes an optical alignment system that tracks deformations due to thermal changes and magnetic fields.

In the forward regions ($2 < |\eta| < 2.7$) of the spectrometer's inner layer, MDTs are replaced by CSCs because the hit rate here would be too high for MDTs to operate safely and effectively. CSCs are multi-wire proportional chambers that are read out via segmented cathodes.

The chambers are angled towards the interaction point to ensure tracks are perpendicular to the chamber plates, as inclined tracks would compromise the spatial resolution. CSCs achieve a resolution of 60 μm per layer in the bending plane and 5 mm in the other plane.

The MDTs and CSCs deliver accurate tracking measurements in the bending direction, which is crucial for precise muon momentum (p_T) determination. RPCs, on the other hand, provide muon triggers based on their p_T in the barrel. RPCs are mounted on MDT chambers in three concentric rings.

There are two inner layers surrounding the MDT middle layer and one outer layer attached to the outside of the large MDT chambers. The Level-1 trigger demands a coincidence in the inner layers for low-p_T and an additional coincidence in the outer layer for high p_T.

Each RPC chamber contains two active gas volumes with a readout made up of two plates with orthogonal strips that measure both coordinates. The gas volume is enclosed by two resistive plates maintained at high voltage and ground potential by two graphite electrodes. An ionizing particle passing through triggers an avalanche in the high-potential gradient.

TGCs are composed of two layers mounted near the CSCs and MDTs' inner layer in the end-cap. TGCs offer finer granularity for track reconstruction, which is essential for the lower bending radius of muons in the end-cap compared to those in the barrel at the same p_T.

TGCs are multi-wire proportional chambers with a small gap (1.4 mm) between the wires and cathodes, which is shorter than the gap between the wires (1.8 mm). The smaller gap results in a shorter drift time, crucial for the trigger system's rapid response. The chambers are positioned at an angle that prevents tracks from passing exactly perpendicular between two wires, where the field gradient is minimal, and drift times would be longer. The wires are read out to provide measurements in the η direction, while the cathode strips measure the other coordinate.

The coincidence trigger requires three out of four layers in the doublets or two out of three layers in the triplet modules. Upon fulfilling this requirement, the data is sent to a coincidence matrix, which searches for patterns corresponding to muon tracks above a certain threshold.

At the end caps of the ATLAS detector there is the *New Small Wheel* (NSW), which replaces the old inner endcap layer of MDT, CSC and TGC. The NSW consists of a set of precision tracking detectors, capable of preforming bunch crossing identification at rates up to 15 kHz/cm^2 with a spacial resolution of less than 100 μm per detector plane. The NSW provides the muon trigger system reconstructed track segments with sufficient resolution such that it can be determined whether or not the muons have originated from the collision point. The NSW consists of both *small-strip Thin Gap Chambers* (sTGC) as the primary trigger and *Micromegas* (MM) as the primary precision tracker [17].

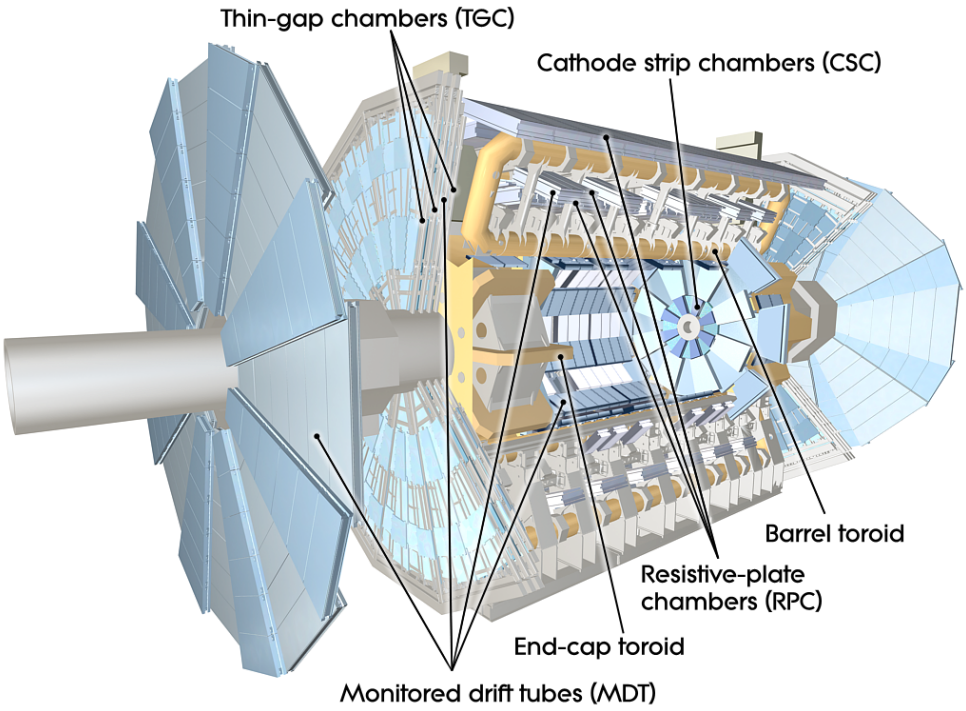


Figure 2.8: ATLAS Muon spectrometer defining the ATLAS overall shape

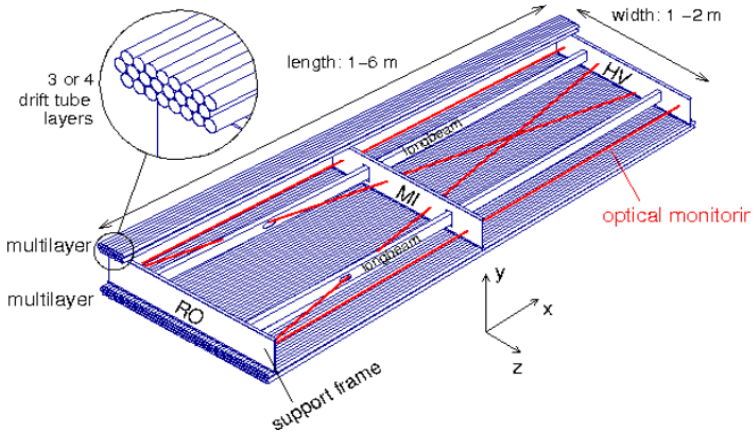


Figure 2.9: MDT chamber

The Phase II Muon spectrometer

The Run 4 muon system will feature a fully covered inner barrel by a layer of RPCs, while the MDT hits will be incorporated in the hardware-level trigger. The RPC and TGC electronics will be redesigned to propagate the buffered information at 40 MHz rate to the L0Muon Trigger Processor.

The NSW is required to increase the readout bandwidth by increasing the number of data collector chips and optical links.

Chapter 3

The ATLAS trigger and data acquisition system

In particle physics, the trigger system is a mechanism that uses criteria to rapidly discriminate data that can be useful for further investigation and data that can be ignored. The events in a particle detector that are deemed interesting are stored to be analyzed offline. Only a small percentage of the total events are recorded. The limiting factor for the limited acceptance is due to real-world limitations in data storage capacity, as well as, data rates.

3.1 Why a trigger system?

The interaction rate at the LHC collision points, R, is shown in equation 3.1, where L is the luminosity and σ the cross-section.

$$R = L \times \sigma \tag{3.1}$$

During Run 3, multiple bunches of protons, each containing a large number of particles, are injected into the beam. These bunches collide at high frequency, leading to a high interaction rate that includes inelastic interactions and pile-up effects within each beam crossing. The specific luminosity and cross-section values determine the overall interaction rate, which reflects the total number of collisions per second.

The data produced by the proton - proton interactions need to be stored. The size of the events that are being stored are approximately 1.5 MB. The data taking sat the

LHC experiments lasts about 10 months. Taking all 4 experiments into account, the data generated by LHC each year can be estimated to around 10^{23} bytes per year.

Storing the vast amount of data generated by the LHC would require an unprecedented and highly distributed infrastructure in a third-party data center that would incur an annual cost exceeding CERN's entire budget. Consequently, CERN operates its own dedicated data center to store the data from all LHC experiments. However, this approach has its limitations, as the data center can store significantly less data compared to data produced by the LHC.

The key to overcoming the storing limitations is the fact that not all the events have information worth analyzing. A Trigger system can introduce a reduction factor on the stored data and reject these events and only store the part of the events worth studying.

3.2 Trigger and data acquisition for Run 3

The current trigger system architecture consists of the Level-1 (L1) trigger and the *High Level Trigger* (HLT). The L1 trigger is built with custom electronics while the HLT system is, for the most part, based on commercially available computers and networking hardware.

The role of the L1 trigger is to reduce the data rate so that the readout is possible, as electronics cannot send out data at 40 MHz. The events that are chosen by the L1 trigger have to satisfy the criteria that allow the search for signatures from high-p_T muons, electrons/photons, jets, and τ-leptons decaying into hadrons. In addition, L1 trigger searches for events with large missing transverse energy and large total transverse energy. To keep up with the data rate, the L1 trigger uses reduced-granularity information from a subset of detectors. The maximum L1 accept rate which the detector readout system can handle is 100 kHz. The L1 decision must be propagated to the front-end electronics within 2.5 μs.

While there are HLT items that read the full detector, there are also some HLT items that take information from both *Regions of Interests* (ROIs). The RoI's are the detector regions where the L1 trigger has detected a possible trigger object within the event. Furthermore, the HLT uses offline-like reconstruction on the events that are fully-built, at this level, to reduce the selected event rate that will be stored for offline analysis. The HLT decreases the event rate down to an average of 3 kHz. The average event processing time is about half a second [20].

The detector data are propagated to the data acquisition system at the L1 trigger rate. The data transmission is performed over point-to-point readout links. The data are then transmitted to the HLT where they are further reduced based on selection criteria and event building is performed. The event data are transmitted to permanent event storage.

L1 trigger system in Run 3

The L1 trigger performs the first event selection utilizing the information from the calorimeters and muon detectors. The goal of the *L1 Calorimeter* Trigger (L1Calo) is to identify high-E_T objects like electrons and photons, jets, and τ-leptons decaying into hadrons. It also aims to identify events with large E_T^{miss} and large total transverse energy.

The L1 muon trigger takes information from RPCs in the barrel and the TGC's in the end-caps. It triggers on patterns of hits consistent with high-p_T muons that originate from the interaction region. Information for each bunch-crossing used in the L1 trigger decision is the multiplicity of muons for each of the p_T thresholds.

In addition to these subsystems, the L1 Topological Trigger (L1Topo) plays a critical role by performing real-time calculations of topological relationships between trigger objects, such as angular separations, invariant masses, and energy sums. These topological criteria enhance the trigger's ability to identify complex event signatures that are indicative of rare or interesting physics processes.

The final decision is made by the *Central Trigger Processor* (CTP). The CTP uses the combined information for different object types.

Calorimeter trigger

The L1Calo is a digital system that processes calorimeter data to identify high-E_T objects such as electrons photons, τ-leptons, and jets, as well as calculate global sums like missing transverse energy (E_{miss}^T) and total transverse energy. For Run 3, the L1Calo trigger was upgraded to perform on-detector digitization of transverse energies from the LAr calorimeter, enabling input data with finer granularity. The calorimeter information is now provided in the form of SuperCells. The SuperCells combine signals from four or eight calorimeter cells. Each trigger tower; a small, predefined section of the calorimeter detector that groups together energy deposits

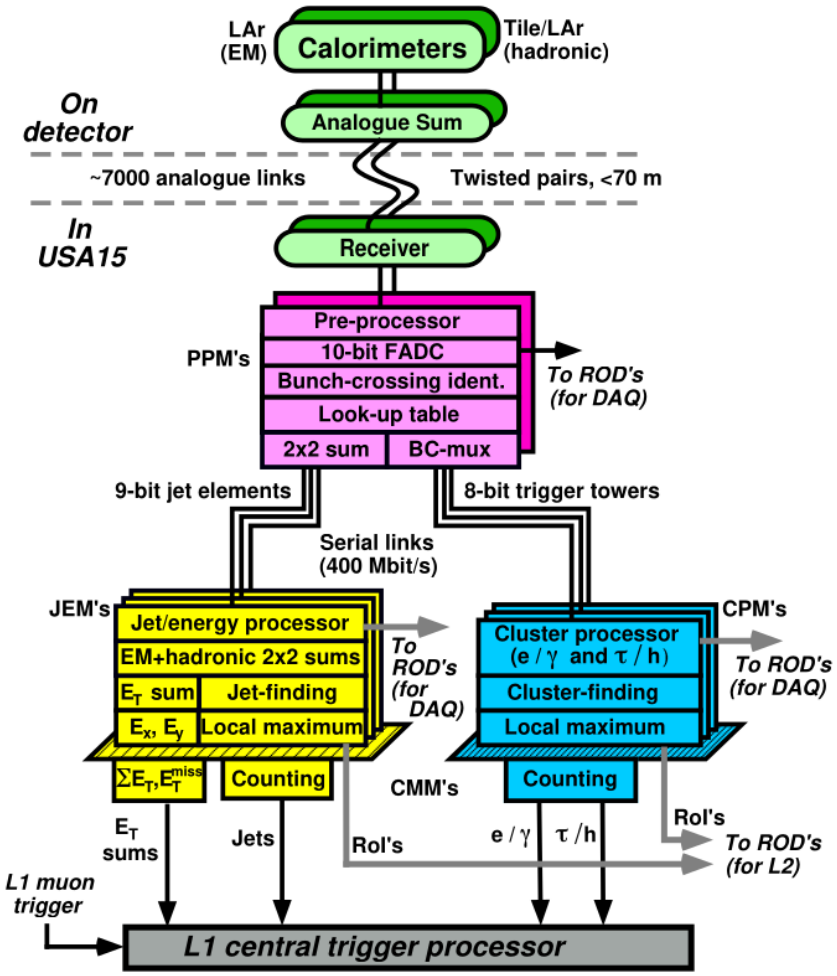


Figure 3.1: Architecture of the L1Calo trigger [18]

(cells) to create a coarser spatial resolution for the L1 trigger system, contains ten SuperCells.

The upgraded L1Calo system has new components to detect electromagnetic signals (eFEX) and jets (jFEX), along with a global detector (gFEX). The eFEX can capture detailed signal information to identify electromagnetic objects and certain types of decaying particles. It also measures shape patterns to help with identification. There are three adjustable levels (loose, medium, and tight) for detecting electromagnetic signals and one for detecting tau particles. Figure 3.1 shows the architecture of the L1Calo trigger.

Muon trigger

The L1 muon trigger uses RPC detectors in the barrel and TGC detectors in the end-caps to identify the bunch crossings that contain a muon candidate. Three trigger stations make the trigger in both the barrel and the end-cap regions. The algorithm applied requires a coincidence of hits in the different trigger stations within a road that tracks the path of a muon from the interaction point through the detector. The coincidence logic allows the concurrent operation with a total of six thresholds. Three of them are associated with the low-p_T and three are associated with the high-p_T trigger. In the Run 3 upgrade, the endcap provides additional bits to increase the granularity and precision of the trigger decision, enabling better identification of muon candidates in high-p_T regions. The trigger signals are propagated to the CTP through the muon-to-CTP interface.

Topological Trigger

The L1Topo system is a component of the Level-1 trigger system in the ATLAS experiment that enhances event selection using topological selections. It was introduced in 2016 during Run 2 to improve background rejection and increase acceptance of physics signal events without raising trigger rates, even as luminosity increased.

The L1Topo receives trigger objects, which contain kinematic and additional qualifying information from the L1Calo system. Subsequently, it applies topological selection. These selections allow from complex event filtering based on spatial correlations and kinematic properties of detected particles. Finally, the L1Topo system transmits its decisions to the CTP.

Central Trigger Processor

The Central Trigger Processor (CTP) processes trigger data from the calorimeter and muon trigger systems, including multiplicities for electrons/photons, *tau*-leptons, jets, and muons, as well as flags indicating which thresholds have been surpassed for total and missing transverse energy, and total jet transverse energy. Additional inputs for specific triggers, such as those based on beam-pickup monitors and scintillation counters, are also received.

This processed information is used to evaluate predefined trigger conditions, such as specific combinations of object multiplicities or energy thresholds, which are con-

figured through look-up tables. For detailed information about the data format and processing logic in the CTP, see Chapter 5 on the Central Trigger Processor.

The CTP then uses look-up tables to create trigger conditions from these input signals. A trigger condition might be, for instance, that the multiplicity of a specific muon threshold exceeds one, meaning at least two muons in the event have passed this threshold. For such an event, this trigger condition would be marked as true. The maximum number of trigger conditions the CTP can handle at once is 512.

These trigger conditions are combined to form up to 512 trigger items, where each trigger condition can contribute to multiple trigger items. An example of a trigger item might be the simultaneous fulfillment of the following conditions: at least two muons exceed a certain threshold, and at least one jet exceeds another threshold. Each trigger item is also assigned a mask and a pre-scaling factor. The L1A signal produced by the CTP is the logical OR of all trigger items.

The dead-time is introduced by the CTP as a means of protection against overflowing the system's memories. There are two types of dead-time: "Simple" dead-time, and "Complex" dead-time. The "Simple" dead-time algorithm allows the generation of a second L1A only after a set time from the generation of the previous one. The "Complex" dead-time is necessary for the BUSY propagation, and it allows a certain amount of L1As within a window of bunch crossings.

The CTP also generates an eight-bit trigger-type word with each L1A signal, which indicates the type of trigger and can be used to select processing options in the front-end electronics and readout chain. Upon receiving each L1A signal, the CTP sends trigger decision information for all trigger items to the high level trigger and the data acquisition system. Additionally, the CTP records the current luminosity block number as part of the readout data. A luminosity block is the shortest interval for which integrated luminosity—corrected for dead-time and pre-scale effects—can be determined. In cases of detector failure, data can be excluded from the last unaffected luminosity block, so the interval should be short to minimize data loss. However, each luminosity block should contain enough data that the uncertainty in luminosity determination is limited by systematic effects rather than statistical ones. For ATLAS, this interval is typically on the order of one minute. The CTP initiates a luminosity block transition by briefly pausing trigger generation, such that there is significant deadtime introduced, incrementing the luminosity block number in a register on the CTP decision module, and resuming trigger generation. This number is then included in the readout data for each event. At each transition, a set of scalers is read from the CTP

and stored in a database, marked with the luminosity block number. These scalers track the number of triggers generated, those surviving the pre-scale veto, which is in place to prevent the prescale mechanism from rejecting rare or interesting events, and those surviving the dead-time veto. These counter values are necessary to later apply corrections to the luminosity value for each luminosity block. For monitoring, the CTP provides bunch-by-bunch scalers of inputs and, when integrated over all bunches, scalers of trigger inputs and trigger items before and after pre-scaling.

Beyond its role in the selection chain, the CTP also serves as the timing master for the detector. The clock signal synchronized to the LHC beams is received by the CTP and is distributed along with the L1A and other timing signals to all subsystems.

High level trigger

Once an event is accepted by the L1 trigger, it undergoes processing by the HLT, which utilizes more detailed calorimeter data, accurate measurements from the muon spectrometer, and tracking details from the ID, information that is not accessible at the L1 stage. The HLT reconstruction process can be conducted within the RoIs identified by L1 or across the entire detector, depending on the need. In either scenario, the necessary data is retrieved on demand from the readout system. To minimize processing time, many HLT triggers employ a two-stage method: a rapid initial reconstruction to filter out most events, followed by a more thorough reconstruction for the remaining ones. The majority of HLT functionalities are executed on standard, rack-mountable, server-class PCs, running CentOS7/Alma9 Linux, and connected via multi-layer gigabit-Ethernet networks—one dedicated to control tasks and the other to data transfer.

For Run 3, the HLT has a dedicated HLT computing farm of approximately 56k cores with hyperthreading, running up to 90k HLT *Multi-Process Processing Units* (HLTMPPU). The processing nodes are equipped with one or more data collection managers or a few HLTMPPU selection applications. An overview of the ATLAS Run 3 TDAQ system is shown in figure 3.2.

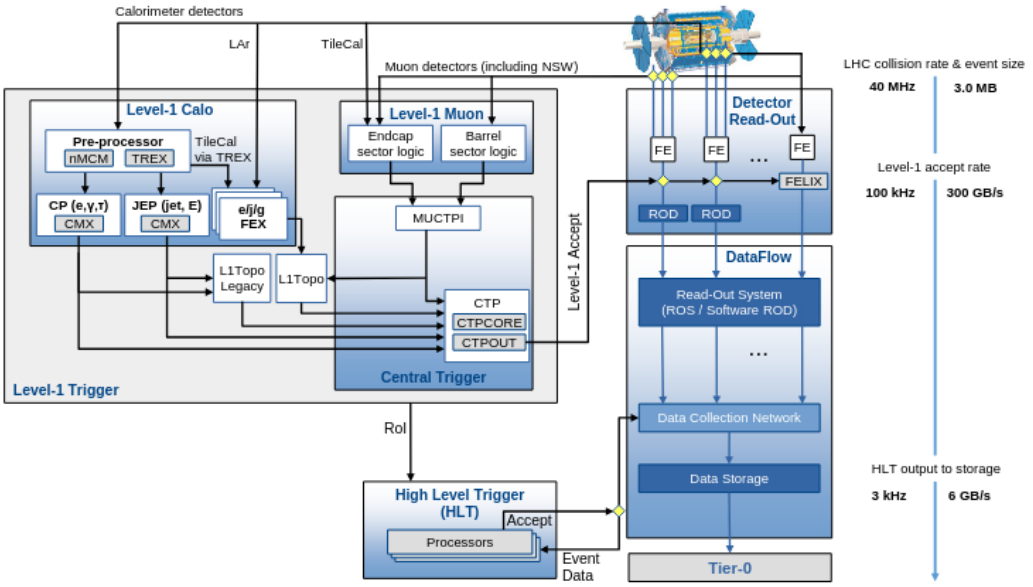


Figure 3.2: ATLAS Run 3 TDAQ system [20].

Trigger menu

The events that are selected by a trigger system are determined by the trigger menu. The trigger menu is a record of criteria. Every time an event satisfies one or more of the criteria defined in the trigger menu, a trigger is asserted. Trigger menus are defined in such a way that the trigger efficiently is reduced enough that the data acquisition system and the storage is not overwhelmed by the rate and the size of the data. At the same time, trigger menus need to allow enough triggers to save data that are useful. This is important as the data from the events that don't assert a trigger, are lost forever.

The LHC experiments' main focus is the discovery and measurement of physics phenomena. Trigger menus are sensitive to a wide range of signatures that correspond to theoretical models that many times have very low signal rates. Additionally, trigger menus need to be conscious of new physics that is not described by the theoretical models.

The triggers in the experiments are primarily driven by general signatures, such as events containing one or more high-p_T electrons, photons, muons, τ-leptons, or jets. These general signatures are further refined using adjustable p_T thresholds that decrease as the object multiplicity increases. This approach ensures efficient coverage

across a wide spectrum of physics programs, balancing the need to capture rare events with the constraints of the trigger bandwidth.

Some of the main points of focus are, Higgs searches, SUSY searches, searches for other physics and studies of Standard Model processes. Table 3.1 shows an example of approximate total rates for physics triggers by signature at luminosity of $1.8 \times 10^{34} cm^{-2} s^{-1}$ and $\sqrt{s} = 13\ TeV$.

Table 3.1: Approximate total rates for physics triggers grouped by signature [20].

Signature	Rate per Stream [Hz]
Electron	270
Photon	120
Muon	290
Tau	160
Missing transverse momentum	140
Unconventional tracking	40
Jet	490
Jet with *b*-hadrons	190
Combined	240

3.3 ATLAS Phase II TDAQ system

In order to keep the data readout rate to existing levels during the HL-LHC operations, the trigger thresholds would have to be increased, resulting in the reduction of the event acceptance rate to a level that would undermine the performance of the HL-LHC. Thus, a new upgraded trigger system is introduced, improving the event selection.

The ATLAS *Trigger and Data Acquisition* (TDAQ) system is made up of the Level-0 Trigger System, the *Data Acquisition System* (DAQ) and the *Event Filter* (EF) System.

The baseline design of the ATLAS TDAQ system in Phase II is shown in figure 3.3. The blocks highlighted with blue are the interfaces between the frond end detector electronics and the TDAQ system. They are grouped into three categories based on the part of the detector that they interface with, the *Inner Tracker* (ITk) that includes the pixel and strip detectors, the calorimeters with liquid argon and Tile calorimeters and the Muon System.

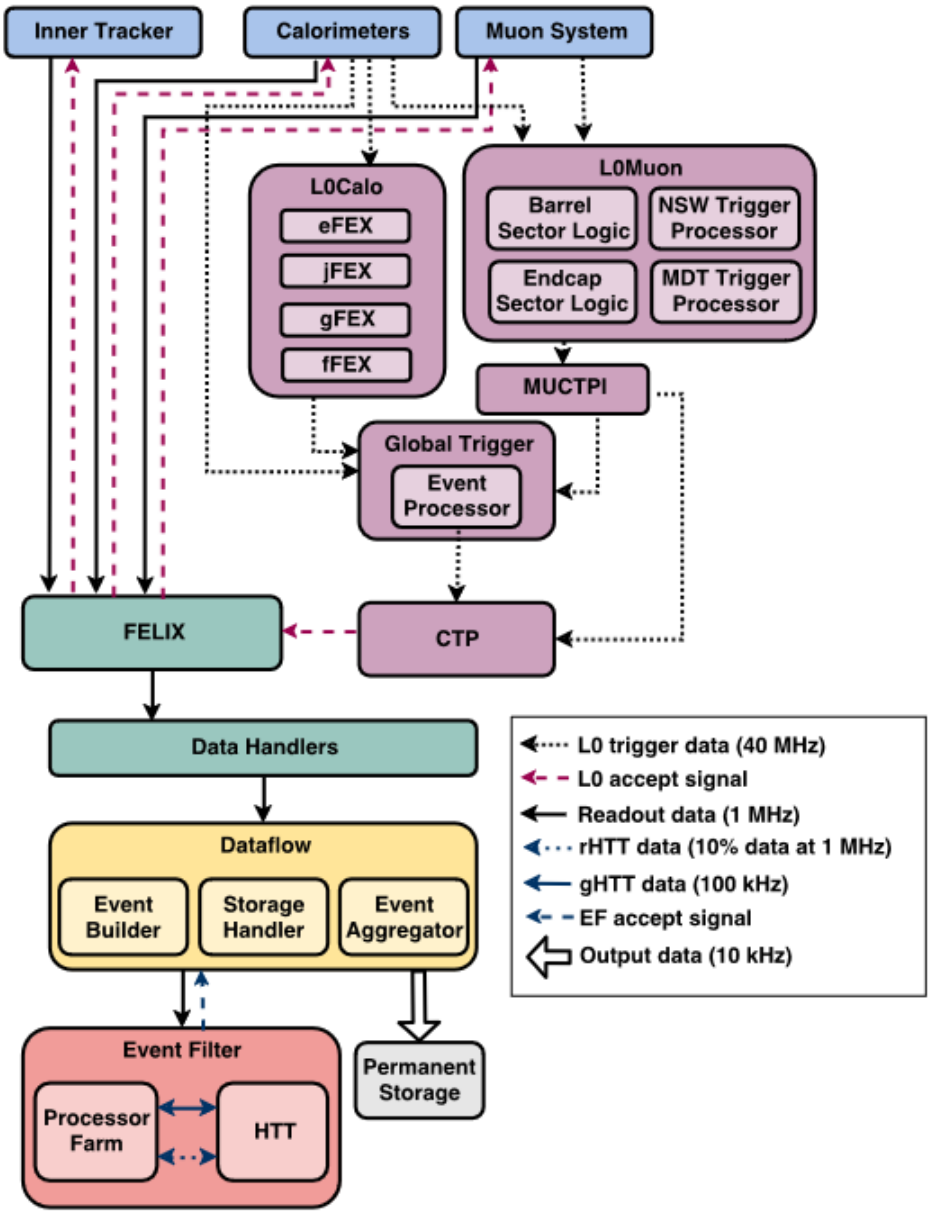


Figure 3.3: Baseline design of the ATLAS TDAQ system in Phase II [19].

A few key trigger and readout parameters in the baseline architecture are given in table 3.2.

Table 3.2: Main trigger and readout parameters in the baseline architecture [1]

Parameter	Phase II value
Clock frequency	40.08 MHz
Level-0 trigger rate	1 MHz
Level-0 latency	10μ*s*
Calorimeter data reception in L0Calo and L0Muon processors	1.7μ*s*
High Granularity Calorimeter data reception in Level-0	1.7μ*s*
Seeding Muon detector data reception in L0Muon	1.7μ*s*
Precision Muon detector data reception in L0Muon	2.8μ*s*
Deadtime	$<0.1\%$

Level-0 Trigger System

The L0Calo, L0Muon, Global Trigger and the Central Trigger subsystems consist the Level-0 Trigger system. The L0Calo and the L0Muon, are the evolutions of the L1Calo and L1Muon respectively that provide the initial event selection and identify features that are investigated to a greater extent further down the TDAQ pipeline. Both systems operate at 40 MHz and the L0Muon trigger information will include MDT, RPC, TGC, and NSW. The L0Calo system utilizes the coarse-granularity data from the calorimeters in order to identify electron, tau lepton and jet candidates as well as calculate E_T^{miss}. The L0Muon identifies muon candidates by receiving data from the muon subsystems that include data from the MDT. It interfaces with the Global Trigger and the CTP through the *Muon CTP Interface* (MUCTPI). The MUCTPI is responsible for the overlap removal, which is the removal of the muon candidates that have been counted twice in the L0Muon system, and calculates multiplicities for various transverse-momentum thresholds. In addition, the MUCTPI merges data from multiple optical fibers into fewer output streams, optimizing the transmission of muon trigger information to the Global Trigger.

The Global Trigger is a newly introduced Level-0 Trigger system. It processes calorimeter data on full granularity, and it is designed to make topological clusters for refined jet algorithm and pile-up subtraction. In addition, it identifies topological signatures that can include a wide variety of four-vector combinations. By processing the information received by the L0Calo and MUCTPI subsystems, the Global Trigger refines the e/gamma, tau, muon and jet selections.

The final block of the L0-Trigger system pipeline is the CTP. The CTP is respon-

sible for taking the final Level-0 decision based on the trigger menu, the prescale factors, and the dead-time requirements. The *Level-0 Accept* (L0A) signal is distributed by the *Local Trigger Interfaces*(LTI), the *Trigger, Timing, and Control systems* (TTC) distribution network, and *Front-End Link eXchange* (FELIX), to the detector systems.

The detector information is propagated via custom point-to-point serial links to the FELIX subsystem. The FELIX system was introduced as a scalable, detector-agnostic solution, designed to facilitate seamless upgrades. Operating as a PC-based gateway, FELIX integrates radiation-tolerant optical links from front-end electronics through PCIe Gen3 cards to a standard switched Ethernet network. This approach allows for a reduction in custom electronics, shifting towards software-based solutions on commercial servers. Positioned as the backbone for the ATLAS Phase-II upgrade, FELIX efficiently manages data from all ATLAS sub-detectors at a data rate of 1 MHz and a total throughput of 4.6 TB/s. Post-Level-0 trigger decision, detector data swiftly traverse custom point-to-point serial links to the FELIX subsystem, marking the initial stage of the Readout subsystem within the DAQ system. FELIX serves as a vital interface, bridging the gap between detector-specific custom point-to-point serial-links and the downstream commodity multi-gigabit data network. Before reaching the Dataflow subsystem for buffering, the data are received by the Data Handlers, where formatting processes and monitoring take place. The Readout operates on 1 MHz event rate. The data are buffered, transported, collected and compressed by the Dataflow before sent to the EF System.

The Event Filter system

The second level of the Phase II TDAQ system, is a software based EF system. In the baseline scenario, the EF system will leverage commercial hardware acceleration that provides high-level trigger functionality. This is supported by a large CPU farm, which processes the events selected by the TDAQ system in real time, ensuring efficient decision-making at high data rates. The events that are accepted by the TDAQ system are permanently stored at the ATLAS offline computing system.

The event filter plays a critical role in identifying and retaining the most interesting collision events for analysis, while discarding the majority of the data that are do not provide particularly useful or interesting information. The physics motivation behind this filter lies in the need to study rare processes which occur inside an overwhelming background of high-frequency, lower-energy interactions. The event

filter uses advanced algorithms to quickly analyze data from the detector and focus on events showing signs of rare phenomena, like high-energy particles or unusual energy patterns. This helps collect the most valuable data while staying within the limits of storage and processing resources, increasing the chances of finding new discoveries. It is designed to sustain a maximum input rate of 1 MHz, and select events with a maximum output rate of 10 kHz.

The new ITk reconstruction software prototype in combination with the updated ITk layout has achieved a reduction of the time for reconstruction by a factor of 8 with minimal compromises in the tracking performance. Furthermore, the delay of Run 4 introduced a new factor by the latest developments on the CPU market that have resulted in significant reduction in the predicted CPU costs.

EF alternative architectures

The *Hardware Track for Trigger* (HTT) is an alternative hardware based approach that was introduced by the technical design report for the EF, based on custom hardware. The HTT subsystem, performs hardware-based track reconstruction that allows a fast rejection of single high-p_T lepton and multi-object triggers, resulting in a rate reduction to approximately 400 kHz. Subsequently, global tracking is performed again making use of the HTT to achieve even further rejection. This approach was later abandoned as evolution of the technology allowed to more versatile and easier to develop architectures than the custom hardware design.

Reduced physics motivation, and complexities in implementation led to reconsideration of the original TDAQ system design. One of the main reasons was the difficulty in developing the ITk Pixel Front-End ASIC, which created significant technical risks for the project. In addition, the ITk Pixel Detector's throughput was more limited than initially expected, resulting in the weakening of the physics motivation for pursuing the original design that relied on higher data rates.

Another reason driving the reconsideration of the system was the latency of the L1 trigger. The proposed "evolution" scenario involved increasing the L0 trigger rate and adding a second fast selection stage L1 to process the data. However, this two-step trigger system raised concerns about potential delays, particularly given the limitations imposed by the detector front-end buffer sizes. Additionally, the technical challenges of implementing the L0/L1 system, combined with the complexities of the existing L0 system, made the project both more intricate and riskier.

ATLAS decided to drop the “evolution” option, which removed the need to design a custom-based track trigger for L1. Advancements in track reconstruction software and commercial accelerator technology further reinforced the decision. ATLAS committees recommended a shift to a commercial solution for Event Filter (EF) tracking at the High-Luminosity LHC. This commercial approach, which could include hardware accelerators like FPGAs and GPUs, was seen as more cost-effective and capable of saving power, further reducing the need for the custom-built system described in the original TDAQ TDR.

Chapter 4

Digital system design

The need for an advanced TDAQ system to handle the high data rates generated by the LHC, that is impossible to handle with CPU/GPU based architecture, has already been highlighted in the previous chapter. This section will introduce the different technologies available to support this system.

Over the past few decades, there have been significant advancements in digital design. The number of transistors in chips has increased exponentially, making them faster, cheaper, and more efficient. As a result, digital systems have evolved beyond just computational applications to include control and communication applications that store, process, and transmit information in high rates, while offering flexibility due to more accessible tooling.

4.1 Device technologies

Digital systems can be implemented in hardware either by using standard *Application Specific Integrated Circuits* (ASICs) allowing the design down to the transistor level or with architectures like *Field Programmable Gate Array* (FPGA) which are re-programmable. All of them have their trade-offs, the most obvious being the chip area, speed and power versus the cost of design, implementation and production.

There are different types of device technologies for building custom digital circuits. A key feature of these technologies is how they are customized. An IC is made from layers of doped silicon, polysilicon, metal and silicon dioxide, built on top of one another, on a thin silicon wafer. The FPGAs have all their layers set in advance. These devices can be customized later by downloading a connection pattern to their internal memory or by burning the internal silicon fuses.

In contrast, ASICs require one or more layers to be specially customized for a specific application. These layers are part of the physical structure of the chip and typically include the metal interconnect layers, which define how different components of the chip are connected, and sometimes the polysilicon and diffusion layers, which influence the fundamental transistor design. This customization involves creating specialized photolithographic masks to define the patterns for these layers and fabricating them through complex semiconductor manufacturing processes. The entire process requires high precision, advanced equipment, and is carried out in specialized factories called foundries or fabs. Each layer corresponds to a specific step in the manufacturing process, and its design must be carefully optimized to meet the application's performance, power, and area requirements.

4.2 Technology evaluation criteria

During the research and development phase of an application, a decision has to be made on which device technology is the better match for the needs of the project at hand. The main criteria for choosing are area, speed, power, and cost.

Area: This is the size of the chip. A smaller chip uses fewer resources, is easier to test, and has a better yield. An FPGA chip often needs two to five times the area of an ASIC chip [25].

Speed: This is how fast the circuit performs a function, usually measured by the worst-case delay between input and output signals. Faster circuits are better, especially for computation-heavy applications. Faster operation can be achieved with a more sophisticated and parallelized design, which requires a larger area. As the ASICs are custom designs, they are optimized for performance, making them the faster technology compared to an FPGA.

Power: This is the energy consumed by the circuit. Low power is important for applications like battery-operated handheld devices. A smaller chip with fewer transistors usually consumes less power. Thus, an ASIC uses the least power, and an FPGA chip uses the most power.

Cost: The cost includes production cost, development cost, and time-to-market cost. The most cost-effective option between an FPGA and an ASIC implementation depends on the number of chips that are needed. The ASICs have a lower cost of production per unit, however, the cost of development and the time-to-market cost is higher. Therefore, if the total amount of chips is below a certain threshold, the FPGA

are more cost-effective. On the other hand, when the total amount of chips rises above that threshold, the ASICs become the more cost-effective option. In addition, reprogrammability is a factor that has to be taken into account for choosing the technology, as this is a reason that FPGA might be preferred over ASICs.

Each technology has its strengths and weaknesses. The best technology depends on the needs of a specific application.

4.3 Abstractions

Technology advancements allow the deployment of a number of transistors so large that the description of hardware becomes a very complex task. A way to simplify this process is to describe the hardware in a higher, more abstract level. Different methods of abstractions can be used to simplify the models of the system that show only the selected features and ignore the associated details of the hardware design. While the lower levels of abstraction are a more complex expression of the system, they do offer more accurate description representation of the system.

There are four levels of abstraction in digital system development based on the basic building block

Transistor level: Some of the most common building blocks on the transistor level are the transistors, resistors and the capacitors. It is an analog system and its behavior can be described by current-voltage diagrams. A system in the transistor level is described by a layout of its components and its interconnections.

Gate level: The building blocks of the gate-level abstraction are logic elements including AND, OR, XOR, NOT, multiplexers and basic memory elements. The signal propagated through the design are logic signals '0' and '1'. The logic values are determined by the voltage in the entry of the logic elements. A voltage threshold discriminates and classifies the voltage value to a logic value by comparing it with a voltage threshold. A gate level system is described with the placement of the logic gates and the routing of the interconnection wires.

***Register Transfer Level* (RTL):** The RTL's main building entities are simple logic block, memory elements, and routing components. This level of abstraction allows the grouping of the signals into data types that can be handled and propagated between registers.

The registers remain in sync by following the rising or falling edge of a common clock signal. The layout of the RTL is the floor plan, which is a representation of the

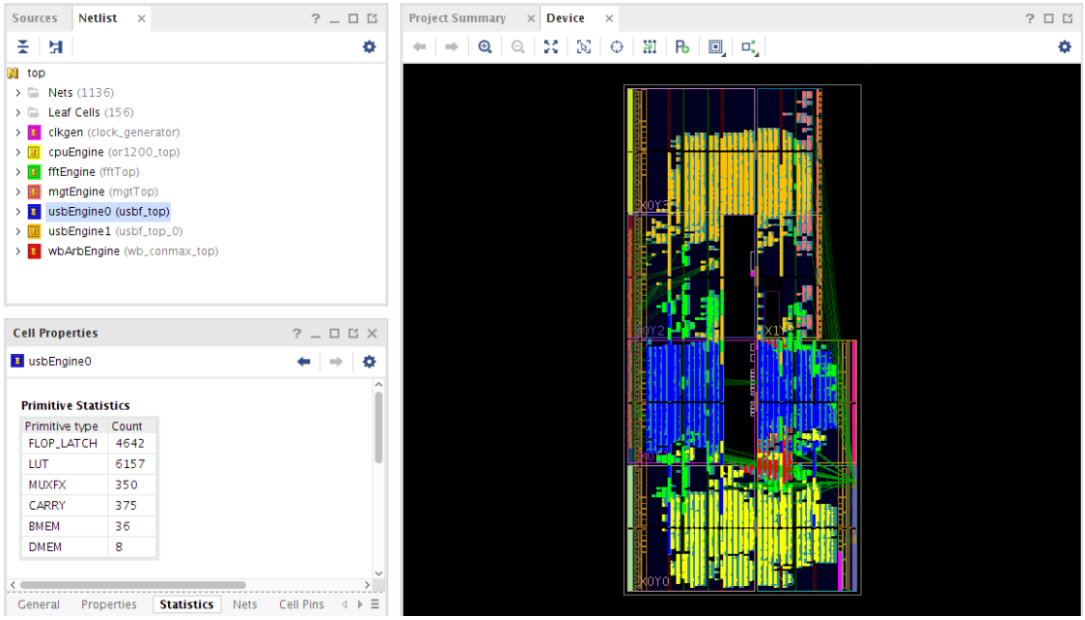


Figure 4.1: FPGA floor plan, highlighting the hierarchy [26].

placement of the components and their interconnections of the silicon. An example of a floor plan is shown in figure 4.1. The floor plan allows the detection of the slowest path between the registers to determine the clock period that allows the seamless function of the system.

System-Level Design: This is the highest level of abstraction. The building block of logic in this case is an *Intellectual Property* (IP) core. The most common IPs are processors, memory blocks, bus interfaces. The behavioral description includes computation steps and communication processes. A floor plan is also the physical layout of this level of abstraction, yet the components have a much bigger size compared to the RTL floor plan.

The abstraction level characteristics are outlined in the table 4.1 below.

4.4 Field Programmable Gate Arrays

Field Programmable Gate Array devices (FPGAs) are integrated circuits that users can program multiple times. They contain logic blocks with programmable logic gates and memory elements, such as simple flip-flops or more complex memory blocks. FPGAs also have interconnect resources, allowing these blocks to be wired together to implement various digital circuits or systems. A block diagram of the architecture of an FPGA is shown in figure 4.2. Figure 4.3 shows the architecture of

Table 4.1: Abstraction level characteristics [25].

	Typical blocks	**Signal representation**	**Time representation**	**Behavioral description**	**Physical description**
Transistor	Transistor, resistor	voltage	continuous function	differential equation	transistor layout
Gate	and, or, xor, flip-flop	logic 0 or 1	propagation delay	Boolean equation	cell layout
RTL	adder, mux, register	integer, system state	clock tick	extended FSM	RT-level floor plan
System Level Design	processor, memory	abstract data type	event sequence	algorithm in C	IP-level floor plan

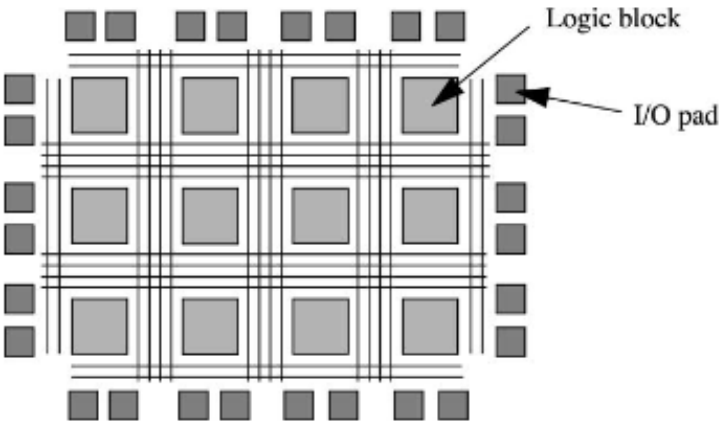


Figure 4.2: FPGA architecture [37].

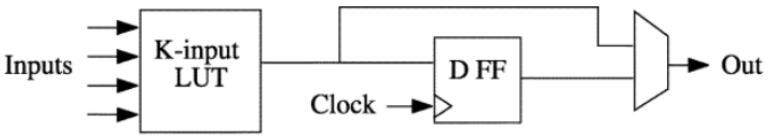


Figure 4.3: Basic logic element [37].

a basic logic element, which consists of a look up table a flip-flop and a multiplexer. Multiple basic logic elements make up a logic cluster, shown in figure 4.4.

The basic structure of an FPGA includes:

1. Look-up tables (LUTs) for performing logic operations.

2. Flip-Flops (FFs) that are synchronous memory elements used for register im-

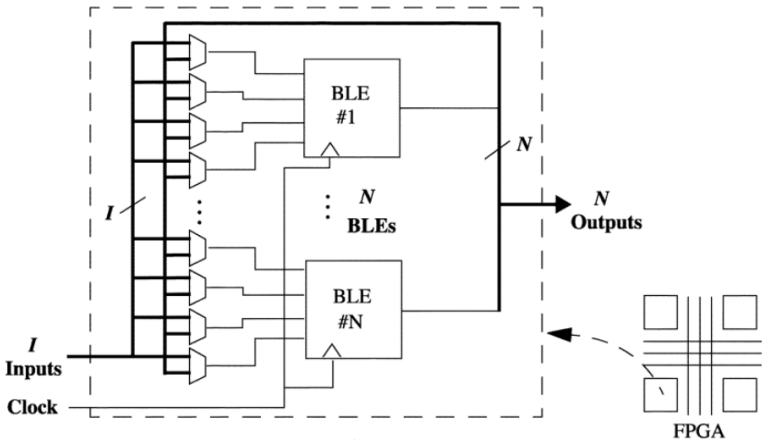


Figure 4.4: FPGA logic cluster [37].

plementations.

3. Wires for interconnection.
4. IO pads for data input and output.

Modern FPGAs include additional elements for improved efficiency:

1. Embedded memories (Block RAMs) for data storage.
2. Phase-locked loops (PLLs) for varying clock rates.
3. External or embedded microprocessors.
4. Digital Signal Processing blocks DSPs.

These additions result in a more complex architecture. FPGAs are cost-effective and quick to deploy for low to medium volume productions, unlike (ASICs), which require significant design time and resources. FPGAs can be configured in under a minute and are relatively inexpensive. Recent FPGAs support partial reconfiguration, allowing dynamic changes to a portion of the FPGA while the rest continues to operate, enabled by Dynamically Reconfiguration Ports (DRP). The applications presented in this thesis target both Xilinx and Intel FPGA devices.

4.5 FPGA Technologies

FPGA reconfigurability and reprogrammability depend on the technology used for logic blocks and interconnections. The main technologies of FPGAs are static memory (SRAM), flash, and antifuse.

1. SRAM-based FPGAs: These store logic cell configurations in static memory arrays. They are popular due to their reprogrammability and use of standard CMOS technology. However, they occupy more area and they are more sensitive to radiation. The Xilinx Virtex is an example of an SRAM FPGA-based family.

2. Flash-based FPGAs: These use flash or *Electrically Erasable Programmable Read-Only Memory* (EEPROM) for configuration storage. EEPROM is a type of memory widely used in embedded systems. It is a non-volatile memory, meaning it retains stored data even when the power is turned off. Both flash and EEPROMs are even more power efficient compared to SRAM-based FPGAs, and they are radiation-tolerant, and they are already programmed at power-up. Their drawbacks include limited reconfiguration cycles, meaning that they cannot be reprogrammed as many times as a regular SRAM-based FPGA, and reliance on non-standard CMOS processes.

3. Antifuse-based FPGAs: The antifuse-based FPGAs such as the Microchip Accelerator FPGAs [27] can be programmed only once. They benefit from lower area occupancy and low parasitic capacitance. An antifuse initially does not conduct current but can be permanently programmed to do so. These FPGAs are robust against radiation, making them ideal for use in high-radiation environments, such as space or particle physics experiments. However, once programmed, they cannot be reset.

4.6 Programming languages for digital systems

Hardware description languages

Hardware description languages program the behavior of an electronic circuit. VHDL and Verilog are the two languages primarily used by engineers for the application development. VHDL stands for *VHSIC Hardware Description Language* where VHSIC is *Very High Speed Integrated Circuits*. As VHDL is the language used for the application development shown in this thesis, the attention is centered on it, however most of the subject discussed apply to all the hardware description languages.

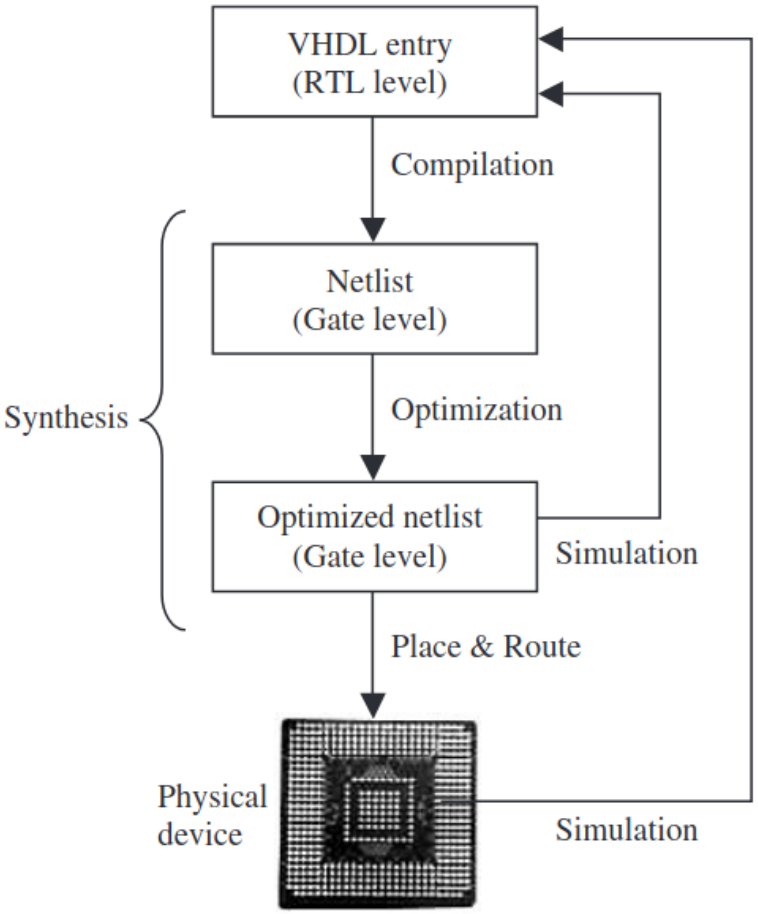


Figure 4.5: VHDL design flow [38].

VHDL is synthesizable but can also be simulated. Not all constructs are synthesizable but they can be utilized for simulation. Synthesis is the process of converting high-level abstract descriptions of a digital circuit, written in a hardware description language, into a gate-level representation that can be implemented on hardware such as FPGAs or ASICs. The design flow of VHDL is shown in figure 4.5.

Opposite to the more common programming languages, VHDL is capable of describing hardware design which is inherently concurrent. Sequential code can be executed in *processes, functions and procedures* [38].

HLS

An alternative to the low level languages is the *High-Level Synthesis* (HLS). HLS is typically written in a language like C or C++, and is converted into a hardware

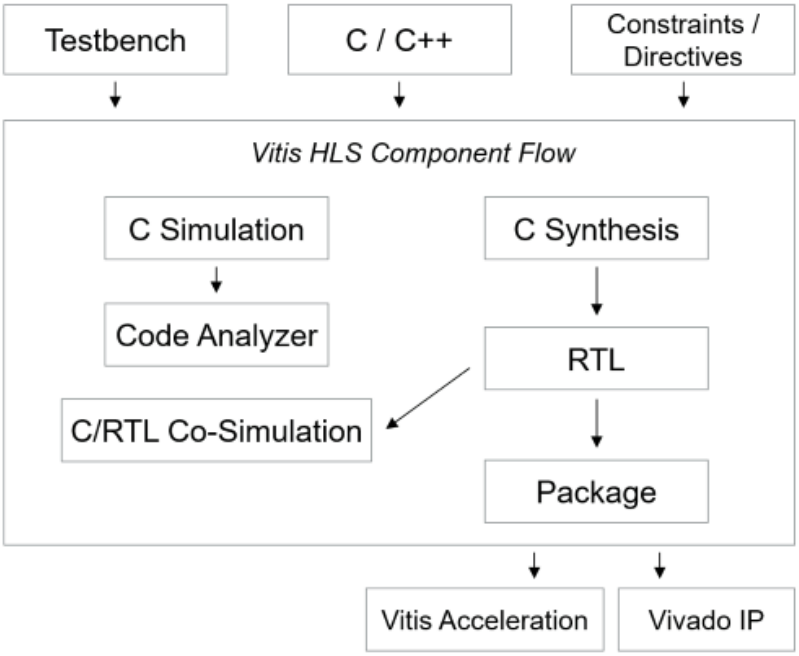


Figure 4.6: HLS design flow [39].

description language like VHDL or Verilog. HLS facilitates the code validation as the functionality can be tested in C-level and then verify the functionality of the HLS C-simulation.

The fine-tuning of the design on the RTL level can be achieved with optimization pragmas. Designers can adjust settings, called pragmas, to fine-tune performance and resource usage. This flexibility helps explore different design options without rewriting the code. The main advantage of HLS is faster development and easier debugging. The HLS design flow is shown in figure 4.6.

Pragmas in HLS help optimize the design and control hardware implementation. Below are examples of how they are used:

- Loop unrolling: This directive duplicates loop iterations to improve parallelism, reducing execution time.
- Pipelining: Pipelining allows multiple loop iterations to execute simultaneously, improving throughput.
- Array partitioning: Splits an array into smaller sections to enable parallel processing, optimizing memory access.

- Interface control: Defines how function inputs and outputs interact with the hardware, reducing latency in FPGA implementations.
- Dataflow optimization: Enables task-level parallelism by allowing different functions to run concurrently, when this is possible.

The ease of development of a design and the exploration of different implementations by configuring different pragmas, is a major advantage of the HLS approach over the HDLs. However, the HDLs allow the developers to have full control over the implementation and optimize the design which results in designs that optimize the chip utilization.

4.7 Development flow and verification

VHDL code that is synthesizable, can program an FPGA or an ASIC. The synthesis and implementation steps are as follows.

1. Compilation: This step converts the RTL design described in VHDL into a gate level netlist.

2. Optimization: The RTL conversion is followed by an optimization. The optimization can target either speed or area.

3. Place and route: The final step of the flow is the place and route where the physical layout is generated to program the FPGA.

Verification is crucial during the development flow. It is the process of ensuring that a design meets its specifications and performance goals. It involves checking the accuracy of both the initial design and any changes made during development. There are two main aspects of verification: functionality and performance.

Functional verification ensures that the system produces the correct output. Performance verification checks if the system meets timing constraints, ensuring it operates within the required time limits. Verification is carried out at various stages of design and at different levels of detail. The process typically begins with simulation, where the HDL code is tested against a set of predefined testbenches. Testbenches simulate various input scenarios and edge cases to verify the functionality of the design under all expected operating conditions. Once functional correctness is verified, formal verification may be applied to mathematically prove that the design adheres to specified properties, such as absence of logic bugs. Additionally, timing analysis ensures the design meets the clock frequency and timing constraints, using tools like

static timing analyzers. For FPGAs, resource utilization (e.g., logic elements, memory blocks) is also checked to ensure the design fits within the target device. After simulation and synthesis, the design is verified in hardware by implementing it on the FPGA and applying actual inputs to confirm its behavior in real-world scenarios. Verification at each stage is critical to catching issues early and ensuring reliable FPGA functionality.[38]

Chapter 5

Firmware development for the Phase II upgrade of the Central Trigger Processor

This chapter presents the author's contribution to the firmware studies for the CTP readout for Run 4. It introduces the current Run 3 trigger system and focuses the Phase II CTP upgrades. Finally, this chapter describes the development of the CTP readout firmware and discusses the results of the firmware studies of the proposed design.

5.1 Central trigger system

Level 1 trigger system

The ATLAS Level-1 trigger is based on information from the calorimeter and muon trigger processors. The calorimeter information coming from both the hadronic and electromagnetic calorimeters is processed by the L1Calo system. The L1Calo system processes data with coarse granularity from the calorimeter towers and identifies electrons, photons, taus and jets. It counts the number of objects above a number of energy thresholds, as well as transverse energy sum and missing transverse energy. The L1Muon system processes information from the RPC and TGC to identify muons in the barrel and end-caps. The L1Topo selects events based on their topology. Information from both L1Calo and L1Muon includes multiplicities for candidate electrons/photons, taus/hadrons, jets, and muons. It also includes flags that act

as predefined conditions of interest for total scalar transverse energy, total missing transverse energy, and total scalar jet transverse energy. In addition, information that includes the p_T, ϕ, and η of each reconstructed candidate is passed to the L1Topo. The purple blocks of the Figure 3.3 provide an overview of the ATLAS Level-1 trigger.

The calorimeter and muon trigger processors supply trigger information to the Central Trigger Processor (CTP). The CTP generates the Level-1 Accept (L1A) decision and distributes it to the Timing, Trigger, and Control (TTC) partitions of the experiment. Each TTC partition has one *ATLAS Local Trigger Interface* (ALTI), a TTC system, and a hierarchical structure of ROD_BUSY modules. The CTP also provides trigger summary information to the High Level Trigger (HLT) and the data acquisition system. The ATLAS Online system configures, controls, and monitors the CTP. The ALTI is the upgraded system for Run 3 that replaced the legacy *Local Trigger Processor Interface* and the *Local Trigger Processor* that are still partially used. The ALTI consists of a single *Versa Module Eurocard bus* (VMEbus) [22] module.

The ALTI allows the use of trigger and timing signals from the CTP but can also generate these signals locally. To enable multiple TTC partitions to operate independently of the CTP, an ALTI module is used. This allows concurrent calibration runs of the calorimeter detectors with the calorimeter trigger and of the muon detectors with the muon trigger without requiring any re-cabling.

The Central Trigger System consists of the CTP and the Muon-to-CTP interface (MUCTPI), which connects the detector-specific logic of the barrel and end-cap muon trigger systems to the CTP. The TTC system, which transmits trigger and timing signals to the detector front-end systems, is also discussed in this context.

Trigger information from 208 L1Muon sectors are used to calculate the multiplicity of muon candidates for up to six momentum thresholds, and sends them to the CTP. The CTP is the system at the end of the pipeline that forms, as already said, the L1-Accept signal and transmits it to the sub-detectors.

When a sub-detector readout system is approaching its full capacity, the data acquisition system sends a BUSY signal to stop the L1A. This mechanism is in place to prevent the sub-detector's buffers from overflowing.

All CTP elements are located in the ATLAS control room (USA15) in standard racks and operate under standard conditions, avoiding exposure to radiation and magnetic fields. Most components are implemented on 9U or 6U VME boards, while the MUCTPI is implemented in an *Advanced Telecommunications Computing Architec-*

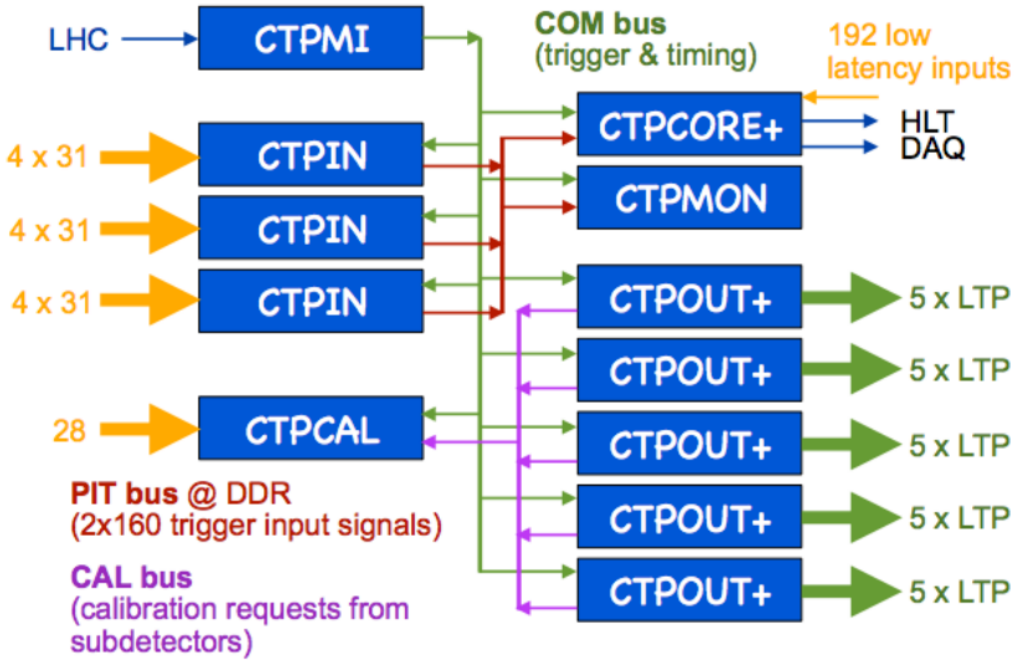


Figure 5.1: CTP VME modules and their connections [23]. All modules are housed in a VME crate.

ture (ATCA) board.

Central Trigger Processor

CTP, the last system of the L1 trigger, forms the L1 decision after processing the information from the L1Topo, L1Calo and L1Muon. Once the L1 decision has been made, the CTP distributes it to the sub-detector readout systems along with the LHC timing signals via the TTC network. The CTP is a VME base system, housed in a single 9U VME crate. Figure 5.1 shows a schematic overview of the CTP modules and their connections.

The first VME module shown in figure 5.1 is the *CTP Machine Interface* (CTPMI). It distributes the signals received from the LHC to the other modules. The *CTP Input Modules* (CTPIN) receives and aligns the trigger inputs from L1Calo and L1Muon.

There are three CTPIN modules that receive 124 trigger inputs over four cables, making 372 trigger inputs. The trigger inputs are routed by a switch matrix, and they are sent to the CTPCORE+ and CTPMON modules through the *Pattern In Time* (PIT) back-plane.

The *CTP Core Module* (CTPCORE+) receives 320 trigger signals from the CT-

PIN modules over the PIT bus and 192 trigger signals via optical links. Figure 5.2 shows the processing stages of the CTP. *Look Up Tables* (LUTs) form the 512 trigger items by logically combining the 512 input trigger signals. The LUTs are capable of implementing any truth table. They select object multiplicities and can perform OR operations, while *Content Addressable Memory* (CAM) performs AND and NAND operations. The full trigger logic, which includes the LUT and CAM, is programmable via software and each item can be prescaled individually. Based on a programmable list, every trigger item's coincidence with the LHC bunch-crossing is verified. A random prescaling algorithm can prescale trigger items. A prescale of N means that only 1 out of the N trigger items will be accepted. If the prescale for a certain trigger item is set to N=2 and then, on average, one out of every two events passing that trigger item's conditions will be accepted. Prescaling is applied independently to each trigger item and evaluated for every event, ensuring that the reduction in accepted events is systematic and consistent with the defined rate. Therefore, on average only one of the two events will be chosen [42]. The prescale factor can be changed during a run to adjust the trigger rate. Trigger veto can be asserted by the CTP. The veto can be triggered both by the sub-detector busy signals or by the CTP internal dead-time generation. The dead-time is contributed by both a simple dead-time algorithm and 4 leaky bucket algorithms.

The leaky bucket algorithms prevent the sub-detector frond-end buffers from becoming full. The logic of the leaky bucket algorithm follows the principle of a bucket, which leaks at a constant rate, while it is filled with a different rate, which in the case of the trigger system is the L1A. When the bucket gets full the system asserts the preventative deadtime to prevent it from overflowing. The parameters of the input and output rate in the leaky bucket algorithm are configurable and determining the period and the maximum number of L1A that can be accepted during a given period [41]. Both the veto gate and the enable/disable mask are first applied to each trigger item. The logic OR forms the L1A signal that is propagated to the sub-detectors by the CTPCORE+.

Figure 5.3 shows the CTPCORE+ module block diagram. The CTPCORE+ consists of two Virtex 7 FPGAs. One of the FPGAs is responsible for processing the trigger signals while the second FPGA is a dedicated monitoring processor, that brings significantly improved capability compared to the previous system, enabling higher efficiency and precision in handling increased data rates and complex event selection. The trigger rate is monitored in each step of the pipeline, which is the trigger items

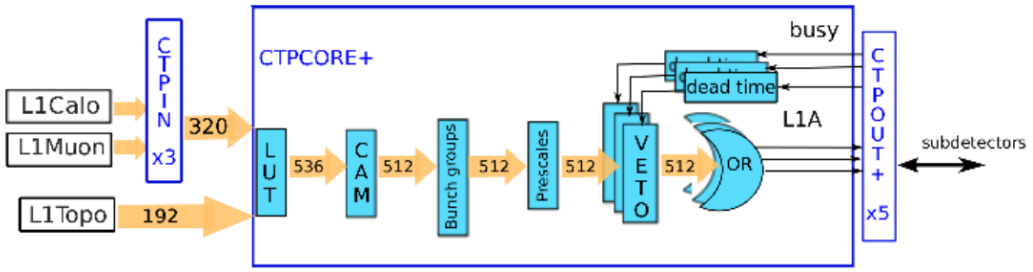


Figure 5.2: Processing stages of the CTP [1].

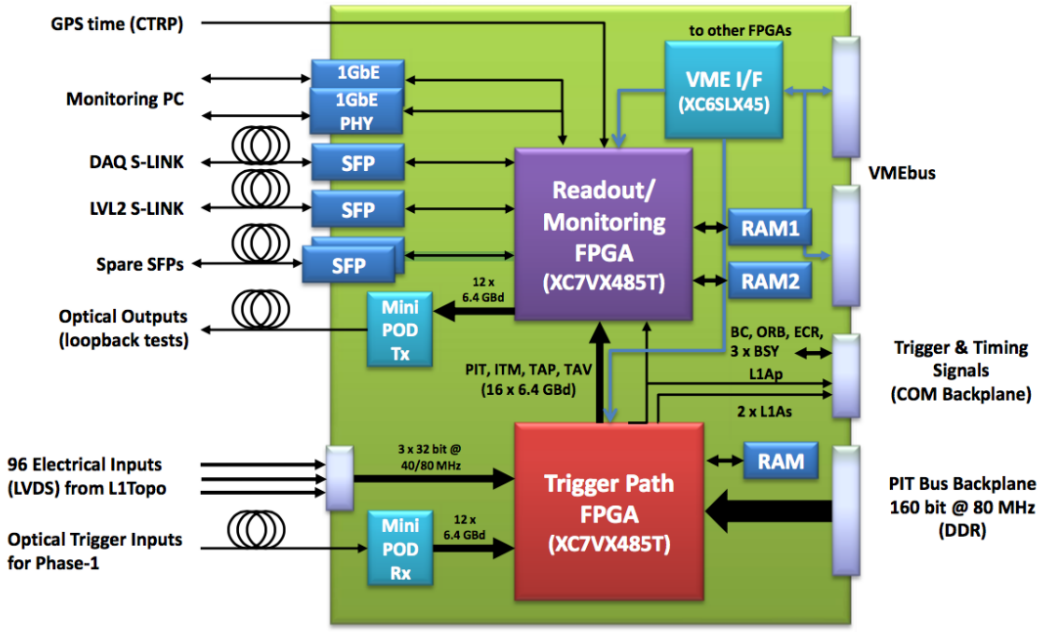


Figure 5.3: CTPCORE+ module [1].

before prescale (TBP), after the prescale (TAP) and after veto (TAV).

The inner structure of the CTP Out Modules (CTPOUT+) can be seen in figure 5.4. CTPOUT+ distributes the L1A and timing signals to the sub-detectors. At the same time, it receives their calibration and busy signals. The CTP Monitoring module (CTPMON) is responsible for the per-bunch monitoring of trigger signals on the PIT back-plane while the CTP Calibration module (CTPCAL) accepts and transmits the sub-detector calibration requests to CTPCORE+. The common back-plane (COM bus) is responsible for the distribution of the trigger and timing signals between the CTP modules.

The CTP is partitioned into multiple logical sections. The partitioning allows the

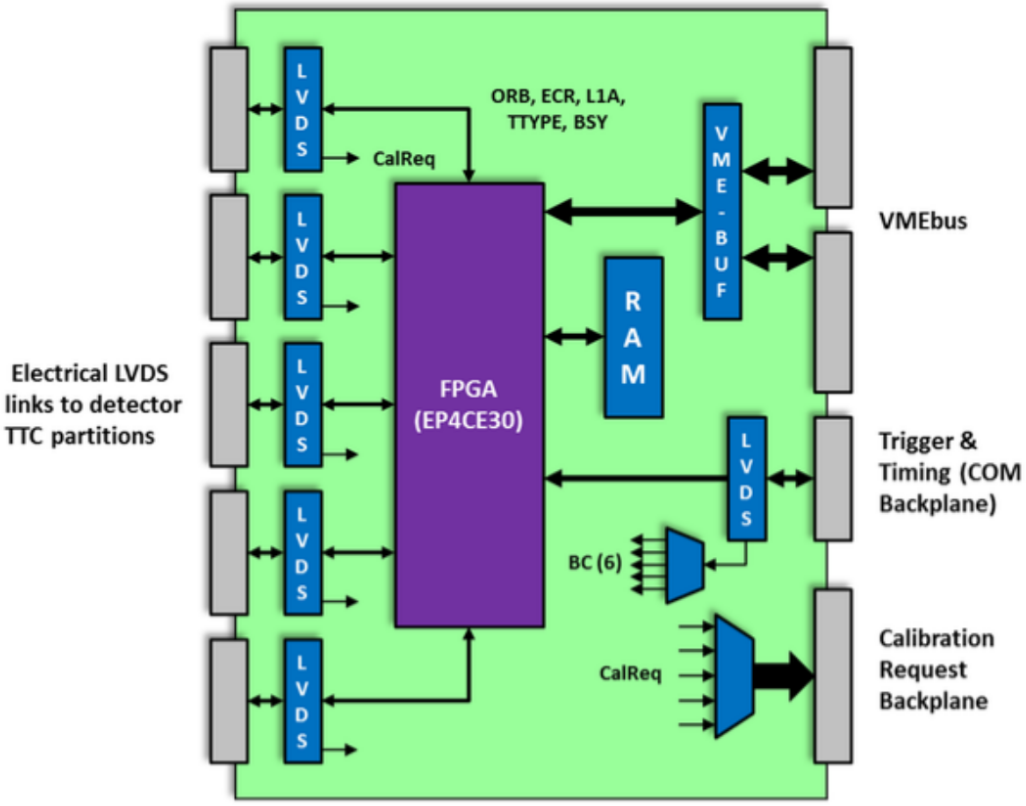


Figure 5.4: CTPOUT+ module block diagram [1].

CTP to commission and calibrate different sub-detectors in parallel. By partitioning, it is possible to logically separate all data taking. More specifically, every trigger item is assigned to only one data-taking session, allowing the different configurations of prescales and bunch group patterns, as well as, different dead-time configuration to better address sub-detector requirements.

MUCTPI

The ATLAS Level-1 trigger uses the multiplicities of tracks found in dedicated muon trigger detectors to form the final trigger decision.

The muon trigger detectors consist of resistive plate chambers (RPC) in the barrel region and thin-gap chambers (TGC) in the end-cap and forward regions of ATLAS. Coincidences of hits in different detector stations are used to identify muon candidates. The muon trigger electronics considers the transverse momentum (p_T) of the muon candidates and classifies them according to six programmable p_T thresholds.

The muon trigger detectors are grouped into 208 sectors: 64 in the barrel, 96 in the

end-cap, and 48 in the forward region. Each sector can provide information on up to two muon candidates, prioritizing the highest-p_T candidates. For each bunch crossing, the detector-specific sector logic sends the position and p_T (thresholds passed) of the muon candidates to the MUCTPI. This results in a 32-bit word per sector and bunch crossing, representing more than 266 Gbits/s in total.

The MUCTPI combines the information from all muon trigger sectors and calculates the total multiplicity for each of the six p_T thresholds, resolving possible double counting of muon candidate tracks that traverse more than one detector region. It forwards these multiplicity values to the CTP for the final Level-1 decision.

In addition to inputs for the CTP, the MUCTPI provides data to the HLT trigger and the DAQ system for events selected at Level-1. The DAQ system receives a formatted copy of the information on candidate muon tracks and the computed candidate multiplicity. In Run 3, the HLT trigger system processes muon candidates using the DAQ path. The DAQ path allows for the direct analysis of data streams, providing access to muon candidates and other trigger information without predefining RoIs. This approach ensures greater flexibility and efficiency in handling high data rates and complex trigger conditions.

The MUCTPI must avoid double-counting muons that traverse more than one set of trigger chambers to prevent a high rate of fake di-muon triggers. This is achieved through overlap removal algorithms that identify and resolve cases where a single muon is detected in multiple regions. The algorithm uses information about the spatial and temporal patterns of hits in the trigger chambers, as well as the geometry of the detector and the magnetic field configuration, to correlate tracks and ensure that each muon is counted only once. This issue is significant for low-p_T muons, as their tracks are considerably deflected in the magnetic field of the toroid magnets. For instance, muons may be seen in both the barrel and end-cap detectors.

5.2 Phase II CTP

Central trigger system

The interfacing with the muon detectors in the upgraded central trigger system for Run 4 will be performed by the same MUCTPI hardware used for Phase I, with upgraded firmware to keep up with the new requirements introduced by Phase 2.

During the HL-LHC data taking, the CTP will need to be able to accommodate

bigger trigger menus. The TTC system along with the LTI modules, which exploit the high-bandwidth optical-link technologies, will incorporate the detector BUSY collection function.

CTP will have optical serial inputs that receive information from the global trigger. The Phase II CTP has 1024 trigger items based on 1024 usable single-bit inputs. The CTP combines trigger inputs from the Global Trigger and the Phase I L1Calo trigger systems, to form trigger items. Trigger items can be simple requirements such as inclusive selections of objects above a given p_T threshold, or complex requirements combining multiple objects. Trigger algorithm flags from the Global Trigger, MUCTPI and the Phase I legacy inputs are combined to form the final trigger decision. The CTP applies additional requirements such as limiting triggers to the subset of bunch crossings in the LHC train, deadtime and pre-scaling.

Finally, the Central Trigger System distributes the LHC timing signals and the trigger decision, with the associated data, to the LTI modules of the corresponding sub-detectors. The LTI is responsible for reformatting the TTC data and potentially to add asynchronous and synchronous sub-detector data and commands over a programmable interface and, subsequently, propagate it to the sub-detector FELIX systems. In addition, the LTIs send the BUSY signals from the sub-detectors back to the CTP.

The FELIX system is the part of the Readout that interfaces all the detector-specific electronics over custom point-to-point serial-links. Furthermore, FELIX interfaces the Data Handlers, monitoring, control and configuration, and DCS via commodity multi-gigabit network. The LTI sends the TTC information to the FELIX system, which are then propagated to the front end electronics systems after they are formatted appropriately. The main advantage of a system such as FELIX is the development of a modular system with the capability to independently upgrade or modify aspects of the system.

Level-0 CTP

The basic architecture of the trigger CTP is shown in figure 5.5. The CTP receives the trigger inputs from the same bunch crossing, from the various resources at different points in time. Therefore, it is required to buffer the information that arrives at an earlier stage and align it with the subsequent receptions. The buffering is performed by programmable elements

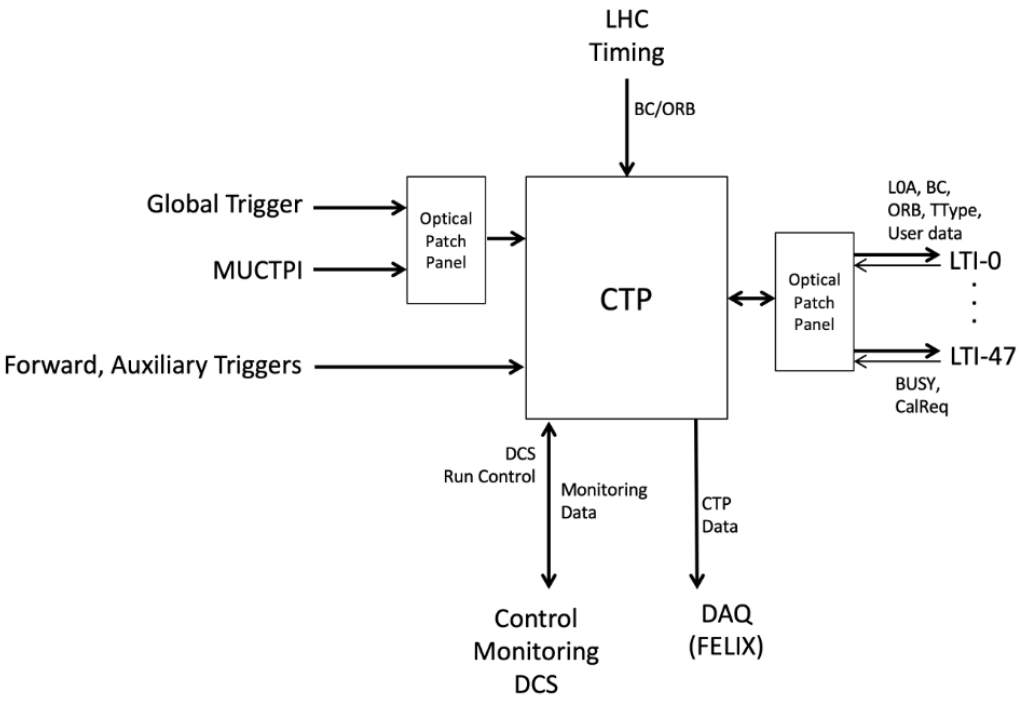


Figure 5.5: Conceptual design of the Level-0 CTP and its interfaces [28].

The CTP building blocks are custom-built ATCA blades, and the whole system will consist of a single ATCA shelf. Figure 5.6 shows a detailed block diagram of the CTP architecture and it's interfaces, while figure 5.7 shows a detailed overview of the inputs and the outputs of the module. An overview of the Phase II upgrades is listed below:

Increased input capacity: The Phase II CTP will support a larger number of inputs from the detectors, making use of the higher granularity of the upgraded detectors.

Enhanced processing power: The chosen FPGA is a Virtex UltraScale+ [44], which is bigger than the FPGA from the same family that was used in Run 3.

Advanced trigger logic: Incorporation of more complex algorithms for decision-making.

Increased output bandwidth: Support of high-throughput outputs to downstream trigger systems for efficient data transferring.

Scalability: Designed for scalability and modularity to accommodate future upgrades and evolving detector requirements.

CTP Machine Interface

The CTP Machine Interface (CTPMI) receives the timing signals from the LHC. More specifically, it will receive the 40 MHz bunch clock and/an the ORBIT signal. It will host an FPGA that will process the input signals such that it will ensure the synchronization of the ORBIT signals, fine-tune the phase of the clock signals, switch between clocks, provide an internal 40 MHz clock and perform jitter cleaning.

CTPIN

The CTPIN is the input board that receives non-latency-critical optical and electrical trigger, such as optical trigger inputs from the Phase I trigger processors and from the MUCTPI. Two 12-way ribbon fiber receiver modules allow one to 24 optical links, each with line rates of up to 9.6 Gb/s. The signals are synchronized and aligned in time by an FPGA. Up to 1024 bits are selected and are propagated to the output. A 12-way ribbon fiber transmitter module allows one to send the 1024 selected trigger signals to the CTPCORE.

CTPCORE

The CTPCORE architecture is based on FPGAs mounted on ATCA boards. It is required that the FPGAs support numerous on-chip multi-gigabit transceivers (MGTs). A Virtex UltraScale+ VU13P has been selected as it offers 6 times more logic resources compared to the Virtex-7 FPGA used for Run 3. In addition, it has 128 GTY transceivers [29]. An SoC FPGA will be used for configuration, control and monitoring. It will be running on Linux OS (CentOS) and application software. It will interface to processing FPGA using AXI chip-to-chip bridge via Aurora MGT serial link. The device that will be eventually used has not been defined yet.

The CTP receives the trigger inputs over 24 optical serial links that are grouped into two 12-way ribbon fiber receiver modules. They are going to be operated at 9.6 Gb/s. It can receive 1024 input bits per BC from the Global Trigger system and up to 1024 trigger inputs from the MUCTPI, the Legacy L1Topo system, forward detectors, calibration, and test systems.

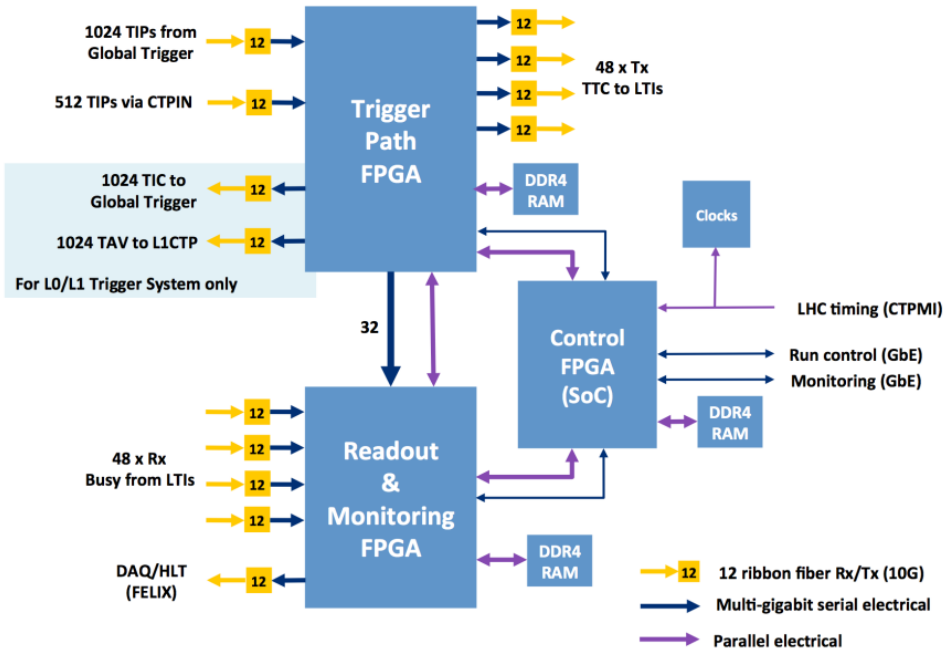


Figure 5.6: CTP architecture and its interfaces [28].

The MUCTPI information sent to the CTP is limited to muon multiplicity and will not contain muon-only topological triggers. The MUCTPI system will handle the detector hemispheres independently. Therefore, the CTP will merge the muon multiplicities from the two hemispheres.

The CTP will receive optical trigger signals via 8 optical fibers from the Legacy L1Topo system with the same format as in Run 3.

It is foreseen that the input signals will be synchronized to the selected BC in the CTP and provisions will be made to allow for a robust synchronization procedure. The trigger input word from each optical fiber and each electrical input signal will be individually delayed in steps of 25 ns. Figure 5.7 shows the inputs and the outputs of the CTP

The CTPCORE's trigger formation begins with the receipt of 1536 trigger inputs. 1024 of the trigger inputs are selected for use in the L0A trigger formation. Additional programmable triggers, like random triggers and bunch pattern flags, are integrated at this stage. The selected 1024 inputs are then directed to the Trigger Logic block.

The Trigger Logic block is organized using LUTs and ternary CAM. This scheme allows flexible and programmable logical combinations of the inputs. This design,

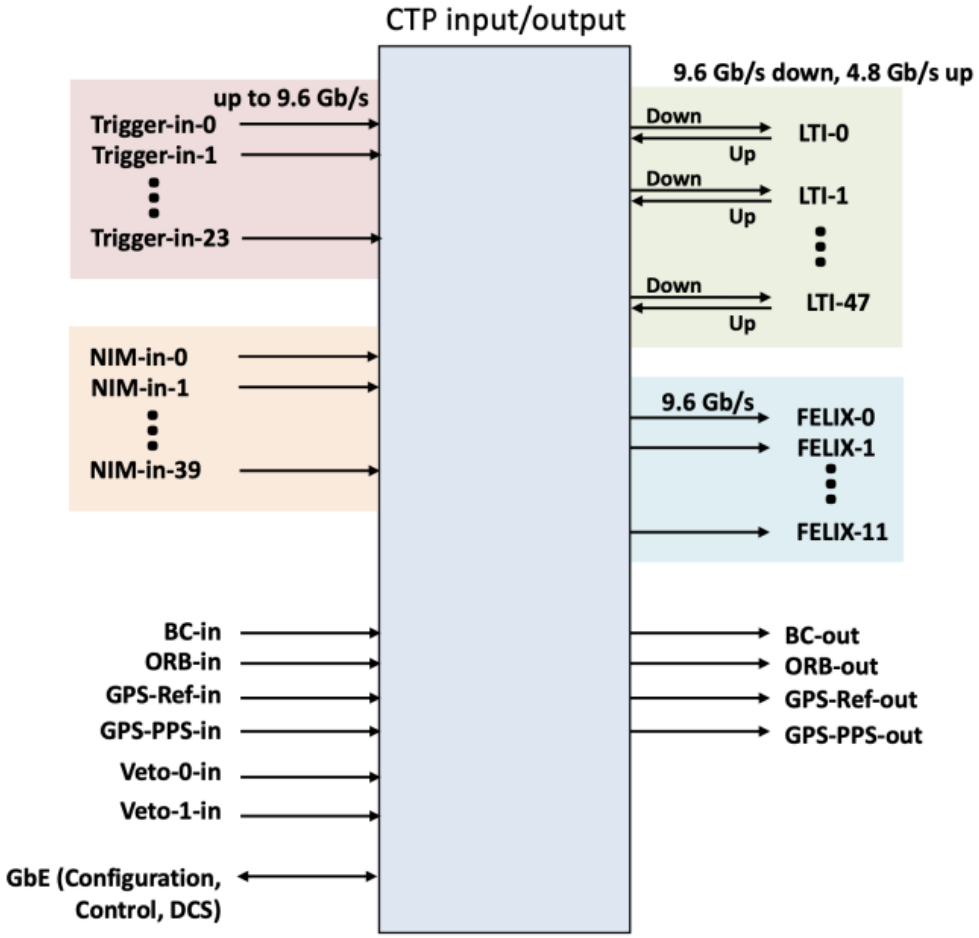


Figure 5.7: CTP inputs and outputs [28].

which builds on previous successful implementations, allows for easy updates to the trigger menu without needing to change the firmware.

After processing in the Trigger Logic block, the 1024 trigger items, that are now called *Trigger items before Bunch Groups* (TBG), are gated with bunch group masks that can be programmed as needed. The gating process creates 1024 *Trigger items Before Pre-scale* (TBP). Subsequently, the trigger items are then pre-scaled using individually programmable pseudo-random scalers to produce 1024 *Trigger items After Pre-scale* (TAP). The TAP signals are gated using a VETO signal. This step allows or prevents specific trigger items from being processed further. The final step combines these signals to generate the L0A signal and a corresponding 16-bit Trigger Type word. The Deadtime block uses the L0A signal history and BUSY signals from subdetectors to generate the VETO signal. This step ensures that the system can handle

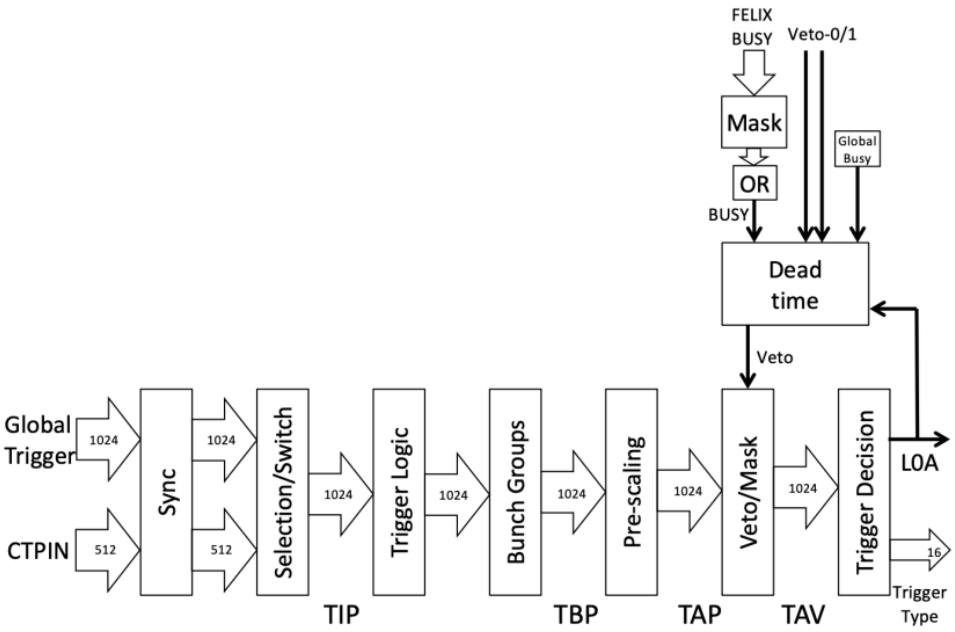


Figure 5.8: CTP real-time trigger formation [28].

trigger bursts without issues. Figure 5.8 shows the real-time trigger formation for the CTP.

The CTP is designed in a way that it will separate the outputs into three different divisions and assign each output to a different partition. This method allows many data-taking segments to run simultaneously. This is especially helpful when it comes to testing and calibration. The trigger path in this configuration is shared by all partitions, including the pre-scaling phase. There will be separate Deadtime, Trigger Decision, and Veto/Mask blocks for every partition. Lastly, FELIX may receive CTP event readout fragments for each distinct partition.

5.3 CTP Readout firmware studies

The CTP plays a critical role in managing the system's deadtime, ensuring that it meets the needs of both the front-end systems and the DAQ system. As already mentioned, in order to prevent buffer overflows, the DAQ system can send a BUSY signal, which acts as a feedback mechanism indicating that specific sub-detector readout systems are approaching their capacity. When this happens, the generation of L0A signals is intentionally stopped. These BUSY signals are sent through the FELIX

systems and passed along via TTC uplinks. The signals are then aggregated by the LTI modules, which combine them using a logical OR operation before transmitting them to the CTP.

The CTP consolidates all these BUSY signals into its BUSY collection tree. The CTP's readout can generate its own BUSY signal, which is also integrated into this tree. During data taking, the CTP is required to sustain an average L0A rate of 1 MHz, introducing an average of not more than 0.1% of deadtime.

To ensure that the CTP meets the stringent performance requirements, firmware studies are essential for developing a design capable of sustaining the target L0A rate of 1 MHz while adhering to the system's operational constraints. These studies focus on optimizing the firmware to handle the aggregation, processing, and generation of signals with minimal latency and deadtime. Testing and validation are necessary to verify that the firmware design satisfies all performance criteria, including the correct handling of BUSY signals, efficient integration with the FELIX system, and adherence to the average deadtime constraint of 0.1%. These tests will confirm that the proposed firmware implementation meets both functional and operational specifications under real-world conditions.

CTP trigger path

Preventative deadtime

For front-end systems that do not send a BUSY signal, the CTP applies what is known as "preventive deadtime" to avoid buffer overflow. This is an artificial deadtime that is introduced based on the timing of L0A signals. The purpose of this is to protect the buffers in the front-end systems by limiting bursts of L0A signals. In the earlier LHC runs (Run-1, 2, and 3), this preventive deadtime was managed using a combination of simple and complex algorithms. Simple deadtime works by inhibiting L0A signals in the N bunch crossings immediately following an L0A signal. On the other hand, complex deadtime involves more sophisticated algorithms, such as the leaky-bucket and sliding-window methods. The leaky-bucket algorithm models a front-end buffer with a set capacity, while the sliding-window algorithm limits the number of L0A signals within a specific time frame.

The CTP is equipped with several programmable features to regulate L0A signals according to the needs of the front-end systems:

- A maximum of 4 L0A signals within 5 consecutive bunch crossings,

- No more than 8 L0A signals in a 0.5 µs window,
- A cap of 128 L0A signals within a 90 µs window.

The second and third requirement means that 8 events out of a window of 20 bunch crossings can be accepted and 128 events out of 3600 bunch crossings can be accepted respectively. When this requirement is not satisfied, then the preventative deadtime shall be asserted and no more events can be accepted.

Furthermore, the CTP is designed to be flexible enough to incorporate additional complex deadtime algorithms in the future, should the need arise due to new sub-detector constraints. The CTP's programmable facilities for deadtime generation, which are based on the L0A signal, include:

- Four programmable sliding-window complex deadtime algorithms, with configurable parameters that allow for a maximum trigger count ranging from 2 to 255 within a time window of 2 to 4095 bunch crossings.
- Three leaky-bucket algorithms, consistent with those used in the Run-3 CTP.
- One simple deadtime algorithm, adjustable from 1 to 4095 bunch crossings.

To ensure the CTP can adapt to future requirements, adequate FPGA resources will be reserved to allow for the addition of new preventive deadtime algorithms based on the experiences gained during commissioning and early operations.

Trigger VETO

Experience from previous LHC runs has demonstrated the usefulness of temporarily VETOing triggers through external signals. For instance, during Run-2, this feature was successfully used to interface with external hardware designed to prevent trigger bursts, thereby protecting the end-cap muon trigger system from noise bursts.

The CTP contains a deadtime block that processes the L0A signal, the BUSY signal, a programmable Global Busy signal, and two external VETO signals. This block houses the preventive deadtime algorithms and generates an internal VETO signal for the VETO block. The internal VETO signal is the result of a logical OR operation that combines the outputs of the preventive deadtime algorithms, the aggregated BUSY signal, the Global Busy signal, and the two external VETO signals. Contributions that are not needed can be masked off as required.

CTP Readout firmware

The input of the readout is the TIP, the trigger logic, the TAP, and the TAV. This results in 4096 bits. In addition to the event word, there is a 1 bit used to flag the L0A, and a 128 bit header word, which includes information like the bunch group ID, the L0A ID, the orbit ID, the readout window size and other metadata that come with each trigger event. Using buffers for this size of data would require bigger sized bunches of memories that would potentially make the implementation more complex. In addition, the readout interfaces with FELIX over 12 optical readout links, operating in FELIX full-mode at 9.6 Gb/s line rate. As the FELIX interface is 32 bit, the 4096 bit words from the readout would have to be broken down into smaller words to match the FELIX interface size.

Each of the 3 CTP partitions is assigned 4 of these links. This allows a design where the readout is splits the 4096 bit input into 4 x 1024 bit inputs. This implementation would result into simpler logic and more optimal usage of the hardware that narrows down the readout word size. Figure 5.9 is showing the CTP readout and monitoring, as well as the three CTP partitions.

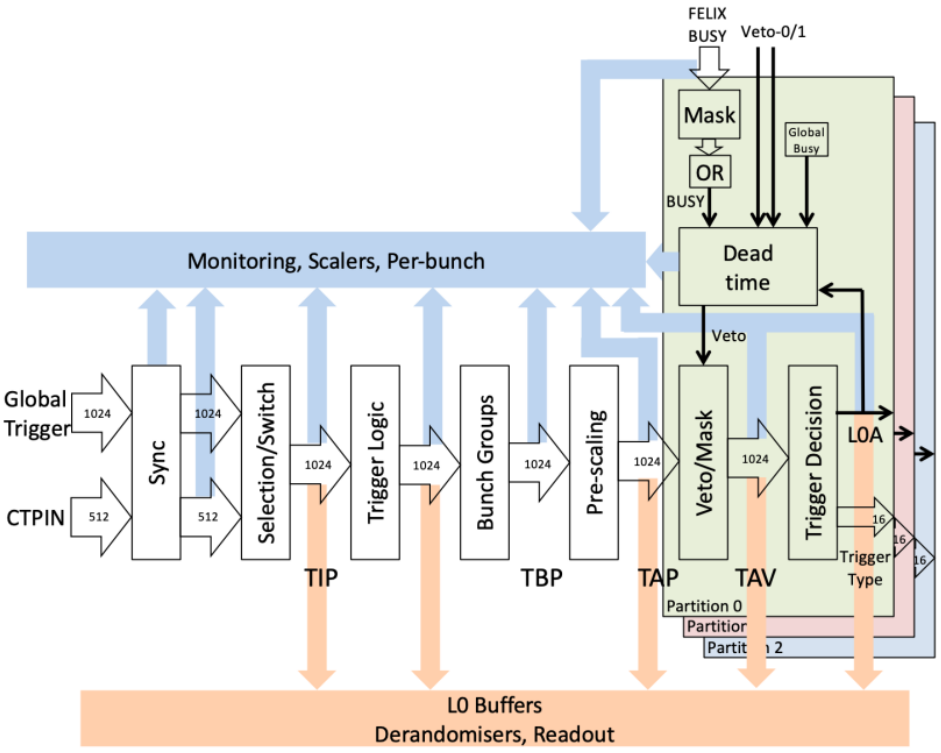


Figure 5.9: CTP readout and monitoring [28].

During data tacking, the CTP readout shall propagate three events to the readout, one before the one with the trigger and one after. For testing and calibration purposes, this window of three events may be possible to be bigger. The size of the readout window shall be configured during the normal configuration phase of the CTPCORE FPGA. The size of the window word is 5 bits, hence the window size at maximum can be 15 events before and 15 events after. The bigger window sizes are used for calibration and synchronization with the LHC machine.

It has been decided by ATLAS that for the Run 4 the trigger system shall be capable to accept events from consecutive bunch crossings. Therefore, simple deadtime cannot apply anymore. As mentioned in the previous paragraph, a window of events must be readout for every accepted event. That would mean that in the case of consecutive events, an event may need to be readout multiple times depending on the window size as an event with a L0A or an event before or after a L0A.

As the system requirements allow up to four consecutive L0A signals to be accepted by the CTP during data taking, each event word can be readout three times, as a trigger event word, as an event word before the trigger and as an event word after the trigger. Every time the same word is readout for a different L0A, then this word needs to be buffered in a different FIFO. Figure 5.10 shows the scenario where the four consecutive L0A are received, resulting in two event words to be buffered for three different L0As. As this is the worst case scenario, the system requires three FIFOs to buffer all the events for a readout window of three events.

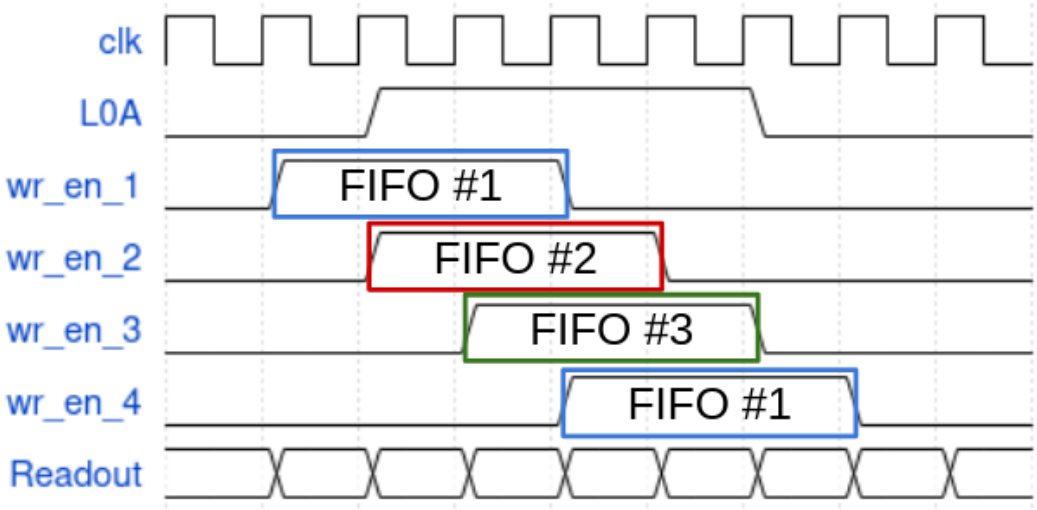


Figure 5.10: Multiple readout of an event, worst case scenario.

For the cases where the CTP is configured to operate with a bigger window, the preventative deadtime would have to be adjusted such that it would satisfy the condition that an event would keep getting registered up to three times. The reason for this limitation is that the number of buffers in the CTP, where the events are stored, shall not be configurable. Such a condition is not necessary, as in the case of calibration and testing, the CTP is not required to accept consecutive events.

A block diagram of the readout firmware is shown in figure 5.11. As already mentioned, the input is a 1 bit word representing the L0A, 1024 bit words representing the events word and the header that is a 128 bit word with a 40 MHz rate. In order to have the ability to register the event before the L0A that comes before the event with the L0A, the input has to be delayed such that the older event words can be accessed. This condition applies for all the readout inputs.

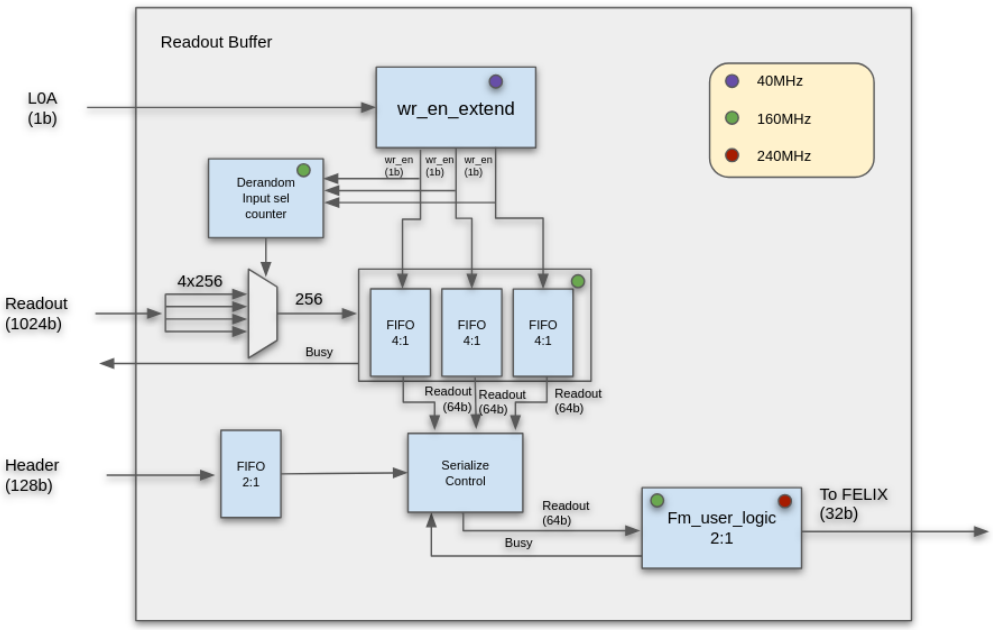


Figure 5.11: Block diagram of the readout firmware.

Write enable extension

The L0A acts as write enable to the first stage of buffering into FIFOs. Therefore, the write enable signal is triggered with the L0A and gets extended for a number of clock cycles that is equal to the size of the readout window. The readout input words, as well as, the header words are also buffered to be in sync with the write enable. Unlike the readout words that register as many events as the readout window, the header is only required to register the one header word that is associated with the L0A.

Input selection

It has been deemed that the input words narrowing will be done in steps. This process results in an architecture that distributes the resource usage more evenly and doesn't require big chunks of memory that might be difficult to implement. Therefore, the input words are already divided into four 256 bit words. There is no need for buffering in this step as it is being performed at a frequency of 160 MHz, which is 4 times higher than the frequency that the input is received. A multiplexer is being used for selecting which of the 256 bit words are going to be propagated to the next stage. A loop counter is used for controlling the select input of the multiplexer. The counter

is enabled by the extended write enable signal that is generated by the write enable extension block.

Write enable distribution

As the input events may have to read out up to three times, there has to be three separate buffers such that no information gets lost in the worst case scenario. Therefore, the write enable signal has to target a different FIFO for every event and distribute the input in a round trip fashion. In addition to that, the logic that controls the write enable has to take into account that an event can potentially need to be written two or three times on all three derandomizing FIFOs.

Derandomizer FIFOs

The three derandomizing FIFOs are asynchronous, such that they reduce the size of the words from 256 to 64 bits. The derandomizing FIFOs operate on a 160 MHz clock to match the input rate of the data sent from the input multiplexer.

Header FIFO

The header FIFO is also asynchronous and controlled by the write enable block. In this case the extended write enable is not used as there is no need to propagate the header of all the events inside the readout window.

Serialize control

The serialize control component is responsible for the read enable of both the derandomizer FIFOs and the header FIFO. It is implemented as a Finite State Machine (FSM). The states of the FSM is shown in figure 5.12.

When the FSM is in the idle state, it examines if the header FIFO has a value. When the header FIFO is not empty, the state changes to the read header state. The read header state will assert the header FIFO's read enable and stay asserted for as many clock cycles as it is required to read out the full header. Subsequently, the FSM will move the state that reads the readout word from the derandomizer FIFO. Similarly to the header read out, the read enable stays asserted until all the event words are read out from the FIFO. This state has to take into account the window size to count correctly. When the window has been read, the FSM moves forward and checks again if the header FIFO is empty to start again the same cycle.

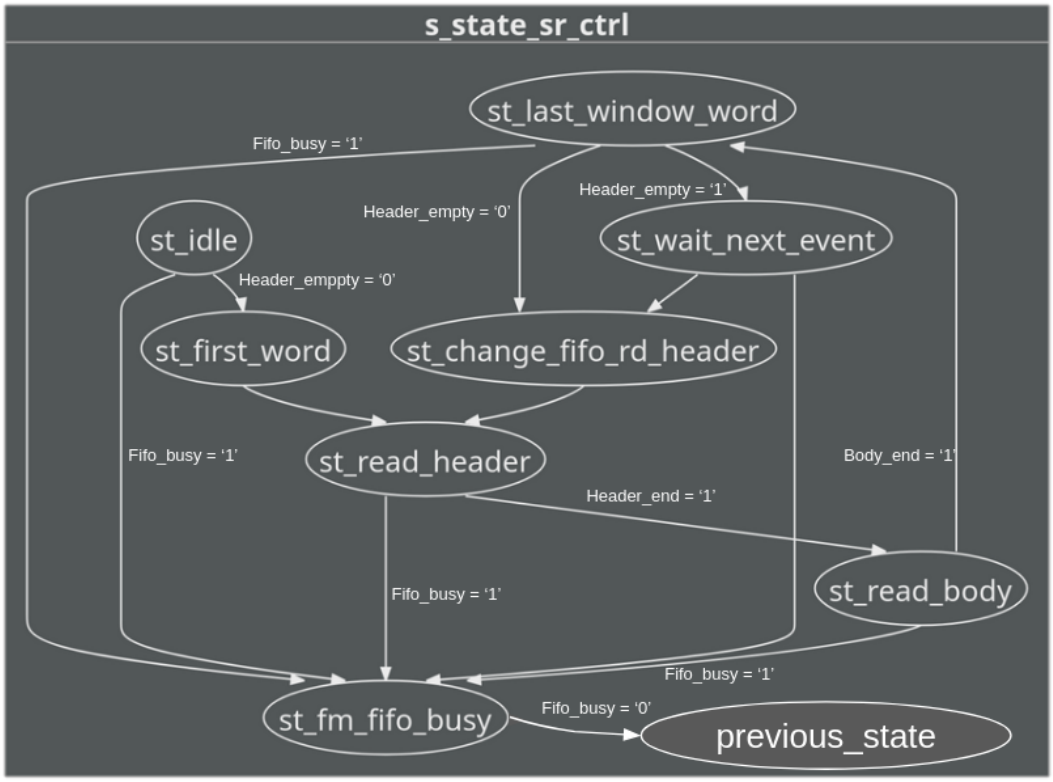


Figure 5.12: Serialize control finite state machine.

The FSM takes into account if there is a busy signal has been propagated back from the following block of the pipeline such that it stop the process of reading the FIFO to the next buffer and prevent it from overflowing. There are also a couple in between states for the first and last words of an event that are necessary for keeping the data flow and the FSM in sync.

FELIX full mode user logic

The FELIX full mode user logic is a component already developed and tested. It was added to the design to interface with the FELIX system. It operates with two clock domains. Both the readout buffer FIFOs which is 160 MHz and 240 MHz which is the frequency that FELIX operates. The full mode user logic uses a 2 to 1 asynchronous FIFO on the two different clock domains that outputs 32 bit words to match the FELIX interface.

Information about the type of data is extracted as they are being processed and they are encoded in the data words that are being outputted by the full mode user logic in a format that is expected by the FELIX system.

Busy propagation

When the full mode user logic FIFO is getting full, a busy signal is asserted. It is propagated back to the serialize control FSM to stop the readout FIFO reading. As the readout FIFOs are not read out, they are filled at a higher rate. A threshold has been set at the index of the memory of the derandomizer FIFOs that signifies that the FIFO is getting full. When the FIFO gets full to this threshold, the busy signal is asserted and outputted such that the CTPCORE will not allow any more triggers and the FIFO gets overflown. There is a delay between the busy assertion and the pause of the trigger. For this reason, the threshold must be set at an early enough stage such that the FIFO can accommodate the events that will be received until the triggers are stopped.

As the input bandwidth of the readout buffer is higher than the output bandwidth, the question of whether the readout buffers are getting full during the operation is raised. An even more strict requirement that needs to be satisfied is that the busy cannot be asserted for longer than 0.1% of the time. As the data rate is dictated by the trigger rate, the need for a simulation that could replicate the conditions of a normal LHC run, as well as, corner cases and worst case scenarios was considered necessary.

CTP readout testbench

A test harness has been developed to evaluate if the requirement of the 0.1% of busy time is met. Figure 5.13 shows a block diagram of the test harness. The OSVVM [30] library has been utilized to generate random input for testing. It also offers more advanced logging that helps with validation and debugging.

Test control

The test control block configures the testbench. It sets up the test and chooses the mode of the data generator and the sequence of the different types of input modes.

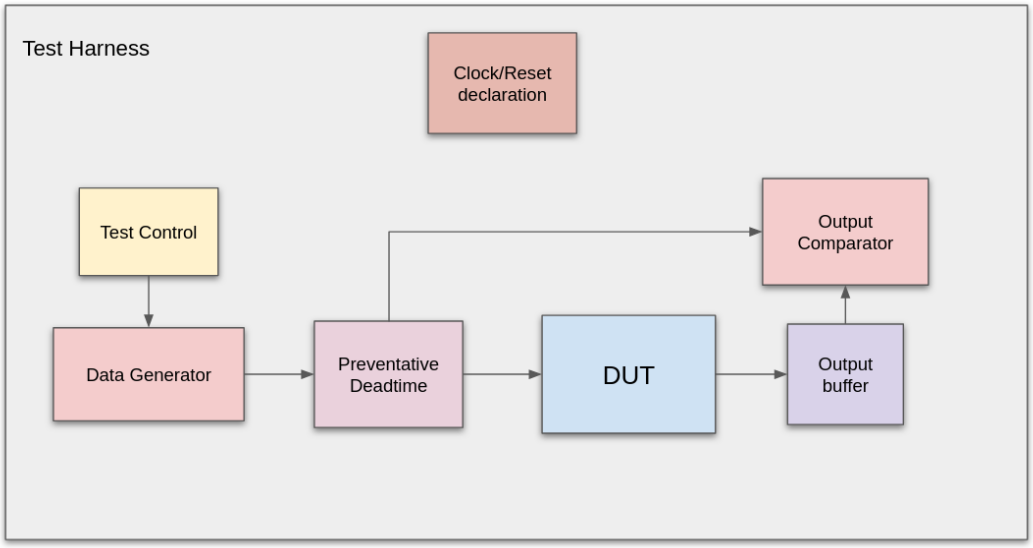


Figure 5.13: Test harness block diagram.

Data generator

The data generator produces the input for the design under test. There are different modes of data generation.

Random trigger: Generates a random trigger with a configurable average rate.

Deadtime assertion: Generates triggers that will target the assertion of the chosen deadtime requirement. The trigger generation is done in a way that will only trigger the deadtime requirement set to assert, without asserting stricter deadtime assert conditions.

Continuous trigger: In this mode, the L0A is set to '1'. This mode is used for debugging and for stressing the design under test.

Bunch group pattern: Masks the L0A that is generated with a bunch group pattern that can be imported from a csv file. This is used to demonstrate more realistic scenarios.

Preventative deadtime

The preventative deadtime block applies the preventative deadtime conditions. It is developed in a generic way such that a new condition can be easily set up from a

configuration file. The conditions that might need to be set are used for the test cases that used bigger readout window sizes.

Design Under Test

The Design Under Test (DUT) is the instantiation of the Readout component described in the firmware section.

Output buffer

The output buffer receives the output from the DUT. It is an FSM that decodes the information and constructs the words back to the format produced by the data generator. The reformatting of the output facilitates the verification and debugging. In addition, it helps with the automation of the validation by the output comparator.

Output comparator

It receives the events from the data generator that have been filtered by the preventative deadtime block. It temporarily stores them until they are compared with the output fo the output buffer.

Validation and testing

The Questasim 2023.1 [36] simulator is used for running the simulation. It is used to visualize and debug simulation results through waveforms. The horizontal axis represents the unit of time. Listed on the left are the signals that are visualized. Figure 5.14, shows the waveform from a test validating the format of the input that is configured by the test control to produce continuous L0A. The signal on top is the 40 MHz clock pulse. Further down is the L0A, which is the input of the component that introduces the deadtime. The signal "pdt_l0_a_or_i" is the same signal after the deadtime has been applied. The configuration generates L0As that follow the same pattern as a bunch group pattern that was used during data taking in Run 3. The two signals at the bottom are the header and the payload of the events.

The left most pat of the waveform shown in figure 5.15 shows a test with the same conditions as figure 5.14 that is configured to be followed by a specific L0A pattern that will trigger only the looser preventative deadtime requirement.

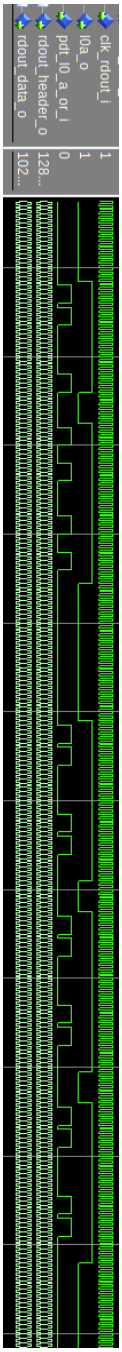


Figure 5.14: Continuous L0A simulation waveform. Run 3 bunch group masking applied.

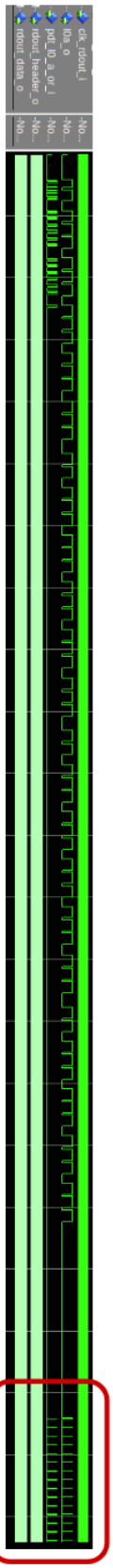


Figure 5.15: Continuous L0A simulation waveform followed by requirement trigger pattern. Run three bunch group masking applied.

Several tests have been performed to model the rate that the derandomizer FIFOs are filled. The test take into account both the window size of the normal data taking, which is 3 events, meaning that along with the event that generates the L0A, both the event before and the event after that are read out, and a window size of 5 that may be used for testing and calibration. Another parameter that is used in the tests is the masking of the input L0A with the Run 3 bunch group pattern. Nevertheless, the design has been tested even without the group pattern masking to test corner cases and find the limits of the system.

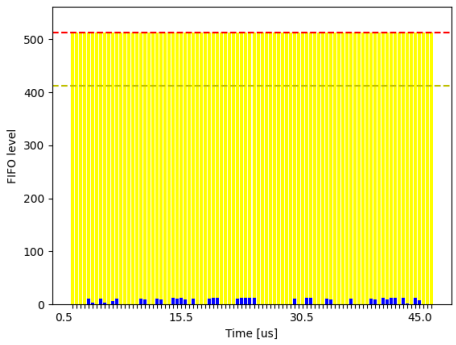
Figure 5.16a shows a plot of the FIFO level with an input of L0A of 5 MHz which results in an average L0A output rate of 1.3 MHz, which is approximately the rate that the CTP has to support during data taking. The red dashed line indicates the size of the FIFO, while the yellow dashed line shows the level of the FIFO where the Busy signal is asserted. Each yellow bin contains the maximum level that the FIFO has reached during the simulation, which is shown in blue. The bins where the Busy signal is asserted are depicted with orange instead of yellow. From the plot, it is evident that the FIFO level is significantly lower than the Busy assertion level that is set at the index 412, which is 100 bits lower than the maximum capacity of the FIFO.

The two following figures 5.16b, 5.16c repeat the same test and increase the average frequency of the L0A to 6 MHz and 7 MHz. The two input rates result in an average 1.38 MHz and 1.4 MHz output rate, approximately. The FIFO level remains virtually the same as the bunch group pattern, and the preventative dead time regulate the rate with which the FIFOs are being filled.

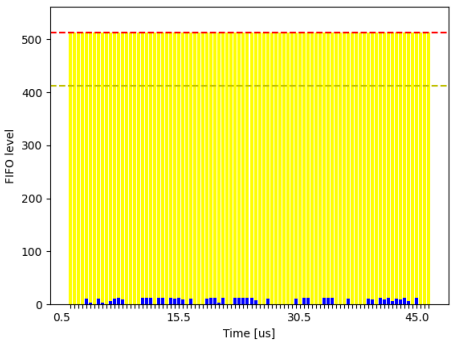
Figure 5.16d is the plot showing the FIFO level for random input with a frequency of 6 MHz with an event window of 5. It is evident that the occupancy of the FIFO is higher. However, it still remains significantly lower than the Busy trigger level.

The last test that is being presented is a test for an event window of 5 and continuous L0A. In this case, there are no restrictions. The purpose of this test is the verification of the Busy generation and whether triggering at the position 412 gives the system enough time to reduce the FIFO level before overflowing. Figure 5.17 shows the plot that verifies Busy generation and that the system stops accepting new events. In addition, it is evident that the level of the FIFO where the Busy signal is triggered is low enough such that the FIFOs don't overflow.

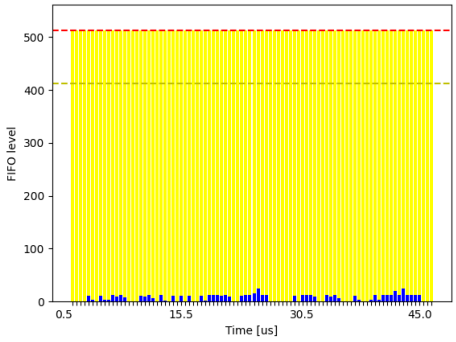
The tests presented above have been repeated for longer simulation times to verify with higher certainty that the Busy signal remains less than 0.1%. The following Table 5.1 summarizes the tests and shows the percentage of the Busy assertion during



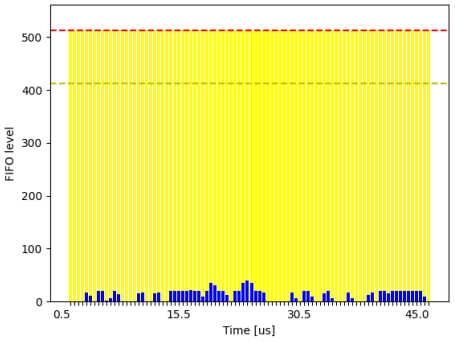
(a) FIFO level with an average of 5 MHz L0A and an event window of 3.



(b) FIFO level with an average of 6 MHz L0A and an event window of 3.



(c) FIFO level with an average of 7 MHz L0A and an event window of 3.



(d) FIFO level with an average of 5 MHz L0A and an event window of 5.

Figure 5.16: Derandomizer FIFO filling patterns for different input frequencies and register windows.

a full test.

The Busy percentage in the data taking conditions was never asserted during the longer simulations. This means that the developed design satisfies the set requirements. The FIFO depth is the lowest that can be configured, as the FPGA vendor sets it to a minimum of 512 address deep. Thus, the design is tested with the minimum memory that can be allocated and no further decrease in memory usage can be achieved.

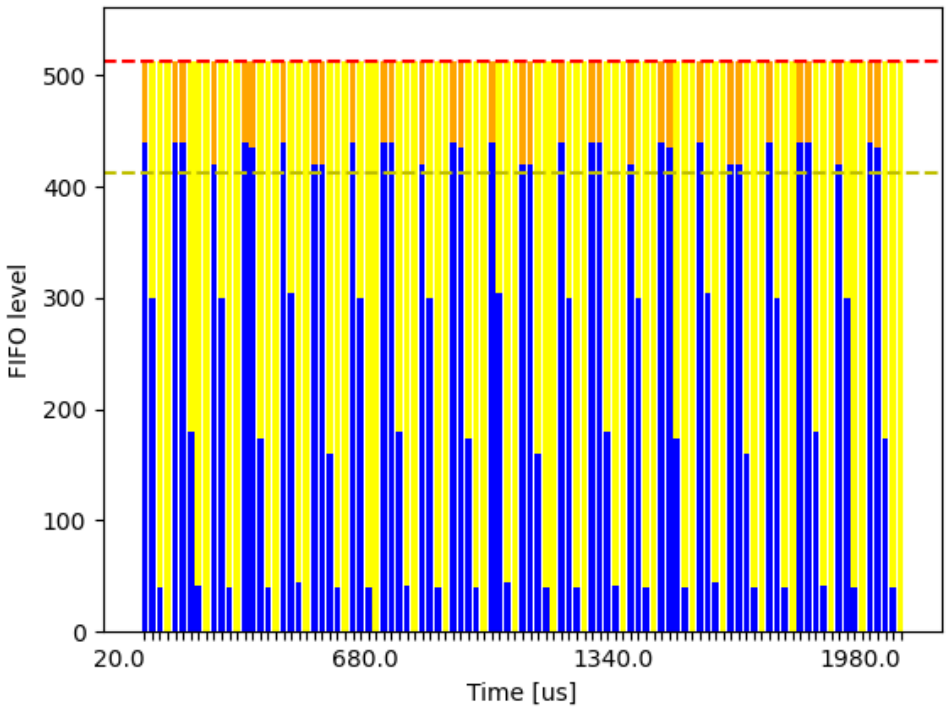


Figure 5.17: FIFO level for continuous L0A and an event window of 5.

Table 5.1: Busy signal percentage for test with various conditions.

Frequency	Window	Busy assertion
5 MHz	3	0%
6 MHz	3	0%
7 MHz	3	0%
5 MHz	5	0%
40 MHz(No BG pattern)	5	2%

5.4 Conclusions

In this study, extensive firmware investigations were conducted to ensure that the CTP readout system meets the requirements of LHC Run 4. The functionality of the system was evaluated, focusing on its ability to handle high data rates, implement deadtime algorithms, and ensure integration with FELIX. Key features, such as the prevention of buffer overflows, the regulation of L0A signals through programmable deadtime mechanisms, and robust handling of BUSY signals, were validated. Test-bench simulations were performed to assess the firmware under realistic and extreme conditions. The tests verified compliance with the defined constraints, including the maintenance of less than 0.1% deadtime. These results confirmed that the proposed firmware design is capable of achieving the required performance required for Run 4.

Chapter 6

PRM HTT firmware development and hardware testing

This chapter presents the collaborative efforts of the firmware development team for the Pattern Recognition Mezzanine (PRM) board, which is a component of the ATLAS *Hardware Track for the Trigger* (HTT) system. As part of this team, I contributed to the development of several firmware modules, including the data generator, the output comparator, and the input block that prepares the data format for further processing.

In addition to firmware development, I conducted hardware tests to validate the performance and functionality of the PRM board. This work was carried out as a research and development proof of concept, laying the groundwork for future advancements in high-speed data processing for particle tracking in the ATLAS experiment.

6.1 Hardware Tracking for the Trigger

In the interest of making use of the information received from the new upgraded tracking detector, the ATLAS trigger system is posed with the challenge to make use of the ITk information as early as possible in the trigger selection. The original baseline scenario to achieve this goal is a system that uses associative memory blocks and FPGAs, the HTT.

The HTT performs tracking both regionally and globally. The regional HTT (rHTT) system provides fast, localized tracking in specific regions of the detector. It processes smaller regions of the detector to reconstruct particle tracks more quickly,

focusing on areas where there is already a known region of interest (ROI), such as areas with identified particles like electrons, muons, or jets. The global HTT (gHTT) system provides high-resolution, full-scan tracking for the entire detector. It performs a more detailed track reconstruction across all regions, covering all possible particle interactions in the detector, not just in localized regions of interest.

The rHTT system is used to track particles produced by the high-energy collisions at the full data acquisition rate of 1 MHz at the Level-0 trigger. It uses a reduced set of layers from the Inner Tracker (ITk), which allows the sufficient tracking precision to identify tracks down to energies of 2 GeV. Even though the rHTT can detect tracks that confirm the presence of objects like electrons, muons, and tau particles, this reduced layer usage limits the ability to perform b-tagging, that would require more precise track resolution by involving more detector layers.

A core use case of the gHTT is b-jet finding. This is important in physics analyses involving flavor specific decays such as Higgs boson decays, supersymetric (SUSY) searches, and exotic decays. In addition, the gHTT is capable of identifying additional soft jets. These are jets with low momentum. They are important for certain physics analyses, such as Higgs boson decays or multi-jet events in high-multiplicity triggers.

HTT architecture

Fig. 6.1 shows an overview diagram of HTT system architecture. The HTT is organized as an array of independent tracking HTT units. The main building block for this architecture are ATCA boards called *Tracking Processors* (TPs). Depending on the mezzanine mounted on each board, the TP are divided into two different types of processors with different functionalities. The TPs that host a *Pattern Recognition Mezzanine* (PRM) are the *Associative Memory Tracking Processors* (AMTPs) and the TPs with a *Track Fitting Mezzanine* (TFM) are the *Second Stage Tracking Processors* (SSTPs). HTT is partitioned such that each TP is equipped with six AMTPs and one SSTP. The partitioning ratio, six to one, allows the balanced distribution of the data to each TP. A set of dedicated servers, the *HTT Interface* (HTTIF), propagates the data from the network. The data that are processed by the HTT are ITk hits, transferred by commodity network.

The HTT is capable of performing regional tracking in 10% of the detector at 1 MHz, resulting in the Level-0 output rate reduction to 400 kHz. The HTT $\eta - \phi$ region differs from the *Region of Interest* (RoI) concept, as it comprises all ITk

detector elements intercepted by tracks with track parameters η and ϕ falling within a specified range. These tracks must have p_T above certain thresholds (2 GeV for rHTT and 1 GeV for gHTT), with $|z_0| < 15$ cm and $|d_0| < 2$ mm. Consequently, adjacent HTT $\eta - \phi$ regions in ϕ share ITk detector elements due to track curvature, while those adjacent in η also share elements because of the track z_0 range they cover. Figure 6.2 shows the ITk layers that are used for the first and second stage fitting.

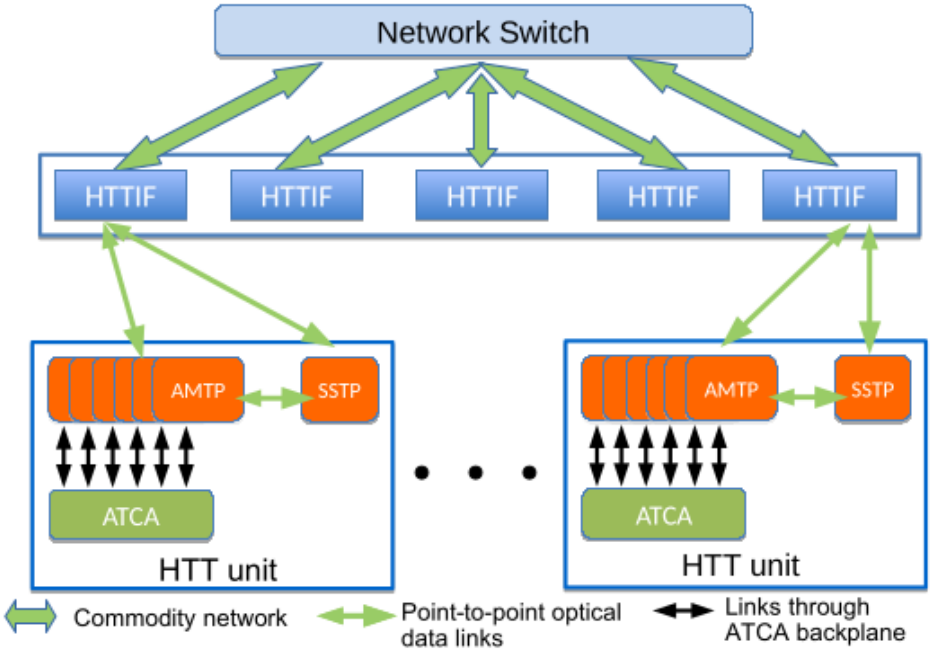


Figure 6.1: HTT system architecture.

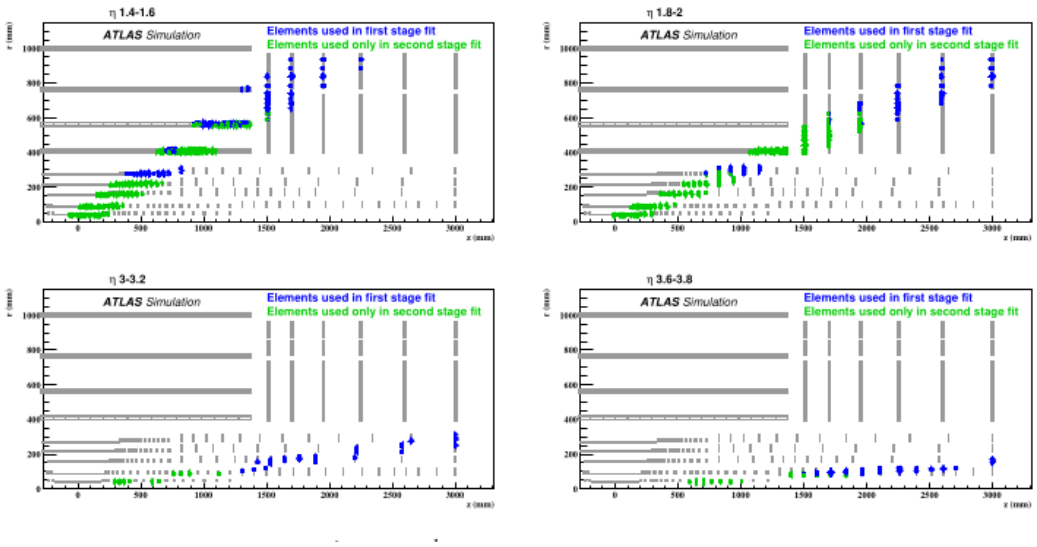


Figure 6.2: ITk layers used for 1^{st} stage fitting in blue and 2^{nd} stage fitting in green in four η-regions.

Functional description of HTT

The HTT pattern recognition method builds upon the existing FTK [24] approach developed for the Run 2 ATLAS Inner Detector. It is adapted to the ITk configuration and the increased pile-up levels expected at the HL-LHC. This technique involves creating template patterns from numerous single-muon tracks in simulated training events, employing coarse-grained *Superstrip IDentifiers* (SSIDs) across multiple layers of the ITk. These template patterns, known as pattern banks, are crucial for efficient track reconstruction.

In the hardware setup described in the previous section, these pattern banks reside in Associative Memory (AM) ASICs within the PRM, where the pattern matching takes place. Each pattern consist of a sequence of eight SSIDs from various detector layers. The selection of layers impacts the number of required patterns, false matches, and the precision of track fitting. The approach treats both sides of double-sided strip staves as distinct layers, while each physical pixel layer accounts for one layer in the pattern bank. The choice of layers depends on the track's pseudorapidity, with specific sets allocated for the barrel, transition, and end-cap regions. At higher pseudorapidities, where strip end-cap layers lack hits, multiple pixel end-cap hits become necessary.

The generation of pattern banks is done by Monte Carlo muon simulations. Each

event undergoes cluster recording from ITk layers intersected by the training muon. Only clusters attributed to the primary interaction, indicated by a bar-code, are considered. Clusters are then transformed into superstrips. The width of each superstrip is configurable during simulation to adjust the number of parameters in the AM ASICs. Duplicate hits within a pattern are ignored, and patterns are not exhaustively generated for all possible combinations. Successful patterns, along with variations in superstrip width, are inserted into sets. The usage count of each pattern informs subsequent steps, including space optimization within the hardware. The removal of patterns with low usage minimally impacts signal efficiency but significantly reduces storage requirements.

First-stage track fitting

The first stage track fitting occurs in firmware. It is executed in an FPGA within the PRM. The PRM is a custom board that hosts an Intel Stratix 10 MX FPGA and four groups of (AM) ASICs that function independently with respect to one another. The process involves taking full-resolution hits from the roads. These hits are passed by the pattern matching. Then, the track parameters and χ^2 of the fit are calculated. The track parameters p_i are determined via linear interpolation

$$p_i = \sum_{j=1}^{N} C_{ij} x_j + q_i \tag{6.1}$$

The constants C_{ij}, q_j are unique per sector. A sector combines one module from each of the 8 layers. These constants are derived from the simulated muon tracks. They match the parameter ranges and distributions of the generated patterns.

The fit quality is assessed using a linearized χ^2 method. It's efficient for FPGA computation. Additional constants ($A_i j$, k_i) are required per sector. Many sectors, each with its set of constants, are needed for fitting a 0.2×0.2 region.

$$\chi^2 = \sum_{i=1}^{N-5} (\sum_{j=1}^{N} A_{ij} x_j + k_i)^2 \tag{6.2}$$

Second-stage track fitting

The second-stage fitting takes place in a TFM. For each track, it computes 5 helix parameters and the χ^2. The TFM receives tracks from 6 PRM cards and hits from unused detector layers.

Two main functions operate on the TFM, the Extrapolator and the Track Fitter. The Extrapolator identifies hits close to PRM tracks. The Track Fitter fits these hits to PRM tracks, checking χ^2. Accepted track candidates undergo duplicate removal before reaching the Event Filter.

The Data Organizer is the first step in the Extrapolator. It builds a hit database to swiftly retrieve hits near PRM tracks. The Extrapolator then uses PRM track hits and sector information to search for hits in other layers. These hits, along with PRM tracks, are fed to the Track Fitter. For each candidate, the Track Fitter calculates the helix parameters and the χ^2 using hit coordinates and sector-specific constants.

The Pattern Recognition Mezzanine

The PRM PCB is designed for mounting onto its ATCA motherboard TP. It features a powerful FPGA, the Intel Stratix 10 MX1, along with 20 AM ASICs. Additionally, an Intel MAX 10 FPGA is installed for implementing the power-up sequence of the Stratix 10 and for board control and monitoring. The PRM connects to the TP via eight Samtec Z-Ray connectors of type ZA8H-24-0.33-7. These connectors provide power and high-speed serial data links, as well as SGMII, JTAG, and other control interfaces.

PRM functional description

A PRM block description is shown in Fig. 6.3. The PRM conducts the initial track reconstruction and selection for the HTT system. It processes data from 8 detector layers, comprising a maximum of four silicon pixels and the remaining silicon strip layers, depending on the ITk detector region. The configuration for this study involves 1 pixel and 7 strip layers. The Intel Stratix 10 MX FPGA serves as the PRM's core, managing communication with the TP and ASICs, and implementing track reconstruction algorithms. The PRM receives clustered ITk detector data via high-speed serial links from the TP using FPGA transceiver blocks, with up to 32 input and 16 output channels. Twenty AM ASICs on the PRM are grouped into four sets of five daisy-chained ASICs, capable of processing different cluster sets simultaneously. These ASICs collectively store 7.864 million patterns, each consisting of eight SSIDs capturing adjacent silicon strip or pixel channels for coarse position identification.

The super-strip width, tuned in simulation, not only optimizes the number of patterns within the AM ASICs but also reduces bandwidth between FPGA and ASICs,

affecting average fit combinations. The FPGA's block diagram showcases the event-based data processing synchronized cluster data converted into SSIDs, then sent to AM ASICs for pattern matching. FPGA-ASIC communication occurs via source-synchronous parallel LVDS buses at 1 Gbps per line.

Within the FPGA, clusters are temporarily stored in a Data Organizer database under corresponding SSIDs. After AM ASIC event processing, FPGA receives matched pattern addresses (roadIDs) for Data Organizer retrieval from High-Bandwidth-Memory (HBM). SSIDs and associated clusters are then processed in the Track Fitter block. This block employs a Principal Component Analysis (PCA) algorithm to determine track parameters within specific sectors identified by sectorIDs stored in HBM.

The Track Fitter relies on multiply-accumulate (MAC) operations, efficiently executed by the DSPs integrated in FPGAs. A χ^2 cut is applied to reject erroneous cluster combinations before computing track parameters, selecting about 8% of tracks. Finally, track parameters, χ^2 value, and associated cluster information are transmitted from PRM to TP via high-speed serial links for each track.

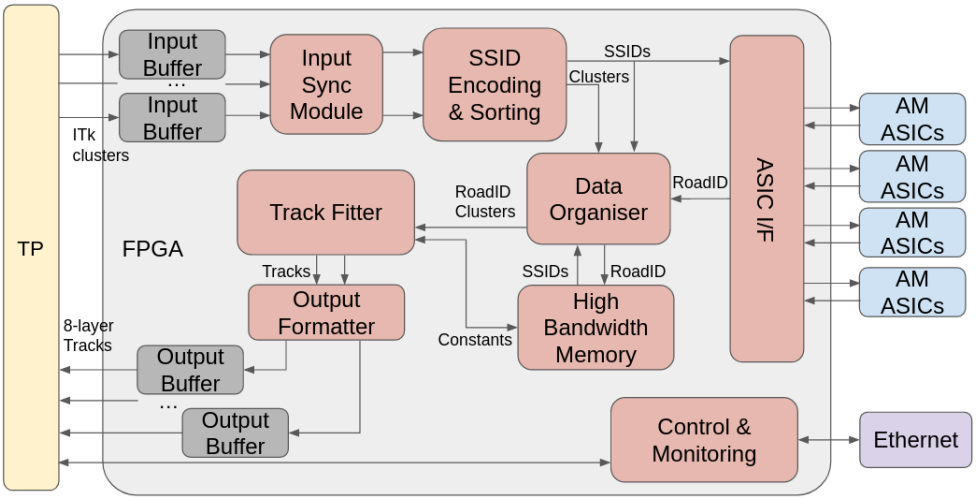


Figure 6.3: PRM FPGA block diagram.

PRM requirements

Given the expected data flow of pile up $< \mu >= 200$ that HTT needs to support, table 6.1 shows an overview of the requirements for the PRM in the TDAQ baseline architecture with HTT in the Event Filter.

Parameter	Requirements per group of ASICs
Peak cluster rate	95 MHz
Roads processing rate	77 MHz
Constants readout rate (χ^2 calculation)	41 MHz
Fit rate (χ^2 calculation)	714 MHz
Constants readout rate (parameter calculation)	41 MHz
Fit rate (parameter calculation)	57 MHz
Track output rate	57 MHz

Table 6.1: PRM requirements for event request rate of 240 kHz [32].

PRM standalone firmware

The PRM firmware is developed and tested in a "standalone" set-up in order to be able to test the PRM firmware without the need of a host tracking processor board and the AM ASICs, that at the time were still under development. The transceivers and the ASIC functionalities are emulated through a Data Generator and an ASIC Emulator block. Figure 6.4 shows the firmware diagram.

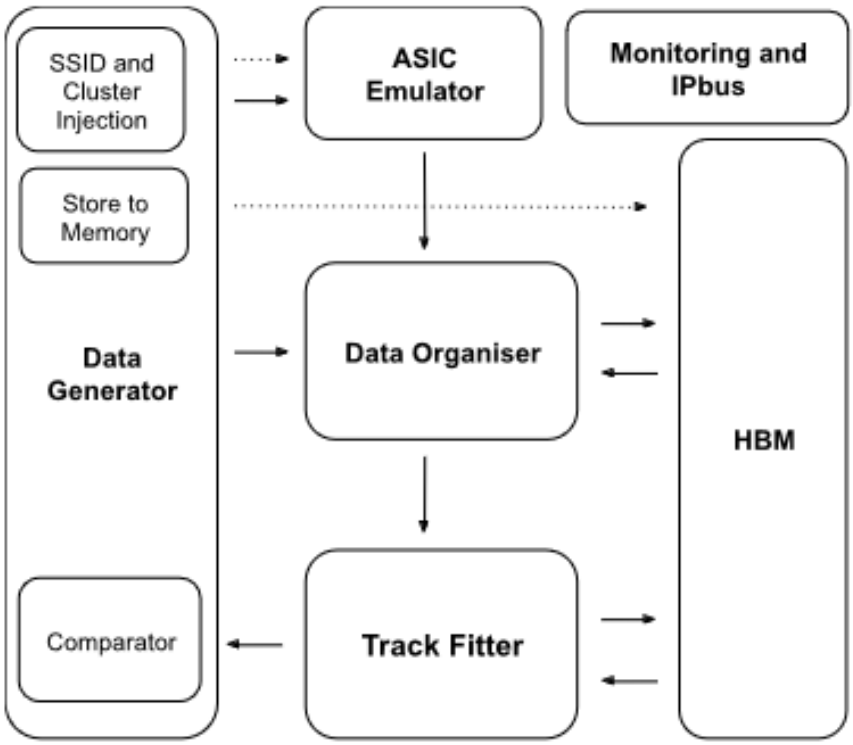


Figure 6.4: PRM standalone block diagram. Solid lines show the data paths. Dashed lines show the data path for the board initialization [32].

The Data Generator block provides different test modes and initializes PRM internal memories with constants. It injects data and checks results. Figure 6.4 displays the block diagram of the PRM standalone firmware that includes the data generator.

The control signals block handles testing modes. Two injection modes exist: with and without the SSID Encoder block. Parameters like injection frequency and event number can be controlled. Fake clusters and SSIDs can be added to emulate real detector behavior.

PRM internal memories initialize with a memory initialization command. Constants from Constants RAM are uploaded into High-Bandwidth Memory. Track patterns are sent to the ASIC Emulator block and HBM.

Injection starts after memory initialization. PRM Input RAM contains clusters in TP motherboard format. PRM Extended Input RAM contains simulated true track χ^2 and helix parameters for comparison.

The ASIC Emulator implements AM pattern matching. It stores 16 patterns per instance, configurable at run-time. A daisy chain of five emulator instances mimics a single group of AM ASICs on the PRM.

The Data Organizer is an on-the-fly database for fast access to cluster position and SSID. Clusters and SSIDs are stored in FIFOs.

The High-Bandwidth Memory stores patterns and constants for track fit computation. FIFO buffers handle clock domain crossing and back pressure.

The Track Fitter assesses track candidates by computing the χ^2 value and extracting track parameters. Constants for χ^2 computation are retrieved from HBM. All constants are in floating point mode.

The TF generates candidate tracks for fit. Chi Square Unit computes the χ^2 value and track parameters. Only 8% of the generated track candidates are expected to pass the selection. Parameter Calculator block computes parameters.

6.2 PRM Firmware

Data Generator

The data generator block is part of the PRM standalone project. Its task is setting up the PRM memories for testing, injecting the input data and verifying the output of the firmware by comparing it with a pre-calculated output. The block diagram of the Data Generator can be seen in figure 6.5

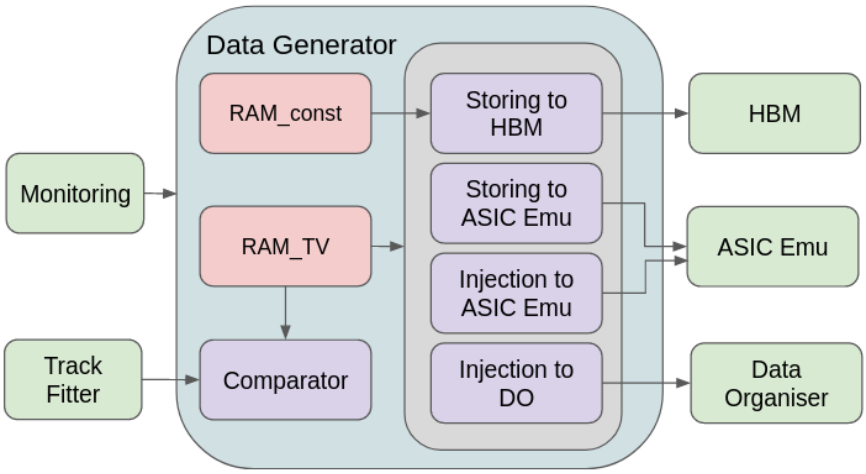


Figure 6.5: Data Generator Block Diagram.

Input data

A set of data is used by the data generator in order to initialize the memories of the PRM firmware and emulate the input from the TP. The data that set up the PRM memories are coefficients for the calculation of the χ^2 and the five helix parameters, as well as the data for the pattern banks along with the pre-calculated χ^2 and helix parameters. The data that are used for the injection, emulate the input from the TP, and they carry the information of the clusters of each Inner Tracker layer.

The files with the information for the χ^2 and the parameter calculation contain both the scalar and the vector coefficients of all the sectors in a single file. Scripts have been developed in order to convert the information from the file and generate a memory initialization file.

The information for the roads meant to be stored in the pattern banks are included in a test vector along with more information such as their identifier, the corresponding clusters and the pre-calculated output.

The final piece of input data is the input of the PRM. It is a test vector that emulates the stream of data propagated from the TP. Along with some overhead information, this test vector contains the cluster information for each layer.

Test vectors with both the road and the PRM input information have been generated by the software simulation team. In addition, logfiles with data for the construction of new test vectors have been provided. Scripts have been implemented also in this case in order to convert the log file information into a memory initialization format.

Tables 6.2 shows the structure of the input stream. The first word of the stream is a header, followed by the cluster modules. Finally, the stream ends with a footer. The inner structure of the modules both for pixel and strips, are shown in the tables 6.3 and 6.4. Both the pixel and the strip modules consist of 64 bit words. The module blocks have two header words and one footer word. The pixel cluster information is 32 bits wide while the strip cluster is 16 bits wide.

Table 6.2: Cluster data stream structure.

Event header
Module #1 Block
...
Module #N Block
Event Footer

Table 6.3: Pixel module block.

bit-63 bit-32 bit-31 bit-0

Module header word #1	
module header word #2	pixel cluster #1
pixel cluster #2	pixel cluster #3
pixel cluster #4	pixel cluster #5 LB
pixel cluster footer	zero padding

Table 6.4: Strip Module Block.

bit-63 bit-32 bit-31 bit-0

Module header word #1			
module header word #2		strip cluster #1	strip cluster #2
strip cluster #3	strip cluster #4	strip cluster #5	strip cluster #6
strip cluster #7	strip cluster #8	strip cluster #9	strip cluster #10
strip cluster #11 LB	strip cluster footer	zero padding	

Data Reading

Three RAM block are instantiated and initialized with the test vectors and the coefficients for the track fitting calculations on compilation time. The data from the RAMs are read out and recognized by the data register block. The register block is a finite state machine that reads the input data stream, recognizes what piece of information each word represents and registers the information to the appropriate VHDL record, to be propagated to the rest of the pipeline. The data register block is capable of handling various sizes of input events and accounting for missing hits.

Memory Initialization

The first task of the data generator is to set up the PRM firmware memories, the HBM and the AM ASICs. The roads are sent to both the HBM and the AM ASICs while the coefficients are uploaded just to the HBM. In the PRM complete project the data are uploaded to the PRM memories over IPBus protocol. The same protocol is used for the PRM standalone project so that it resembles the complete project.

An IPbus timing diagram can be seen in figure 6.6. Strobe indicates the valid data. The ack signal is the reply of the receiving end of the transmission, verifying that the data have been received. Addr is the target address. Write defines the write and not the read mode. Wdata is a 32 bit channel that propagates the data, while rdata is also a 32 bit channel that propagates the data to be read.

The roadID is used as the address for sending the patterns to the AM ASICs as shown if figure 6.7. As the SSID information is 16 bit, two SSIDs are concatenated into a 32 bit word that is sent in a single clock cycle such that the IPbus bandwidth is used more efficiently. An acknowledge signal is being received from the AM ASICs after 4 wdata words corresponding to eight SSIDs have been transmitted.

Roads and coefficients are sent over the same IPbus interface to the HBM. The first set of data, which is referred in the figure 6.8 as PC-MASK, that are being sent during a transmission discriminates between the two types of data. An acknowledge will be received from the HBM only after nine wdata words are sent. The coefficient transmission takes place first. The address in this case is the sectorID shifted by 5 bits leaving the 5 least significant bits free to be used as a counter. This is necessary as the number of coefficients is larger than the number that can be stored into a single address in the HBM. In this way the coefficients from the same sector will be stored in neighboring addresses. The coefficients are 32 bit words limiting to a single

word transmission every clock cycle. The coefficient transmission follows the pattern transmission. As in the case of the AM ASICs, the roadID is used as an address. Apart from the SSIDs, wdata also sends the sectorID.

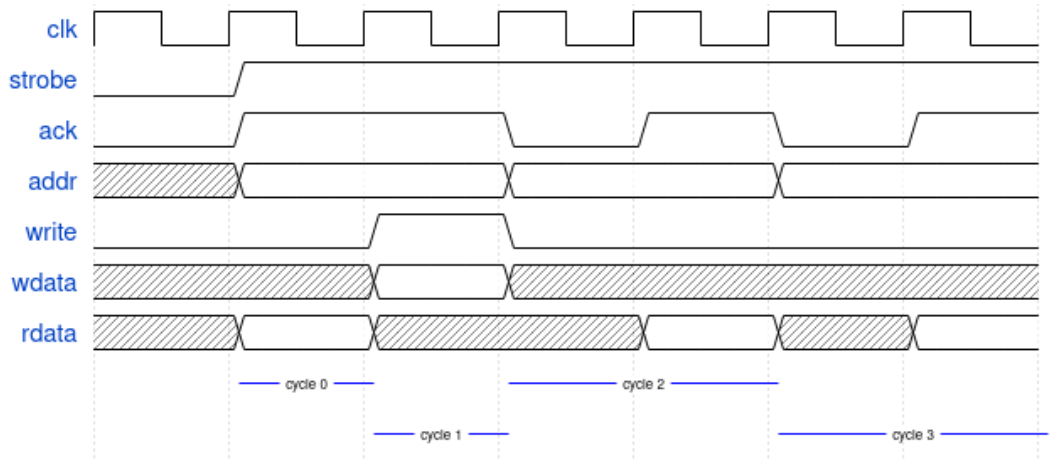


Figure 6.6: IPbus Protocol.

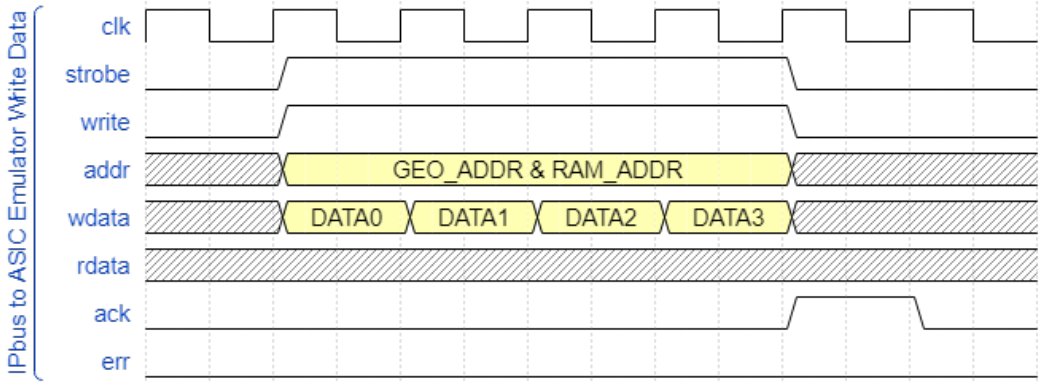


Figure 6.7: ASIC Emulator Initialization.

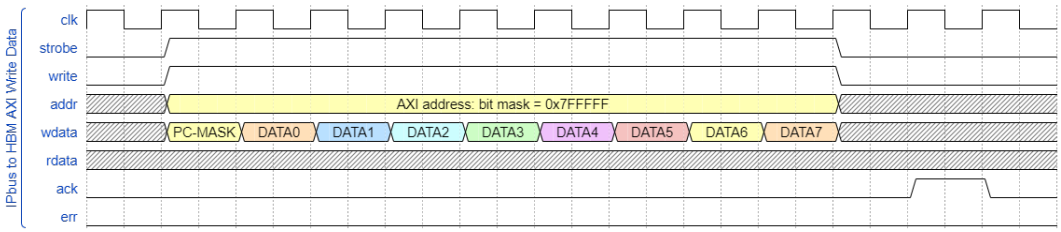


Figure 6.8: HBM Initialization.

Injection

Once the PRM memories have been set up, the data generator can inject the input data. The data are injected towards the DO and the AM ASICs. The eight layers are injected in parallel. As shown in figures 6.9 and 6.10 the beginning of an injection is indicated with an event start signal and the end of an event with the event end signal. The eventID channel propagates the identifier of the event, which is generated by the data generator. The error word channel indicates the validity of the event. Each layer is paired with a valid signal.

Both the clusters and the corresponding SSIDs are injected towards the DO, while only the SSIDs are injected towards the AM ASICs. A restriction that applies only for the injection towards the AM ASICs is that all layers must either be valid or not valid for a given clock cycle. In the case where in one layer there are fewer SSIDs than others, then the last SSID is injected multiple times to satisfy this condition.

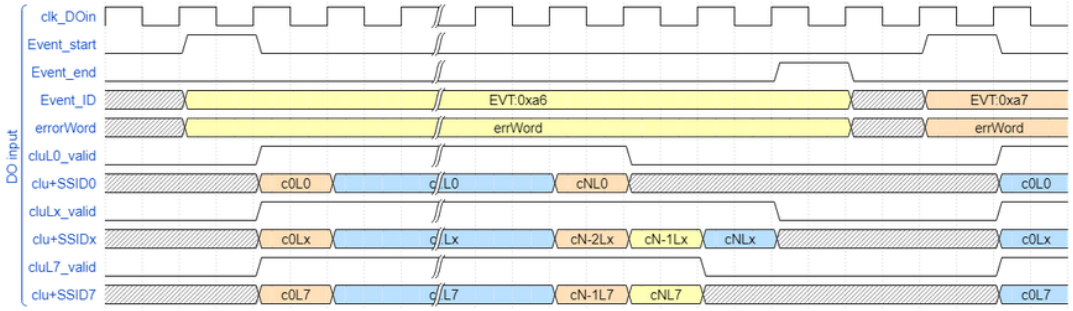


Figure 6.9: Injection to DO.

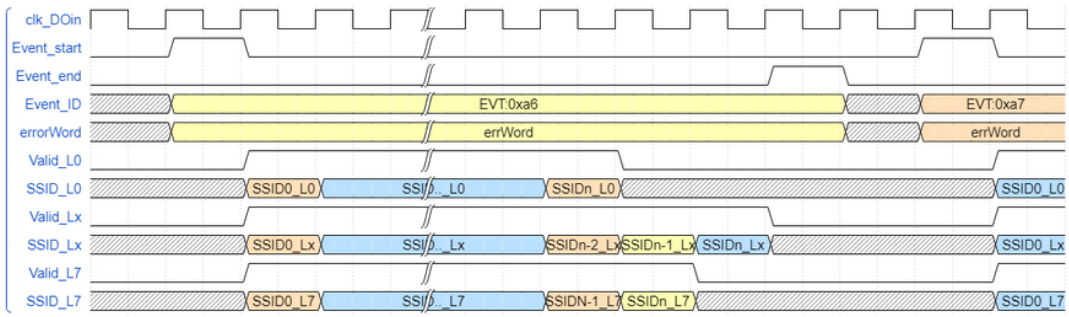


Figure 6.10: Injection to AM ASICs.

Output Comparator

The output comparator is the final block of the PRM standalone firmware pipeline. It receives the χ^2 and the parameters from the track fitter in order to compare it with output calculated by the software simulation. The order in which the events are outputted by the firmware is the same as the order of the injection, while the order of the tracks inside an event may differ. To deal with the random order of the incoming tracks from the TF, the comparator has to compare every single output of the TF with the pre-calculated output of all the tracks from the same event in the test vector.

The test vector is read by the registering block, which assigns the incoming data to the appropriate signals. A RAM is instantiated for the χ^2 and for each one of the five helix parameters. The values of each one of these elements from every track is concatenated and stored in a single address in the corresponding RAM. This is a way to compare every output of the track fitter with the corresponding elements of every track in the test vectors in a single clock cycle. The comparator asserts a flag whenever the output matches the golden data, thus, validating the functionality of the PRM firmware.

The data generator uses very little resources compared to the resources available at the FPGA. The table 6.5 shows the FPGA occupancy.

Table 6.5: Data Generator Resource Utilization.

Total dedicated logic registers	4096
Total block memory bits	17024/140.2 Mb (<1%)

SSID encoder

The SSID encoder is part of both the PRM complete project and the PRM standalone project. It receives the clustered data from the TP and encodes the SSIDs of each cluster. Finally, it groups the clusters per their SSID. A block diagram of the structure of the SSID encoder's firmware is shown in figure 6.11.

At compilation time, it can be decided if it is going to be included in a standalone project in order to be tested. Excluding the SSID encoder from the standalone firmware gives the opportunity to be more agile while testing and gives the opportunity to test the limits of the PRM firmware functionality without the need of testing with big test vector files, which results in limiting the simulation time.

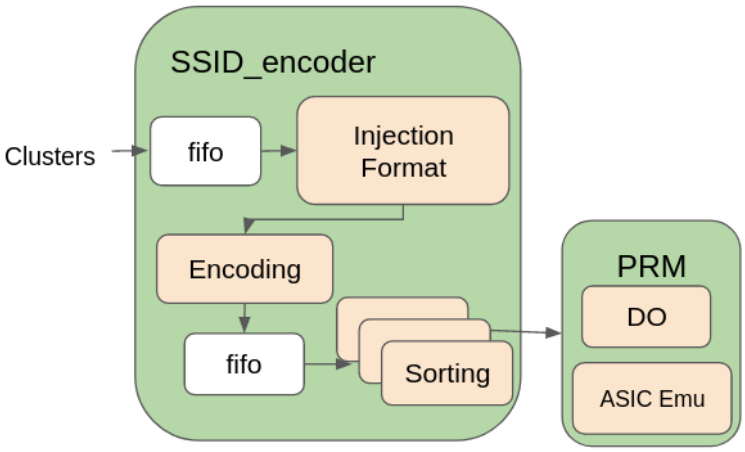


Figure 6.11: SSID encoder block diagram.

SSID Encoding

The output of the TP is buffered before entering the SSID encoder. The SSID encoder:

- reads the stream of the 64 bit words,
- recognizes the type of clusters that are being received and the layer they are coming from,
- recognizes the beginning and the end of an event,
- generates the event start and event end signals needed for the communication between the PRM firmware blocks,
- generates the event ID, which is an increasing number,
- reads the information of each cluster and encodes the SSIDs,
- and finally, outputs each cluster and the corresponding SSID on the same clock cycle and distributes them to the proper layer.

SSID Grouping

The clusters with their SSIDs are propagated to the SSID grouping block. This block groups the clusters per their SSID. This is needed by the Data Organizer block to properly store all the necessary information in its memories. The cluster sorting per SSID helps also the AM ASIC processing, as in this way the repetition of the same

SSID is avoided. An example of a cluster sorting per SSID is shown in the timing diagram in figure 6.12

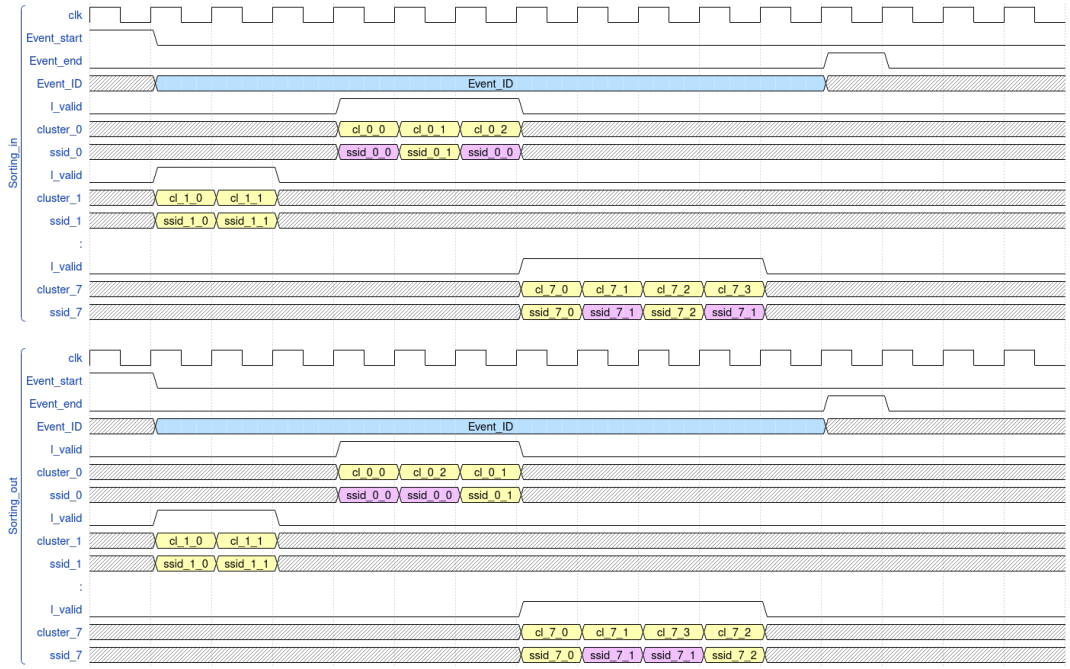


Figure 6.12: Cluster Sorting per SSID.

As part of the PRM complete project, SSID encoding and sorting have to have high throughput and low latency while using a reasonable amount of resources. Different approaches were discussed to achieve high performance. The approach that was selected to be implemented is based on a FIFO where the sorting takes place.

The building block of the sorting design is shown in figure 6.13. A FIFO is used as a buffer at the input of the block. The cluster - SSID pairs that belong to the same module are propagated to the ordering FIFO. The values written to the sorting FIFO are then read out. Each new input value is compared with all the previous values stored in the ordering FIFO at its output by the FSM connected to the sorting FIFO's output. If a match is detected, then the multiplexer lets the new value be stored in the sorting FIFO. Next the multiplexer changes and the output of the FIFO gets written again in the sorting FIFO into the address next to its new matching input. In the case where the values don't match, the outputted value gets written again without writing the new value as in the previous case. The values in the ordering FIFO continue to be read and written until either a match is found or a full circulation of the values in the ordering FIFO has occurred. After a full circulation, the new value is added in the sorting FIFO. Finally, when a full module of SSIDs has been written to the ordering

FIFO, the block moves to the reading process. During this process the FIFO reads out all the values sequentially,

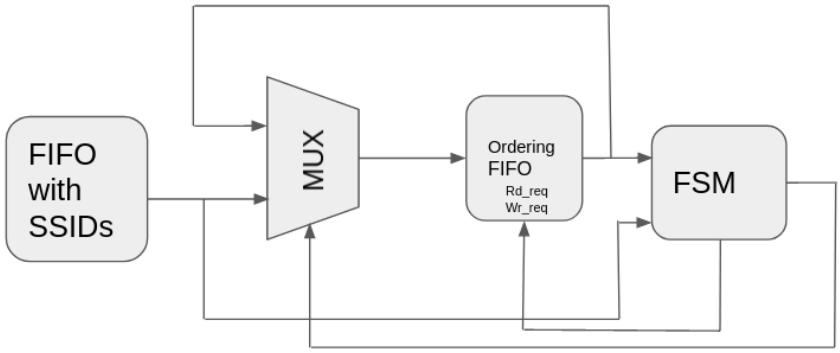


Figure 6.13: Cluster Sorting per SSID Block Diagram.

The sorting block can be instantiated multiple times to accelerate the procedure of sorting, as shown in figure 6.14. Each new value is compared with multiple cluster-SSID pairs already stored in the sorting FIFO. In the case of a match, the new value is stored in the same FIFO as its match into an address next to it. If a match is not found, then the new value gets stored in the FIFO with the least content, keeping the parallel comparison more efficient. During the reading phase, the sorting FIFOs read their content in sequence. Since the content of each FIFO is already sorted in each sorting block, and the same SSID cannot be stored in more than one FIFO, the output remains sorted.

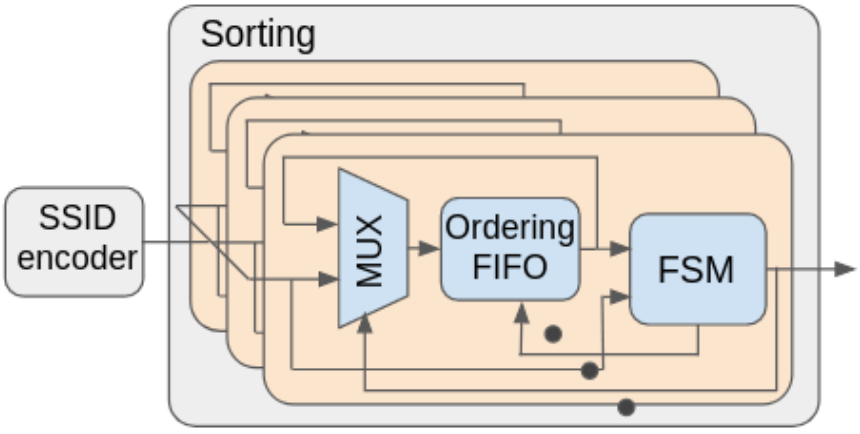


Figure 6.14: Cluster Sorting per SSID in Parallel Block Diagram.

The advantage of this specific implementation is its scalability. The number of sorting modules can be adjusted to keeping the trade-off between performance and

resource utilization at the desired level.

Table 6.6: Resource Utilization.

Arithmetic Logic Units	260/702720 (<1%)
Total dedicated logic registers	5208
Total block memory bits	47.9 kb/140.2 Mb (<1%)
Total RAM Blocks	46/6847 (<1%)

The PRM firmware is required to run with a clock frequency of 250 MHz. The SSID encoding and sorting blocks achieve this timing requirement. Table 6.6 shows the resource utilization of the sorting block when there is only one instance of the sorting building block. Studies have been conducted in the context of the Technical Design Report amendment [19] showing that, for single muons in pile-up 200, an average of 272 clusters per layer would be expected in each event and approximately 1.2 of them match with a Road in the AM ASICs.

Every instance of a sorting building block utilizes 165 arithmetic logic units and, 13312 block memory bits. The PRM has been tested with one and two sorting blocks instantiated. The latency of the event sorting relies heavily on the size of the event, as the sorting block has to parse through all the input values and compare them with all the previous ones before finalizing the sorting.

For an event containing two modules, one with 6 clusters and one with 11 clusters, the latency of the sorting process is 528 ns. The same test, conducted with two sorting blocks, gives a latency of 424 ns, improving the latency by 20%.

Simulation

Before the verification on the hardware, the project was simulated in Questasim [36]. TCL scripts are used to simulate the VHDL code, as well as the IP core emulation files that were used for the substantiation of various types of memories by the Vivado suite. Simulation is a very important step before testing on hardware, especially for bigger projects. It not only facilitates the firmware development, but also, helps with the debugging of the full project as it allows the developer to view internal signals throughout the simulation timeline, something that is not possible during the hardware testing.

Questasim waveforms that verify the functionality of the data generator and SSID sorting block are shown in figures 6.15, 6.16, 6.17, 6.18, 6.19, and 6.20. These waveforms confirm that the data generator accurately produces the expected data patterns under simulated conditions, and the SSID sorting block correctly organizes and prioritizes these patterns as intended. This validation demonstrates that the implementation meets the design functionality requirements, ensuring compatibility with downstream processing stages.

6.3 Hardware Testing

The firmware's verification in simulation follows the verification on hardware. As the standalone project doesn't make use of the hardware components of the PRM, it can be tested on a commercial board with a Stratix 10 FPGA. Hardware testing is a very important part of the verification, as the simulation tools used to test the functionality cannot fully replicate the real-world conditions and timing constraints of the hardware environment.

IPbus Protocol

The IPbus is protocol described in section 6.2 is a packet-based control protocol designed to read and to modify the memory–mapped resources of FPGA hardware devices supporting A32/D32 bus.

Hardware

The board used for the PRM hardware test is the Stratix 10 MX devkit - 1SM21BHU2 F53E1VG. The device is programmed via jtag. The software provided by Intel for developing and programming the device is Quartus Prime Pro 19.2.

A set of input registers has been defined to control the behavior of the application. The list of registers can be seen in figure 6.21. The input can be set by a Linux machine over Ethernet communication. A conversion from Ethernet to QSFP is required, as the board doesn't feature an Ethernet port. The transformation of the signal is done using a converter, which is a separate hardware module. Figure 6.23 is an image of the setup, while figure 6.22 shows a block diagram with the interconnections between the different pieces of hardware.

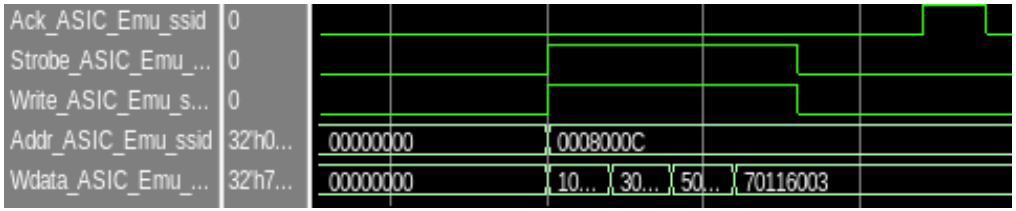


Figure 6.15: Initialization of the ASIC Emulator over IPBus protocol. The waveform depicts the control signals, the data and the acknowledge from the ASIC Emulator.

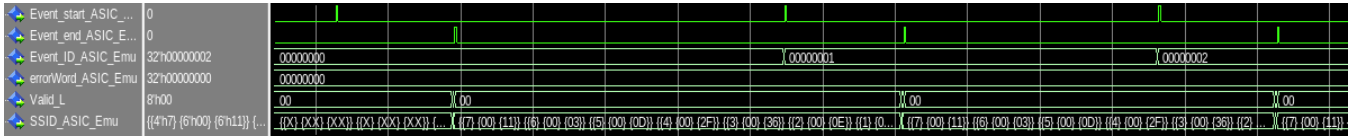


Figure 6.16: ASIC Emulator injection. The waveform depicts the start and end injection signals, the data and the metadata injected to the ASIC Emulator.

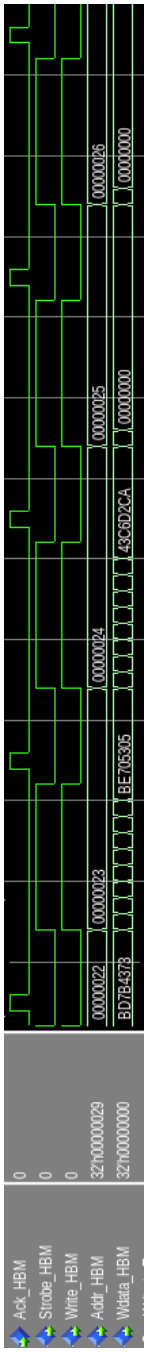


Figure 6.17: Initialization of the High Bandwidth Memory. The waveform depicts the control signals, the data and the acknowledge from the ASIC Emulator.

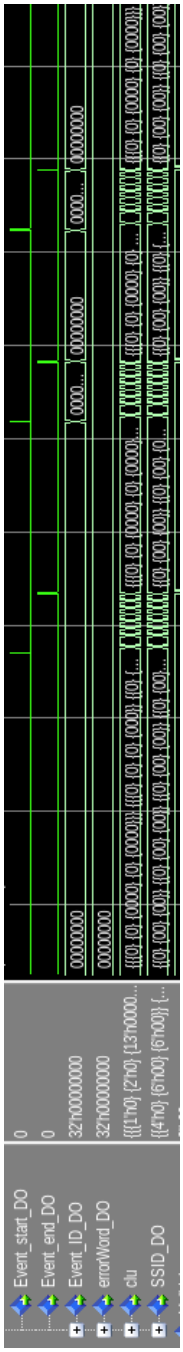


Figure 6.18: Data organizer injection. The waveform depicts the start and end injection signals, the data and the metadata injected to the ASIC Emulator.

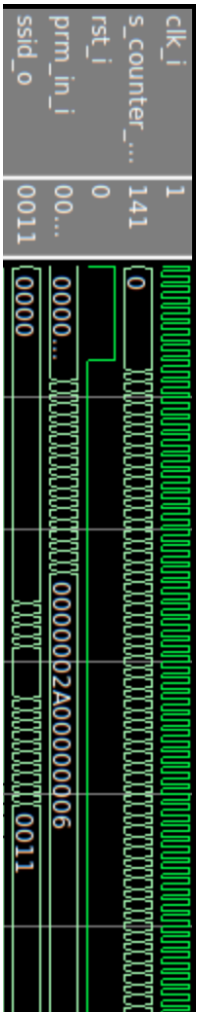


Figure 6.19: SSID sorting.

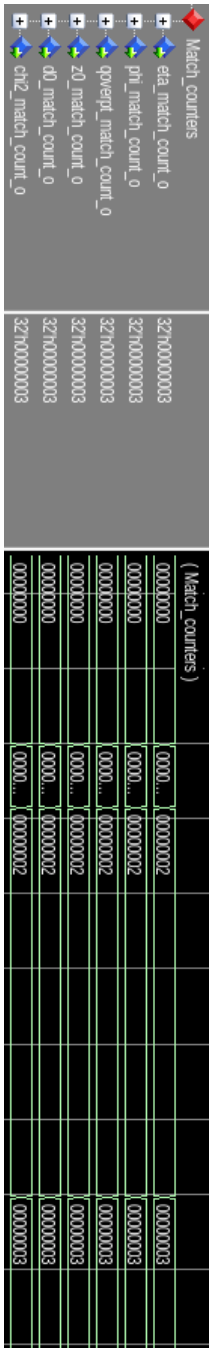


Figure 6.20: Output comparator.

Variable	Address	Bit-by-bit	Register mode	Comments
Start	0x00000012	Bit(0)	Write/Read	
Stop	0x00000012	Bit(1)	Write/Read	
Reset injection	0x00000012	Bit(2)	Write/Read	
Reset inject flag	0x00000012	Bit(3)	Read-only	
Fakes enable	0x00000012	Bit(4)	Write/Read	
Injection delay	0x00000012	Bit(14 downto 5)	Write/Read	
Max num of events enable	0x00000012	Bit(15)	Write/Read	
Max num of events	0x00000012	Bit(24 downto 16)	Write/Read	
Comparator valid	0x00000012	Bit(25)	Read-only	
Chi2 match	0x00000012	Bit(6)	Read-only	

Figure 6.21: List of control registers.

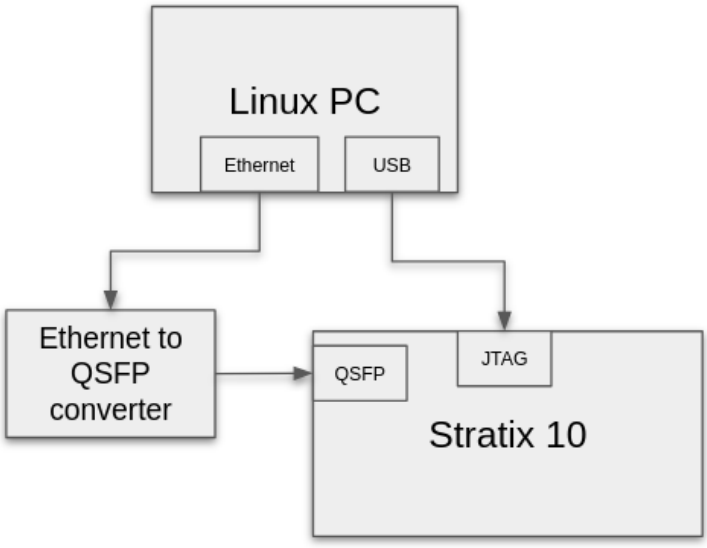


Figure 6.22: PRM developer's kit setup block diagram.

Figure 6.23: Hardware Testing Setup.

Setting up the hardware

The Vivado software suite allows for the initialization of the instantiated RAMs in the project at synthesis time. The RAMs that are used in the PRM are IP cores that are provided by the vendor and can be configured from a configuration wizard. The bitstream generated during the project compilation is uploaded to the FPGA platform over jtag. Figure 6.24 is a schematic view of the dual port RAM used in the design to store the values of the inputs that are used for the tests.

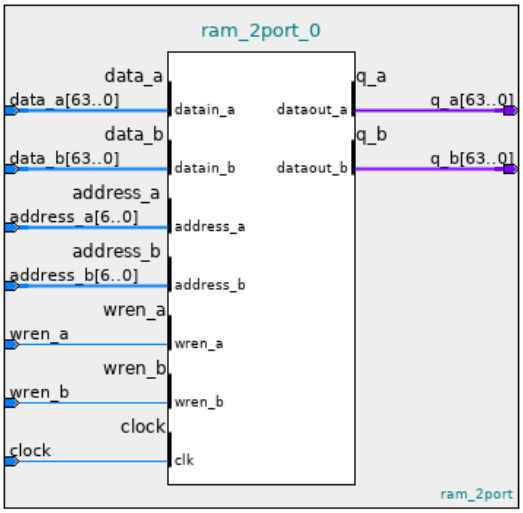


Figure 6.24: Dual port RAM Intel IP core input and output configuration.

The values that initialize the memories are converted to hex format, which can be digested by the software tools. The memory initialization values can be seen listed in the Quartus configuration tool, which is shown in figure 6.25.

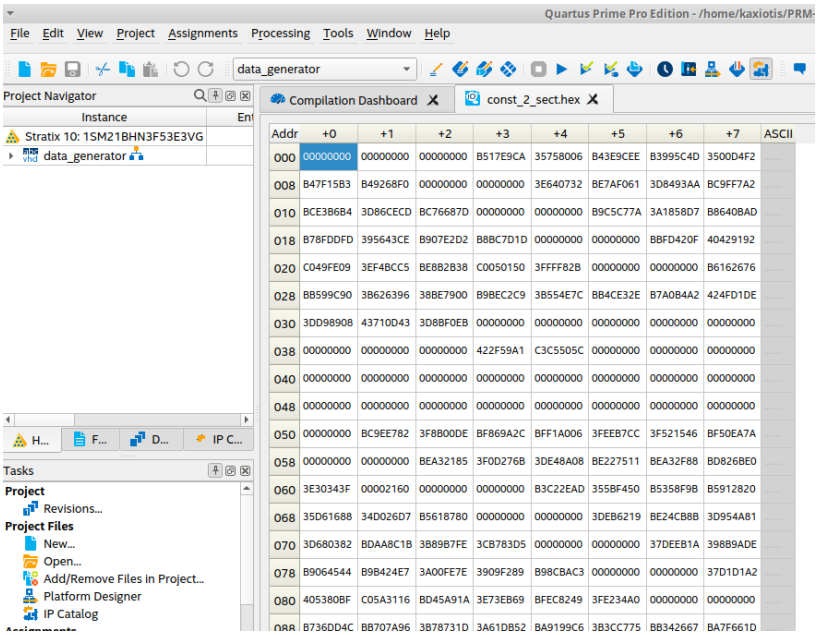


Figure 6.25: Initialization values list of the memory IP core.

Signal Tap

Both input and output as well as any internal signal of the firmware can be monitored using Signal Tap. Signal Tap is a tool featured in Quartus software. Tests were performed with the available Test Vectors and the system recognized the tracks and calculated χ^2 and the helix parameters just like the simulations done on QuestaSim. Figure 6.26 shows the waveform created by the Signal Tap that shows the counter of the output values generated by hardware that match the expected values.

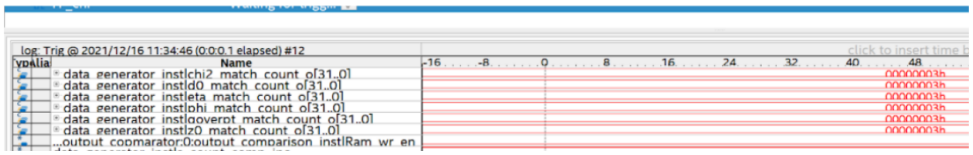


Figure 6.26: Hardware Test Output

6.4 Conclusions

In the context of the HTT project, a custom hardware-based tracking system was developed, utilizing associative memories for pattern recognition and FPGAs for data organization, communication, and track fitting. Although the project was ultimately discontinued within ATLAS, extensive studies were conducted to evaluate its feasibility through hardware demonstrators and simulations. Within the HTT system, the PRM played a key role by integrating pattern matching and track fitting functionalities.

The PRM firmware faced significant challenges, including achieving high processing rates and maintaining low-latency requirements. To address these requirements, the development of an innovative SSID Encoder was undertaken. This module converts the cluster data into SSIDs and efficiently groups the clusters per their SSID, playing a role in enhancing the overall performance of the PRM firmware.

This work outlines the design and validation of the PRM firmware, which was tested in simulations and on an Intel Stratix 10 MX development board. The data generator and the output comparator allowed the validation of the PRM standalone project. Resource utilization measurements confirmed compatibility with the requirements of a complete PRM board. While the full design specifications set for the PRM were not entirely met, this study demonstrates the feasibility of achieving the neces-

sary functionality with modern FPGA technologies equipped with High-Bandwidth Memories.

Chapter 7

Testing FPGAs as CPU accelerators

This chapter explores the performance and testing of heterogeneous systems within the TDAQ framework. The work presented includes testing of hardware-software co-designed components, simulation-based validation, and hardware testing to evaluate system robustness. Key focus areas include the performance of firmware and software modules on heterogeneous platforms.

7.1 Heterogeneous computing and Alveo devices

Updates on Track Reconstruction in Software

The current status of the ATLAS detector is the result of a research and development initiative to design and implement the ITk layout and enhance the Phase-II track reconstruction software. In parallel, a novel track reconstruction method that is tailored to high pile-up events in the ITk has been developed [45]. This method leverages the precision and redundancy of the five-layer Pixel System. It begins by identifying track candidate seeds from triplets of hits within the Pixel layers, using the additional layers to efficiently discard fake combinations before initiating the more computationally demanding combinatorial Kalman filter-based track finder for each seed [43]. These developments create space for developing new tracking methodologies, as well as, making use of new technologies.

The industry trends show that a CPU is no longer the only unit of computation for many workloads such as data streaming, analytics, and artificial intelligence applications. In order to meet rack space, electrical power, and cooling constraints, the heterogeneous systems integrate a variety of computational units, including multi-

core CPUs, GPUs, DSPs, FPGAs, and ASICs. This allows the systems to perform necessary computations more quickly (lower latency) and/or achieve higher performance with lower power consumption. The idea of a heterogeneous commodity system makes it easy to think of exciting new technologies in the future. While FPGAs have been used for many years for low latency applications in a variety of domains, their role as accelerators within data centers is relatively new yet rapidly expanding. There are multiple examples of applications that accelerate machine learning inferences with FPGAs such as the Configurable Cloud-Scale DNN Processor for Real-Time AI [50] and the *Heterogeneous Accelerated Compute Cluster* (HACC) [51] which is an initiative to support novel research in adaptive compute acceleration for high-performance computing. FPGAs, in the context of data centers, are used as accelerator cards that connect to a server's PCIe slot. In parallel, GPU performance has seen significant advancements, supported by modern compilers that allow complex, feature-rich code to be executed on GPUs.

Looking ahead, unified memory architectures represent a promising area of innovation. These architectures could integrate various processing units, such as CPUs, GPUs, and FPGAs, onto a single chip that shares high-speed memory. The continuous evolution of programming interfaces for diverse hardware for CPUs, GPUs, and FPGAs, further enhances the appeal of heterogeneous systems.

FPGA accelerator platforms

FPGAs present several benefits over traditional CPU/GPU acceleration, including a customizable architecture that can implement any function executable on a processor, leading to improved performance with reduced power consumption. Unlike processor architectures, the programmable logic (PL) fabric in AMD devices allows for a high level of parallelism during application execution.

The following key principles are essential for developing accelerated applications on FPGAs, offering significant performance enhancements over CPUs by introducing parallelism and fixed latency:

- Applications designed for CPUs differ substantially from those for FPGAs, necessitating a rewrite of the functions intended for FPGA acceleration. While CPUs execute functions sequentially, FPGAs require inferred parallelism for optimal performance.

- To accelerate an application, the software is divided into a host application that runs on the CPU and compute functions, or kernels, that operate on an FPGA accelerator card. The XRT runtime library provides an API that facilitates interaction between the host application and the kernels on the accelerator cards. The XRT library is discusses in the subsection 7.1.
- Data transfers between the host and global memory can introduce latency, potentially impacting the application's overall performance. For true acceleration, the hardware acceleration kernels must deliver performance gains that exceed the latency introduced by these data transfers.
- Developers should profile the original application to pinpoint functions suitable for acceleration. After identifying these functions, it is important to establish a performance target for each kernel to ensure the overall application meets its performance objectives.
- The memory hierarchy is critical to the application's performance. Kernels should manage memory reads and writes through separate functions using a load-compute-store model, access contiguous memory whenever possible, and minimize memory accesses by eliminating redundancy or employing a local cache.

XRT and OpenCL

The *Xilinx Runtime* (XRT) library is an open-source software framework that simplifies the management of FPGA and *Adaptive Compute Acceleration Platform* (ACAP) devices, which are advanced processing platforms that combine programmable logic (such as FPGAs), multicore processors, and AI engines within a single chip. The XRT library supports common programming languages like C, C++, and Python. Developers use XRT to write host code that communicates with FPGA/ACAP hardware. The library provides a consistent set of APIs that work across PCIe-based data center systems and embedded platforms such as ZYNQ UltraScale+ MPSoC and Versal ACAP. XRT is a key part of the Vitis and Alveo product families, making hardware-software integration more efficient.

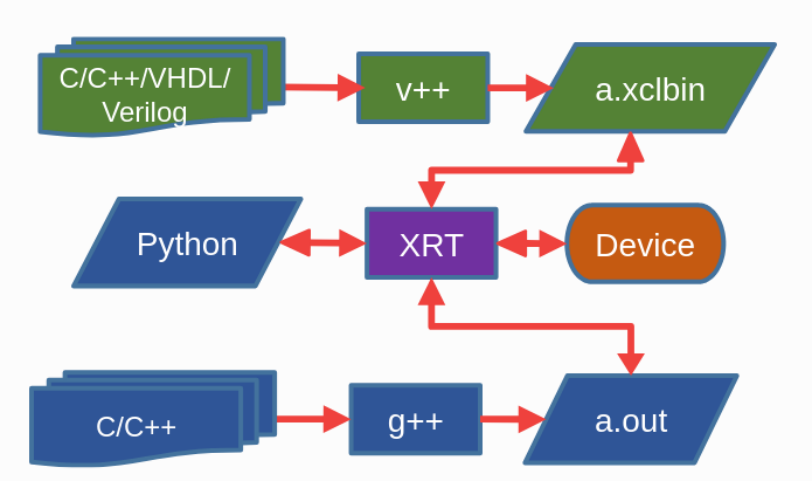


Figure 7.1: Application compilation and execution flow [52].

Compilation and Execution flow

Figure 7.1 shows a conceptual overview of the application compilation and execution flow, using XRT. The user's application consists of host and device code. Host code can be written in C, C++, OpenCL, or Python. Device code is developed in C, C++, OpenCL, or hardware description languages like VHDL or Verilog. To compile and link device code for the target platform, users use the Vitis compiler, v++. Host code in C, C++, or OpenCL is compiled using gcc or g++. Python can also be used for host code, either with PyOpenCL or Python XRT, which offers built-in bindings.

OpenCL

OpenCL is a C based programming language capable of describing data-parallel kernels and tasks that can be executed on heterogeneous systems. Similarly to a dynamic library, OpenCL applications call the functions, which are exported by a dynamic library from their code, directly. The applications, on the other hand, cannot call an OpenCL kernel directly. They queue the execution of the kernel to a command-queue created for a device. The execution of the kernel takes place asynchronously with the application code that is executed on the host [46].

In the context of Vitis acceleration flow, OpenCL acts as an XRT wrapper, adding more layers of abstraction. OpenCL is used widely by both the industry and academia, and it is not limited to only Xilinx applications. Therefore, OpenCL was chosen as the language for the host code development.

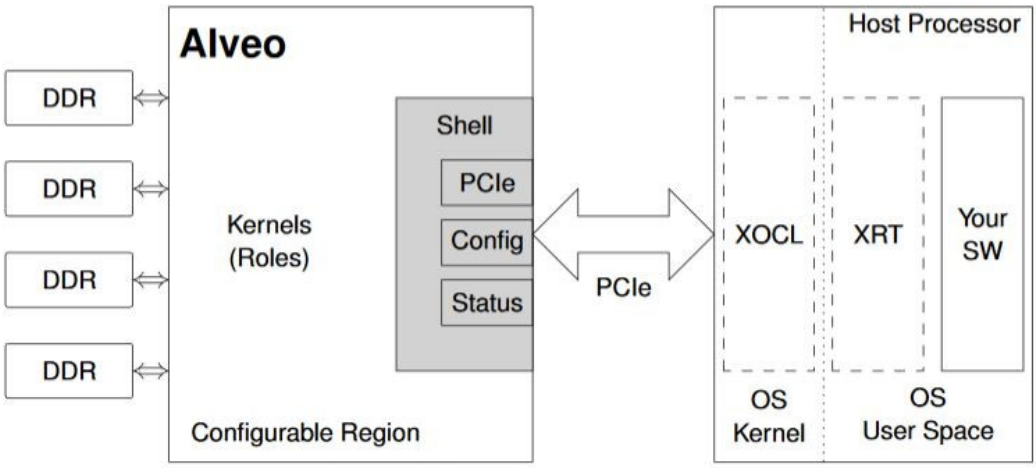


Figure 7.2: Alveo topology [54].

PCIe platforms

PCIe based FPGA platforms work with x86_64, PPC64LE, and AARCH64 host architectures. These platforms are divided into two main sections: the Shell and the Role. The Shell has two key functions. The first is the privileged *Physical Function* PF0, also called the management PF. The second is the non-privileged PF1, known as the user PF. The Shell provides the basic infrastructure for the Alveo platform. The User section, also known as the PR-Region, holds the user-compiled binary. XRT uses Dynamic Function Exchange (DFX) to load the user-compiled binary into the User section. This enables the platform's dynamic functionality [52].

Alveo platforms

Alveo cards consist of an FPGA or ACAP acting as the accelerator, a high-bandwidth DDR4 memory bank, and a PCIe link that interfaces with the host server. They are split into the shell and the role conceptual model. The shell accommodates the static functionality that include external links, configuration and clocking, while the role executes the custom logic. Figure 7.2 show the conceptual topology of an Alveo card.

The FPGA on the Alveo board consists of multiple *Super Logic Regions* (SLRs). This design choice enables higher performance by partitioning the logic across multiple independent regions, reducing routing congestion and improving signal timing. It also allows for better parallelism and scalability, as each SLR can handle separate portions of the design workload efficiently.

The board hosts multiple on-card DDR4 memories. The DDR4 memories, also referred to as global memory, offer high bandwidth interfaces with the FPGA and the host CPU.

The PCIe interface between the accelerator board and the host is the bottleneck of the data transferring, as it adds significant latency. Therefore, it is favorable to transfer data over the PCIe as infrequently as possible. A way to work around this limitation is transferring the data while the kernel is processing a previous set of data. In this way, both the FPGA and the PCIe interface stay busy continuously.

7.2 Data transfer tests

Studies have been initiated for the ATLAS trigger exploring the use of heterogeneous computing methods. The focus of the studies is on data center configurations for the Event Filter system in preparation for Phase II as the latency restrictions are not as strict as in the lower level trigger. As there is no prior experience with such an approach, a testbed was set up to conduct experiments and build expertise in this area. The components of the Alveo U250 FPGA accelerator platform used in the testbed are the following:

- PCIe Gen3x16, capable of transferring about 16 GB of data per second
- High bandwidth DDR4 memory with:
  - 16 GB
  - 2400 MHz
  - 512 bit wide interface
- FPGA accelerator: There are three FPGA accelerators available that allow the verification of the applications on different hardware. The accelerators and their key features are shown in table 7.1.

In the context of the Event Filter tracking, the data processing shall be distributed between the CPU and the accelerator, where only the processes that benefit from the architecture of the accelerator shall be transferred to it. In a data center architecture, all the data are received by the CPU, which subsequently transfers the portion intended for the FPGA accelerator. The transferring procedure comes with a cost of a delay that can be significant.

Table 7.1: FPGA accelerators and key characteristics.

Alveo U50	Built on the UltraScale+ architecture. Offers 100 Gbps networking I/P, PCIe, and 8 GB HBM Resources: – LUTs: 872K – HBM Memory Capacity: 8 GB – Internal SRAM Capacity: 28 MB [47].
Alveo U250	Built on the AMD 16 nm UltraScale architecture. Offers up to 90x higher performance than CPUs on key workloads. Resources: – LUTs: 1,728K – Off-chip Memory Capacity: 64 GB – Internal SRAM Capacity: 54 MB [48].
Versal VCK5000	Build on the AMD 7 nm Versal adaptive SoC architecture. Optimized for AI developments. Offers near 100% compute efficiency per watt in standard AI benchmarks. Optimized for CNN, RNN, and NLP acceleration Resources: – LUTs: 900K – Off-chip Memory Capacity: 16 GB – Internal SRAM Capacity: 23.9 MB [49].

The PCIe Gen3x16 has a bidirectional bandwidth of 16 GB/s, resulting into 8 GB/s per direction. The FPGA accelerator cards' vendor offers a validation test that performs a data transferring test. The bandwidth of the process of writing data to the FPGA with this test is around 8.3 GB/s. The reading process has a higher bandwidth that is approximately 11.5 GB/s. There are a few reasons for the reduced bandwidth reported in the test. In practice, data transfer rates are reduced due to PCIe protocol overhead that includes transaction layer packets, which carry the headers, and acknowledgments and flow control, especially for the read operations. Another factor affecting the bandwidth is the CPU memory access latency. The PCIe transactions

involve data transfers between the system's RAM and the FPGA. If the host memory is not well optimized, it can limit the direct memory access performance.

This work primarily focuses on identifying and addressing bottlenecks in data transfer. More specifically, it focuses on the challenges associated with transferring data over PCIe to accelerator cards. In this initial stage, the algorithms used for testing are simplified, dummy algorithms. The algorithms that are going to be executed by the Event filter accelerators will execute operations that will include matrix multiplications and vector additions. Therefore, it was decided to perform tests with simple matrix multiplication and vector addition kernels. The primary focus is on the mechanisms and strategies for transferring data. The experiments investigate the effects of varying data sizes and the methods by which data is sent to the accelerator, with the aim of optimizing the data transfer process and determining whether such a technology is adequate in the context of the ATLAS trigger.

Data transferring and execution duration

In order to break down the kernel execution steps and further understand where the execution time is spent, a series of tests has been performed and the execution time of each step have been recorded. The execution time has been monitored with the Vitis Timing Analyzer. The plot in figure 7.3 shows the kernel execution time in the same plot with the data transferring duration for a vector addition. The inputs and the output are vectors of integers. The horizontal axes show the size of the buffer that was instantiated for every execution. The bigger the buffer size, the larger number of integers is transferred to the accelerator and back for the addition. The transfer time and the execution time increase linearly with the size of the data that are being processed. The transfer time of the input is about two times higher than the output due to the fact that the input consists of two vectors of integers while the output consist of only the result, which is one vector of integers.

Figure 7.3: Vector addition execution time.

Figure 7.4 depicts the same plot for a matrix multiplication kernel. As this is a much more compute intensive process, the size of the matrices and, therefore, the sizes of the buffers are significantly smaller. This plot shows that the execution time is notably larger compared to the time spent for transferring the data. The rate at which the execution time increases with the buffer size is also significantly higher.

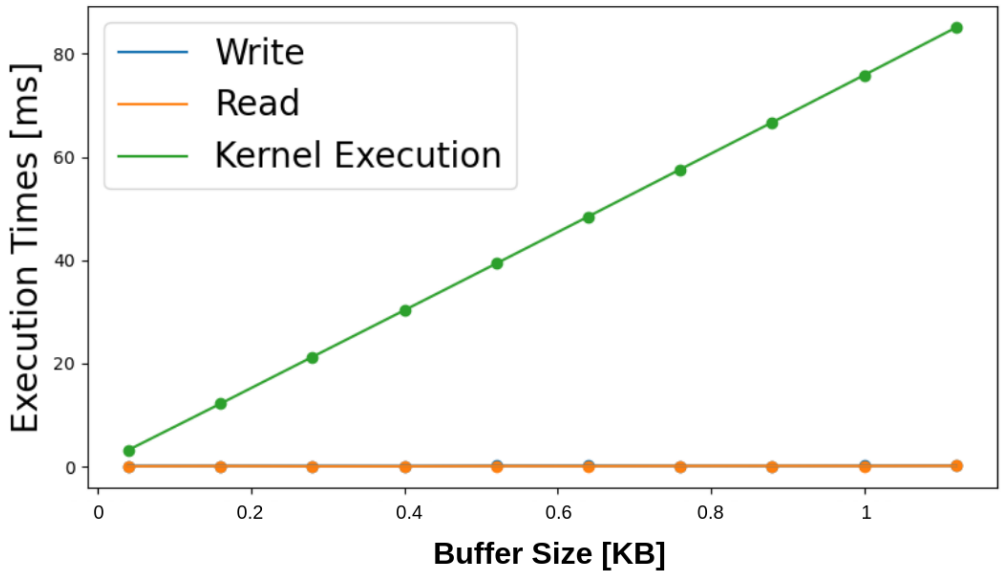


Figure 7.4: Read, Write and execution time for matrix multiplication.

As a means to understand further the behavior of the system, a set of test have been performed with the vector addition kernel to examine whether there is relation between the writing and reading of the data. The following figure 7.5 shows that there is no relation between the read and write, while it shows again that the read time is twice as long compared to the write time.

Understanding and addressing these read/write timing differences will ultimately help the Event Filter system process data more effectively, improving both the quality and speed of data processing in real-time.

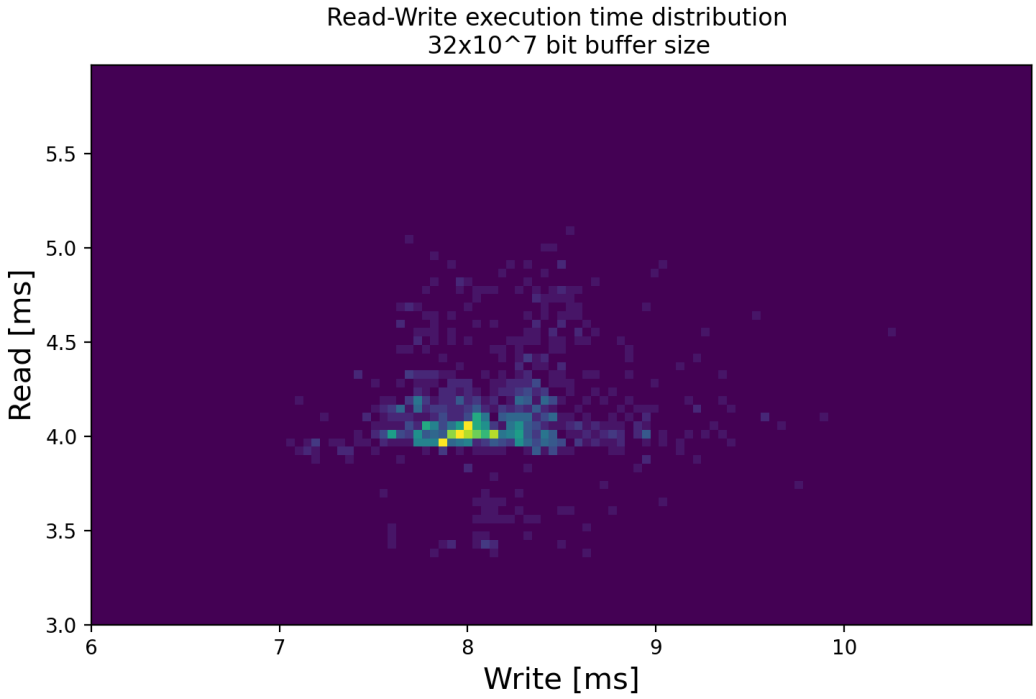


Figure 7.5: Read and write process distribution.

The two following figures 7.6, 7.7 show the distribution of the time spent on reading, writing, and execution time on the vector addition kernel that was used in the previous test. The distribution of the kernel execution is significantly narrower. This is expected as the data transferring as the latter can be affected by the CPU processes that may run in parallel while the kernel execution on the accelerator is remains unaffected.

To further investigate this behavior, the same test was performed while the CPU was forced to be busy with other processes at the same time. The results in figure 7.7 confirm that the data transferring is indeed affected by this, and the already wide distribution is amplified by that.

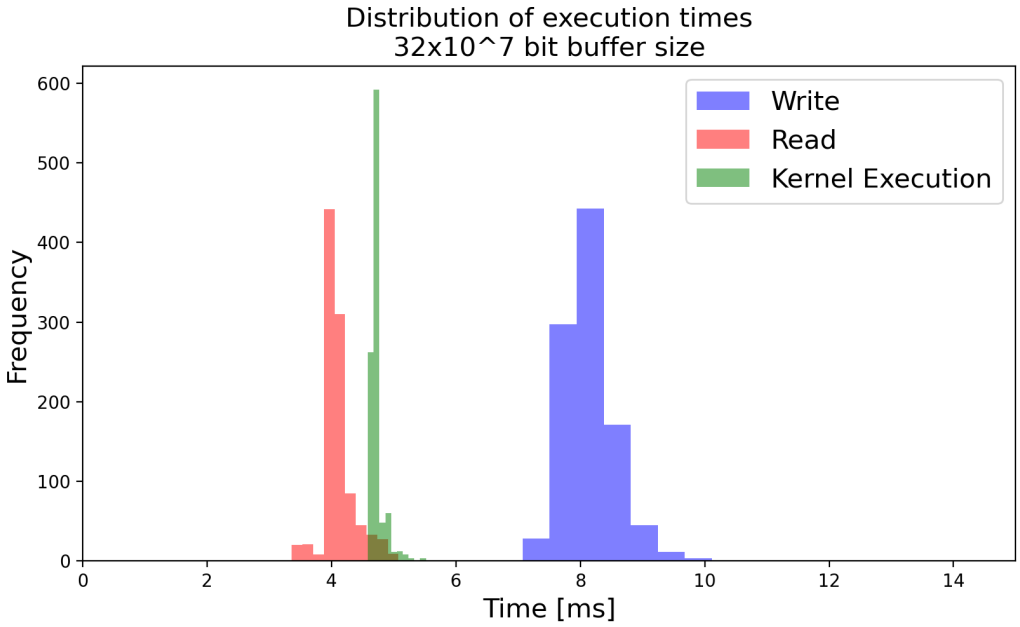


Figure 7.6: Read, write and execution time distribution for idle CPU.

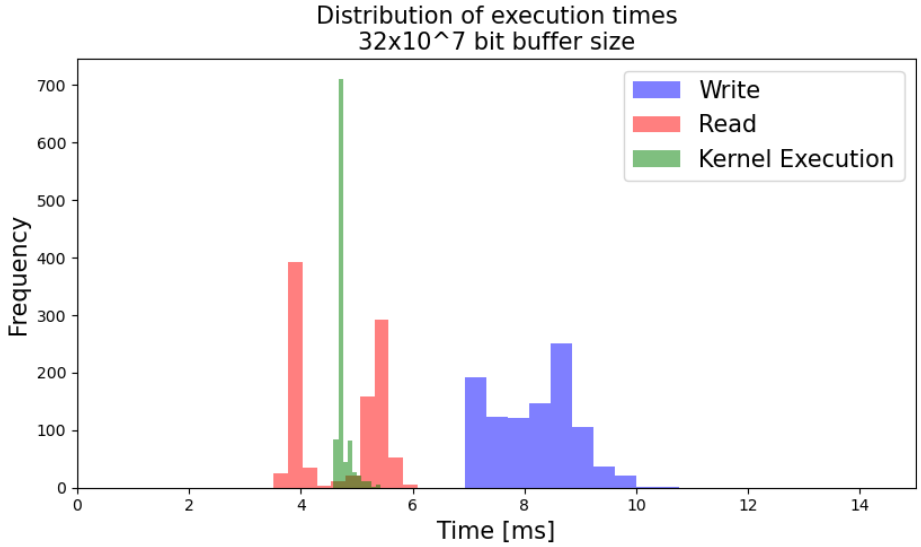


Figure 7.7: Read, write and execution time distribution for busy CPU.

Hardware vs Software

The amount of data that is processed by the accelerator significantly impacts the performance of the system, as the more the data that are sent for processing to the accelerator, the more insignificant the transferring time cost is. The reason for that, is that the acceleration of the process for more data outweighs the cost of transferring more

data over the PCIe. In order to understand the behavior of the system, tests have been performed to identify the conditions where the accelerator improves the performance of the system and when it is redundant. At this stage, dummy algorithms have been used again to hint at the behavior of more sophisticated algorithms that will be used in the Event Filter. The results of the comparison between the execution of the same function implemented in software and for hardware acceleration as shown in figures 7.8 and 7.9, including tests for different sizes of buffers. It is apparent that the execution of vector addition with the accelerator becomes beneficial for buffer sizes over 1.5 MB. Similarly, the kernel execution for the matrix multiplication becomes faster on the accelerator for higher buffer sizes.

This observation can be explained as for smaller amount of data it is not worth to pay the price of moving the data to and from the accelerator but rather perform the process in the CPU. As the matrix multiplication is a more compute intensive process, this observation becomes apparent for smaller data sizes.

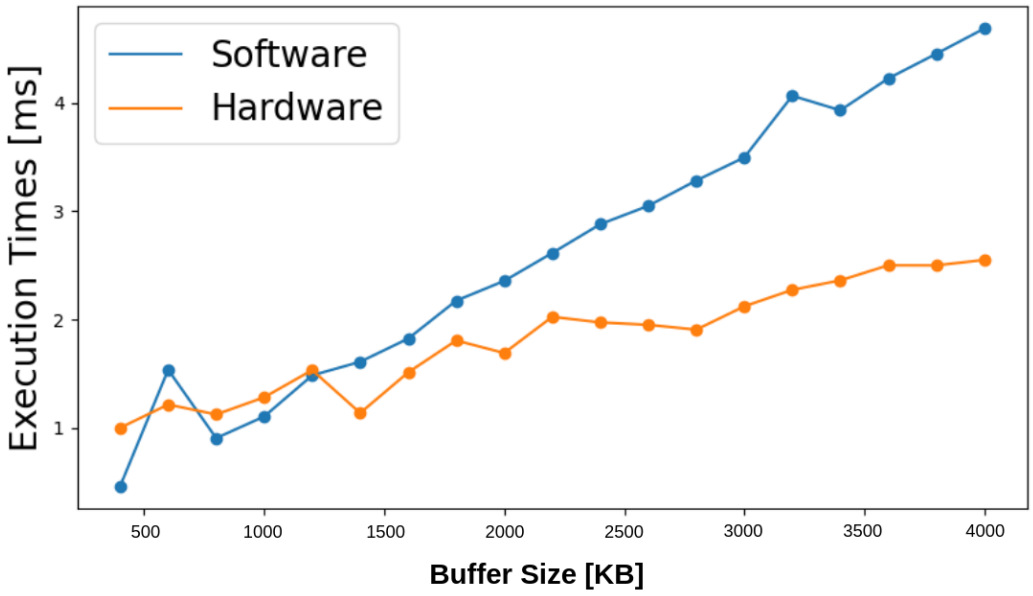


Figure 7.8: Vector addition execution time.

Figure 7.9: Matrix multiplication execution time.

Buffer numbers

Finally, tests have been performed to evaluate the behavior of the setup using multiple buffers for the data transfer. The multiple buffers allow the data to be transferred in bunches, allowing the beginning of the execution of the kernel at an earlier stage. The subsequent bunches of data are transferred while the kernel is executed. This results in a pipelined procedure that speeds up the kernel execution.

The following figure 7.10 shows that by using one and two buffers, the results are not significantly different. For larger numbers of buffers, the execution time gets bigger. An explanation for this result may be that the kernel is too simple and takes so little time to execute that the data transferring in bunches deteriorates the performance.

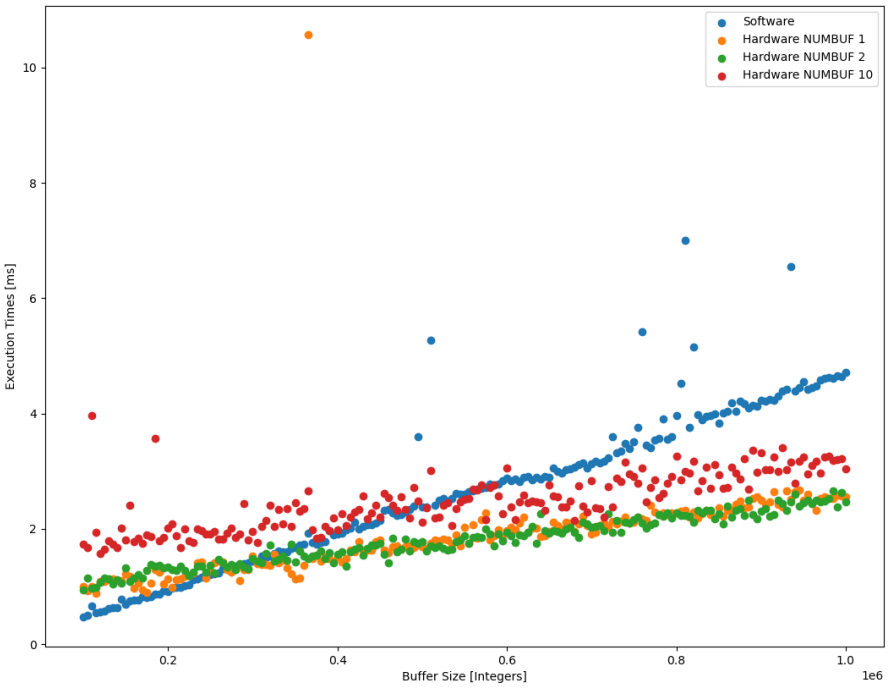


Figure 7.10: Execution time as a function of the number of buffers.

The bandwidth of the data transaction of the test above was measured and a difference with the vendor's bandwidth test is observed. In the test's performed above, the write speed to the FPGA is 9 GB/s, which is higher than the vendor's 8.3 GB/s, while the read speed is only 6.7 GB/s, much lower than the vendor's 11.5 GB/s. The increase in write speed may be due to larger write transactions or reduced software overhead, while the vendor's test might be more conservative. The lower read speed in the data transfer tests could be caused by smaller read requests, leading to increased overhead. The vendor's bandwidth test might handle multiple read requests at the same time, making it more efficient.

7.3 Conclusions

As the technology trends shift towards heterogeneous architectures and hardware acceleration, studies were conducted on a data center system consisting of a CPU and an off the shelf FPGA accelerator in the context of the EF for the ATLAS Phase II

upgrade. As the focus of the studies was on the data transferring between the CPU and the accelerator, dummy algorithms were used as Vitis kernels.

The tests performed included time measurements of the data transferring between the host CPU and the FPGA accelerator. The kernels used for these tests were a vector adder and a matrix multiplexer, developed in HLS. The tests were performed with different sizes of data. As the matrix multiplexer is more complex and relies more on older values, the data transferring is less significant and the execution time of the test is dictated by the kernel execution. Unlike the matrix multiplication, the vector adder's execution time is affected heavily on the data transferring. Subsequently, test were performed to study the variance of the transfer duration. The test showed that when the CPU is busy with some other task, the data transfer duration will be affected while the kernel execution will remain the same. Furthermore, the tests showed that the read and write operation delays are not correlated. Finally, the studies on sending the data in bunches to pipeline the procedure of write, execute, read, showed that there were no significant changes in the system's performance.

The next step, to further investigate the system, was to test it with a more sophisticated algorithm that would stress the system more and would give more real world results. This idea led to a collaborative work to develop a machine learning algorithm that would be executed on an FPGA accelerator. The authors' contribution to this project is discussed in chapter 8.

Chapter 8

Machine learning accelerator backend for HLS4ML

The ATLAS experiment is increasingly leveraging FPGA-based accelerators to meet the stringent demands of real-time data processing and to explore their capabilities for hosting advanced machine learning algorithms. By integrating multiple algorithms into FPGA platforms, ATLAS aims to enhance computational efficiency and scalability, ensuring optimal resource utilization in high-throughput environments. Its goal is to introduce more and more sophisticated algorithms and explore the capabilities of machine learning in the context of the Trigger. This chapter introduces the development and verification of a novel backend for the HLS4ML library that automates generation of machine learning applications for FPGAs in data center configuration

8.1 HLS4ML

HLS4ML is an open source library for python that generates machine learning inferences for FPGAs. The generated firmware is HLS code. The models that are translated by HLS4ML are translated versions of open source machine learning package models.

The idea of HLS4ML is the automation of the translation of machine learning models for training machine learning algorithms to HLS. Subsequently, the HLS code is synthesized to an IP that can be executed on FPGAs.

HLS4ML stemmed from the CERN community. More specifically, it was developed to support the LHC experiments' needs of upgrading the trigger systems to

support more sophisticated online algorithms, including machine learning inferences [53].

HLS4ML takes a trained machine learning model from libraries like TensorFlow, Keras, PyTorch, or scikit-learn. It then converts the model into C++ code using HLS. This code describes how the model will run on the FPGA. After that, the code is compiled, synthesized, and loaded onto the FPGA, turning it into a fast, hardware-based inference system.

Controlling the design

The users can control how the FPGA design is generated by adjusting configuration settings.

- Configure the size of the design by compressing the implementation.
- Set the precision of the calculation in the model.
- Control the resource reuse by setting the level of pipelining.
- Perform quantization aware training to increase performance at low precision using tools such as QKeras, and parse the QKeras model automatically during inference.

By changing these settings, users can balance accuracy, speed, and resource consumption.

Automatic code and pragmas generation

HLS4ML automatically generates HLS code, using code templates to build the code of the project, and includes pragmas, which are special instructions for the compiler. These pragmas guide the synthesis process, helping optimize the hardware implementation. For example:

- Loop unrolling pragmas help speed up computations by executing multiple operations at once.
- Pipeline pragmas allow different parts of the model to work in parallel.
- Resource pragmas control how FPGA resources like multipliers and memory are used.

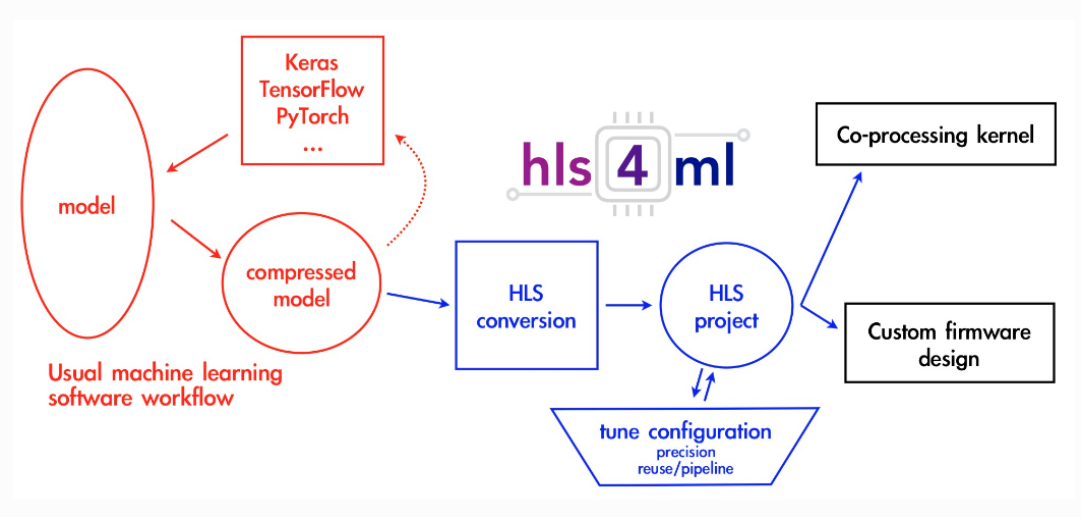


Figure 8.1: HLS4ML workflow [53].

Instead of writing all these optimizations manually, HLS4ML adds them automatically based on the user's configuration. This makes it much easier to generate efficient FPGA designs without deep hardware knowledge.

8.2 Vitis accelerator backend for HLS4ML

HLS4ML offers three backends that target Xilinx devices. The backend is selected by the HLS4ML project generation script. The backends available are the following:

- **Vivado backend:** The Vivado backend supports the standard Vivado IP creation, which is primarily suited for embedded system applications.
- **Vivado accelerator backend:** Is also based on the standard Vivado IP flow and doesn't support the latest Vitis tools that target the Alveo and Versal accelerator cards.
- **Vitis backend:** Finally, the Vitis backend is designed to work with Vivado IP flow, however, it lacks direct support for kernel expression, which is a function executed by the kernel within a hardware accelerator. In the context of heterogeneous systems, it typically defines the operations performed by the accelerator on input data.

A new Vitis accelerator backend is developed to address the limitations of the existing backends. The objective of this new backend is to enable direct Vitis acceler-

ator kernel expression, that can potentially be utilized during ATLAS event filtering or in other applications requiring high throughput.

Furthermore, the Vitis accelerator backend does not only leverage the FPGA technology for acceleration development purposes, but also facilitates prototyping and testing of kernels on hardware.

Vitis accelerator backend structure

Kernel: The input and output of the application is transferred over *Advance eXtensible Interface* (AXI) communication protocol. There are three types of AXI interfaces.

- AXI4 (Full AXI4): Offers high performance memory mapped requirements.
- AXI4-Lite: Similarly to Full AXI4, AXI-Lite is based on memory mapped communication, however, it is suited for simpler and low-throughput applications.
- AXI-Stream: Suitable for high-speed streaming communication.

The Vitis accelerator backend inherits from the existing Vitis backend. As shown in figure 8.2 it wraps the HLS project generated by the Vitis backend with the input and output stream blocks. The wrapper converts the input AXI memory mapped to AXI stream protocol to provide the input data to the Vitis project. In contrast, the output is converted from AXI stream to AXI memory mapped to propagate the data back to the host via the PCIe interface.

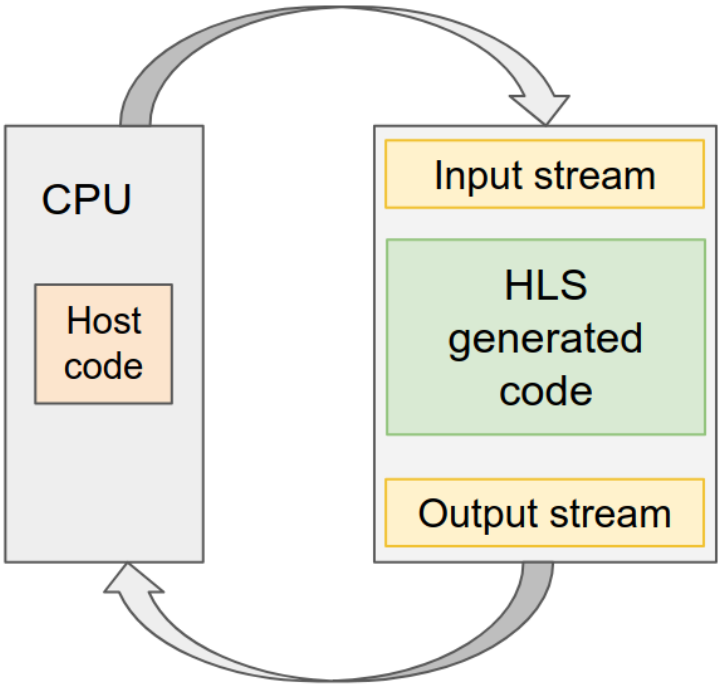


Figure 8.2: Vitis accelerator backend kernel wrapper.

Vitis flow: When configuring the Vitis environment with the Vitis GUI, there is the option to select the Vitis Kernel Flow in the HLS Component wizard. The Vitis accelerator backend utilizes the alternative way to configure the targeted Vitis flow by specifying the `flow_target=vitis` in the *tcl* file that builds the project.

Host: Along with the kernel that runs the application on the FPGA accelerator, the Vitis accelerator backend generates a host code that controls the application. The host code is written with OpenCL. OpenCL was chosen over the XRT library as it is more widely used and offers a more software friendly user experience.

Templates: Following the architecture of the existing backends, the Vitis accelerator backend makes use of code templates to generate the source files of the generated projects. The templates are snippets of the source code that remain unchanged. It also contains keywords and comments that are later replaced by a *Writer* class that shape the final code according to the configuration that has been set by the script that generates the project.

Project generation: There were a few considerations regarding the choice of the technology that compiles the host code and performs the compilation and the linking of the FPGA kernel.

- **Xilinx Software Command-Line Tool:** The *Xilinx Software Command-Line Tool* (XSCT) is a scripting language based on Tcl. XSCT can be run interactively, or it can be scripted for automation purposes [55]. However, XCST doesn't support cloud acceleration flows on Alveo platforms, therefore it couldn't be integrated in the Vitis accelerator backend.
- **Automation from HLS4ML:** The second consideration for building the project was the automation from HLS4ML. This option would fully automate the building process and simplify the initial project setup, however, it would be less convenient for further customization and usage as the user wouldn't be able to run specific commands.
- **Makefile:** The Makefile approach, allows the configuration even after the project generation while fully supporting Alveo platforms. Therefore, the Makefile was chosen as the way to automate the project compilation.

8.3 Verification with QDIPS algorithm

As a first step towards leveraging FPGA-based accelerators and exploring more sophisticated algorithms and machine learning solutions. The QDIPS algorithm, a quantized version of the *Deep Impact Parameter Sets* (DIPS) [56], was selected as a proof of concept to illustrate the potential of the Vitis accelerator backend for HLS4ML. QDIPS is an ideal candidate for this demonstration due to its quantization-aware design, which aligns well with FPGA resource constraints, and its straightforward yet powerful architecture, which makes it well-suited for implementation and evaluation. This chapter details the development and performance of the Vitis backend, using QDIPS to validate its functionality and to pave the way for deploying a broader range of algorithms on FPGA platforms in the ATLAS experiment.

Dips algorithm

The Deep Sets algorithm is a machine learning architecture designed to operate on sets of unordered, variable-sized data, making it ideal for tasks involving collections

of objects without an intrinsic sequence. In the context of jet identification for the ATLAS experiment at the LHC, this structure is particularly well-suited for processing the tracks within a jet, which can vary in number and lack any natural ordering. The Deep Sets architecture, denoted by the DIPS is described by the equation 8.1, where $O(p_1, ..., p_n)$ represents the b-, c-, and light-flavor class probabilities derived from the inputs for the n tracks in the jet, processes each track independently through a neural network (denoted as Φ). DIPS aggregates the outputs using a permutation-invariant operation (such as summation), and then passes the result through a second neural network (denoted as F) for further processing. This enables the model to capture both the individual properties of the tracks and the overall structure of the jet, while maintaining efficiency and scalability.

$$O(p_1, ..., p_n) = F(\Sigma_{i=1}^{n} \Phi(p_i)) \tag{8.1}$$

Figure 8.3 illustrates the DIPS architecture for jet classification. It depicts the way the algorithm processes track information in a jet using a set-based neural network model. The architecture is divided into two main components, the Φ network and the F network.

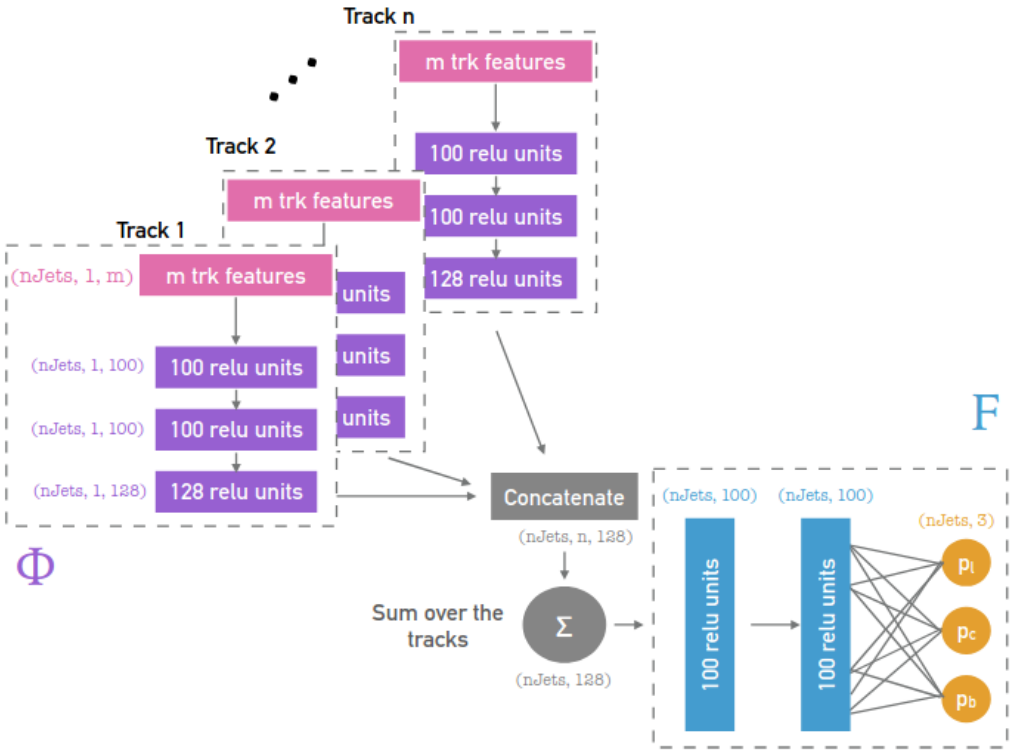


Figure 8.3: Dips algorithm architecture.

Φ(Track Network): The left side of the image shows the Φ network, which processes each track in a jet independently. For each track, a set of track features (m features) is fed into a series of fully connected layers. Each layer consists of 100 *Rectified Linear Unit* (ReLU) activation functions, which introduce non-linearity into the network by applying the function $ReLU(x) = max(0,x)$ to the input. This activation function is widely used in neural networks due to its simplicity and effectiveness in accelerating training and mitigating the vanishing gradient problem. Subsequently, the track features are further processed into 128-dimensional latent space representations. This process is repeated for every track within the jet (Track 1, Track 2, ..., Track n). All these representations maintain the same dimensionality (nJets, 128), where nJets refers to the number of jets being processed simultaneously.

Aggregation (Σ): The latent representations of the tracks are then summed over all tracks in the jet. This summation step is a permutation-invariant operation that ensures the architecture does not rely on the ordering of the tracks. It results in a single 128-dimensional representation for each jet (nJets, 128). This repre-

sentation encodes both the individual track information and track multiplicity within the jet.

F (Jet Network): The aggregated track information is then passed through the F network, which consists of another set of fully connected layers with 100 ReLU units. This part of the network combines the track information and predicts the class probabilities for the jet (pb, pc, pl), corresponding to the likelihood of the jet being a b-jet, c-jet, or light-flavor jet.

QDips and Vitis accelerator

QKeras

QKeras is a quantization extension of Keras designed to create quantized versions of neural networks. It follows Keras' principles of being user-friendly, modular, and extensible while aiming to be minimally intrusive to Keras' native functionality. Quantization refers to reducing the precision of a model's parameters and activations, which can make models more efficient in terms of memory and computation, especially useful for deployment on hardware with limited resources like edge devices.

QDIPS

QDIPS employs QKeras for training the Deep Sets model with quantization awareness, ensuring that the model remains effective even when constrained to lower precision, which is crucial for deployment on hardware with limited resources. QDIPS architecture, depicted in 8.4 resembles the architecture of the DIPS algorithm.

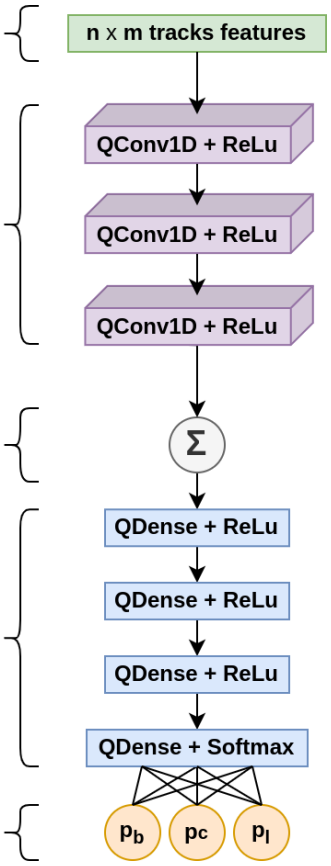


Figure 8.4: QDIPS architecture.

QDIPS implementation and results

The Vitis accelerator backend has been used for a first implementation and performance optimization of QDIPS. The Vitis accelerator backend allowed for kernel instantiation and resource management by offering the configuration option of multiple kernel instantiation to optimize the performance given the available resources. In addition, it provided the host OpenCL code that controls the application and all in all enabled the QDIPS kernel instantiation flexibility and efficient data transfer management. The implementation has been tested on a data center setup. The setup is described below:

- AMD Epyc 744P (24 cores, 256 GB RAM, 5x PCIe gen4x16)
- AMD/Xilinx VCK5000
- Ubuntu Linux - Vitis/Vivado 2023.2

The model that is implemented allows the instantiation of two kernels in the VCK5000. The execution reports are shown in figure 8.7a, while the floor-planning of the same implementation is shown in figure 8.5. Each inference has a latency of 10μs. The latency includes the transfer to and from the PCIe and the DDR memory loading from the kernel.

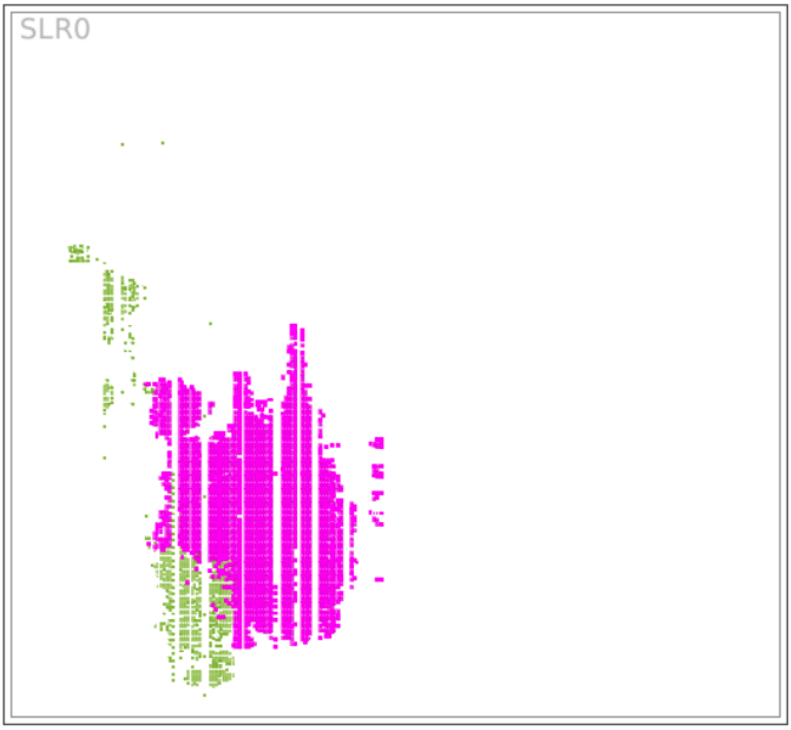


Figure 8.5: FPGA floor-planning with 1 kernel instantiation.

The performance of the two kernel implementation and execution is shown in table 8.1. The latency, measured by the host code, in this occasion is limited to 5 μs. The significant performance increase observed when using two kernels instead of one indicates an efficient use of parallel processing resources. The acceleration strategy minimizes data transfer overhead, allowing for higher throughput. The floor-planning of the implementation is shown in figure 8.6.

Although the DSP usage shown in figure 8.7 is relatively high in these implementations, the performance results are promising. The increase in DSP utilization from 15% to 31% is consistent with the additional logic required to instantiate multiple kernels. This reflects the synthesis tool's resource allocation strategy when implementing multi-kernel configurations. Future studies could focus on optimizing and reducing DSP utilization in subsequent iterations of the QDIPS model.

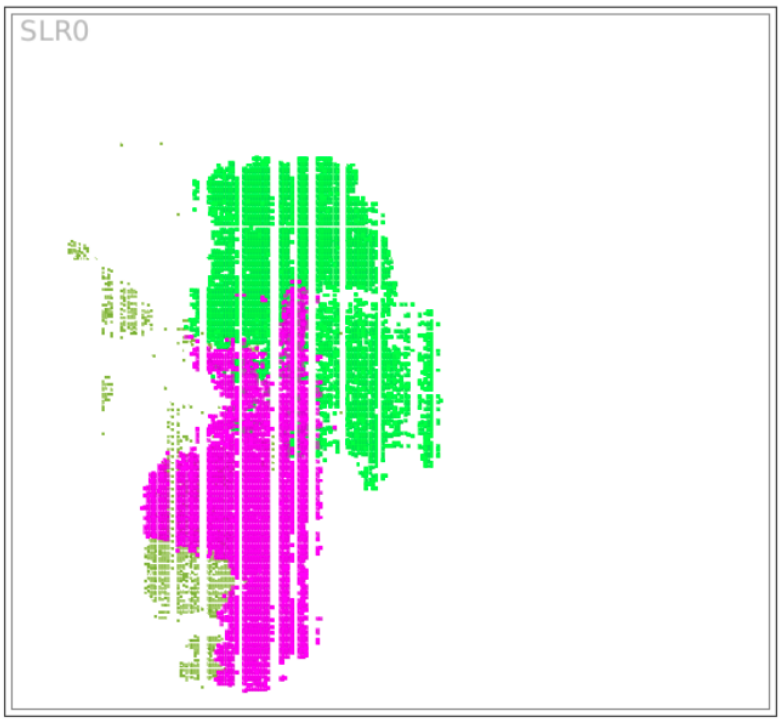
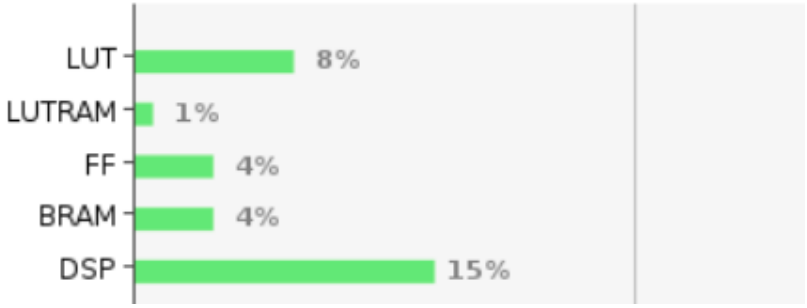
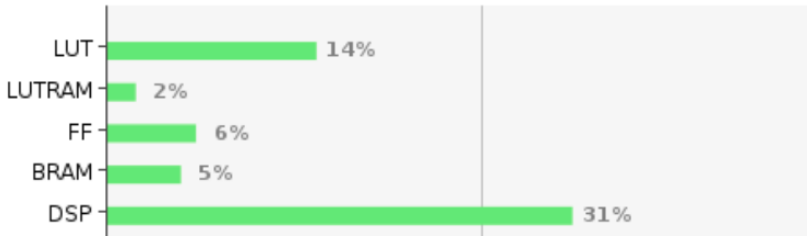


Figure 8.6: FPGA floorplanning with 1 kernel instantiation.



(a) FPGA utilization with 1 kernel instantiation.



(b) FPGA utilization with a 2 kernel instantiation.

Figure 8.7: Resource utilization 1 and 2 kernel instantiation.

Table 8.1: System performance with one and two kernel instantiations.

	1 Kernel	2 Kernels
Frequency	300 MHz	300 MHz
Batch size	256	256
Throughput	100,000 prediction/s	210,000 prediction/s

In conclusion, the Vitis accelerator backend for HLS4ML has proven to be a valuable tool for running machine learning algorithms efficiently on FPGA hardware. Using the QDIPS algorithm as a test case, this work showed how the backend can handle resources, set up flexible kernel designs, and manage data transfers effectively. These results demonstrate that the backend is a solid foundation for running other machine learning models on FPGAs, helping the ATLAS experiment explore new ways to process data faster and more efficiently. This project also provided useful lessons on how to improve the use of FPGA resources and reduce processing delays, laying the groundwork for future advancements and the use of more algorithms in ATLAS.

8.4 Conclusions

Following up chapter 7, chapter 8 presents the development and evaluation of the Vitis Accelerator backend for HLS4ML. The backend automates the generation of an HLS4ML such that it will target an FPGA accelerator in a data center configuration. To test and evaluate the backend a QDIPS algorithm was used as a test case. The selection of the Makefile approach for project automation enabled flexibility and compatibility with Alveo platforms, facilitating efficient kernel instantiation and resource management. The modular architecture of QDIPS maintained computational efficiency while addressing FPGA resource constraints, making it a robust choice for tasks like jet classification in the ATLAS experiment.

Performance benchmarks are presented using an AMD/Xilinx VCK5000 setup, showcasing the ability of the Vitis backend to support multiple kernel instantiations and optimize throughput. The backend was later validated using more FPGA accelerators like the Alveo U250 and the Alveo U50. The work presented above is a proof of concept and the Vitis backend can facilitate the optimization of the resource utilization and latency reduction.

Chapter 9

Summary and conclusions

The subject of this thesis is the development of FPGA-based solutions for the ATLAS TDAQ system. It showcased the challenges of the high data rates and the real-time processing for the ATLAS experiment at the LHC. The end goal was to enable more precise and efficient data handling that are essential for understanding fundamental physics.

Firmware studies for the CTP For the CTP firmware studies, firmware was developed and validated for the CTP readout system. In addition, tests were performed to verify that the memory buffers handled high data rates effectively and confirm the system's reliability.

Firmware development for the PRM Designed and implemented key firmware components, including the Data Generator, the SSID encoder and the output comparator. Validated the firmware with simulation and on hardware platforms, ensuring that the system performed well under realistic conditions.

Data transfer studies for FPGA accelerators In this project, tests were performed to recognize the challenges and understand the mechanisms of using off-the-shelf FPGA accelerators. The set-up was tested under different configurations to evaluate efficiency in different conditions.

Backend development for machine learning for FPGA accelerators Developed the Vitis accelerator backend for the HLS4ML library, that streamlines the generation of HLS4ML projects that target FPGA accelerators. Verified the solution with the

QDIPS algorithm and verified the feasibility of the implementation of such a project while facilitating the process of optimization of the implementation.

Challenges The main challenges that were faced during the studies presented in this thesis include managing high data rates while ensuring reliable operation and balancing the performance with resource constraints. Another challenging part of this work was working with newly introduced technologies with limited information and availability of resources.

Impact of the presented work and final thoughts The work presented in this thesis contributed to the studies and the improvement of the performance of FPGA systems for the ATLAS TDAQ system. It demonstrated the reliability of the firmware implementations with thorough testing and, lastly, it enhanced the integration fo machine learning models on FPGA accelerator platforms.

This work contributes mainly to the ATLAS Phase II upgrade, and it supports the efficient data handling for the HL-LHC. This is critical for studying phenomena at unprecedented energy levels, opening the door to deeper insights into the Standard Model of particle physics and potential new physics beyond it. Furthermore, it showcases how FPGA systems can be used in machine learning.

By enabling more detailed studies of the experiment, the work contributes to finding the answer of fundamental questions about the universe, such as the nature of dark matter, the imbalance between matter and antimatter, and further understanding the Standard Model and the properties of the Higgs boson.

Glossary

ACAP Adaptive Compute Acceleration Platform.

ALTI ATLAS Local Trigger Interface.

AMTP Associative Memory Tracking Processors.

ASIC Application Specific Integrated Circuits.

ATCA Advanced Telecommunications Computing Architecture.

AXI Advance eXtensible Interface.

CAM Content Addressable Memory.

CERN European Organization of Nuclear Research.

CTP Central Trigger Processor.

CTPCORE CTP Core Module.

CTPIN CTP Input modules.

CTPMI CTP Machine Interface.

DAQ Data AcQuisition.

DIPS Deep Impact Parameter Sets.

DSP Digital Signal Processing block.

EEPROM Electrically Erasable Programmable Read-Only Memory.

EF Event Filter.

FCAL Forward Calorimeter.

FELIX Front-End Link eXchange.

FPGA Field Programmable Gate Array.

gHTT global HTT.

HACC Heterogeneous Accewlerated Compute Cluster.

HEC Multi-Process Processing Units.

HLS High Level Synthesis.

HLT High Level Trigger.

HLTMPPU Multi-Process Processing Units.

HTT Hardware Track for Trigger.

HTTIF HTT Interface.

IP Intelectual Property.

ITk Inner Tracker.

L1Calo Level 1 Calororimeter.

LAr Lead Liquid Argon.

LASP Liquid Argon Signal Processor.

LH-LHC High Luminosity LHC.

LHC Large Hadron Collider.

LIU LHC Injectors Upgrade.

LS1 Long Shutdown 1.

LS2 Long Shutdown 2.

LS3 Long Shutdown 3.

LTI Local Trigger Interface.

LUT Look Up Table.

MM Micromega.

MUCTPI Muon to CTP Interface.

NSW New Small Wheel.

PF Physical Function.

PLL Phase Locked Loop.

PRM Pattern Recognition Mezzanine.

ReLU Rectified Linear Unit.

rHTT regional HTT.

ROI Region Of Interest.

RPC Resistive Plate chamber.

RTL Register Transfer Level.

SLR Super Logic Region.

SSID SuperStrip IDentifier.

SSTP Second Stage Tracking Processors.

sTGC small-strip Thin Gap Chambers.

TAP Trigger items After Pre-scale.

TBG Trigger items before Bunch Groups.

TBP Trigger items Before Pre-scale.

TDAQ Trigger and Data AcQuisition.

TFM Track Fitting Mezzanine.

TGC Thin Gap Chamber.

TP Tracking Processor.

TTC Trigger, Timing, and Control systems.

VHDL VHSIC Hardware Description Language.

VHSIC Very High Speed Integrated Circuits.

VMEbus Versa Module Eurocard bus.

XRT Xilinx Runtime.

XSCT Xilinx Software Command-Line Tool.

Bibliography

[1] G. Apollinari, I. Béjar Alonso, O. Brüning, P. Fessia, M. Lamont, L. Rossi and L. Tavian, *High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report V. 0.1*, CERN, Geneva (2017), https://cds.cern.ch/record/2284929.

[2] Yuval Grossman and Yossi Nir, *The Standard Model: From Fundamental Symmetries to Experimental Tests*, 2023, https://press.princeton.edu/books/hardcover/9780691239101/the-standard-model?srsltidĀfmBOop7G7dZ7GGaByxQFc_DwMwLhWB4YRRIpiLz4_S5___4FLRQzBkd.

[3] ATLAS Collaboration, *Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC*, CERN, Geneva (2012), https://arxiv.org/abs/1207.7214.

[4] John Terning, *Modern Supersymmetry: Dynamics and Duality*, Oxford University Press, 2005, https://academic.oup.com/book/11943.

[5] CERN, *LHCb*, CERN, Geneva, https://home.cern/science/experiments/lhcb.

[6] Fabiola Gianotti, *Physics at the LHC*, CERN, Geneva (2004), https://www.sciencedirect.com/science/article/abs/pii/S0370157304003382.

[7] ATLAS, *Standard Model Summary Plots June 2024*, CERN, Geneva (2024), https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PUBNOTES/ATL-PHYS-PUB-2024-011/.

[8] CERN, *The High-Luminosity LHC (HL-LHC) Project*, CERN, Geneva (2016), https://cds.cern.ch/record/2199189/.

[9] CERN, *A new ATLAS for the high-luminosity era*, CERN, Geneva (2023), https://atlas.cern/Updates/Feature/High-Luminosity-ATLAS.

[10] CERN, *The High-Luminoisty LHC (HL-LHC*, CERN, Geneva, https://voisins.web.cern.ch/en/high-luminosity-lhc-hl-lhc.

[11] Karolos Potamianos, *The upgraded Pixel detector and the commissioning of the Inner Detector tracking of the ATLAS experiment for Run-2 at the Large Hadron Collider.*, CERN, Geneva (2016), https://cds.cern.ch/record/2209070..

[12] *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST 3 (2008), https://doi.org/10.1088/1748-0221/3/08/s08003.

[13] ATLAS Collaboration, *ATLAS Phase-II Upgrade Scoping Document*, CERN, Geneva (2015), https://cds.cern.ch/record/2055248.

[14] ATLAS Collaboration, *Technical Design Report for the ATLAS Inner Tracker Pixel Detector*, CERN, Geneva (2017), https://cds.cern.ch/record/2285585

[15] ATLAS Collaboration, *Technical Design Report for the ATLAS Inner Tracker Strip Detector*, CERN, Geneva, (2017) , https://cds.cern.ch/record/2257755.

[16] Joao Paquenao, *Computer Generated image for the ATLAS calorimeter*, CERN, Geneva (2008), https://cds.cern.ch/record/1095927

[17] Stelzer, Bermd (Simon Fraser U.), *The New Small Wheel Upgrade Project of the ATLAS Experiment*, ATLAS Collaboration, Geneva (2016), https://cds.cern.ch/record/1958265/

[18] ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, CERN, Geneva (2008), https://iopscience.iop.org/article/10.1088/1748-0221/3/08/S08003.

[19] ATLAS Collaboration, *Technical Design Report for the Phase II Upgrade of the ATLAS TDAQ System*, CERN, Geneva (2017), https://cds.cern.ch/record/2285584.

[20] ATLAS Collaboration, *The ATLAS Trigger System for LHC Run 3 and Trigger performance in 2022*, CERN, Geneva (2024), https://arxiv.org/abs/2401.06630.

[21] PICMG, PICMG 3.0 Revision 2.0, *AdvancedTCA Base Specification*, http://www.picmg.org

[22] IEEE, *1014-1987 - IEEE Standard for a Versatile Backplane Bus: VMEbus*, https://ieeexplore.ieee.org/document/172897

[23] Glatzer, Julian Maximilian Volker, *Operation of the Upgraded ATLAS Level-1 Central Trigger System*, https://cds.cern.ch/record/2014707

[24] The ATLAS Collaboration, *Fast TracKer (FTK) Technical Design Report*, CERN, Geneva (2013), https://cds.cern.ch/record/1552953

[25] PONG P.CHU, *RTL hardware design using VHDL*, https://theswissbay.ch/pdf/Gentoomen Library/Digital Design/RTL Hardware Design Using VHDL.pdf

[26] Xilinx, *RTL hardware design using VHDL*, https://docs.amd.com/r/en-US/ug906-vivado-design-analysis/Highlighting-Placement

[27] Microchip, *Antifuse FPGAs for Space or Other Harsh-Environment Application*, https://www.microchip.com/en-us/products/fpgas-and-plds/antifuse-fpgas

[28] Thilo Pauly, *ATLAS Phase-2 Central Trigger Processor Requirements and Specification*, https://edms.cern.ch/ui/#!master/navigator/document?D:101153280:101153280:subDocs

[29] Xilinx, *UltraScale Architecture GTY Transceivers*, https://docs.amd.com/v/u/en-US/ug578-ultrascale-gty-transceivers

[30] OSVVM, *Open Source VHDL Verification Methodology (OSVVM)* , https://github.com/OSVVM

[31] Intel, *High Bandwidth Memory (HBM2) Interface Intel FPGA IP User Guide*, https://www.intel.com/content/www/us/en/docs/programmable/683189/ 21-3-19-6-1/about-the-high-bandwidth-memory-hbm2.html

[32] A. Camplani, *Intel Stratix 10 FPGA design for track reconstruction for the ATLAS experiment at the HL-LHC*, https://arxiv.org/abs/2302.13609

[33] Intel, *Intel Stratix 10 MX FPGA Development Kit*, https://www.intel.com/content/www/us/en/products/details/fpga/development-kits/stratix/10-mx.html

[34] Intel, *Intel® Quartus® Prime - FPGA Design Software*, https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html

[35] Intel, *About the SignalTap II Logic Analyzer*, https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProjects/program/ela/ela_view_using.htm

[36] Siemens, *Questa Advanced Simulator*, https://eda.sw.siemens.com/en-US/ic/questa/simulation/advanced-simulator/

[37] IEEE, *The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and ensity*, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1281800

[38] Volnei A. Pedroni MIT Press, *Circuit Design with VHDL*,

[39] Xilinx, *Vitis High-Level Synthesis User Guide (UG1399)*, https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

[40] The ATLAS collaboration, *Operation of the upgraded ATLAS Central Trigger Processor during the LHC Run 2*, https://cds.cern.ch/record/2063035?ln=en

[41] The ATLAS collaboration, *The ATLAS central level-1 trigger logic and TTC system*, https://iopscience.iop.org/article/10.1088/1748-0221/3/08/P08002

[42] The ATLAS collaboration, *ATLAS Software Documentation*, https://atlassoftwaredocs.web.cern.ch/athena/trigger/prescales/

[43] The ATLAS collaboration, *Technical Design Report for the Phase-II Upgrade of the ATLAS Trigger and Data Acquisition System - Event Filter Tracking Amendment*, https://cds.cern.ch/record/2802799?ln=en

[44] Xilinx, *Virtex UltraScale+ Boards, Kits, and Modules*, https://www.xilinx.com/products/boards-and-kits/device-family/nav-virtex-ultrascale-plus.html

[45] The ATLAS collaboration, *Fast Track Reconstruction for HL-LHC*, https://cds.cern.ch/record/2693670

[46] Aaftab Munshi, *OpenCL Programming Guide*, https://www.amazon.com/OpenCL-Programming-Guide-Aaftab-Munshi/dp/0321749642

[47] Xilinx, *AMD Alveo U50 Data Center Accelerator Card*, https://www.amd.com/en/products/accelerators/alveo/u50/a-u50-p00g-pq-g.html

[48] Xilinx, *AMD Alveo 2U50 Data Center Accelerator Card*, https://www.amd.com/en/products/accelerators/alveo/u250/a-u250-a64g-pq-g.html

[49] Xilinx, *VCK5000 Versal Development Card*, https://www.xilinx.com/products/boards-and-kits/vck5000.html

[50] Jeremy Fowers, *A Configurable Cloud-Scale DNN Processor for Real-Time AI*, https://www.microsoft.com/en-us/research/publication/a-configurable-cloud-scale-dnn-processor-for-real-time-ai/

[51] Javier Moya, *Heterogeneous Accelerated Compute Cluster - ETH Zürich*, https://www.microsoft.com/en-us/research/publication/a-configurable-cloud-scale-dnn-processor-for-real-time-ai/

[52] Xilinx, *Vitis Unified Software Platform Documentation: Application Acceleration Development (UG1393)*, https://docs.amd.com/r/en-US/ug1393-vitis-application-acceleration/Getting-Started-with-Vitis

[53] HLS4ML, *HLS4ML Documentation*, https://cds.cern.ch/record/2063035?ln=en

[54] Xinlinx, *Get moving with Alveo: Accelerationo Basics*, https://www.xilinx.com/developer/articles/acceleration-basics.html

[55] Xinlinx, *Xilinx Software Command-Line Tool*, https://docs.amd.com/v/u/en-US/ug1208-xsct-reference-guide

[56] The ATLAS collaboration, *Deep Sets based Neural Networks for Impact Parameter Flavour Tagging in ATLAS*, https://cds.cern.ch/record/2718948