Readout Electronics for the Upgraded ITS Detector in the ALICE Experiment

Simon Voigt Nesbø

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2022

UNIVERSITY OF BERGEN

Readout Electronics for the Upgraded ITS Detector in the ALICE Experiment

Simon Voigt Nesbø

Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 03.06.2022

Year: 2022

Title: Readout Electronics for the Upgraded ITS Detector in the ALICE Experiment

Name: Simon Voigt Nesbø

Print: Skipnes Kommunikasjon / University of Bergen

"How hard can it be?"

Jeremy Clarkson

Abstract

ALICE is undergoing upgrades during the Long Shutdown (LS) 2 of the LHC to improve its performance and capabilities, and to prepare the experiment for the increases in luminosity provided by the LHC in Run 3 and Run 4. One of the most extensive upgrades of the experiment (and the topic of this thesis) is the replacement of the Inner Tracking System (ITS) in its entirety with a new and upgraded system. The new ITS consists exclusively of pixel sensors organized in seven cylindrical layers, and offers significantly improved tracking capabilities at higher interaction rates. And in contrast to the previous system, which would only trigger on a subset of the available events that were deemed "interesting", the upgraded ITS will capture all events; either in a triggered mode using minimum-bias triggers, or in a "trigger-less" continuous mode where event data is continuously read out.

The key component of the upgrade is a novel pixel sensor chip, the ALPIDE, which was developed at CERN specifically for the ALICE ITS upgrade. The seven layers of the ITS is assembled from sub-assemblies of sensor chips referred to as staves, and the entire detector consists of 24 120 chips in total. The staves come in three different configurations; they range from 9 chips per stave for the innermost layers, and up to 196 chips per stave in the outer layers. The number of control and data links, as well as the bit-rate of the data links, differs widely between the staves as well.

Data readout from the high-speed copper links of the detector requires dedicated readout electronics in the vicinity of the detector. The core component of this system is the FPGA-based Readout Unit (RU). It facilitates the readout of the data links and transfer data to the experiment's server farms via optical links; provides control, configuration and monitoring of the sensor chips using the same optical links, as well as over CAN-bus for redundancy; distributes trigger signals to the sensor, either by forwarding the minimum-bias triggers of the experiment, or by local generation of trigger pulses for the continuous mode. And the field-programmable devices of the RU allows for future updates and changes of functionality, which can be performed remotely via several redundant paths to the RUs. This is an important feature, since the RUs are not easily accessible when they are installed in the cavern of the experiment and will be exposed to radiation when the LHC is in operation. Radiation

tolerance has been an important concern during the development of the FPGA designs, as well as the RU hardware itself, since radiation-induced errors in the RUs are expected during operation. Techniques such as Triple Modular Redundancy (TMR) were used in the FPGA designs to mitigate these effects. One example is the radiation tolerant CAN controller design which is introduced in this thesis. A different challenge, which is also addressed in this thesis, is the monitoring of internal status and quantities such as temperature and voltage in the ALPIDE chips. This is performed over the ALPIDE's control bus, but must be carefully coordinated as the control bus is also used for triggers.

The detector and readout electronics are designed to operate under a wide set of conditions. Considering events from Pb–Pb collisions, which may have thousands of pixel hits in the detector, a typical pp event has comparatively few pixel hits, but the collision rate is significantly higher for pp runs than it is for Pb–Pb runs. And the detector can be used with two triggering modes, where the continuous trigger mode has additional parameters for trigger period. A simulation model of the ALPIDE and ITS, presented in this thesis, was developed to simulate the readout performance and efficiency of the detector under a wide set of circumstances. The simulated results show that the detector should perform with a high efficiency at the collision rates that are planned for Run 3. Initial plans for a dedicated hardware, to handle and coordinate busy status for the detector, was deemed superfluous and the plans were canceled based on these results. Collision rates higher than those planned for Run 3 were also simulated to yield parameters for optimal performance at those rates. For the RU, which was designed to interface to three widely different stave designs, the simulations quantified the amount of data the readout electronics will have to handle depending on the detector layer and operating conditions. Furthermore, the simulation model was adapted for simulations of two other ALPIDE-based detector projects; the Proton CT (pCT) project at University of Bergen (UiB), a Digital Tracking Calorimeter (DTC) used for dose planning of particle therapy in cancer treatment; and the planned Forward Calorimeter (FoCal) for ALICE, where there will be two layers of pixel sensors among the 18 layers of Si-W calorimeter pads in the electromagnetic part of the detector (FoCal-E). Since the size of a calorimeter pad is relatively large, around 1 cm^2, the fine grained pixels of the ALPIDE (29.24 µm $\times$ 26.88 µm) will help distinguish between multiple showers and improve the overall spatial resolution of the detector. The simulations helped prove the feasibility of the ALPIDE for this detector, from a readout perspective, and FoCal was later approved by the LHCC committee at CERN.

Acknowledgements

Well, here it is. Finally. The culmination of almost six years of work, two years over time. I hope you will enjoy it. If not, you can always keep it by your bedside as a sleeping aid. But at least it is out of my hands now.

I guess I did not fully grasp what I was taking on when I decided to pursue a Ph.D. degree. Back in 2015, I was perfectly content with my Master's degree and job in the electronics industry, and I did not anticipate that I would ever pursue a Ph.D. What would I even research, and how would I get funding for it? But then I ran into Johan Alme one day, whom I knew from when I studied for my Bachelor's at Høgskulen på Vestlandet (HVL)[1,2]. Johan would be supervising a Ph.D. position at HVL and tried to persuade me into applying. Although it seemed like a once-in-a-lifetime opportunity for me, I was reluctant at first. But I guess the appeal of referring to myself as Dr. Nesbo in the future was too strong to ignore, so after some time I decided to give it a go. And how hard can it be to do a Ph.D., anyway, I thought.

The Ph.D. was essentially a collaboration between three institutions. The research project itself was part of a much larger project in the ALICE experiment at CERN's Large Hadron Collider (LHC). The UiB would award the Ph.D. degree. And I would be employed and paid by HVL.

The Ph.D. project would revolve around some major upgrades that were planned for the ALICE experiment. The details were a bit scarce, but it would involve research and development of instrumentation and electronics, which were likely to feature FPGAs. Now, you may wonder what an FPGA is, and to answer the question I have come up with this scientific definition: *An FPGA is a particularly nerdy type of programmable microchip. There are, of course, many types of programmable chips, but none that are not as cool as FPGAs*. Needless to say, I was excited about the opportunity to work on a real-world FPGA project. And, let us not forget, it was a project at CERN, one of the world's most famous and highly regarded research organizations.

I had my first day as a Ph.D. student on April 1st, 2016. There was not much time to adjust as we had scheduled the first trip to CERN in my second week. Fortunately, Johan and Håvard, two of my supervisors on the Ph.D., came along and were there

[1]Known as the Western Norway University of Applied Sciences in English.

[2]The campus in Bergen was formerly known as Høgskolen i Bergen (HiB), or Bergen University College in English.

to help me register and get started. And, not to mention navigating the maze of buildings and corridors at the CERN complex. More details about the project had been carved out at this point. Along with my supervisors and other colleagues at HVL and UiB, I would be working on the upgrades for the Inner Tracking System (ITS) of the ALICE experiment. The project had already been in progress for several years, and the new system would be fully based on a custom pixel sensor chip that had been developed at CERN for this project. We had a chance to meet some of our new colleagues at CERN, in particular the group of people responsible for the development of the readout electronics for the detector, with whom we were going to join forces.

That first trip to CERN really set the pace for the next couple of years. There was never a dull moment. My schedule was very busy with travel which I had to carefully balance between my teaching duties at HVL and courses at UiB. I attended conferences and workshops in California, USA; Antwerp, Belgium; Santiago de Compostela, Spain; Adelaide, Australia; as well as a few in Norway. But most of my travel was to CERN, where I went several times a year to perform testing and development, present our work and discuss the project, as well as to perform shift duties in the ALICE control room until the LHC shut down for the upgrades in December 2018.

It was all very overwhelming, with so many activities and responsibilities, probably more than one person could realistically handle. But I am very fortunate to have had three excellent supervisors whom I could always turn to for help. I have been able to meet with them frequently, it has never been hard to find time for a discussion, and they have all been actively involved in my Ph.D. project as well as ALICE and ITS as a whole. I also think the different backgrounds and areas of expertise, and experience among my three supervisors, combined really well to guide me in the best possible way.

I want to thank Johan Alme, my main supervisor, for having so much faith in me and convincing me to pursue the Ph.D. in the first place. For the last six years, I have been bugging him with emails and messages at practically any time of the day. At times he has had to put up with my rants and negativity, my weird sense of humor, yet he always remained positive and had a unique ability to motivate me and make me see the light at the end of the tunnel. His knowledge of electronics, instrumentation, and particularly FPGAs, has been an important resource for me. He is also an excellent teacher, who has had an enormous impact on my career and education ever since I was a Bachelor's student at HVL.

Håvard Helstrup, my supervisor from HVL, was usually the person I would go to with more theoretical questions in physics, and I thank him for that. And he deserves

extra thanks for his detailed review of this thesis and for spotting several small mistakes that most people would have overlooked. He has also been enormously helpful with some of the practical matters relating to the Ph.D., CERN, as well as teaching duties and my employment at HVL. I also want to highlight the great work Håvard has done as the Ph.D. Programme Coordinator at HVL. Such as the annual Ph.D. seminars he organized at Geilo in the winter and making sure that the voices of the Ph.D. students are heard, which has meant a lot for me and the other Ph.D. students.

And last, but not least, thanks to my supervisor Dieter Röhrich. He does an impressive job coordinating CERN-related projects in heavy-ion physics across multiple institutions in Norway, as well as research in medical physics. His vast knowledge about subatomic physics, and the CERN experiments, makes him the person I would go to after a long discussion to find out what the true answer is.

I want to thank all three of them for all the time and effort they spent reviewing several drafts of this thesis. Thanks to their feedback and suggestions, the end result is much better than anything I could have written purely on my own.

There are, of course, several other people that helped me along the way, and deserve to be mentioned. My colleagues in the ALICE collaboration at CERN have been most welcoming. There are too many names to mention individually, but a thanks goes out to Luciano Musa, Piero Giubilato, Gianluca Aglieri Rinella, Joachim Schambach, Paolo Martinengo, Felix Reidt, Marcel Rossewij, and Krzysztof Marek Sielewicz. And I would especially like to thank Matteo Lupi, Matthias Bonora, and Arild Velure, whom I have been working with on a daily basis, both remotely and when I have visited CERN. I can not emphasize enough how hard they were working and how dedicated they were to making the ITS upgrade a success. But there was still time for the occasional coffee break, usually brewed with Matteo's Espresso machine, which he had carefully configured to produce an Espresso so strong that the aftertaste would linger for days. And for Arild and I, I think it was almost tradition to end the day with a fine meal of french fries and a glass of Cardinal beer in CERN's Restaurant 1.

I also want to give a special thanks to Ruben Shahoyan at CERN, for helping me generate input data for my own simulations using AliRoot, and to Ilker Meric at HVL for providing me access to a computing cluster that became crucial for the simulations I was running. And at UiB, I would also like to thank Attiq Ur Rehman, who has also collaborated with us on the readout electronics for the ITS. In the early days of my Ph.D., when I still had no idea what I was doing, he would often patiently explain things to me. I should also mention Kjetil Ullaland, who until recently was the head of the microelectronics program at UiB. Although he was not directly involved in my Ph.D. project, I have had many interesting discussions with him, and he has been administering a lot of software and licenses that I have relied on for my work.

At UiB I shared an office with Ola Grøttvik, Magnus Rentsch Ersdal, Are Haslum, and Shiming Yuan, who were all Ph.D. students associated with the microelectronics program. It is great having a bunch of smart people like you nearby when you are stuck on a problem. And I really enjoyed the many fun and nerdy scientific discussions we had over the years, which may sometimes have gotten in the way of the work we were actually supposed to be doing. Often late in the evening over a pizza and some beers, with Magnus and Ola, the music experts, bickering over what constitutes a good guitar solo. We were often joined by Lucas Altenkämper, a fellow Ph.D. student in ALICE, when he was not too busy sweating over his computer because his physics analysis would not work. Lucas has been a great friend these years and deserves special thanks. My wife would jokingly refer to us as "the particle pals." He has been my go-to guy for questions about physics in general and the ROOT framework. And I will never forget the time he helped me study the night before an exam in particle physics which I was a bit ill-prepared for. I think we stayed at the office till 2-3 AM; without his efforts, it would probably have been a disaster.

I was also happy to find great company among the other Ph.D. students and postdocs at HVL. Unfortunately, there are too many to mention individually. But I would like to especially thank Fernando Macias Gomez de Villar, Rui Wang, Andreas Ramstad Urke, and Espen Nilsen. Although unrelated to my project, I had a lot of interesting discussions about IoT and sensor networks with Andreas and Espen during the early days of my Ph.D. As fresh Ph.D. students it also felt like the three of us were in the same boat figuring out what to do for our Ph.D. projects, and we learned a lot from each other. Fernando and Rui were also most welcoming when I started, and I think our friendships grew much closer over the years. Fernando appeared to be one of the few people in this world who appreciates all the delicate nuances of my weird sense of humor. He is always an entertaining person to be around, who can brighten up any room. And Rui was always an interesting person to talk with, and he would often share his culture. Such as when he kindly invited me to his Chinese New Year celebration to enjoy the delicious food he had prepared. At some point, the three of us started organizing a weekly meeting for the Ph.D. students at HVL, a tradition that has since been continued by other Ph.D. students, and we also organized other activities.

Among other colleagues at HVL, I would first and foremost like to thank Kristin Fanebust Hetland, leader of the Department of Computer science, Electrical engineering and Mathematical sciences. I remember the first day of my Ph.D. very well. A clean desk and a new computer awaited me in the Ph.D. students' office space on the fifth floor, which had a fantastic view of Ulriken and the areas surrounding Kronstad. Kristin gave me a nice tour of the building, introduced me to my new colleagues, and

helped me feel welcome. I think she is a great leader who genuinely cares about those under her. Teaching was part of my position at HVL and this was an interesting experience. I came to realize that it is a bit harder than it looks, when my best attempts to liven up my lectures with a joke would fail to elicit a response from the students. The first course I was teaching was about metrology and statistics, a mandatory course for most engineering students at HVL, where I was responsible for the laboratory experiments along with Kjell Eivind Frøysa. I think we made a good team, and I hope our students learned as much as I did from Kjell Eivind. I also taught parts of a programming course together with Pål Ellingsen and Atle Geitung, and I want to thank them all for their guidance.

At this point, I would like to express my utmost gratitude towards Høgskulen på Vestlandet as a whole. It is an excellent educational institution that I hold in very high regard, and I have always felt welcome and at home there. HVL has played an instrumental role in my life ever since I decided to pursue higher education. And I am extra thankful to have been provided with the funding and unique opportunity to pursue a Ph.D. degree.

And finally, I would like to thank my parents and brothers for their support. And my wife, Jillian, for her patience, love, and support.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

As the story famously goes, Sir Isaac Newton was sitting in his garden one day and watching as apples fell to the ground. He pondered over this observation, wondering why they fell straight to the ground. Why straight down, why not a little sideways? Why fall at all? He came to the conclusion that bodies of mass attract, and eventually, he went on to formulate his theory of gravitation. Observations like these, either by serendipity or from purpose-built experiments, have led to enormous advances in physics and other sciences the past few centuries. Newton humbly said of his many achievements, that "If I have seen further it is by standing on the shoulders of giants." And with the advances in physics and other sciences, mankind has reached an unprecedented understanding of the world and the universe. The idea of four elements are gone, and so is the notion of a flat earth. We no longer invoke ideas of witchcraft or magic when faced with unexplained phenomena; it has become second nature for us all to seek out scientific explanations. Today our understanding of nature reaches so far that the rudimentary experiments of the previous centuries can not offer us much in terms of new insight. Much of the research in physics is now focused on subatomic and quantum physics, as well as studies of distant galaxies[1]. Advances in these fields require complex equipment and experiments. It would seem as though today, climbing the shoulders of the giants is an enormous challenge in itself, let alone seeing further. The words of President John F. Kennedy, from his famous speech at Rice University in 1962, resonates well almost sixty years later:

> We choose to go to the moon. We choose to go to the moon in this decade and do the other things, not because they are easy, but because they are hard, because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one which we intend to win, and the others, too.

[1] Of course, there are many other important fields of physics, such as solid-state, medical, fluid, acoustics, to name a few.

It is commonly said that curiosity drives science. But the challenges help make it exciting, and there are few things that appeal more to a scientist than the idea of solving a near impossible task. Progress in physics may require complex experiments, but there is little reason for concern, as the scientific community is more than willing to build those experiments.

1.1 Subatomic Physics

The electron was the first subatomic particle to be discovered in 1897, by J. J. Thompson and his colleagues, in an experiment on cathode rays [1]. It was followed by the first models of the atom. The first half of the 20th century saw the discovery of the proton and neutron, the nucleons of the atomic nucleus. And studies of atmospheric cosmic rays led to the discovery of the positron, which is the positive anti-particle of the electron; the muon, a more massive cousin of the electron; and the pion. The second half of the 20th century saw the discovery of numerous other subatomic particles. Scientists were struggling to make sense of this "particle zoo", as it was commonly called. They were not convinced that they were all elementary particles in their own right. An important discovery was the realization that protons and neutrons have substructure. This led to Richard Feynman's parton model in 1969 [2], which proposes that neutrons and protons consist of several smaller partons. Gell-Mann and Zweig had previously proposed the quark model in 1964 [3], as an explanation to the zoo of particles. Their model did not see wide spread acceptance in the physics community at first, but eventually the partons of the parton model were recognized as the quarks of the quark model. These discoveries eventually gave rise to the Standard Model of particles in 1975 [4], which stands to this day as the de facto model of particle physics. The Standard Model allowed the particles of the "particle zoo" to be identified as either; composite particles consisting of a number of quarks; or as elementary particles in the Standard Model.

The Standard Model

The Standard Model (SM) introduces several groups of elementary particles, as shown in fig. 1.1. Leptons and quarks are the massive particles that all the known matter in the universe is composed of. They come in three generations, with increasingly larger mass per generation, and each lepton or quark has a corresponding anti-particle with opposite electric charge.

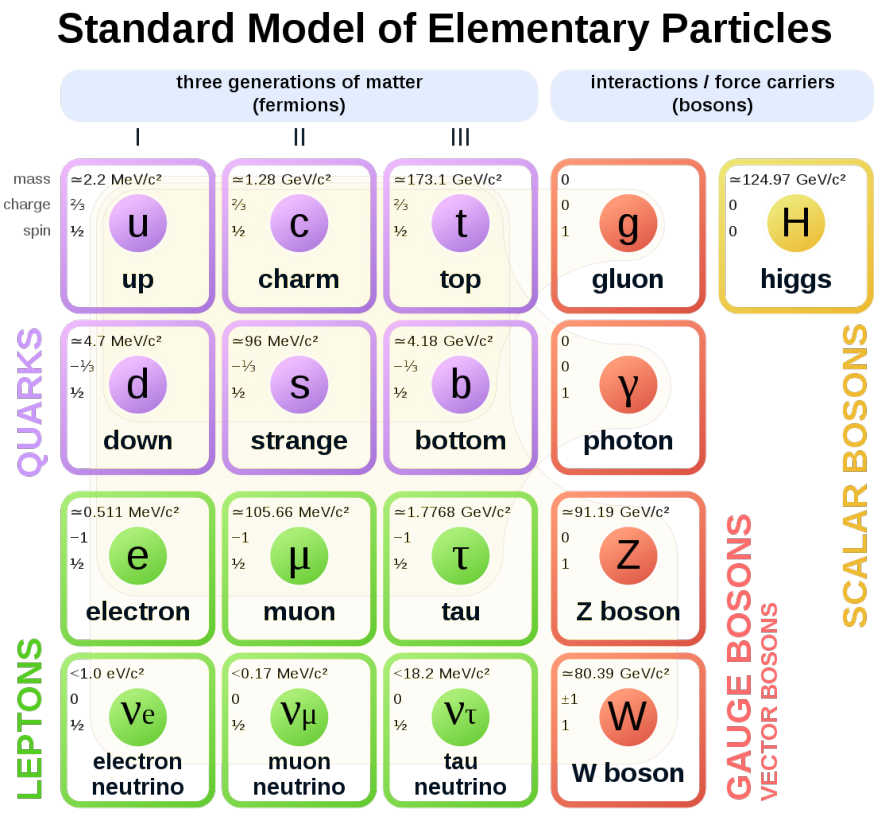


FIGURE 1.1: The Standard Model (SM) of Elementary Particles [4].

Quarks. Individual quarks have non-integer charge (i.e. $\pm\frac{1}{3}e$ or $\pm\frac{2}{3}e$) and do not appear in isolation. At least two or more quarks are confined in a composite particle called a *hadron*, and only certain combinations are possible. For instance, the sum of charges must add up to an integer, typically 0 or ± 1, but higher charge is also possible[2]. Hadrons are further divided into mesons and baryons. Mesons are quark and anti-quark pairs, and baryons consists of three quarks (or three anti-quarks). The pion, which was mentioned earlier, is an example of a meson. It comes in three combinations: π^0, π^+, or π^-, and consists of combinations of up/down and anti-up/down quarks. The most famous examples of baryons is the proton and the neutron, which consist of; two up and one down quark, and one up and two down quarks, respectively.

Leptons. There are three charged leptons; the electron (e), muon (μ), and tau (τ), each carrying a charge of -1. Each of them has a corresponding neutrino; the electron-neutrino (ν_e), the muon-neutrino (ν_μ), and the tau-neutrino (ν_τ). The neutrinos have

[2]More specifically, $+2e$ known to exist for Δ^{++} (and $-2e$ for its anti-particle).

no charge and have a very small rest mass (until recently they were thought to be massless).

Bosons and Forces. Among the bosons in the SM we find:

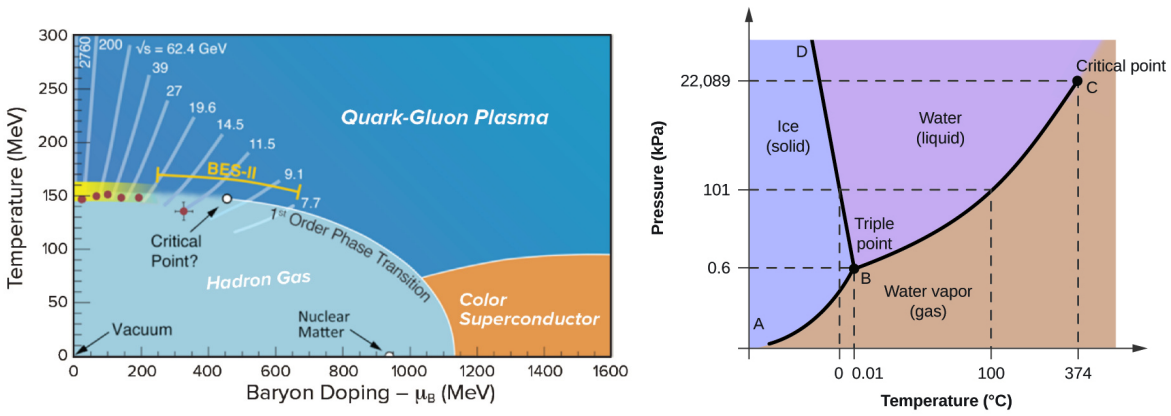
- The photon, which is the carrier of the electromagnetic force. Electromagnetic interactions are governed by the theory of Quantum electrodynamics (QED).
- The gluon, which is the carrier of the strong force (also called the colored force), which keeps the quarks bound together in hadrons. A related force is the nuclear force, which binds the nucleons in the atomic nucleons[3]. Quantum chromodynamics (QCD) is the theory that describes the strong interaction. An important consequence of QCD is the confinement of quarks; free quarks are never observed, they are always confined to a hadron consisting of a multiple of quarks.
- The Z and W bosons of the weak interaction, which is responsible for the radioactive β-decay.
- The Higgs boson. The Higgs mechanism explains why particles have mass.
- The existence of a graviton boson is also hypothesized, and it would be the carrier of the gravitational force. But considering how weak the gravitational force is relative to the strong and electromagnetic force, it is unlikely that we will be able to confirm its existence in the near future, perhaps never.

Quark Gluon Plasma

It was mentioned earlier that quarks are confined to hadrons, however the theory of QCD also predicts that there should be an energy regime where the quarks are no longer confined. Quark matter can appear in different states, as indicated in the phase diagram in fig. 1.2A, in a similar fashion to ordinary elements of matter, such as for water in fig. 1.2B. At temperatures of around 2×10^{12} K, or around 175 MeV, the quarks are able to break free from the confinement of the strong interaction. Quark matter in this deconfined state is referred to as Quark Gluon Plasma (QGP), and the Universe is believed to have been in this state the first few microseconds after the Big Bang. This is commonly referred to as the quark epoch.

Physics experiments that study Quark Gluon Plasma (QGP) produce the plasma by colliding large nuclei at very high energies. As two heavy nuclei collide, they create a hot "fireball" (not in the literal sense), or essentially a drop of liquid QGP

[3]The nuclear force is not an elementary force in its own right; it is a derivative of the strong interaction.



(A) (B)

FIGURE 1.2: Phase diagram of strongly interacting matter (A) [5], and for water (B) [6] included for comparison.

matter. The QGP behaves as a near-perfect fluid, and as this drop of QGP expands it cools and hadronizes, a process in which the free quarks in the QGP form hadrons which they are confined to again.

A couple of experiments are particularly focused at studying the QGP: A Large Ion Collider Experiment (ALICE) at CERN, and the experiments at the Relativistic Heavy Ion Collider (RHIC) at Brookhaven National Laboratory (BNL). The ALICE experiment primarily studies lead ion collisions at energies of 5.5 TeV per nucleon, and collisions with ions of gold and other heavy elements are studied at the RHIC experiments, at energies of 200 GeV per nucleon.

1.2 Particle Detectors

When a beam of radiation passes through a material, it is able to "knock off" electrons in the atoms of the material. In other words, the atoms are ionized. The radiation must be sufficiently energetic for this to happen, and such radiation is typically referred to as ionizing radiation. The radiation itself can be of gamma rays or charged particles. Electrons may be ejected from an atom subjected to gamma rays, if the rays interact with atoms via the *photoelectric effect*, *Compton scattering*, or *pair production*, depending on the energy of the gamma rays. And when a charged particle passes in the vicinity of an atom, it can interact with the atom's electrons via their electric fields and transfer energy to the electrons. If sufficient energy is transferred, an electron can be freed from the grasp of the atom's nucleus.

These principles are utilized in a range of radiation detectors. Early experiments with radiation, including cosmic radiation experiments, in the late 19th and early 20th

century, used photographic films for detection. When the films were subjected to radiation, they would darken due to a chemical reaction following the ionization of the material. This approach to detection offers limited information about the type of radiation and no tracking of the particles. Later experiments from the 1920s used cloud chambers, where droplets are formed in an evaporated liquid, i.e., a cloud, as it is exposed to ionizing radiation. The droplets tend to form around ions in the cloud. When an ionizing particle passes through the cloud, it leaves a trail a of ions, which causes a trail of droplets to form for a brief moment, and these trails were visible to the naked eye. A camera would continuously take photographs of the chamber, which could be studied later. If a magnetic field was present in the chamber, charged particles would curve as they went through it. The curvature of the tracks depends on the mass and kinetic energy of the particle. It can be used to identify the type of particle. From the 1950s and on, the bubble chamber was a popular particle detector. It shared many similarities with the cloud chamber but used a superheated transparent liquid, and several cameras at different angles allowed the events to be captured in three dimensions.

But decades of research since the bubble chamber has seen the emergence of an extensive range of detector technologies, which are now available to modern experiments. The choice of technology depends on the application and type of measurement, such as the type of particle, whether it is for tracking or measuring energy. To stay in line with the topic of this thesis, the following discussion will focus on technologies that have been used for tracking.

Gas detectors

Detectors like the bubble and cloud chambers required photographs to be analyzed by hand. Advances in electronics and computer technology paved the way for the gas-filled Multi-Wire Proportional Chamber (MWPC) in 1968 [7]. In configurations with wires running perpendicularly in several layers, it is possible to measure a particle's incident angle and trajectory, as well as its energy. A further improvement came with drift chambers, which also measured the time it took for the charge to drift to the wires. With the additional information, the full track of the particle in the chamber can be reconstructed. These developments eventually lead to the Time Projection Chamber (TPC), a detector type that is used to this day at some collider experiments. It is a large cylindrical gas detector that encompasses the collision point, with either an MWPC endcap at each end of the cylinder [8], or, in more recent times, a Gas Electron Multiplier (GEM)-based endcap.

Silicon detectors

As with other materials, ionizing particles can liberate electrons in the silicon substrate of semiconductor devices. Silicon strip and drift detectors are the semiconductor equivalent of MWPCs and drift chambers. They are implemented by running lines of *n-doped* and *p-doped* material along the top and bottom of the silicon die and are used for energy measurements and tracking. But superior tracking performance is offered by silicon pixel detectors, which is currently the preferred technology for the innermost trackers at collider experiments. They generally consist of thousands of pixel chips, typically organized in multiple cylindrical layers with close proximity to the collision point. The die of a pixel sensor chip is divided into a matrix of reverse-biased diodes, where each diode forms the sensitive area of a pixel responsible for charge collection. The signals from the diodes are continuously amplified, digitized, and sampled, to detect hits from incident particles.

Detector Electronics and Challenges

It is an unavoidable fact that any particle detector, such as a pixel detector chip, will interact with and affect the particles that it measures. Generally speaking, a detector should be as simple as possible to limit the amount of material a particle must traverse. Only basic signal amplification, and possibly digitization, is performed in the sensitive part of the detector, since most detectors will have external Front End Electronics (FEE) or readout electronics for more advanced data processing. Field-Programmable Gate Arrays (FPGAs) or custom-made Application Specific Integrated Circuitss (ASICs) are often employed in the FEE to process large amounts of data coming from many sensors or channels. Signal integrity considerations usually mandate that the FEE are placed in close proximity to the detector itself. As a consequence, the FEE itself is often exposed to radiation from the experiment, which can cause a range of operational faults and errors.

1.3 The Large Hadron Collider

The Large Hadron Collider (LHC) is a synchrotron accelerator at CERN[4], which, as it name implies, accelerates hadrons and collides them. With a circumference of 27 km,

[4]The European Organization for Nuclear Research (CERN) was founded in 1954 with the original intent of studying the atomic nucleus. It is located right outside of Geneva, Switzerland, on the French-Swiss border. A number of important discoveries have been made at CERN over the years, and the research has spanned far beyond the study of the atomic nucleus.

it is the worlds largest particle accelerator, and it is capable of accelerating particles to energies around 7 TeV.

The accelerator is located a circular underground tunnel, which was originally built for the Large Electron-Positron Collider (LEP). It is built up of sections of parallel beam pipes, surrounded and enclosed by a dipole magnet assembly. Particles move in opposite directions in the two beam pipes, and the beam pipes cross at four points along the LHC ring, at the so-called Interaction Points (IPs). It is at the IPs that the particles moving in opposite directions have a chance of colliding, and it is around the IPs that the 4 main experiments at the LHC are situated (see fig. 1.3), namely:

- A Large Ion Collider Experiment (ALICE)
- A Toroidal LHC ApparatuS (ATLAS)
- Compact Muon Solenoid (CMS)
- Large Hadron Collider beauty (LHCb)

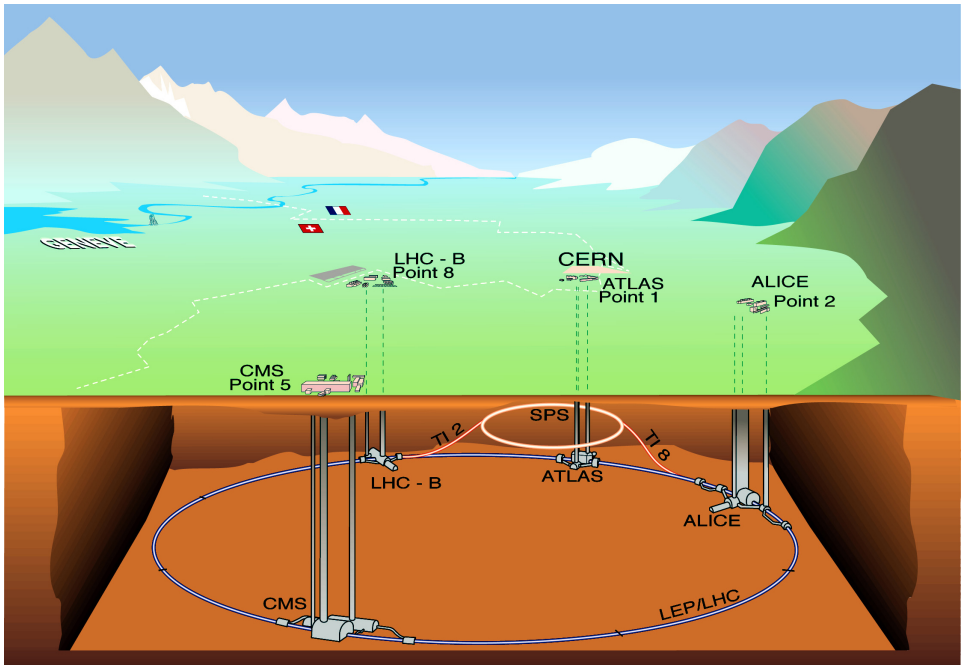


FIGURE 1.3: Map of the LHC and experiments [9].

The experiments consists of different types of particle detectors, to allow them to detect and identify a wide range of particles, and measure their paths and properties such as momentum and mass. Although the experiments share many similarities, they are each uniquely configured and optimized for the particular field of physics they seek to pursue. ATLAS and CMS, famous for the discovery of the Higgs boson, are the largest of the experiments. Interactions between colliding protons (pp) are best suited for their fields of physics, and hence the LHC is primarily running pp.

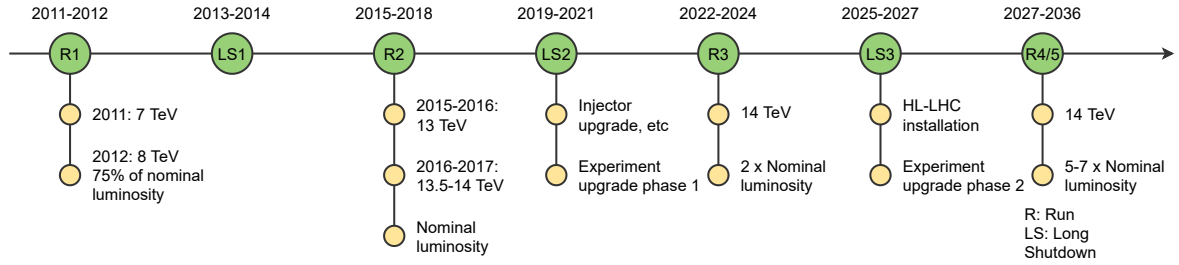


FIGURE 1.4: Timeline of LHC operation.

The LHC and experiments normally operate continuously around the clock, every day of the week. Operation is halted for two to three months during the winter[5]. Besides the winter shutdown and some shorter technical stops for maintenance, the accelerator will only stop for critical problems or maintenance that requires immediate attention. But the long term schedule for the LHC does also include Long Shutdowns (LSs), primarily to allow for major upgrades of the accelerator complex and experiments. A summary of the schedule is shown in fig. 1.4.

1.4 The ALICE Experiment

ALICE is one of the four main experiments at the LHC, and it is situated at Interaction Point 2 (IP2). The primary objective of the experiment is to study QGP and QCD.

Figure 1.5 show the ALICE experiment during Run 1 and Run 2, along with all the subdetectors. Around the collision point at the center of the experiment, we find the Inner Tracking System (ITS), which itself is surrounded by the TPC. These two detectors are responsible for tracking and particle identification. Further out there are patches of calorimeters to measure the energy of a variety of particle types. These are all surrounded by the L3 magnet (in red), which causes charged particles to curve in its magnet field. To the right in the figure, outside of the magnet, we find the detectors for muons, as well as trigger detectors.

Although it is primarily protons that are collided at the LHC, some operational time is also devoted to collisions of lead nuclei with protons (p–Pb), and lead with lead (Pb–Pb). Pb–Pb collisions are of particular interest to the ALICE experiment, as it offers the best data to study QGP. pp is also studied however, and pp also offers a good reference for the studies of Pb–Pb collisions.

[5]The load on the electrical grid and prices of electrical power is the main reason for the winter shutdown. The power consumption of the CERN site is around 80 MW during the winter shutdown, compared to around 200 MW (around a third of Geneva's power consumption) at its peak when the LHC is operating [10].

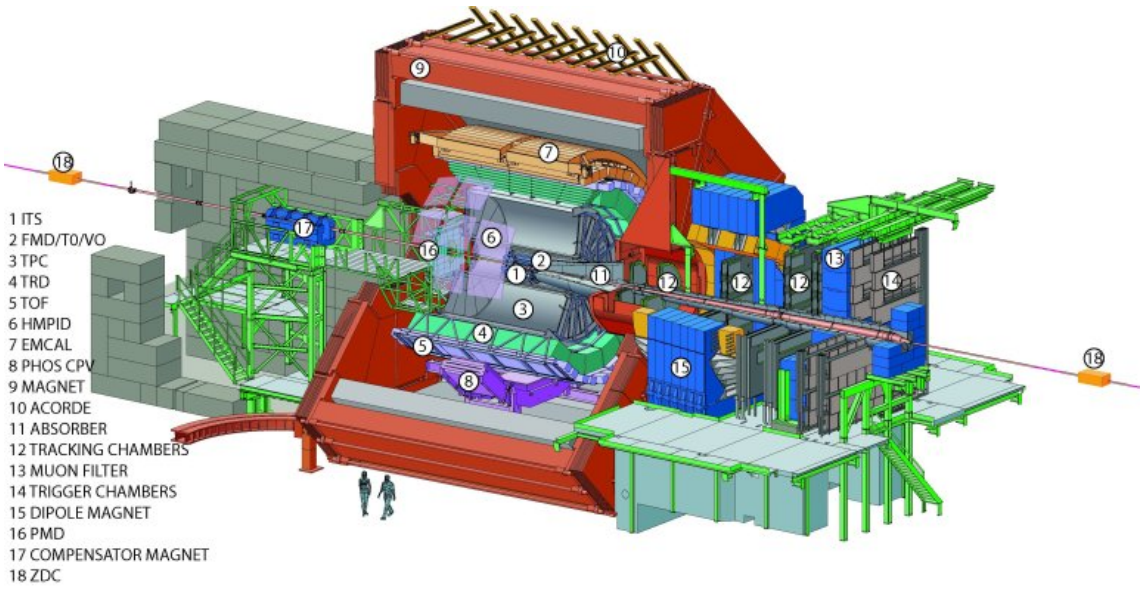


FIGURE 1.5: The ALICE experiment in its Run 1 and Run 2 configuration, with the various subdetectors indicated in the legend. [11].

1.5 ALICE Long Shutdown 2 Upgrades

The first LS of the ALICE experiment primarily saw the installation and upgrades of several calorimeters, such as the Photon Spectrometer (PHOS), Electromagnetic Calorimeter (EMCal), and Di-Jet Calorimeter (DCal), and upgrades to the TPC's readout electronics.

With the LS2 upgrades the ALICE experiment will see substantial upgrades to the trigger system, and an addition of the Fast Interaction Trigger (FIT) trigger detector. But much of the upgrade is focused on the tracking detectors. This includes the installation of an entirely new detector, the Muon Forward Tracker (MFT), and both the ITS and TPC are undergoing significant upgrades: the ITS will be replaced in its entirety with a new tracking system that consists solely of silicon pixel chips, and the TPC will have its MWPC-based chambers replaced with GEM and upgrades to the associated readout electronics.

These upgrades will improve the tracking resolution of these detectors, making it possible to reconstruct the tracks of particles in the experiment with better precision. The new and upgraded detectors are also significantly faster than their pre-LS2 counterparts and can operate at much higher collision rates. In previous runs, the LHC

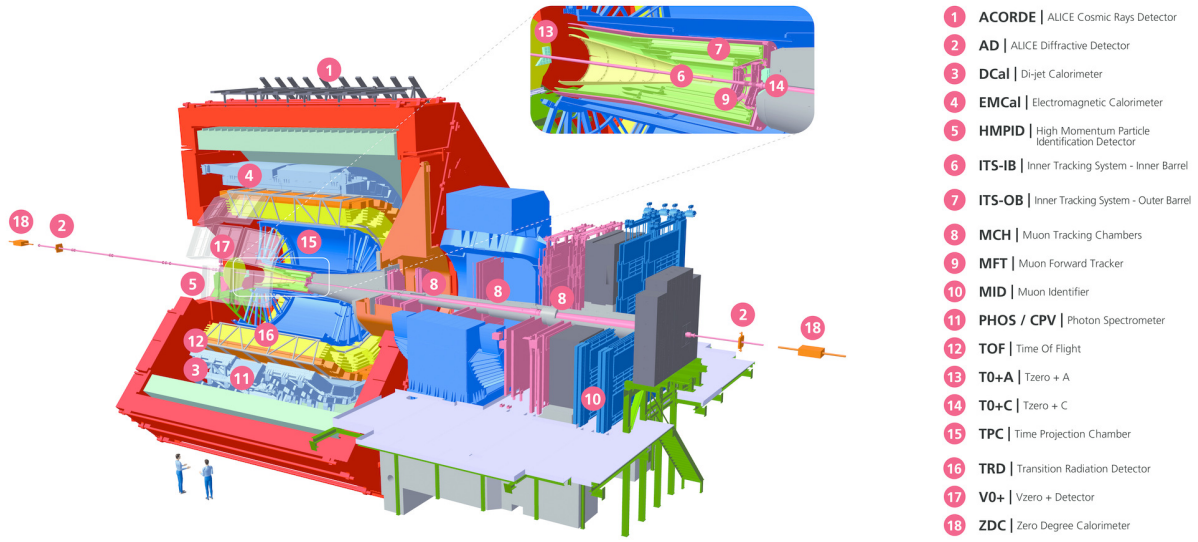


FIGURE 1.6: The ALICE experiment as it will be in Run 3, with the various subdetectors indicated in the legend [14].

could deliver pp collisions at 100 kHz and Pb–Pb at up to 8 kHz [12], but the readout rates in ALICE were substantially lower. With the LS2 upgrades, the ALICE experiment will operate at 500 kHz pp[6] and 50 kHz Pb–Pb [13], using either minimum-bias triggers or a trigger-less continuous mode, thus gathering significantly more data than before. With more data comes improved statistics and smaller uncertainties in the results published by the collaboration, and studies of rare events that were previously out of reach.

1.6 Outline of Thesis and Main Contributions

Research and development for the ITS upgrade for LS2 dates back to 2010. It is a big project involving hundreds of people. I joined the project in 2016, along with other colleagues from the University of Bergen (UiB), and got involved in the group developing readout electronics for the ITS. At this point the group was in the process of drawing up specifications for the readout electronics and finding suitable components. Development of schematics and Printed Circuit Board (PCB) layout for the Readout Unit (RU) started in early 2017, by a group of designers at Nikhef and the University of Utrecht, with the full support of colleagues at CERN, University of Texas, and at UiB, when it came to reviewing the designs. The first prototypes of the RU were completed by the summer of 2017. Work on the FPGA designs for the RU, as well as software development and testing, has been an ongoing activity since

[6] 200 kHz pp was the original target for the ALICE upgrades, but this has been increased to 500 kHz.

the first units were available, and I have been actively involved in those tasks. In addition, I devoted a significant amount of effort into the development of a simulation model for the ALice PIxel DEtector (ALPIDE) and ITS, and used it to run simulations to investigate the necessity of dedicated busy hardware for the readout electronics; determine the amounts of data the RUs have to handle and the readout performance and efficiency of the detector itself, under a range of operating parameters.

This PhD thesis is focused on the activities that were just summarized, with special emphasis on the FPGA designs, and the simulations which supported the development. It will be structured into the following chapters:

The ITS Upgrade. This chapter will give a detailed introduction to the research and development efforts of the ITS upgrade, dating back to the beginning of the project before my involvement started. It will explain the designs of the pixel sensor chips and the hardware for the readout electronics in detail, and provides the necessary context for the chapters that follow.

FPGA Design and Verification Chapters. Two chapters will discuss the FPGA designs for the Xilinx UltraScale and Microsemi ProASIC3 (PA3) FPGAs of the RU. The design for the PA3 FPGA was led from Bergen in its entirety. The design for the UltraScale FPGA was led from CERN, but I collaborated on several parts of the design. Both FPGA designs are introduced in full, along with challenges that had to be overcome to enable reliable operation in a radiation environment. But special emphasis is devoted to the modules that I have been responsible for, which include; Controller Area Network (CAN) controller and associated systems; monitoring of ALPIDE chips; FIFO between the FPGAs for fast transfer of configuration data; debug interface for the PA3 FPGA via Universal Asynchronous Receiver Transceiver (UART).

A third chapter outlines our methodology when it comes to testing and verification of the FPGA design, as well as the electronics in general.

Simulation Model and Results. The next chapter introduces a simulation model I developed of the ALPIDE and ITS. The original purpose was to investigate if the readout electronics would require dedicated "busy units"; a hardware module that communicates with the RUs and aggregates the busy signals of the sensor chips, and consolidates them into a busy map or global busy status for the entire detector. This busy status could be used to coordinate the triggering of the detector to ensure that complete events are recorded; the fear was that poor quality "swiss-cheese" events,

with holes of missing data in place of the busy sensors, would be read out on a regular basis without coordination of the busy and trigger signals. But as the performance of the detector proved to be quite good in the simulations, the focus shifted to quantifying the readout efficiency of the detector under different conditions (also beyond specifications), and to quantify the amounts of data the RUs are faced with.

The model was also adapted to simulate two other detectors which are also based on the ALPIDE; the Forward Calorimeter (FoCal) detector at ALICE, and the Proton CT (pCT) at the University of Bergen (UiB). The chapter contains a brief introduction to both of these detectors and how the simulation model was adapted for them.

The final chapter before the conclusion is dedicated to simulation results, which is mainly focused on the ITS, but also includes some results for the other two aforementioned detectors.

Chapter 2

The ITS Upgrade

This chapter will discuss the ITS upgrade for ALICE in Run 3, with a technical introduction to the Monolithic Active Pixel Sensor (MAPS) chip used in the ITS, the ALPIDE, as well as some details on the development of the chip. The chapter concludes with a discussion of the readout system for the ITS upgrade.

The chapter will serve as an important reference for later chapters in this thesis. In particular chapter 6, which discusses a simulation model for the ITS upgrade and ALPIDE chip.

2.1 ITS Detector in Run 1 and Run 2

The ITS which was used for the first two runs consisted of 6 cylindrical layers. From innermost to outermost layer, there were 2 Silicon Pixel Detector (SPD) layers, 2 Silicon Drift Detector (SDD) layers, and 2 Silicon Strip Detector (SSD) layers. The acceptance of the detector ranged from $|\eta| < 2.0$ and $|\eta| < 1.4$ for the two SPD layers, to $|\eta| < 0.9$ and $|\eta| < 1.0$ for the SDD and SSD layers [15]. The SDD and SSD layers were used for Particle Identification (PID) by measuring the specific energy loss $\frac{dE}{dx}$, while the SPD layers were used for vertex[1] determination and also as a trigger detector [15]. All layers were used for tracking, but the two pixel detector layers offered the best tracking capabilities. They were constructed of SPD staves, where each stave consists of 4 SPD ladders. Each SPD ladder contains a pixel sensor matrix with 256x160 pixel cells, and 5x ALICE1LHCb readout chips which are bump-bonded to the pixel sensor matrix. In total the SPD detector consisted of 60 staves (240 ladders) and 1200 readout chips. The size of each pixel cell was $50\,\mu m \times 425\,\mu m$, and the total number of pixels in the SPD was around 9.8×10^6 [16].

[1] The actual point at which the particles in an event collided, as opposed to the nominal IP of the experiment.

2.2 Long Shutdown 2 Upgrade of ITS

When Run 3 of the LHC commences, the LHC is scheduled to deliver beams to ALICE of significantly higher luminosity than in the previous two runs. The increased luminosity translates into higher interaction rates and more statistics, and new opportunities for the ALICE physics program. Details about how the program will benefit from the upgrade is beyond the scope of this thesis[2]. But briefly speaking, the vast increase in statistics of Run 3 will enable measurements of rare probes that were previously not feasible, as well as improving existing measurements. With interaction rates of 50 kHz in Pb–Pb and 500 kHz in pp, a complete redesign of the ITS was necessary. The existing system was limited to 1 kHz readout [12], [13] and could not possibly cope with the increased interaction rates. Additional requirements for the ITS upgrade was a reduced material budget, and pixel sensors for all layers with a reduced pixel size, which should improve the precision of any measurement in general.

2.3 ITS Upgrade Detector Layout

The upgraded ITS consists of seven cylindrical layers of silicon pixel sensor chips, and is situated directly around the Interaction Point (IP) and beam pipe. In this sense it is a very traditional inner tracker with a design comparable to those of the ATLAS ITk [17], the BPIX detector of CMS' Phase-1 Pixel detector [18], or the planned MAPS-based Vertex Detector (MVTX) detector for sPHENIX [19].

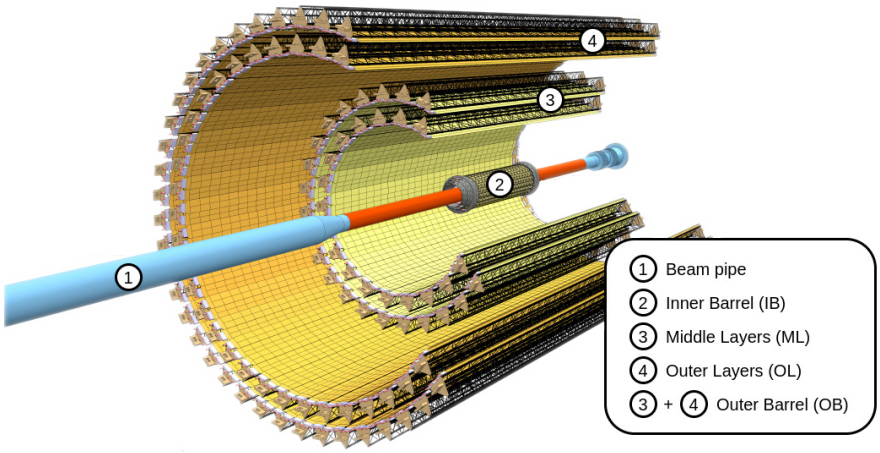


FIGURE 2.1: The upgraded ITS detector [13].

[2]For the interested reader it has been summarized very well in the Technical Design Report (TDR) for the ITS upgrade [13].

TABLE 2.1: ITS detector geometries, and maximum hit density at $\eta = 0$ for minimum bias Pb-Pb events at 100 kHz, including QED background assuming integration time of 30 microseconds, and detector noise of 10^{-5} fake hits/pixel. Table reproduced from table 6.1 in the ALICE ITS TDR [13].

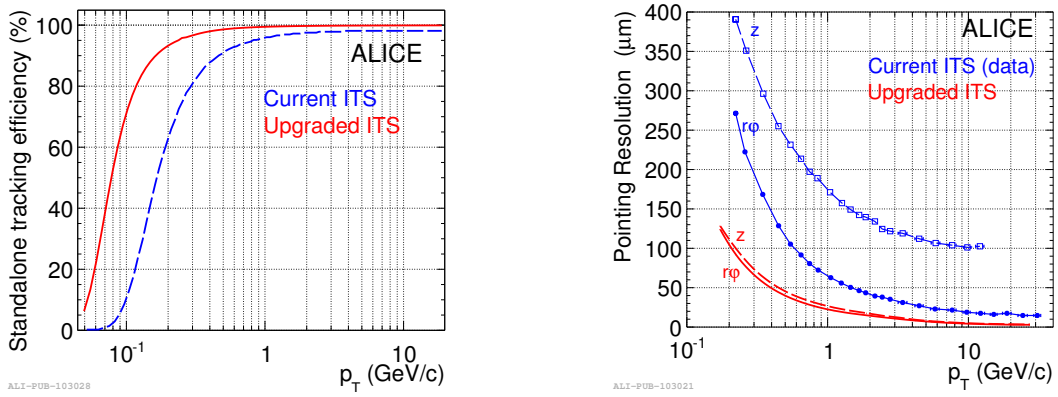
Layer	Length (mm)	Radius (mm)	(Half)-Staves [a] (#)	Chips (#)	Hit density (cm $^{-2}$)
0	271	23	12	108	18.6
1	271	31	16	144	12.2
2	271	39	20	180	9.1
3	843	194	44	2464	2.8
4	843	247	56	3136	2.7
5	1475	353	80	7840	2.6
6	1475	405	92	9016	2.6

As seen in fig. 2.1, the layers of sensor chips in the ALICE ITS are further organized into two barrels; the Inner Barrel (IB), consisting of the three Inner Layers (ILs) and situated closest to the IP; and the Outer Barrel (OB), which consists of the two Middle Layers (MLs) and the two Outer Layers (OLs). "Staves" of sensor chips are the building blocks of each layer, and come in two main configurations: The Inner Barrel (IB) stave, consisting of 9 sensor chips; The Outer Barrel (OB) stave, which comes in a configuration of 112 sensor chips for the MLs, and a 196 chip configuration for the OLs.

The number of sensor chips and staves per layer is summarized in table 2.1, along with the cylindrical layers' radius from the beam pipe, and expected hit densities for minimum-bias Pb–Pb at 100 kHz. As the particles that are produced in collisions shoot out from the IP, the density of tracks (and hence, hits in the sensor chips) is much higher in the innermost layers where the distance to the IP is shorter. Furthermore, the expected hit density in the innermost layers is even higher due to the so-called "QED background" (electron-positron pairs produced in peripheral and ultra-peripheral collisions [20]). These electrons and positrons are typically absorbed in the inner layers and do not contribute significantly to the hit density in the outer layers.

Compared to the previous inner tracker, the upgraded ITS will have a significantly improved tracking efficiency, as shown in fig. 2.2A. Another figure of merit is the impact parameter resolution[3], which is also improved with the new system.

[3]The impact parameter is the distance between the center of two colliding particles, an important quantity to determine if a collision is central or peripheral.



(A) (B)

FIGURE 2.2: Tracking efficiency (A) [21], pointing and impact parameter resolution (B) of the upgraded ITS compared to the old ITS [22].

2.4 The ALPIDE MAPS for the ITS Upgrade

In contrast to the former ITS, which combined a variety of silicon detector chips and readout ASICs, the upgraded ITS was designed to utilize a custom MAPS[4] chip for all seven layers of the detector. It is the single most important component of the ITS, and its development dates back to 2011.

The 180 nm TowerJazz Complementary Metal Oxide Semiconductor (CMOS) imaging process was chosen because it allowed for CMOS-logic in a deep p-well in a p^- epitaxial layer. This is illustrated in fig. 2.3. The sensitive area of a pixel consists of an n-well in the p^- epitaxial layer, surrounded by a p^+ guard-ring. The n-well and p^- epi-layer forms the charge collection diode, which is reverse biased via the pwell that forms the guard-ring. The reverse-biasing increases the size of the depletion region, and the electric field allows charges to drift towards the n-well electrode where it is collected. The digital logic is implemented in a deep p-well, and that effectively isolates it from most of the charge liberated from particle hits [23].

Several early chip prototypes were developed to study the performance of different pixel designs and geometries in the TowerJazz technology, as well as designs for the digital logic, which culminated with the final design for a pixel sensor: the ALPIDE chip.

The ALPIDE has a pixel matrix of 1024×512 pixels which covers a total area of $29.94\,\text{mm} \times 13.76\,\text{mm}$. The size of an individual pixel is $29.24\,\mu\text{m} \times 26.88\,\mu\text{m}$, and each pixel includes a charge collection diode, amplification chain and discriminator

[4] A MAPS chip comprises both the sensitive elements and readout circuits on the same die, as opposed to the hybrid pixel sensors of the previous ITS with the sensitive elements and readout circuits on separate dies.

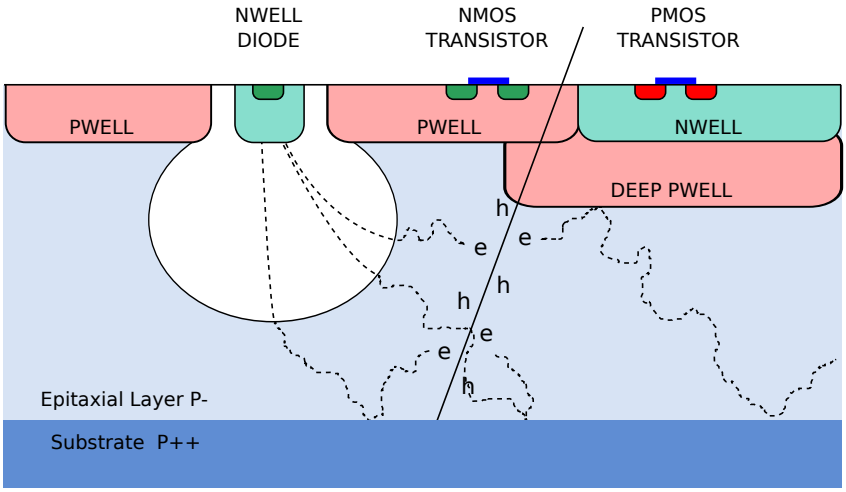


FIGURE 2.3: Cross-section of charge collection diode and CMOS-logic for a pixel in the TowerJazz 180 nm process [13].

circuit. The full size of the chip is 30 mm $\times$ 15 mm, which also includes Digital to Analog Converters (DACs), Analog to Digital Converters (ADCs), and a digital periphery measuring 30 mm $\times$ 1.208 mm at the fringe of the chip. Everything is contained on one die, owing to the MAPS design of the ALPIDE.

Analog Front-End and Pixel Discrimination

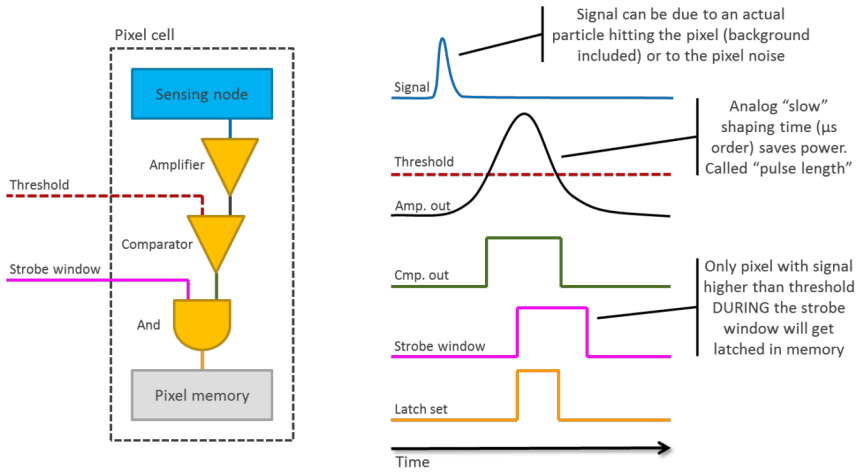


FIGURE 2.4: Illustration of pulse shape output from the pre-amplifier in the analog front-end of the ALPIDE chip, and the digital pulse output from the comparator. [24].

Each pixel in the ALPIDE chip features the amplifier stage in fig. 2.4. In summary the process of detecting and storing a hit goes through the following steps:

1. As a particle traverses through the chip it liberates charge (electrons) in the substrate.

2. An electric field from the substrate to the N-side of the charge collection diode (the "Sensing node" in fig. 2.4) causes the liberated charge to drift and be collected by the charge collection diode [5]. The collected charge, which may be of the order of only a few electrons, gives rise to a weak voltage pulse.

3. A Pulse Shaping Amplifier (PSA) amplifies and shapes the pulse from the sensing node. The width of the pulse after the PSA is typically on the order of 5 µs.

4. Next in the chain is a comparator, which takes the shaped pulse from the previous stage as one input, and a configurable threshold[6] as the other input. When the input pulse is higher in voltage than the threshold, the comparator output is high.

5. The comparator output is gated with the strobe window[7]. One of the three latches in the pixel memory is set when they coincide. Each pixel memory latch is associated with one of the three slices of the chip's Multi Event Buffer (MEB).

Readout

The internal readout logic of the ALPIDE, as well as triggering, framing of events, and data output, is explained in more extensive detail in appendix B.

The chip features a MEB that can hold three events. It essentially consists of three pixel buffers of 1024×512 bits, one buffer per event. When a trigger is received the chip will latch hits into one of the MEB's buffers. The 1024 columns of pixels are divided into 32 regions of 32×512 pixels. Two pixel columns form a double column, and hits in a double column are read out by a priority encoder (one encoder per double column). The priority encoders within a region are read out in a round-robin fashion into a region FIFO, and the 32 regions are read out in parallel. Data from the region FIFOs are then read out, one region at a time, to be transmitted off the chip as data frames.

The data is transported off the chip over a high-speed differential serial link, operating at 1200 Mbps for IB chips and 400 Mbps for OB chips, and employing a standard

[5] The substrate of the chip, which forms the P-side of the charge collection diode's PN-junction, may be biased at negative voltages (down to 6 V) at the chip's PWELL and SUB pads, for enhanced performance.

[6] The threshold is configured using the DACs on the chip.

[7] The length of the strobe window is configurable, and the each strobe window is associated with a trigger.

K28.5 8b10b encoding scheme. For control and triggering of the chip there is a multi-drop serial control link operating at 40 Mbps.

Pixel Clustering

The pixels in the ALPIDE share the same epitaxial layer, with no insulating barrier between them, and the charge that is liberated by a particle hit is typically shared among the surrounding pixels [25]. This generally leads to a cluster of pixels firing per particle hit with a cluster size related to the amount of liberated charge. To reduce the amount of data to transfer off the chip, up to eight neighboring hits in the same double column can be clustered into a single data word, which reduces the amount of data to transfer. This is explained in more detail in appendix B.5.3 of appendix B.

Modes of Operation

The ALPIDE chip is highly configurable with a rich set of registers that can be accessed via the Differential Control bus (DCTRL) link. Most of these configuration options are out of scope for this thesis, but there are a few important modes and features of the chip that needs to be mentioned:

1. Inner Barrel and Outer Barrel modes
2. Triggered and Continuous mode
3. Internal sequencer

Inner Barrel and Outer Barrel Modes. The ITS IB has a much smaller radius than the OB, and consequently the IB ALPIDE chips have a much higher occupancy than those of the OB. A dedicated data link operating at 1200 Mbps is absolutely needed for the IB chips to cope with the hit densities they are subjected to. However, in the OB the hit densities are so much lower that one 400 Mbps data link shared between seven chips is sufficient, and this called for a chip that could operate in two different modes. Of course it would be easier to design a chip with a dedicated data link operating at a fixed data rate, but this would probably have made the OB stave design impossible, as it would have required a dedicated differential data link to be routed to each of the stave's 196 chips.

For this reason, the ALPIDE features an IB mode where the chip transmits data on a dedicated 1200 Mbps data link [8], and an OB mode where seven chips share one 400 Mbps data link.

[8] In IB mode the data rate is 1200 Mbps by default, but can also be configured for 600 Mbps and 400 Mbps. OB chips are limited to 400 Mbps only.

An OB chip can either be configured to be a master or slave. The OB master chip receives data from the slave chips on the 320 Mbps [9] parallel local bus, and transmits their data, as well as its own data, on the 400 Mbps differential serial link that it controls.

The choice of IB, OB master and OB slave is configured by the CHIPID pins, which are typically hardwired on the stave to assign the geographical ID of a chip. The mode can not be reconfigured in any configuration registers.

Triggered and Continuous Mode. Two major trigger modes are planned for ALICE in Run 3: *triggered mode* and *continuous mode*. Events are captured in triggered mode based on minimum-bias triggers distributed from the Central Trigger Processor (CTP), as seen in fig. 2.5, which is analogous to taking pictures with a camera. The opposite analogy would be to record video with the camera, and that is basically how the experiment operates in continuous mode where all events are continuously captured, which is illustrated by fig. 2.6.

For the ITS the ALPIDE chips will be configured to have a short strobe, on the order of 100 ns, for triggered mode. Longer periodic strobes are used in continuous mode, on the order of 10 µs, with a short gap between each strobe.

The ALPIDE chip itself features two modes; triggered mode, and continuous mode. As their names imply, they are intended to be used with the corresponding mode for the experiment. However, it is important to note that the triggered mode and continuous mode in the ALPIDE **have no effect** on strobe length and how triggers are processed in the chip. The main differences between the two modes are listed in table 2.2, and relates only to how the MEB is handled. To summarize the table, new events are dropped in triggered mode when the MEB is full, and continuous mode

[9] Local bus is not encoded. The 400 Mbps serial link is 10b8 encoded, and has an effective data rate of 320 Mbps as well.

prioritizes new events and discards older data when the buffers are full.

TABLE 2.2: Comparison of triggered and continuous modes in the ALPIDE.

Triggered mode	**Continuous mode**
Allows all 3 buffers of the MEB to be used	Allows 2 buffers of the MEB to be used, keeps one buffer free for new events
Busy when all 3 buffers of the MEB are used[a]	Busy when 2 buffers of the MEB are used[a]
Rejects incoming triggers when busy, and sends an empty frame marked with a "busy violation" flag for the rejected trigger	Accept triggers even when busy as long as 1 buffer is available in the MEB[b] but flushes the oldest buffer in that case. Flushed events are marked with a "flushed incomplete" flag.

[a] The chip will also go busy when the frame FIFO is near full.

[b] If all buffers of the MEB are in use this would lead to a busy violation, just like triggered mode. Should not happen with the foreseen triggering schemes for continuous mode.

In principle, it is possible to use the chip in triggered mode but with the experiment running in continuous mode (i.e. periodic triggers/strobes, and long strobe lengths). Chapter 6 includes some simulations where this has been performed.

Internal Sequencer. The ALPIDE chip features an internal sequencer which can generate strobes with a configurable length, and with a configurable gap between the strobes. When the sequencer is enabled, it must be activated by issuing one trigger to the chip. The sequencer will then run indefinitely until it is deactivated via the control bus.

The continuous mode envisioned for ALICE in Run 3 can be achieved using this internal sequencer, along with the continuous mode in the chip. Or alternatively, the periodic triggers for continuous mode can be issued externally, with the internal sequencer deactivated. But the strobe must still be configured for an appropriate length in the chip.

As previously mentioned, the continuous and triggered modes in the chip only affect how the MEB is handled by the chip, so it is also possible to use the internal sequencer with the chip in triggered mode.

2.5 ITS Detector Staves

For the stave assemblies the ALPIDE sensor chips are glued and wire-bonded[10] to FPCs, which routes the clock, control, and data signals to Samtec FireFly connectors at the end of the stave. In addition to the FPCs, the general stave assembly consists of a

[10]Some sources mention laser-soldering the chips directly to pads on the Flexible (FPC), but due to quality issues this technique was later discarded in favor of wire-bonding.

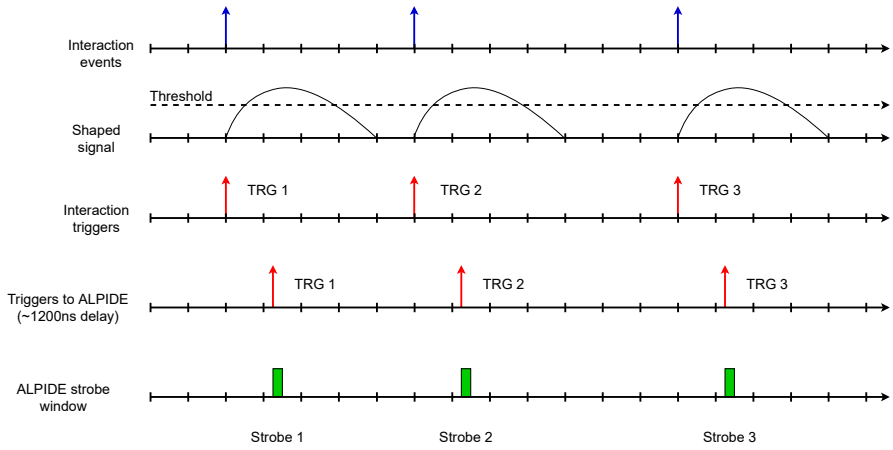


FIGURE 2.5: Waveform illustrating operation with minimum-bias trigger.

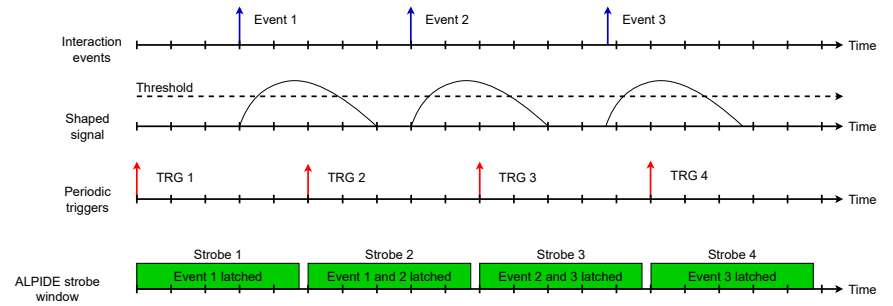


FIGURE 2.6: Waveform illustrating operation with continuous/periodic trigger.

cooling plate with inlets and outlets for coolant, and a carbon-fiber space-frame which holds the assembly together and gives the stave some structural rigidity. Illustrations of the assemblies are shown in fig. 2.7.

The IB staves consist of just one FPC containing all nine chips, and the IB FPC also routes power, ground and back-bias voltage to the chips. Figure 2.8 shows the connections on the stave. Each chip has a dedicated data line, but the multi-drop control bus is shared between the chips.

The OB stave assembly is more complex than the IB stave. It is assembled from four or seven (depending on ML or OL stave) OB-HICs[11]. The HICs are connected at each end, but the density of signals requires power to be routed by dedicated FPCs for a Power Bus (PB) and Bias Bus (BB), which connects to cross-cables on the HICs (see fig. 2.9A).

The connections on a OB half-stave is shown in fig. 2.10. One control line is shared between one row of half-modules, and each half-module has a dedicated data line. A

[11] A Hybrid Integrated Circuit (HIC) in this context refers to an assembly of the FPC with ALPIDE chips, e.g. IB-HIC of 9 chips, or OB-HIC of 14 chips.

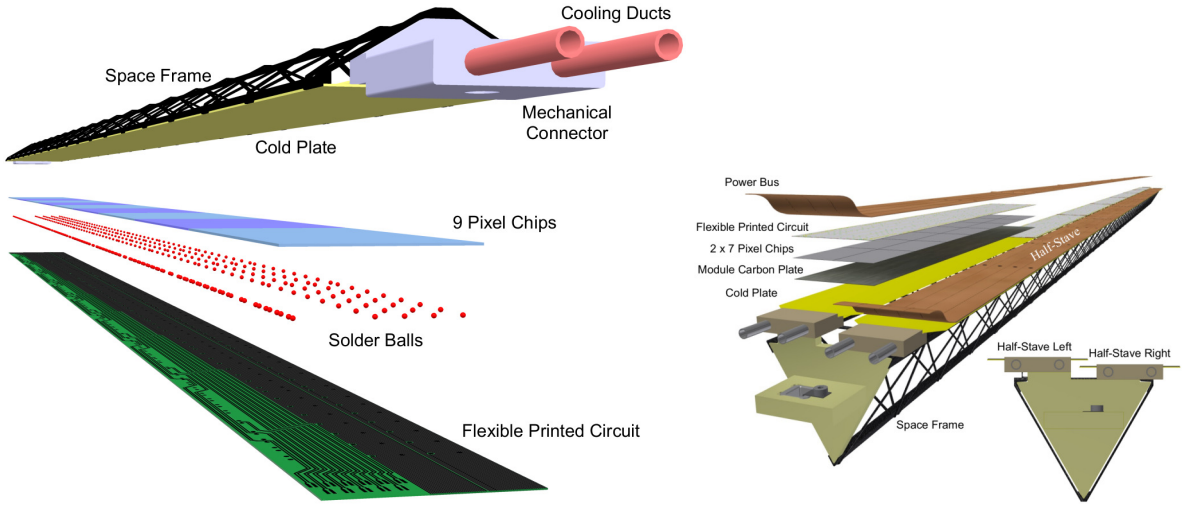


FIGURE 2.7: Illustration of stave assemblies for the IB (A) and OB (B) of the ITS [13].

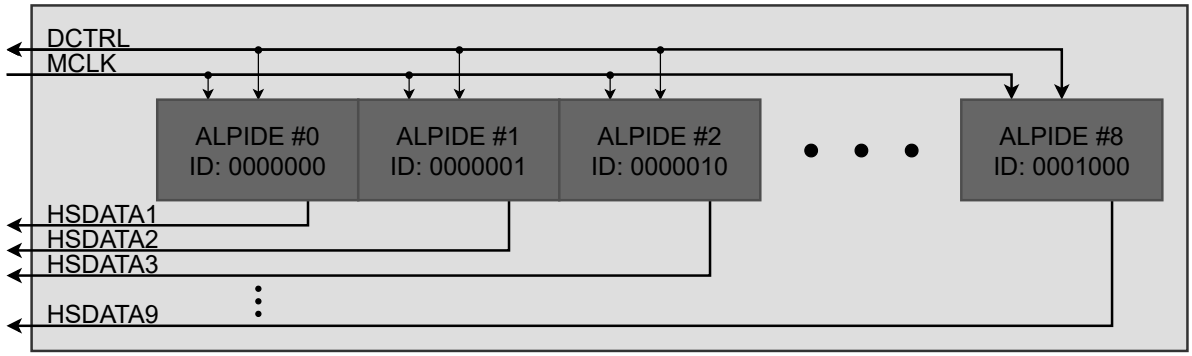


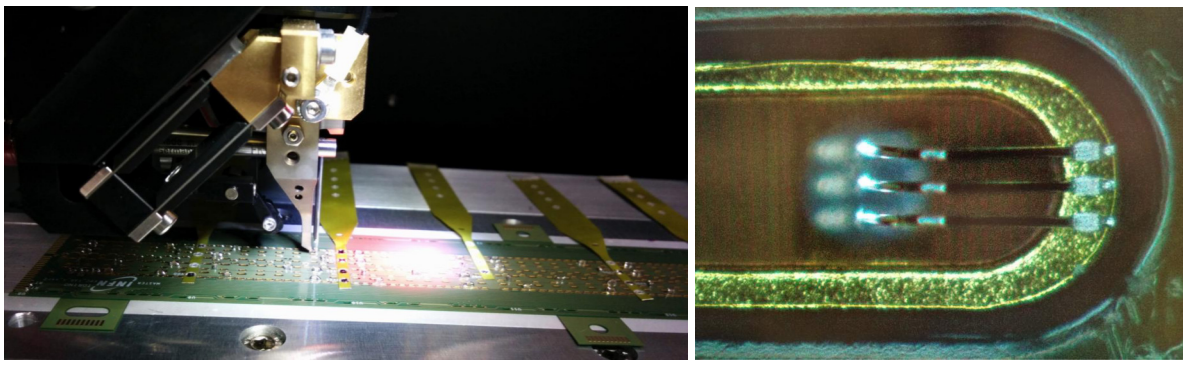
FIGURE 2.8: IB stave schematic (based on schematics from ALPIDE manual [26]). The three Most Significant Bits (MSBs) of the CHIP-ID are all low to put the chips in IB mode.

full OB stave consists of two half-staves like this.

The internal wiring of the OB module is shown in fig. 2.11. The master chip buffers and distributes the clock to the slave chips, and relays the control transactions to and from the slave. Data from the slaves is sent to the master on a shared local bus, which it transmits off the stave on one dedicated data link. A local connection is also used by the slaves to indicate busy status so the master can report it on the outgoing data link.

2.6 Trigger Distribution

The upgrades to the ALICE Trigger and Timing System (TTS) introduces a new CTP and a Local Trigger Unit (LTU). The CTP receives and processes inputs from trigger



(A) (B)

FIGURE 2.9: A: Wire-bonding of ALPIDE chips to the FPC of an OB module [27]. The chips are not visible in the picture, as they have been glued to the FPC and are facing down in the wire-bonding machine. The picture also shows some of the cross-cables for ground, power, and bias, which are connected to the PB and BB during stave assembly. B: Wire-bonding of a ground or power via/pad on the FPC to a pad on the ALPIDE chip (the small square in the middle) [27].

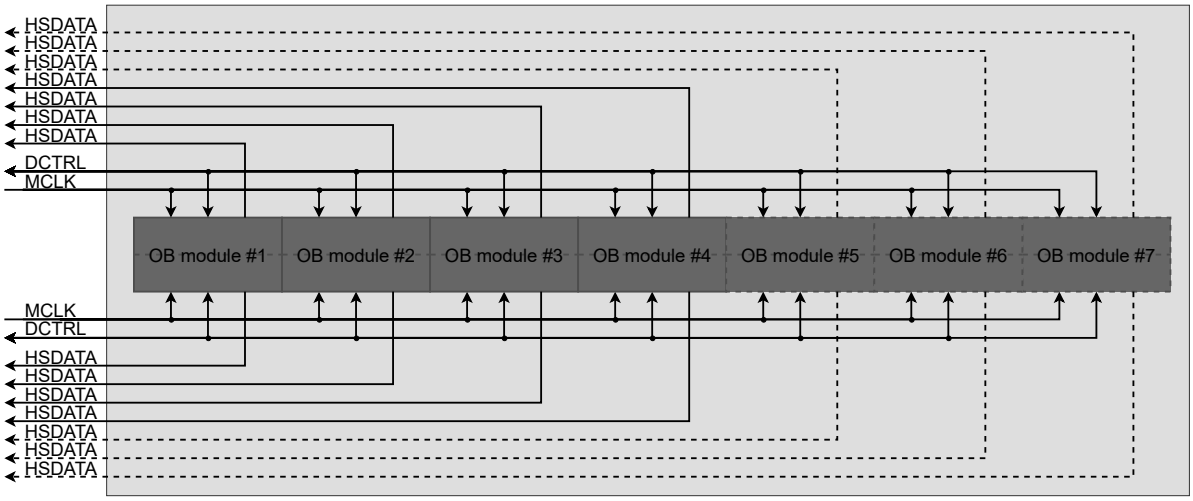


FIGURE 2.10: OB stave schematic (based on schematics from ALPIDE manual [26]). The modules and HSDATA lines with dashed lines indicate the three additional modules for the OL stave compared to the ML stave.

detectors in the experiment, and generates several levels of trigger signals. The triggers are sent to the LTUs which distribute the trigger signals either directly to the detectors or via the Common Readout Units (CRUs)[12]. The same hardware is used for the CTP and LTUs, but in different configurations.

The trigger signals from the CTP and LTUs are used by the ITS in the triggered mode. However, the alternative continuous mode does not use these trigger signals.

[12]The LTU can distribute triggers via a variety of protocols and optical standards, e.g. GigaBit Transceiver (GBT) and TTC-PON, to cater for new and old detectors.

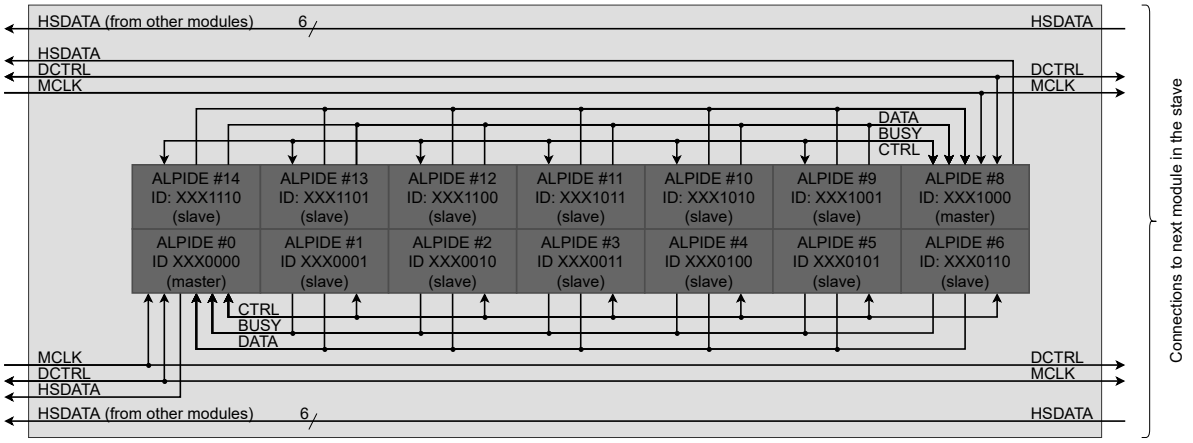


FIGURE 2.11: OB module schematic (based on schematics from ALPIDE manual [26]). Note that CHIP-ID 7 (b0111) is skipped as this ID is reserved for broadcast. All zeroes for the three Least Significant Bits (LSBs) identifies a chip as an OB master chip. The three MSBs indicate the module ID and at least one of them must be 1 to put the chips in OB mode.

All events are captured in this mode and triggers are not tied directly to events; they are continuously generated and sent to the detector at a constant rate.

In either of the two aforementioned modes, the triggers are distributed to the ALPIDE sensor chips by the *main FPGA* (see section 2.7.2) in the RUs using the control links to the chips. The control links do not add significantly to the trigger delay due to the fast trigger decoding of the trigger words (refer to appendix B.4.2).

The CTP is also responsible for the generation of the so-called HeartBeat (HB) triggers [28], which should not be confused with the physics triggers. The concept of HB trigger and frames is explained in more detail in appendix C.2. It is mentioned here because they are also used as a time base for trigger generation by the ITS readout electronics in continuous mode.

2.6.1 Triggered Operation

When the experiment operates in triggered mode the ITS relies on the new low-latency Level Minus (LM) trigger which is introduced with the LS2 upgrades of the TTS [29]. These triggers are sent to the readout electronics on dedicated GBT downlinks from *one* LTU. The LTU has eight optical SFP+ modules installed for the trigger downlinks. The SFP+ outputs are split in a 1:2 ratio, with 16 trigger lines going to 16 sub-racks for the readout electronics. These lines are split again with a 1:16 ratio[13] by a passive optical splitter at each sub-rack, before they are distributed to the RUs in the racks.

[13]The splitting ratios were carefully chosen to ensure that there is a sufficient optical power budget. A lower splitting ratio would have required more than one LTU.

Trigger Delay

Several delays are associated with the trigger signals that are distributed to the sensor chips. When the FIT detects an interaction in the experiment it generates the LM input signal for the CTP. It takes 425 ns for the signal to reach the CTP's inputs. After an additional 190 ns for CTP processing and distribution to LTUs, the trigger is transmitted to the readout electronics over 35 m of optical cables, which delays the signal by an additional 175 ns. Another 250 ns is required for processing and transmission on the ALPIDE control links before it is received by the sensor chips. The total delay adds up to 1230 ns [30].

Trigger Filtering

Since the Time over Threshold (ToT) for a pulse after the PSA is generally around 5 µs (see fig. 6.6), the pulse lives on in the analog front-end for much longer than the trigger delay. As a consequence, when the time between two events is shorter than the shaping time, pixel hits from both events may be sampled in the second trigger, as indicated in fig. 2.12.

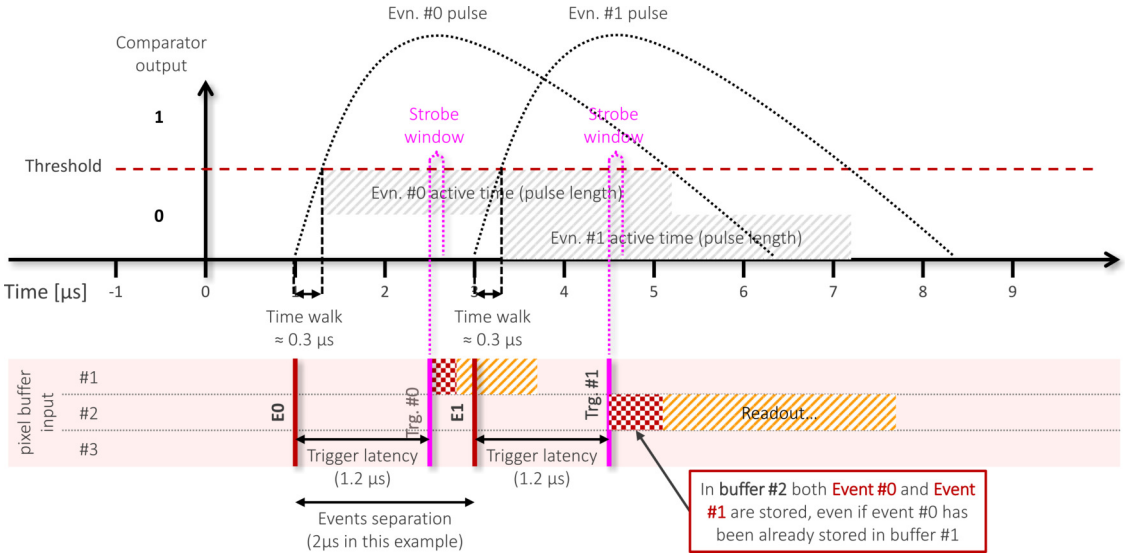


FIGURE 2.12: Two events happen with sufficient time between them so that the first trigger only captures the first event. The second trigger captures both events. [30]

In the case where the spacing between two events is even shorter than the trigger latency, pixel hits from both events may be sampled in each of the two triggers. When the first trigger arrives in the chips, hits from the second event may have gone over threshold, so they are sampled in the first trigger. When the second trigger arrives, the hits from the first event are still over threshold because of the long pulse shaping time, so those hits are also included in the second trigger. This is illustrated in fig. 2.13.

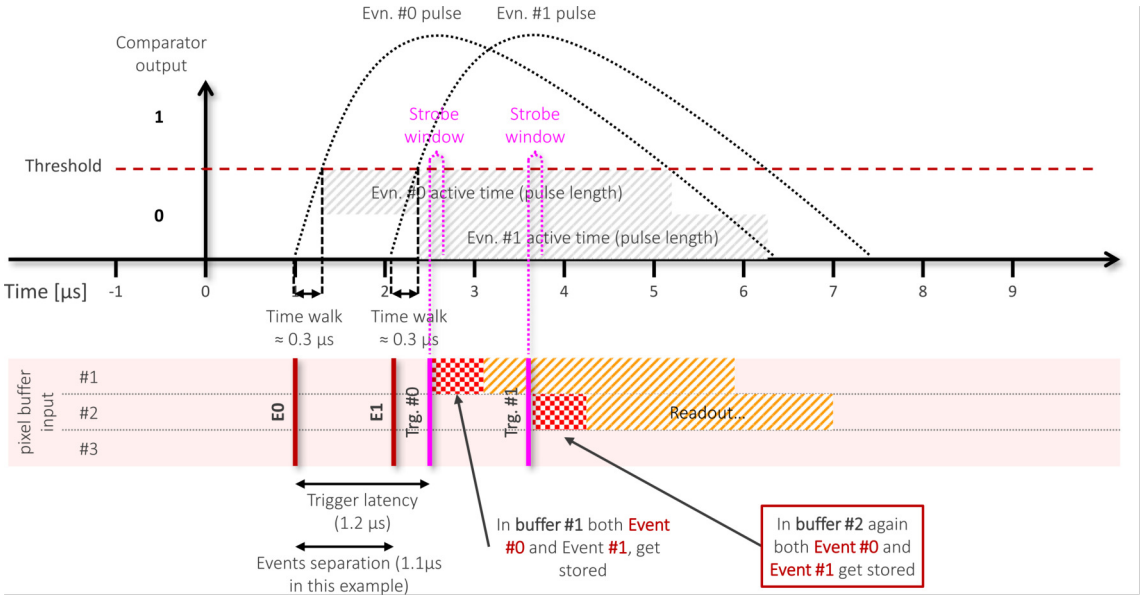


FIGURE 2.13: Two events happen with insufficient time between them, both triggers would capture both events. [30]

Since both triggers would capture the same information in this case, issuing them both is essentially a waste of bandwidth and buffers in the chips. To avoid this the RU can filter out triggers that come too close to the previous trigger (i.e. the time between triggers is less than the trigger latency).

2.6.2 Continuous Operation

The minimum-bias LM-trigger from the CTP is not used when running in continuous mode. Operation in this mode is in principle much simpler. The periodic HB triggers, which main purpose is for framing of detector data, are used as a time base by the RU for generation of the continuous triggers, of which there are several per HB.

2.6.3 Busy Signaling

The sensor chips are in a busy state when they are (temporarily) unable to receive a trigger without subsequent loss of data, typically because all event buffers are in use (see table 2.2) and in some cases because the *frame FIFO* with trigger information is full. More details are available in appendix B. Changes in busy status is reported to the readout electronics using the data link and takes priority over normal event data. Busy status for the sensor chips and readout electronics is reported to the CRUs. In principle, a busy status for the entire ITS can be reported to the CTP which receives an individual busy signal from each detector in the experiment. How these signals are

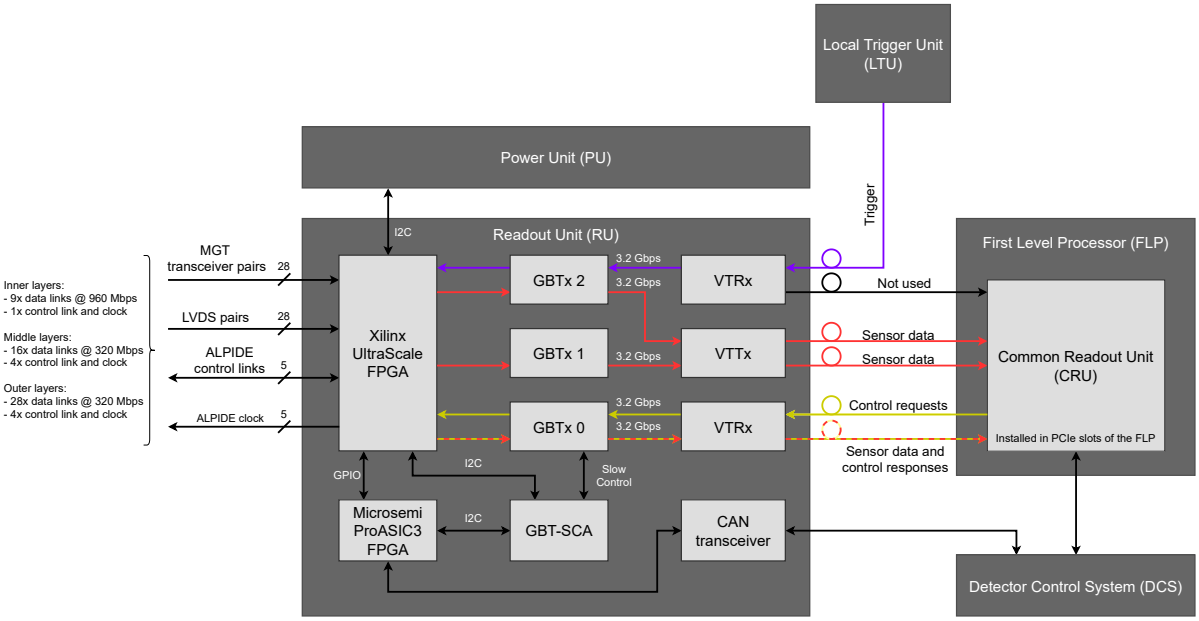


FIGURE 2.14: RU and main interfaces for control, trigger, and data readout.

handled by the CTP will change with the LS2 upgrades: "In contrast to the Run1[14] scenario, a detector which is busy will not interrupt the entire ALICE data acquisition." [29]. But the CTP will not send triggers to a busy detector. For the ITS it has been decided to not report a busy status to the CTP, since the aforementioned behavior is not desirable and would have a negative impact on readout efficiency. The round-trip to report a busy status from sensor chip to the CTP via the CRU, and distribute a trigger from the CTP back to the chips, is too long (order of 2 µs) for efficient busy handling using the CTP. Determining a single busy status for the ITS based on the busy status of over 24 000 sensor chips would also pose a challenge. And simulations have shown that the readout efficiency of the detector is very good under all anticipated operating conditions (see chapter 7).

2.7 ITS Upgrade Readout Electronics

The readout electronics for the ITS consists of the Readout Unit (RU) and the Power Unit (PU). There are 192 RUs, one for each of the 192 staves in the detector. They are responsible for trigger distribution, control and configuration, and of course data readout from the sensor chips. The entire readout chain from pixel detector to the First Level Processors (FLPs), and including trigger distribution from the LTU, is shown in fig. 2.14. The PUs distribute power to the sensor chips, and they are controlled by the

[14] And Run 2.

RUs. A dedicated Busy Unit (BU) was also envisioned at some point, which would monitor the busy status of the detector and use that to influence trigger distribution. However, the BU concept was later discarded; it will be shown in later chapters that the frequency of busy at nominal interaction rates was not high enough to justify its existence.

2.7.1 Radiation Environment

The copper cables used for the ALPIDE data links requires the readout electronics to be placed close to the IP. The signal integrity would deteriorate severely if the distance is too long. Consequently the RUs for the ITS have to be placed in a radiation environment, at around 3 m from the IP and 1 m from the beam line axis. The position of the readout electronics is indicated in figs. 2.15 and 2.16, which shows the levels of Total Ionizing Dose (TID) and high-energy hadron fluence in the experiment[15].

The RUs are expected to be subjected to a TID of up to 10 krad, and a hadron fluence of $1 \times 10^{11}\,\mathrm{cm}^{-2}$ of 1 MeV Neutron Equivalent Fluence (NEF) [31]. These numbers are specified with a safety factor of ten[16]. The hadron fluence should not be a concern with regards to component degradation and reliability. However, hadrons with energies higher than 20 MeV can cause latch-ups and Single Event Upsets (SEUs) in the electronics. The flux of high-energy hadrons is around $1 \times 10^{3}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ [31], also with a safety factor of ten.

2.7.2 Readout Unit

The RU required a high-performance FPGA to handle several gigabits per second of sensor data over the numerous high-speed interfaces. Several high-speed transceivers and IOs were necessary for the data links of the ALPIDE sensor chips. High-speed IOs were also required for the interface between the FPGA and the GBTX chips, which are used for the optical GBT interface.

The choice of FPGA fell on the *Xilinx UltraScale XCKU060*, which is an SRAM-based FPGA (i.e. it uses volatile SRAM memory for its configuration). An SRAM-based FPGA is blank and requires configuration after a *power-on reset*. Configuration of the FPGA can be implemented in different ways (see section 4.3.1), but the configuration image is typically stored in a non-volatile memory device external to the

[15]Note that the TID and hadron fluence has not been simulated for the RUs in these figures; the boxes merely indicate the position of the RUs and were added to the figures.

[16]Figure 2.15 does not show these numbers as the position of the readout electronics is superimposed on the plot. However, one would expect the levels to be similar to the levels of the TPC FEE outside the TPC (excluding the safety factor), estimated to 0.86 krad [32].

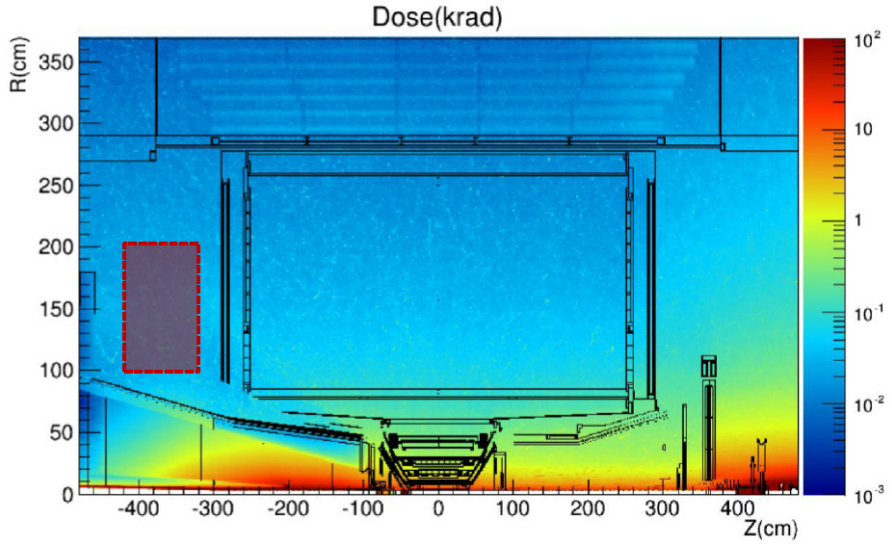


FIGURE 2.15: TID in ALICE for Run 3. The beam-line runs parallel to the Z-axis at $R = 0$, and R is the distance from the beam-line. The red box with the dashed lines was added to indicate the position of the crate for the RUs. Source: [32].

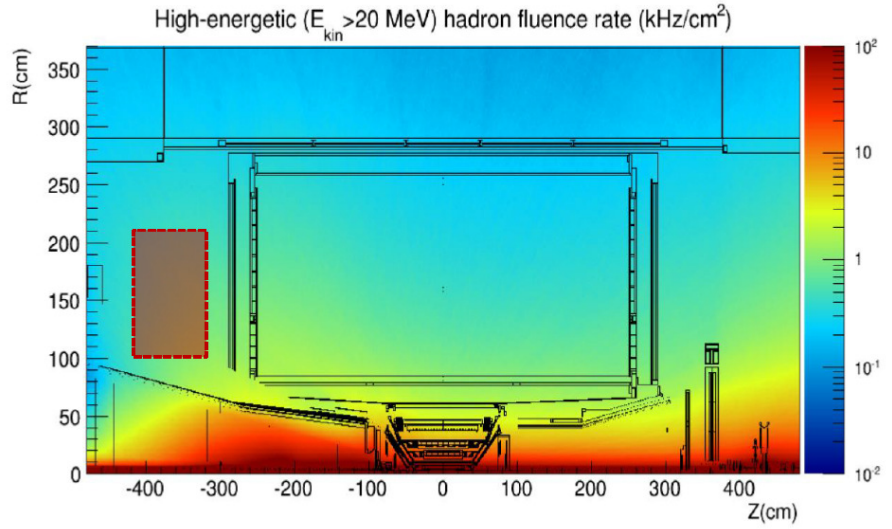


FIGURE 2.16: High-energy (> 20 MeV) Hadron fluence in ALICE and for the ITS RU [32].

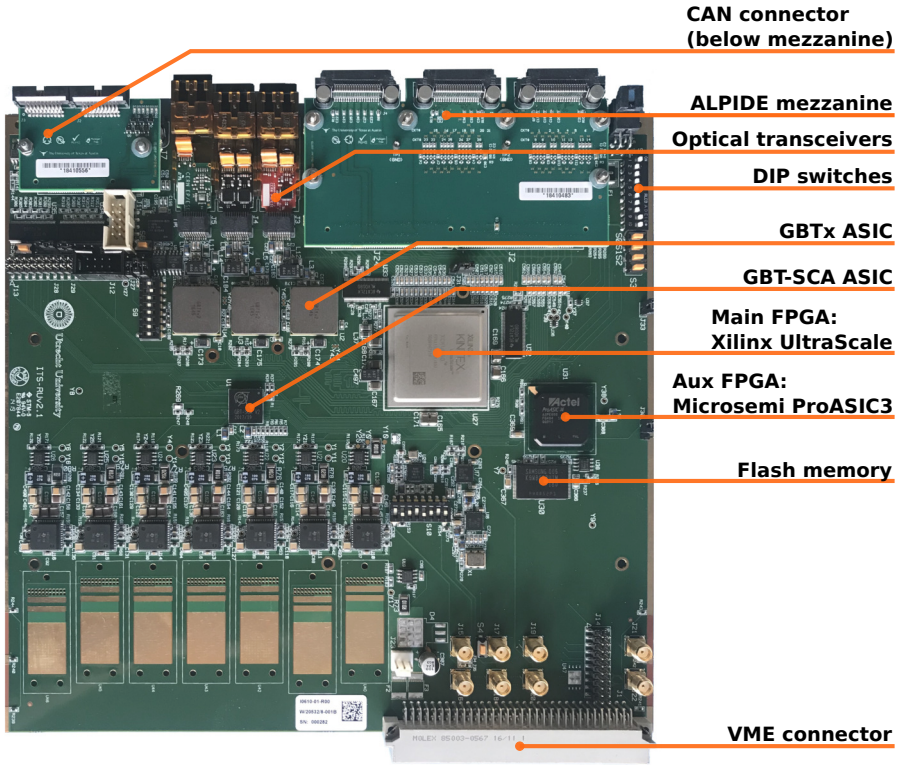


FIGURE 2.17: RU PCB version 2.1 [35].

FPGA. The FPGA's (internal) configuration memory is also susceptible to SEUs and requires *scrubbing*[17] if the FPGA is operating in a radiation environment. For this reason, the RU has an additional *auxiliary FPGA* for configuration and scrubbing of the UltraScale *main FPGA*. A low-end flash-based FPGA, the *Microsemi ProASIC3 (PA3)*, was chosen for the task along with an external flash memory device to store the main FPGA's configuration image. The auxiliary FPGA itself is configured using JTAG and does not require reconfiguration after a power-on reset due to its non-volatile flash-based configuration memory, which is considered immune to SEUs [33]. This is a proven method for external scrubbing which has seen use in other detectors and experiments, such as for the Readout Control Unit (RCU) of the ALICE TPC [34].

A possible alternative to the two-FPGA approach would have been to use a single, high-performance, flash-based FPGA. Some flash-based FPGAs were considered for the RU design, such as the *Microsemi SmartFusion 2*, but ultimately the choice fell on the UltraScale as it offered better performance [18].

The circuit board for the ITS RU is shown in fig. 2.17. The next few paragraphs will discuss the design in more detail.

[17] Continuously refreshing the configuration to restore bits that have flipped due to radiation.

[18] Budgetary considerations also played role in the decision, which for instance ruled out the *Microsemi RTG4*.

Transition Board and Interface to Sensor Chips

A custom *Samtec Twinax FireFly* cable assembly is used to bring clock, control and data signals, to and from the sensor staves. The FireFly-cables connect to the *transition board*, which is a mezzanine board on the RU. The transition board for the ITS is shown in the upper right of fig. 2.17.

Modern FPGAs feature high-speed Multi Gigabit Transceivers (MGTs) which are generally the preferred way to receive data from interfaces of ≥ 1 Gbps, such as the 1200 Mbps ALPIDE data links of the IB staves, although the standard LVDS GPIOs of the UltraScale FPGAs are fast enough to implement a GPIO-based front-end for the links of the IB staves, as proven by the Proton CT (pCT) design at UiB [36], [37]. However, it should be noted that the pCT readout unit required a GPIO-based solution due to the high count of ALPIDE data links per pCT Readout Unit (pRU) and limited number of transceivers available in the FPGA. The FPGA transceiver count is not a limitation for the ITS, and a transceiver-based front-end to the 1200 Mbps ALPIDE data links is preferable. But the GTH/GTY transceivers of the Xilinx UltraScale FPGAs require a minimum line rate of 500 Mbps [38], which means they can not be used in a front-end for the 400 Mbps data links of the OB staves. A GPIO-based front-end is therefore still required for the OB staves.

The transition board connects to the main RU board via two 100-pin connectors. These two connectors provide connections to 28 transceiver pairs on the UltraScale FPGA, as well as 28 standard LVDS GPIO pairs on the FPGA. Signals are routed to the transceiver or GPIO pairs from five Firefly connectors on the transition board; one for the IB stave interface which connects to transceiver pairs; and four for the OB stave interface which connects to the LVDS pairs.

The limited space on the front panel required some of the FireFly connectors to be stacked. Figure 2.18 is an illustration of the RU board and front panel, and shows the stacked FireFly connectors on the left. The connectors could not have been placed directly on the RU PCB to achieve this, it was necessary to place them on the transition board. But the transition board solution also allows the ITS RU to be used by other detectors. For instance the MFT detector in ALICE, which is also based on the ALPIDE pixel sensor [39]. The MFT requires a different number of transceivers for the ALPIDE data links, and a dedicated transition board design for the MFT makes it possible to use the ITS RU.

Optical GBT Interface

The GBT interfaces are based on CERN's *GBT chipset*, which comprises the GigaBit Transceiver ASIC (GBTX) along with the Versatile Twin-Transmitter (VTTx) and Versatile TransReceiver (VTRx) optical transceivers. Based on prior simulations of the readout for the ITS [40], it was determined that three GBT uplinks were sufficient to transfer data upstream to the FLPs. One dedicated GBT downlink is required for control requests from the Detector Control System (DCS) (see section 2.8), and one is required for triggers. Control responses are multiplexed with data on one of the upstream links. As illustrated in fig. 2.14, the optical interfaces are realized with: three GBTX chips; one VTTx transceiver with two uplinks for data; one VTRx transceiver with a downlink for control, and an uplink shared for data and control; and a final VTRx with a downlink for triggers. The latter has an uplink which is not used.

There is also a GBT Slow Control Adapter (GBT-SCA) ASIC on the RU board. It provides some control and monitoring functionality that rely solely on GBTX 0 (see fig. 2.14) using the dedicated *slow control* bits in the GBT frame for communication (more details in appendix C.1). The GBT-SCA is used to monitor voltages and temperatures on the RU board, enables external control of the auxiliary FPGA using one of the GBT-SCA's I^2C interfaces, and allows for remote configuration of the FPGAs using JTAG.

Other Interfaces

Figure 2.18 shows the RU board with the front-panel. The interfaces for the ALPIDE and GBT links have already been discussed, but there is also a number of other interfaces. The power supply is normally connected to the leftmost connector on the front-panel, but can also be supplied via the Versa Module Eurocard (VME) connector at the back of the board. This is the only purpose of the VME connector, it will not be connected when the boards are installed in crates in the experiment. On the right side of the front-panel, there are LEDs for status indication; JTAG connectors for both FPGAs; PU connector; an auxiliary connector to communicate busy status between RUs; and a connector for CAN bus (bottom right). The CAN interface is used by the DCS to control the RUs. This is normally performed using GBT, but an additional control path is provided to the DCS for redundancy in case the GBT interface is down.

Clock Distribution

The two beams that circulate the LHC, in opposing directions, consists of *bunches* of particles (technically, they are not continuous beams) [42], [43]. An orbit is divided

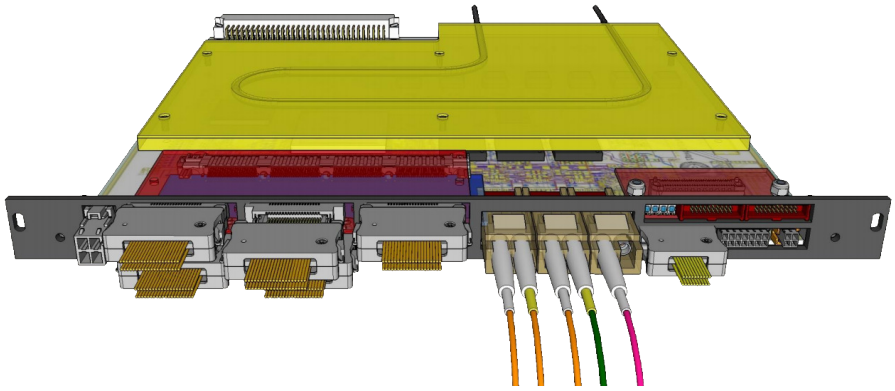


FIGURE 2.18: Illustration of RU design with front-panel and cooling plate. [41]

into 3 564 bunch positions, each separated by approximately 25 ns, and the Bunch Crossing (BC) clock has a frequency of 40.079 MHz[19] (typically referred to as the LHC clock). This clock is distributed to all four experiments and their detectors to keep them synchronized with bunch crossings.

There are two primary clock sources for the FPGAs on the ITS RU:

- A local 160 MHz oscillator on the RU board.
- An external 160 MHz clock provided by one of the GBTX chipsets.

The latter is derived from the LHC clock, as the GBTX recovers the LHC clock from received GBT frames (which are synchronized with the LHC clock) [44].

The main FPGA will primarily use the external clock, when available, but has a clock multiplexer in the design to automatically switch to the local clock if the optical links are down. There are several clock domains in this FPGA. The internal data bus and most of the logic runs at 160 MHz, and there is a 40 MHz clock domain for the ALPIDE control links, as well as some other clocks such as for the GPIO-based ALPIDE data frontend.

The auxiliary FPGA performs no functions that require synchronization with bunch crossings. It runs solely off the local 160 MHz clock, which makes it immune to any potential issues with the external clock. The design for this FPGA has a single 40 MHz clock domain (derived from the local 160 MHz clock using a clock divider module).

[19] For brevity, 40 MHz will be used in place of 40.079 MHz (and 160 MHz for the related 160.316 MHz clock).

Configuration Paths for the FPGAs

Configuration of the two FPGAs is possible via a number of different paths using the different interfaces and features of the RU board. They are illustrated in fig. 2.19. The JTAG interface allows for direct configuration of the FPGAs. The interface is connected to both FPGAs and to the GBT-SCA chip. It allows for local configuration of the FPGAs by connecting a JTAG programmer to one of the JTAG connectors and remote configuration over GBT slow control [45].

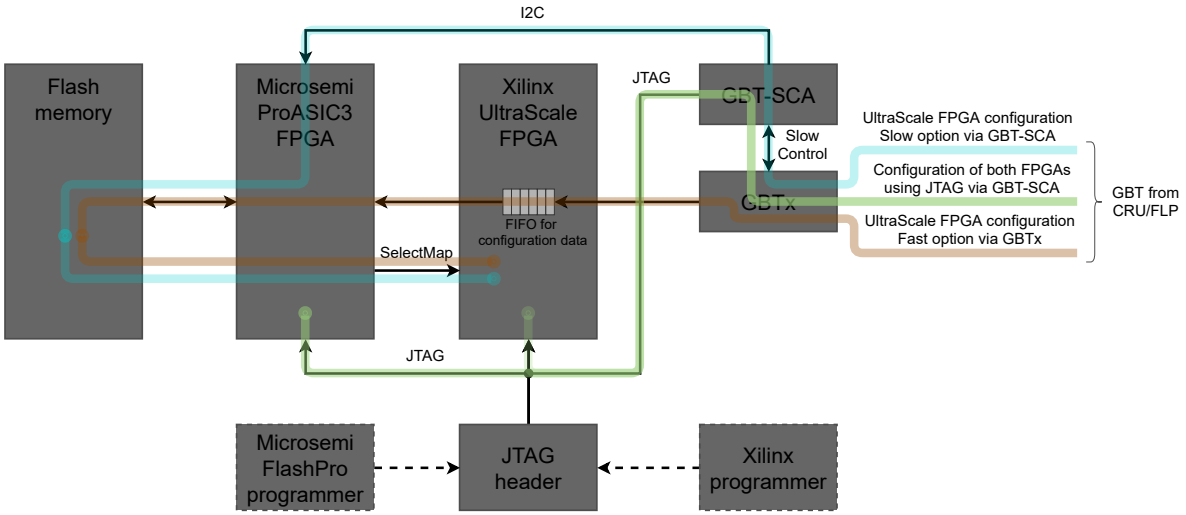


FIGURE 2.19: Readout Unit Configuration Paths for the FPGAs.

It was mentioned earlier that the UltraScale FPGA is normally configured by the PA3 FPGA using configuration data stored in the external flash. There are two paths by which the configuration data in the external flash can be updated: using the GBT-SCA and its I^2C interface to the PA3 FPGA, or via the GBTX to the UltraScale and to the PA3 using a parallel interface between the two FPGAs. The latter option is significantly faster, but requires the UltraScale FPGA to be configured and running already.

Radiation Tolerance

The active components in the RU design must tolerate the expected levels of radiation. The GBTX chipset, which comprises the GBTX and GBT-SCA ASICs, and the VTTx and VTRx transceivers, were designed for radiation tolerance [46]. Their radiation performance has already been tested and characterized by the GBT group at CERN. The other components on the board are Commercial off-the-shelf (COTS). CERN maintains a database of COTS components and their radiation performance, and the COTS components on the RU were primarily chosen from this database. This

includes the PA3 FPGA and Samsung K9WBG08U1M Flash chip, the CAN transceiver, and the DC-to-DC (DC/DC) converters.

In addition to the board design and component choices, techniques for mitigation of Single Event Effects (SEEs) must be employed in both FPGA designs. The measures that were taken in the main FPGA design are described at the end of chapter 3. The scrubbing solution and the mitigation techniques employed in the auxiliary FPGA design are described in chapter 4.

2.7.3 Optional Busy Unit for the ITS

In principle there are two reasons why the ITS would be in a busy status:

1. The ALPIDE sensor chips are busy, either because no event buffers are free, or the frame FIFO reached the busy threshold (see appendix B.5.2).

2. FIFOs in the RUs' datapath are filling up faster than the RUs can transport the data upstream.

The second scenario is unlikely, because the total bandwidth of the three GBT uplinks is higher than the total bandwidth of all the ALPIDE data links. This is true for all three stave types. It should not be possible for the RU's datapath FIFOs to overflow unless the system is run without utilizing all the GBT data uplinks[20].

The primary concern is how the busy signals from the sensor chips should be handled. System simulations have shown that the amount of data loss due to busy sensors is relatively low (see chapters 6 and 7), and it was not necessary to include a dedicated BU or busy logic in the RUs.

Nevertheless, the readout electronics were designed to cater for a BU, although a BU has not been designed for the system. This makes it possible to add a busy solution to the system if the need should arise, for instance if the experiment should run at higher interaction rates than foreseen for Run 3. Some concepts for a busy system is described in appendix F.

2.7.4 ITS Power Board

The ALPIDE sensor chips requires an 1.8 V analog and digital supply, as well as a negative back-bias[21]. On the IL staves the supply lines are routed on the flex-PCB for

[20] This is ignoring the protocol overhead associated with the heartbeat frames, which should be a very small fraction of the links capacity. The capacity of the GBT links are higher than the combined capacity of the ALPIDE data links.

[21] Typically -6 V.

TABLE 2.3: Number of staves, RUs and PBs per layer.

Layer	Stave count	RU count	PB count
0	12	12	6
1	16	16	8
2	20	20	10
3	24	24	12
4	30	30	16
5	42	42	42
6	48	48	48

the chips themselves. The ML and OL staves have dedicated power-bus and bias-bus layers in the assembly, with connections going to flex-PCBs for the half-modules of seven sensor chips. Each half-module is powered individually in the middle and outer layers.

The necessary voltages to operate the sensor chips is generated and supplied by the ITS Power Board (PB) [47]. One PB is an assembly consisting of a sandwich of two near identical[22] PU boards[23], with a cooling plate between them. Each PU has 16 power channels which can be controlled individually. The PU itself is controlled and monitored by the main FPGA of an RU over an I^2C interface. There are also connections for PT100 temperature sensors on the sensor staves, which are continuously monitored by the RUs via the PU board.

The ML and OL staves use one PU channel per half-module [48]. A full PU is used for the 16 half-modules of an OB half-stave, and a full PB for a full OB stave. One PU is sufficient to power a full ML stave, which consists of 16 half-modules (8 half-modules per half-stave). An IL stave requires power from two channels of a PU. In principle, it would have been possible to supply power to eight IL staves with one PU. But each IL stave has a dedicated PU since the PU has to be controlled from the stave's RU.

2.8 Detector Control System and Online-Offline (O2)

The Detector Control System (DCS) is used for control and monitoring of power supplies and voltages, temperatures, pressures, etc., for all detectors and associated systems in the ALICE experiment. It is implemented using WinCC OA, a commercial Supervisory Control And Data Acquisition (SCADA) system [49]. Communication with the ITS RUs is done primarily over GBT via the FLPs, but CAN bus is also used.

[22]The PU comes in a left configuration, and a right configuration, which are essentially mirrored versions of the same board design.

[23]The terminology can be a bit confusing since the term PB refers to an actual assembly of two boards.

The actual detector data is shipped from the FLPs to the server farms of the Online-Offline (O^2) system, where the data is combined into complete events and stored for later analysis.

Further details about these two systems and their upgrades are out of scope for this thesis, but they are introduced here for the sake of completeness.

Chapter 3

Main FPGA Design for the ITS Readout Unit

This chapter provides an overview of the FPGA design for the Xilinx UltraScale main FPGA. The development of the design was led from CERN, and the main contributors were Joachim Schambach, Matteo Lupi, Matthias Bonora, Gianluca Aglieri Rinella, Arild Velure, and Ola Slettevoll Grøttvik. Joachim was the project's leader, and he also developed the GBT communication, control and communication with the PU, and early trigger handler and datapath. Matthias developed the initial versions of the Alpide data decoder and datapath, and created the regression framework for hardware test and simulation. Matteo devised the methods for radiation hardening and triplication that were employed in the FPGA design, and was the main developer and maintainer of the Alpide Control module. The final version of the datapath and trigger handling was developed by Matteo, Arild, Gianluca, and Joachim. Gianluca contributed with his experience and expertise and also initiated development of extensive UVM testbenches for verification of several modules. Arild did a tremendous job setting up the CI build environment for both FPGA projects. He also made countless improvements and fixes to every aspect of the FPGA design, and so did Ola, who joined the project in 2021 and has had the important job of maintaining the project during the late commissioning phase and preparing for the start of Run 3. They deserve credit for all their hard work, and I apologize for any inaccuracies. My main contributions to the project were the modules for the CAN bus interface, the Alpide Monitor, and also the FIFO interface to the PA3 for configuration data.

3.1 General Structure

The design for the main FPGA is illustrated in fig. 3.1. The design does not feature a Central Processing Unit (CPU)[1,2], but is a pure logic design for the programmable fabric of the FPGA, designed at the Register Transfer Level (RTL) using Hardware Description Language (HDL)-code. It implements the main functionality of the RU for triggering and readout, as well as other functionality. To control the FPGA there are three communication interfaces that can be used: GBT, CAN, and Universal Serial Bus (USB). These interfaces allow for direct access to certain registers in the FPGA design via an internal bus.

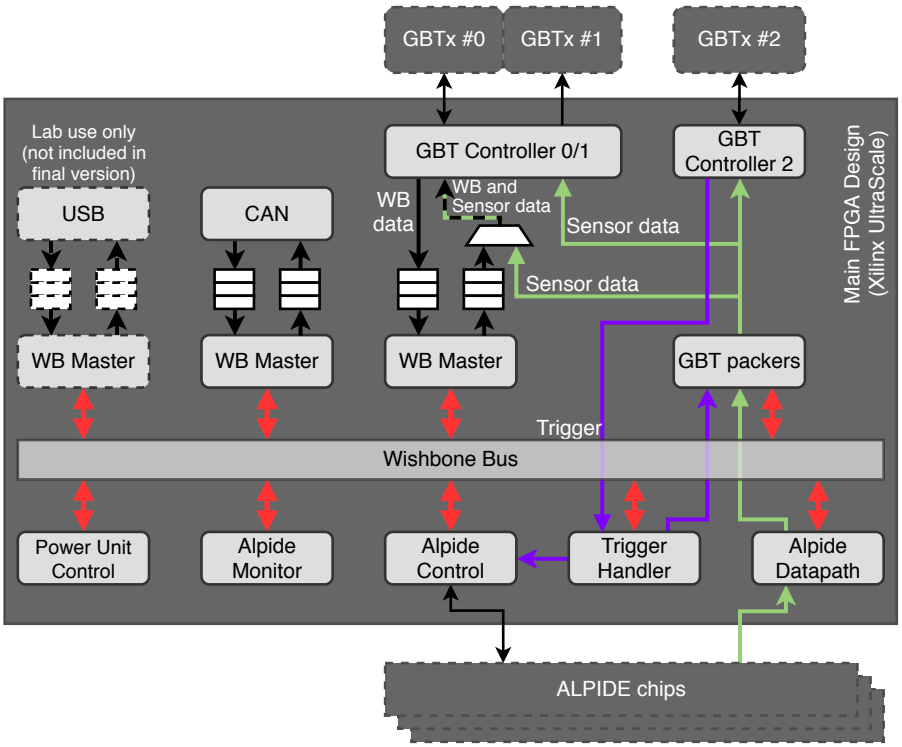


FIGURE 3.1: FPGA design for the Xilinx FPGA [35].

[1]A CPU would need to be protected with Triple Modular Redundancy (TMR), just like the other modules in the design, and would require additional resources in the FPGA. The firmware that runs in the CPU would require dedicated memory, which would also have to be protected, and a way to transfer it to the FPGA is necessary.

[2]The use of a CPU *can* make some aspects of development easier, by allowing complex Register Transfer Level (RTL)-logic to be implemented in firmware for the CPU. But for the main (and also the auxiliary) FPGA design the challenges associated with a CPU were deemed too great compared to the benefits.

3.1.1 Wishbone Bus

The backbone of the FPGA design is a Wishbone (WB) bus with 15-bit address width and 16-bit data width. Access to this bus allows for configuration and control of every module in the design. In total, there are 49 WB slave modules, and 3 masters which are tied to the aforementioned communication interfaces. Because of the bus' importance in the design, the entire bus is protected using TMR (including the slave and master interfaces).

Bus Masters

A generic WB bus master with two FIFO interfaces is used by all the interfaces that provide master access to the WB bus, as indicated in fig. 3.1. The interface to the WB master consists of one FIFO for WB transaction requests, and one FIFO for WB transaction results. A round-robin arbiter (not shown in the figure) coordinates bus access for the masters. The combination of the FIFO interfaces and arbiter allows sequences of WB transactions to be queued, and execution of the transactions to be performed asynchronously with respect to when they were scheduled.

Addressing Space

Internal registers in the modules are addressed using the lower eight bits of the 15-bit address, and the modules are selected by the upper 7 bits in the address. The WB masters' FIFO interfaces use a data width of 32 bits, and each FIFO-entry contains both the data and address for a transaction. The 32^{nd} bit is used to indicate read or write access in the request FIFO, and it is used to indicate bus errors in the result FIFO.

3.1.2 FEE ID

Each RU in the system is configured with a unique ID which identifies which detector stave it is responsible for. This ID is configured using eight Dual In-line Package (DIP)-switches[3] on the RU board. Upstream data transmitted to the CRU is labeled with the FEE ID, and the FEE ID is also used as a node ID in the CAN control interface.

Table 3.1 shows how the IDs are assigned. The stave number in a layer is indicated by the LSBs of the ID, i.e. the bits below the *Layer ID Mask*. As an example, consider the RU for stave number 5 in layer number 4. This RU would have *Layer ID* of *0110 0000*, and the *Stave ID* would be *0000 0101* (i.e. *Stave ID* 5 in decimal notation). The full FEE ID is obtained by computing the boolean *OR* of the *Layer ID* and *Stave ID*, which yields: *0110 0101*.

[3] DIPSWITCH[9:2] on the RU board design.

TABLE 3.1: FEE ID.

Layer	Layer ID	Layer ID Mask	Stave ID Mask
0	0000 XXXX	1111 0000	0000 1111
1	0001 XXXX	1111 0000	0000 1111
2	001X XXXX	1110 0000	0001 1111
3	010X XXXX	1110 0000	0001 1111
4	011X XXXX	1110 0000	0001 1111
5	10XX XXXX	1100 0000	0011 1111
6	11XX XXXX	1100 0000	0011 1111

To find the *Layer ID* and *Stave ID* for an FEE ID, the process is reversed, and considering the *Layer ID* first in this order:

- If the MSB of the FEE ID is 1, then the *Layer ID* is either 5 or 6, and the *Layer ID Mask* of *1100 0000* should be applied.
- If the MSB of the FEE ID is 0, and any combination of the next two bits are 1, then the *Layer ID* is either 2, 3, or 4, and the *Layer ID Mask* of *1110 0000* should be applied.
- If the three MSBs of the FEE ID are all 0, then the *Layer ID* is either 0 or 1, and the *Layer ID Mask* of *1111 0000* should be applied.

After the *Layer ID* has been extracted from the FEE ID, the corresponding *Stave ID Mask* (the inverse of the *Layer ID Mask*) for that layer is applied to find the *Stave ID*.

3.2 Detector Datapath

One of the main purposes of the readout electronics is, as the name indicates, to receive and aggregate data from the sensor chips in the detector, and to transmit the data upstream for storage and analysis. The datapath for the sensor data is consequently one of the most important components of the FPGA design.

A whole range of modules and submodules come together to implement the datapath in the design, but roughly speaking the datapath consists of two major components:

- The **IB and OB datapath** modules, which implement the ALPIDE data lane logic and FIFOs, and the front-end for each corresponding ALPIDE data link. The datapath associated with an ALPIDE data link is referred to as a "lane".
- The **GBT packers** which package data from several ALPIDE data lanes for transmission on a GBT uplink.

3.2.1 Datapath and Data Lanes

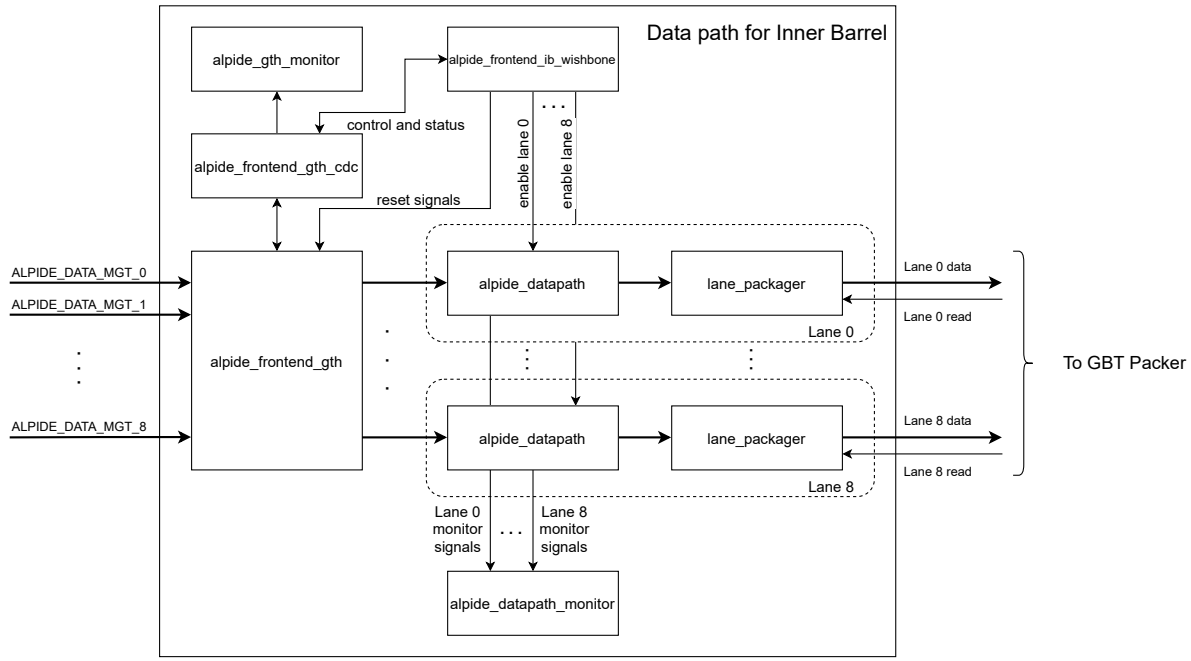


FIGURE 3.2: IB datapath in the main FPGA design.

The datapath for sensor data comes in one configuration for the IB, and one for the OB. The configuration for the IB is shown in fig. 3.2. It has a front-end for the ALPIDE data links which is based on the GTH-transceivers that are available in the Kintex UltraScale FPGA. The front-end receives data from the nine 1200 Mbps ALPIDE data links of the IB, and performs 8b10-decoding of the data stream. The front-end is followed by a *data lane* for each data link. The lane begins with a stage that processes the raw ALPIDE sensor data[4], performing idle suppression (i.e. removal of the ALPIDE's *IDLE* data words), and protocol checking and tracking. The idle-suppressed data is stored in a 16 kB FIFO which is read by the lane packager of the next stage, where the data is packaged in blocks of 9 bytes[5] to be transmitted over GBT.

The datapath implementation for the OB staves is almost identical, but uses a GPIO-based front-end for the slower 400 Mbps data links of the OB. It supports 28 data links and data lanes, but with a smaller lane FIFO size of 4 kB.

[4]The *alpide_datapath* block in the figure. Not to be confused with the full datapath for IB and OB.

[5]The payload size for one GBT frame is 80 bits, i.e. 10 bytes. One byte is reserved to indicate which lane the data belongs to.

3.2.2 GBT Data Packer

The data lanes from the IB or OB datapaths feed into the GBT packer where the data is packed based on physics triggers[6], and framed to conform with the Raw Data Header (RDH)[7] readout protocol that the CRU expects (see appendix C.5 for more details). A block diagram of the GBT packer is shown in fig. 3.3. It consists of three *GBT packer channels* (one channel per GBT uplink). The channels output data to be transmitted on their GBT uplink and multiplex between ALPIDE lane data, when available, and the different control words of the readout protocol over GBT (see appendix C).

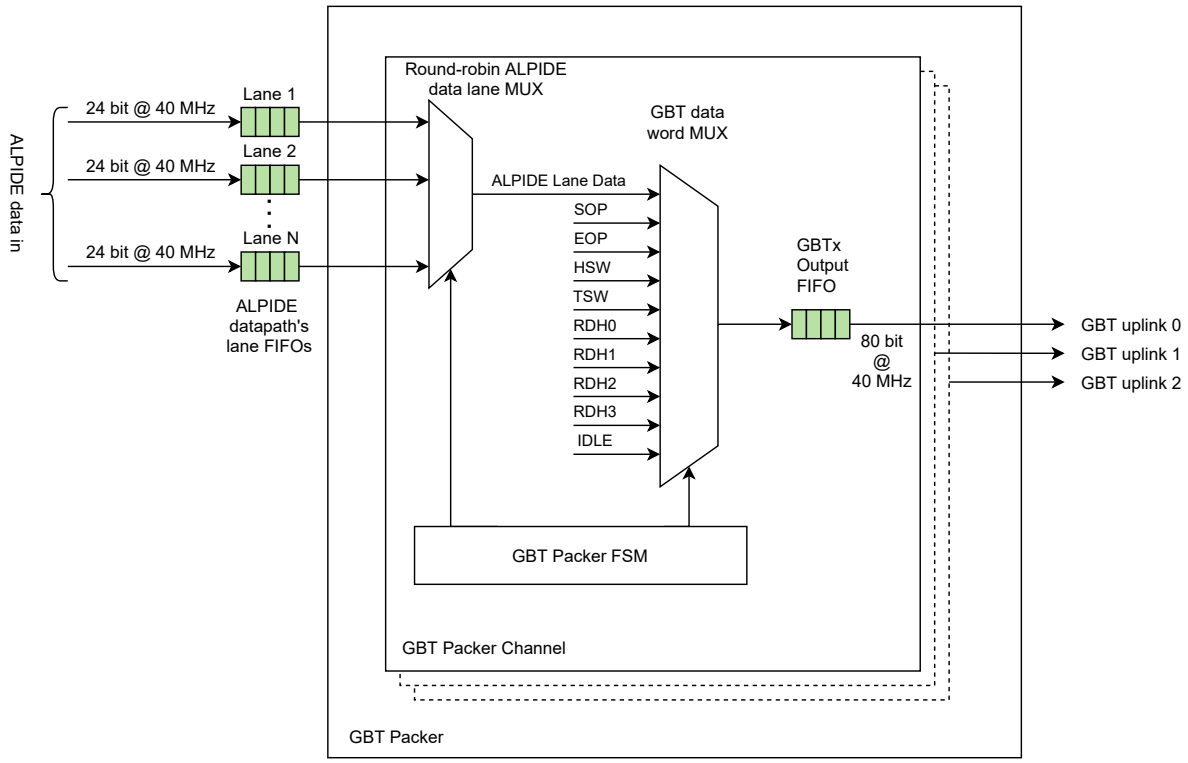


FIGURE 3.3: Block diagram of GBT packer.

The FIFO interfaces of the ALPIDE data lanes are presented to the GBT packer. In principle, the three GBT packer channels share access to **all** of the ALPIDE lane FIFOs, but each channel is configured for a certain range of lanes in the top-level GBT packer depending on which layer of the detector the RU is associated with (this is derived from the FEE ID). The RUs for the IB staves use all three GBT packer channels and GBT uplinks, with three ALPIDE data lanes assigned to each channel/uplink. But

[6]Either the LM-triggers from the CTP or the triggers that are periodically generated by the RU in continuous mode.

[7]At the time of writing, version 6 of the RDH format is used.

for the OB staves, only two of the GBT uplinks and channels are used, where each channel is responsible for half of the ALPIDE data lanes in the stave [8].

Packaged Data Format

The data is packed as illustrated in fig. 3.4 and transmitted on the GBT uplinks. Data for a trigger is split into packages delimited by the Start Of Packet (SOP) and End Of Packet (EOP) data words, and each SOP is followed by an RDH. Up to 512 GBT payloads of ALPIDE data can be transmitted per package [50]. If this is exceeded the page has to be closed with the EOP, and transmission resumed in a new package. The new package should also contain an RDH, and the page counter should have been increased in the RDH.

The GBT frames that carry ALPIDE data reserve one byte for a *GbtId* field, and the remaining nine bytes are used for ALPIDE data [50]. The *GbtId* field identifies which data lane the ALPIDE data in that GBT frame is coming from. This allows data from several ALPIDE data lanes to be interspersed in the same SOP/EOP-package, and the round-robin multiplexing between the lanes allow data to be transmitted efficiently as they come in.

3.3 Trigger System

The trigger system [51] of the main FPGA design receives TTS messages from the LTU via GBTX-2 under normal operation, such as the HB triggers and minimum-bias physics triggers (refer to table C.3 for a complete list of trigger types). An overview of the system is shown in fig. 3.5.

A timebase module has counters for Bunch Crossing ID (BCID) and orbit number, and provides these signals as a timebase for the rest of the FPGA design. It has the option of generating these itself, or synchronizing with TTS messages for orbit and BCID from the LTU. The trigger sequencer has the ability to generate sequences of TTS messages, essentially emulating the CTP/LTU. Together, these two modules can provide inputs for the pulser and trigger handler in the absence of signals from the LTU[9].

The pulser module generates periodic pulses with programmable period and duration relative to the timebase. One notable example is the *INT_TRG* signal which is the periodic trigger signal for the ALPIDE chips in continuous mode. Another is the

[8] 14 ALPIDE data lanes per GBT uplink and packer channel for the OL staves, and 8 ALPIDE data lanes per GBT uplink for the ML staves.

[9] Such as during testing in the laboratory.

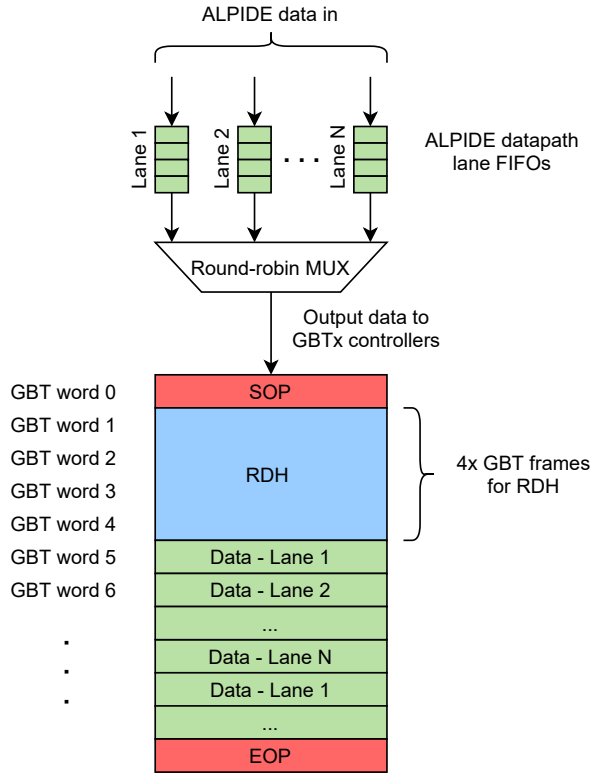


FIGURE 3.4: Illustration of packaged data output from GBT packer (based on an illustration in the specification for the ITS upgrade data format [50]).

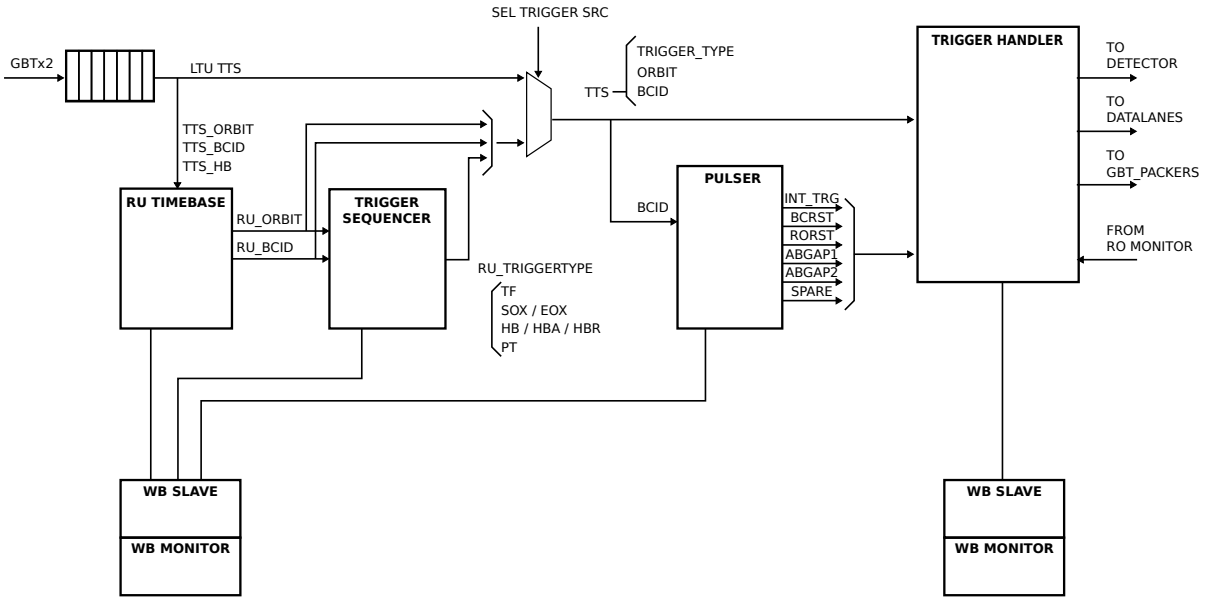


FIGURE 3.5: Trigger system in the main FPGA design [51].

ABGAP1/2 signals which indicates when the abort gaps of the LHC fill pattern are passing through the IP at ALICE. These signals will be used for the planned *Alpide Monitor* module, by allowing it to perform transactions on the ALPIDE control links

during the abort gaps without interfering with the trigger messages.

And finally, the trigger handler distributes the physics triggers[10] to the sensor chips using the *Alpide Control* module, which has a dedicated trigger input[11] to minimize the latency to the ALPIDE sensors. The physics trigger is also distributed to the modules associated with the RU datapath, which need a trigger to associate each frame from the ALPIDE sensors with, along with the HB triggers.

Another responsibility of the trigger handler is trigger filtering in the case where triggers are very shortly spaced (refer to sections 2.6.1 and 6.2.4 for details about the trigger filtering).

3.4 FIFO Interface to the Auxiliary FPGA for Configuration Data

One of the possible configuration paths for the main FPGA is via the GBT[12] to the main FPGA itself, and out on a parallel interface to the auxiliary FPGA which updates the external flash memory with the new configuration image (see section 2.7.2). A simplified block diagram of the module is shown in fig. 3.6.

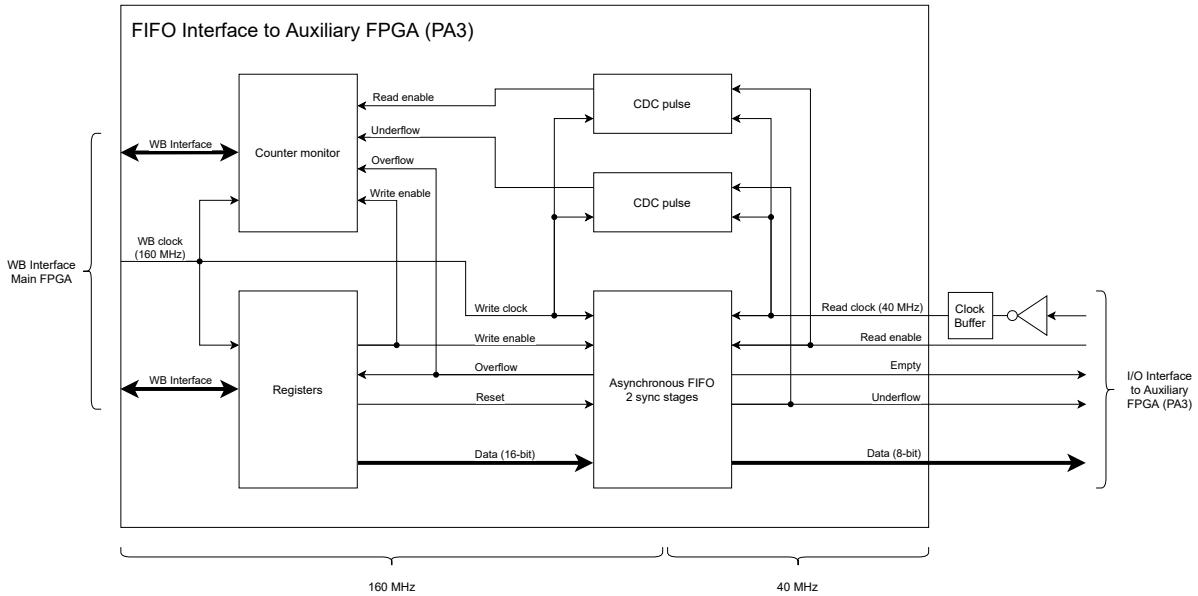


FIGURE 3.6: FIFO interface to the auxiliary FPGA for configuration data.

[10]Physics trigger from the CTP in triggered mode, or INT_TRG from the pulser module in continuous mode.

[11]Normal read and write transactions on the ALPIDE control bus are initiated via the Alpide Control module's WB interface.

[12]In principle, any of the WB master interfaces (GBT, CAN, USB) can be used to transfer configuration data, although the FIFO interface is only intended for use with GBT.

The parallel interface is a direct interface to the read-interface of the FIFO, i.e. the data output (which is parallel), empty signal, and the read control signal. The auxiliary FPGA itself controls when it should read the FIFO.

Writing to the FIFO is performed with a dedicated data register available from the module's WB interface. Any data that is written to this register will be put on the FIFO automatically. The FIFO itself is implemented using an asynchronous FIFO from the Xilinx Parameterized Macros (XPM) library, and Block RAM (BRAM) is used for the internal memory. The data width out of the FIFO is 8 bits, which is limited by the number of ports connected between the two FPGAs. But the data width into the FIFO matches the WB data width (16 bits), which allows data to be written to the FIFO more efficiently. The number of bytes that have been written or read, along with overflow and underflow, is counted by the module's counter monitor. The *full* signal of the FIFO is not used since the auxiliary FPGA should read out the FIFO faster than it can be written to.

The FIFO uses two Flip-Flop (FF)-stages for synchronization between the clock domains, and the signals from the 40 MHz side that are counted and monitored are passed through *CDC pulse blocks* from the XPM library [52]. Since the IO routes between the two FPGAs were not length-matched, it is possible that the rising clock edge would lead the *read enable* signal, and this poses some challenges when it comes to defining proper timing constraints for the interface. But since the interface is relatively slow, a simple solution was to invert the 40 MHz clock input, which places the rising clock edge in the middle of the *read enable* pulses with around $25\,\text{ns}/2 = 12.5\,\text{ns}$ of wiggle room.

Radiation induced errors in the module and data are not expected, since the FIFO will not be used during operation of the LHC. The module is still protected with TMR, but this is primarily to prevent radiation induced errors from affecting other modules in the design via the WB bus. The FIFO does not have built-in Error Correction Code (ECC) protection of its data, but the configuration data is typically ECC-encoded beforehand, and the auxiliary FPGA calculates a Cyclic Redundancy Check (CRC) checksum when configuration data is sent to the main FPGA over SelectMAP, which allows the correctness of the data to be verified at a later point.

3.5 Board Control Interfaces and DCS

The readout boards are controlled by accessing registers on the WB bus. There are three interfaces that provide access to the WB bus via a generic WB master module: USB, GBT, and CAN (see fig. 3.1). The USB interface is only used for testing and debugging in a lab setup, and will not be used when the detector is commissioned. The

optical GBT interface is the primary control path for the DCS, but a secondary control path is offered by the much slower CAN interface [53], [54]. And even under normal circumstances the CAN interface is used by DCS for monitoring of temperatures and voltages. The DCS has a software interlock that will cut power for a rack if abnormal temperatures are detected, and this system relies on communication over CAN bus.

3.5.1 GBT

Each GBTX interface is controlled by a set of 10 *E-links* which operate at 320 MHz [44]. The E-links are bidirectional, and each E-link consists of an output for downlink (receive) data and an input for uplink (transmit) data. The 10 E-links essentially work in parallel to provide 80 bits of payload data, in each direction, per LHC clock period. A *SerDes* in the GBTX performs the necessary serialization/de-serialization of the data.

Access to registers on the WB bus is implemented using Single Word Transaction (SWT) frames over GBT, with a custom protocol on top of the SWT frames. The relevant payload bits in the SWT frame are pushed and popped directly to and from the FIFOs between the GBTX controller and the WB master module. The implementation allows for one WB transaction per SWT, but since each WB transaction only requires 32 bits[13], the full 32 bits of a WB transaction is duplicated in the SWT frame which allows for an extra check for bit errors and utilizes the 80-bit GBT payload better.

3.5.2 CAN Bus

The CAN interface offers an additional control path for the DCS, and provides some redundancy in situations where the GBT links may not be available, such as when there are no beams, during maintenance periods, or because of hardware problems. In a similar fashion to the GBT interface, the CAN interface allows for access to the WB bus using a custom high-level protocol on top of the CAN frames.

CAN High Level Protocol for DCS

The High Level Protocol (HLP) for the CAN bus allows for access to any internal WB register in the main FPGA, and also implements an addressing scheme that allows multiple RUs to share a CAN bus line. The protocol was designed with the ITS RU in mind, and is implemented in the payload of standard 11-bit CAN frames. It is based on a similar protocol designed by Schambach et al. for DCS in the Time Of Flight (TOF) detector in the STAR experiment [55]. The simplicity of this protocol made it an appealing starting point for our custom protocol, instead of implementing a standard

[13] 16 bit data, 15 bit address, and one bit to indicate read or write, for a total of 32 bits.

protocol such as *CANopen* [14] or *DeviceNet*. These protocols are more complex, and an implementation in the FPGA design would likely have required more resources, and had a larger SEU cross-section.

In principle all nodes on a CAN network are masters and can initiate transactions, with the arbitration field of the package used for arbitration of the bus to avoid collisions. However, in our high-level protocol the DCS acts as a master on the bus which initiates the HLP read and write transactions, and the RUs are slaves that respond.

The three LSBs of the 11-bit ID of the CAN frames are used to indicate the type of command, and the upper eight MSBs of the ID are used for the node ID. The node ID corresponds to the FEE ID that was introduced earlier, which is configured using the DIP-switches on the RU board and should be unique for each RU. The payload field of the CAN frame is used for WB register address and data.

The available commands in the current version of the protocol are shown in table 3.2. An earlier version of the protocol was described in a previous publication [35].

Write and Read Commands. The most important transactions are the commands to write and read WB registers, and their corresponding responses. The DCS initiates a WB transaction by sending a HLP *write command* or *read command*, and the RUs responds with a *write response* or a *read response*.

TABLE 3.2: CAN High Level Protocol (HLP) commands and payload.

Command	Size	Byte 0	Byte 1	Byte 2	Byte 3	Bytes 4:7
Alert (0x0)	2	Data[15:8]	Data[7:0]	N/A	N/A	N/A
Write Command (0x2)	4	Addr[14:8]	Addr[7:0]	Data[15:8]	Data[7:0]	N/A
Write Response (0x3)	4	Addr[14:8]	Addr[7:0]	Data[15:8]	Data[7:0]	N/A
Read Command (0x4)	2	Addr[14:8]	Addr[7:0]	N/A	N/A	N/A
Read Response (0x5)	4	Addr[14:8]	Addr[7:0]	Data[15:8]	Data[7:0]	N/A
Status (0x6)	4	Data[31:24]	Data[23:16]	Data[15:8]	Data[7:0]	N/A
Test	8	0xAA	0xAA	0xAA	0xAA	0xAA

Other Command/Message Types. The *alert* and *status* messages in the protocol offer a way for the nodes (RUs) to communicate status changes to the DCS. Transmission of these messages is initiated by the node itself, and a custom payload can be sent with the message. The alert message is intended as a fast and reliable way for the node to report problems; it allows for two bytes of payload, and it is sent with the highest

[14] CANopen is limited to 127 nodes with its 7-bit node ID (one ID is reserved for broadcast). Although all 192 RUs are not physically connected on the same CAN bus line, it would have complicated the addressing of RUs.

possible priority. Status messages, in contrast, allows for four bytes of payload data, and is sent with a lower priority.

The alert message utilizes the fact that the arbitration field of a CAN message is transmitted with the LSB first. Since the HLP command is located at the LSBs of the arbitration ID, commands with lower IDs have higher priority. When a node wants to send an alert message, the alert will be sent before any other message type from any other node (the exception being alert messages from nodes with lower node ID).

The lowest priority message is the test message. When a node is put in test mode it will continuously send these messages. The payload of the test message is eight bytes, which is the maximum allowed payload in a CAN frame, and consists of alternating bits (0xAA-bytes). These messages produce many low-to-high and high-to-low transitions on the CAN bus lines, and can be used for eye diagram measurements, but have also proven useful for debugging of the CAN installation.

Implementation of CAN HLP

The implementation of the HLP protocol over CAN bus consists of two layers. First, there is the CAN controller which can transmit and receive standard CAN messages. The HLP protocol is implemented on top of that, with a Finite State Machine (FSM) that communicates with the CAN controller, implements the protocol logic and performs the WB transactions using the generic WB master module. In addition, the HLP layer has a WB slave for configuration.

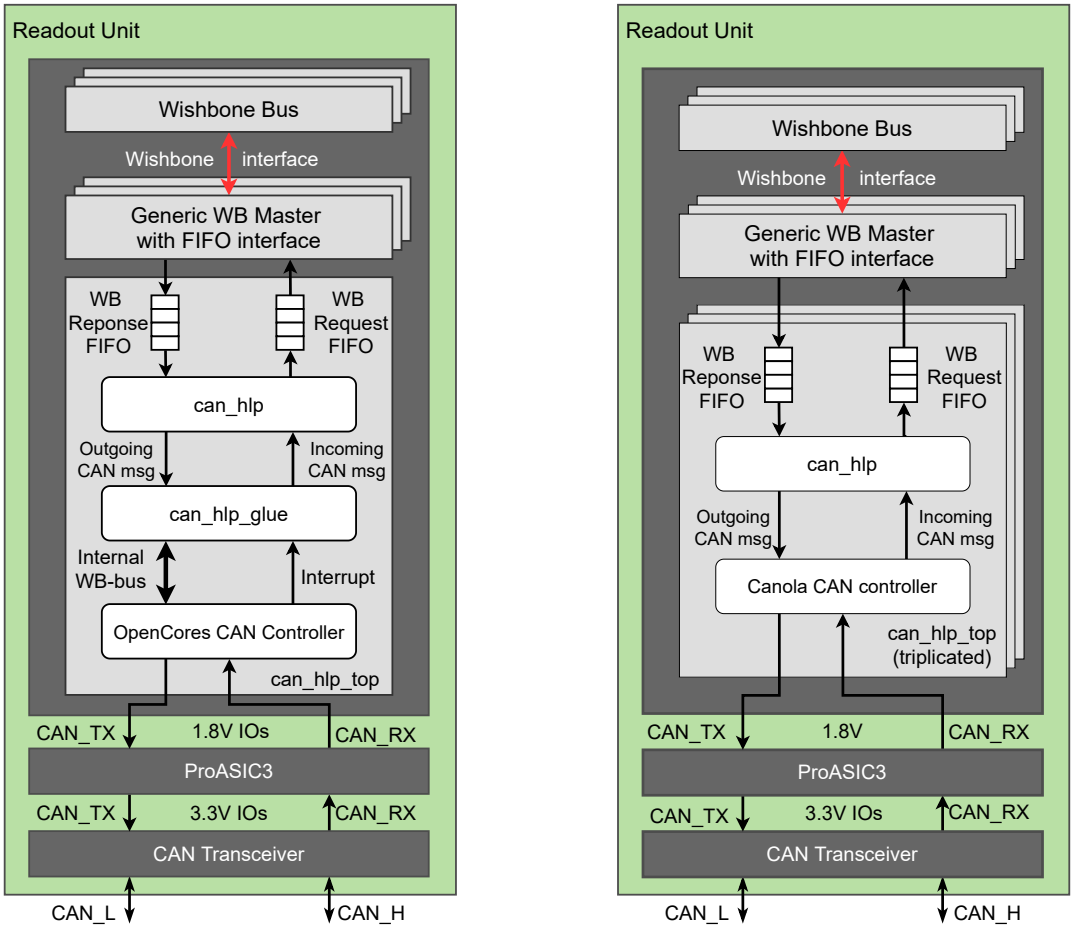
An early implementation of CAN bus and HLP is illustrated in fig. 3.7A. It was based an open source CAN controller available at the *OpenCores* website [56]. Interfacing with this module is performed via a bus interface (WB or Intel 8051) [15], and this required an additional layer of "glue logic" between the HLP layer and the CAN controller in the FPGA design.

The HLP logic and FSM is implemented in the *can_hlp* block in the figure. It used the glue logic in the *can_hlp_glue* block to send and receive CAN frames, and interfaced with the WB master module using the two FIFOs. The glue logic itself communicated with the CAN controller on a local WB bus interface [16]. Initial testing of the CAN interface and HLP for DCS was performed using this version.

Current Version Based on a Custom CAN Controller. The CAN controller that was used in the early version dates back to the year 2002. Although the controller is fully

[15] Documentation for this CAN controller is a little scarse, but the bus interface is compatible with the SJA1000 CAN controllers, so it is essentially pretty well documented by referring to the SJA1000 datasheet.

[16] This WB interface was internal in the CAN HLP module, it is not connected to the main WB bus of the FPGA design.



(A) Early version of CAN HLP, based on OpenCores CAN controller and without TMR.

(B) Current version of CAN HLP, based on custom CAN controller with full TMR.

FIGURE 3.7: Implementation of CAN-based interface to the WB bus for DCS.

functional and has stood the test of time, it is a bit dated from a design perspective: The controller was implemented purely using RTL and structural modeling, and does not employ any of the higher level behavioral constructs that are available in the most recent standards of HDLs. There are no recognizable FSM constructs in the code, and for SEE mitigation using TMR this poses a challenge because the mitigation techniques employed in the main FPGA design can not be applied easily (see section 3.8). The controller could possibly have been mitigated using some of the automatic tools available, but these tools typically employ Local (LTMR) at the register level [57]. This is suitable for flash based FPGAs, but is generally not the best approach for SRAM based FPGAs such as the Xilinx UltraScale [58].

There were not many suitable CAN controllers that could be used as a replacement. One of the few alternatives is *HurriCANe*, a radiation hardened CAN controller IP developed by the European Space Agency (ESA). However, this IP is closed source

and also rather old, the code appears to date back to 1999. And commercial CAN controller IPs are rather expensive, and were avoided for budgetary reasons. Additionally these IPs are typically encrypted and closed source, which poses a challenge when it comes to protecting them from radiation effects.

As a result, it was decided to design a custom CAN controller for the ITS RU. The new controller is designed with TMR in mind, using the same techniques as the remainder of the main FPGA design. The controller has a simple interface to transmit and receive messages, which allowed the glue logic of the previous version of CAN HLP to be eliminated. The other blocks of the HLP design remain relatively unchanged, and the new CAN HLP design is shown in fig. 3.7B. Details about the design of the new CAN controller is located in section 3.9 at the end of the chapter.

3.6 Alpide Control

A simplified block diagram of the Alpide Control module is shown in fig. 3.8. It communicates with the ALPIDE chips via a half-duplex transceiver on the RU board (*SN65MLVD080*, also shown in the figure). The module supports control transactions (see appendix B.4) on five sets of control signals, allowing it to be used with the one DCTRL line for IB staves, or all four DCTRL lines for OB staves[17]. The output stage simply serializes and outputs one byte at a time on the bus. Write transactions and triggers can be broadcasted on all connectors simultaneously. The input stage, which operates independently, performs phase alignment, Manchester decoding and deserializes the data received for read transaction results, for one connector at a time. The *BCRST*, *TRIGGER*, and *PULSE* inputs to the module are used to issue the corresponding hardware opcodes to the sensors. For other transactions, i.e. read and write registers, the WB bus is the primary interface. But the figure also shows an interface that is planned for the *Alpide Monitor* module, to allow the latter to perform read and write transactions directly without going through the WB bus.

3.7 Alpide Monitor

During operation of the experiment the DCS will have to continuously monitor temperatures and voltages on all the 24 120 ALPIDE sensor chips, with a reasonable refresh rate (1 Hz is a typical rate for devices read by the DCS [54]).

[17] The ALPIDE transition board has a total of five DCTRL connectors. One is dedicated for IB staves, the remaining four are dedicated for OB staves.

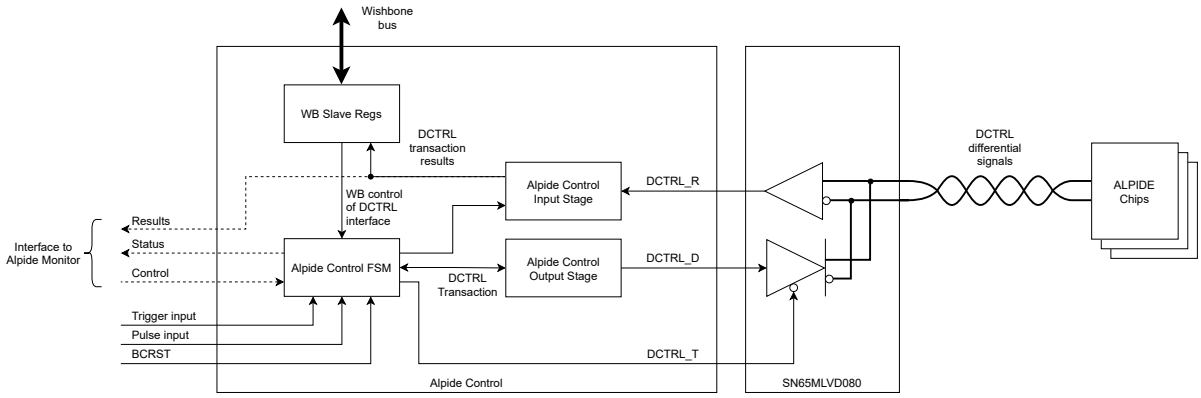


FIGURE 3.8: Alpide Control module.

In principle the monitoring could have been performed with the DCS initiating control transactions via the Alpide Control module's WB interface. However, this would require several WB transactions to set up and initiate a control transaction and an additional WB transaction to read out the result. This also requires polling a status register, or waiting sufficiently long, to ensure that the transaction has been performed before the results are read out. And more critical is the fact that the Alpide Control module is used for triggering as well as control and monitoring, and care must be taken to avoid having triggers blocked by ongoing control transactions for monitoring.

As a consequence of these challenges, it was decided to develop a dedicated module responsible for the monitoring of the ALPIDE sensor chips, which fulfills the following specifications: Autonomous monitoring of a configurable set of status registers, temperatures, and voltages, with a refresh rate of at least 1 Hz (for 10 monitored values). The module should be split in two independent submodules: A *sequencer* with a programmable instruction memory, which initiates the transactions on the ALPIDE control bus using a direct interface to the Alpide Control module. And a *sniffer* that stores the results of read transactions in a result FIFO which can be accessed via the WB bus. The entire module should be protected against radiation induced upsets and effects, and the instruction memory and result FIFOs should both employ ECC to protect against bit errors.

The term *monitoring* may imply that only read operations on the ALPIDE control bus are required. But write operations are also necessary to use the ADC in the ALPIDE. The ADC can be used with several analog inputs such as for temperature or onboard voltages [26]. It requires configuration to start sampling and conversion, as well as switching between analog inputs[18]. And the support for write operations

[18]There is an auto-mode which is supposed to perform automatic measurement of all input sources (placing the result in dedicated result registers for each source), but this mode does not work properly.

may open for other uses in the future, such as configuration of pixel masks for the ALPIDE[19].

The general structure of the Alpide Monitor is shown in fig. 3.9. The sequencer and sniffer components of the monitor each have a dedicated WB slave, and are controlled by two FSMs. The WB bus operates at 160 MHz in the RU, but the Alpide Monitor also communicates with the Alpide Control module which runs at 40 MHz. A choice had to be made as to whether the FSM logic in the Alpide Monitor should operate at 40 MHz or 160 MHz. Allowing all the logic in the Alpide Monitor to run at 160 MHz and crossing clock domains at the interface to the Alpide Control module appeared to be the most clean solution. This is implemented in the CDC block in fig. 3.9, using combinations of *xpm_cdc_single*, *xpm_cdc_single_array*, and *xpm_cdc_pulse*, which are Xilinx's parameterized macros for single and single-bit array synchronizers, and pulse transfer, respectively [52].

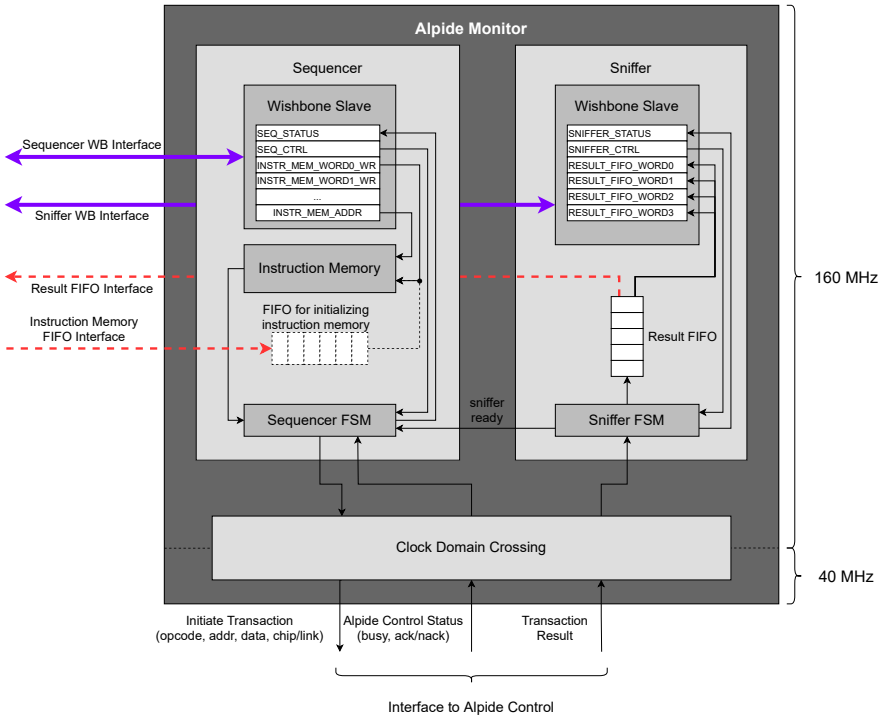


FIGURE 3.9: Alpide Monitor block diagram.

3.7.1 Sequencer

The sequencer has a programmable instruction memory where sequences of read and write operations to be performed on the ALPIDE control bus are configured. An FSM

[19]There may be a number of dead pixels in a chip, which will increase during the chips' lifetime as they are exposed to radiation. Dead pixels which are always on are problematic since they will contribute fake hits to each event, and these will be masked out as they are discovered.

(fig. 3.10) controls the execution of the sequences from the instruction memory, and access to the Alpide Control module. It fetches an instruction word from the memory (with two wait states to account for BRAM read delay) and decodes the opcode to determine which operation should be carried out. After execution, the sequencer proceeds to fetch and execute the next instruction, until an *end* instruction is encountered, which marks the end of the sequence. Depending on whether the sequencer is configured for *single-shot* or *continuous* operation, it will either stop at the end of the sequence, or start over at the beginning of the instruction memory.

The transactions on the ALPIDE control bus are initiated via a direct interface to the Alpide Control module, which primarily includes the follow set of signals: Opcode (either read or write), register address, and data (for write operations). A busy signal indicates if the Alpide Control module is already performing a control transaction, or if there is not sufficient time to perform another control transaction before the next bunch crossing with collisions. The Alpide Control module will always prioritize trigger signals (from the trigger module), in the event that a trigger is requested on the same clock cycle as a normal control transaction. The control module will indicate if a request is executed or not with the acknowledge signals, so the sequencer knows if it should retry the transaction when the control module is not busy again, or if it can move on to the next instruction.

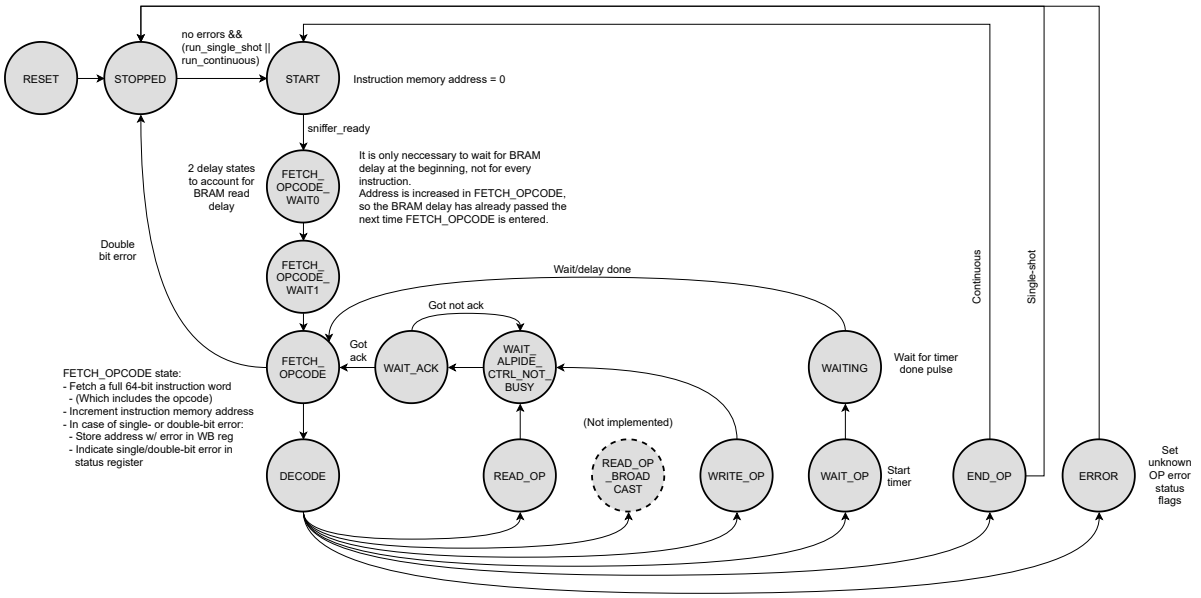


FIGURE 3.10: Alpide Monitor Sequencer FSM Diagram.

Instruction Words and Opcodes

Each instruction word is 64 bits. Four opcodes[20] are implemented in the sequencer: *Read*, *Write*, *Wait*, and *End*. The *Read* and *Write* opcodes (figs. 3.11 and 3.12) initiate a register read or write. Read can only be performed for one ALPIDE chip at a time, but a write to several chips can be performed if a broadcast chip ID is used. The *Wait* opcode (fig. 3.13) delays further execution of the sequencer for the specified delay[21], in units of 6.25 ns, and finally the *end* opcode (fig. 3.14) indicates the end of the sequence.

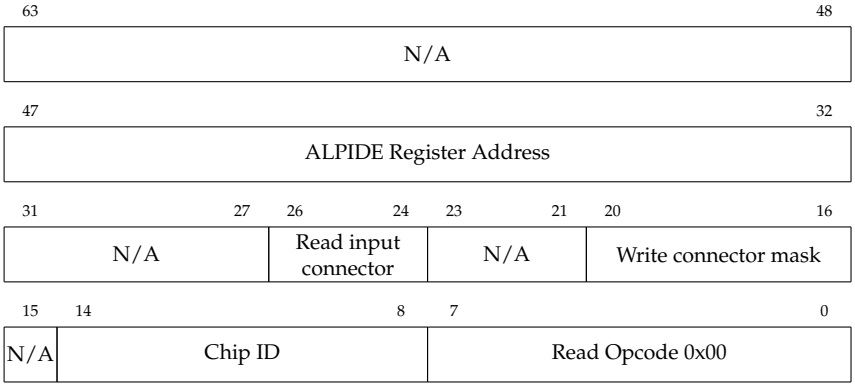


FIGURE 3.11: Sequencer Read Instruction Word.

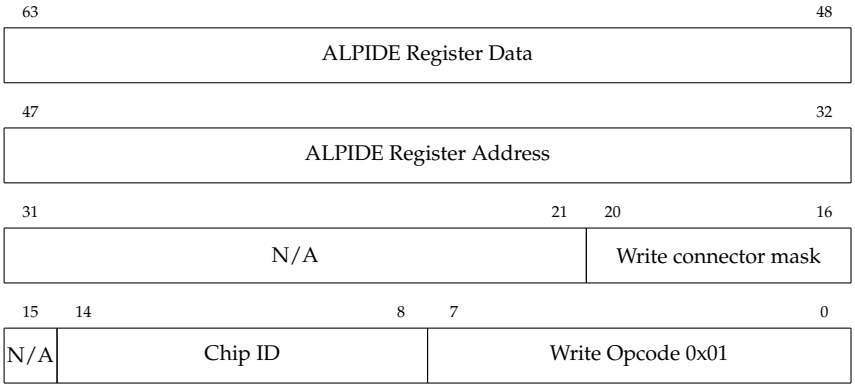


FIGURE 3.12: Sequencer Write Instruction Word.

Instruction Memory

The instruction memory is implemented using a Xilinx dual-port RAM primitive, which allows for built-in ECC to be used. The built-in ECC requires a 64-bit internal data width regardless of the chosen data width for the dual-port RAM. As a consequence, a narrower data width would not allow for efficient use of the BRAM blocks used by the dual-port memory. For instance, if a 32-bit data width was chosen, the

[20] Not to be confused with the opcodes on the ALPIDE control bus.

[21] Its primary use is to implement a necessary delay before reading ADC results after sampling and conversion has been started.

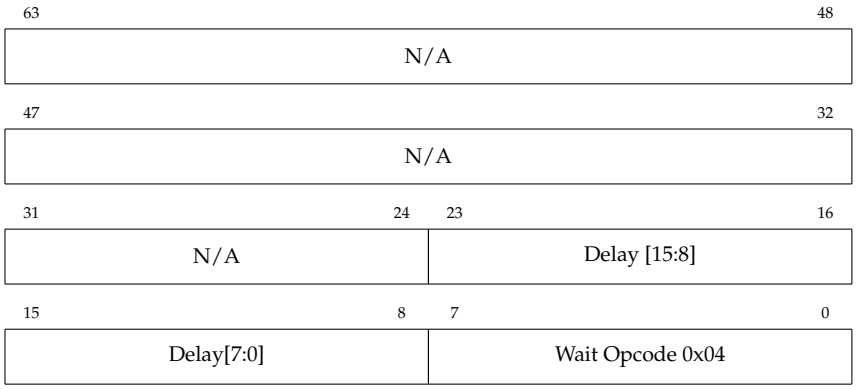


FIGURE 3.13: Sequencer Wait Instruction Word.

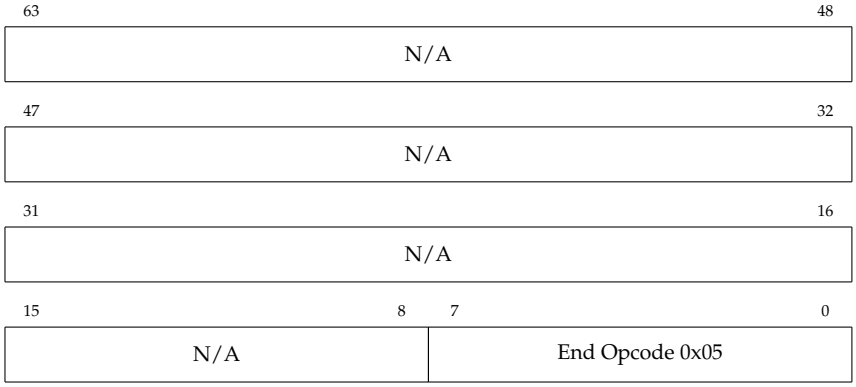


FIGURE 3.14: Sequencer End Instruction Word.

memory would still use 64 bits per address internally to allow for the ECC, but only half of those bits are available to be used. The instruction words have a fixed with of 64 bits for that reason. It made the design simpler, and the BRAM could not have been utilized efficiently with a shorter width unless a custom error correcting scheme was implemented.

Another limitation of this dual-port primitive is that, with ECC enabled, it only allows for read on one port and write on the other. The read port obviously connects to the sequencer state machine so that it can read and execute instructions. The write port connects to the WB bus to allow for configuration of the instruction memory. Therefore, it is not possible to read back the instruction memory from the WB bus. It is likely that the instruction memory will experience bit errors at some point during operation of the experiment, so it is important to have the ability to detect and correct these errors in the instruction memory. This has been solved in the sequencer by monitoring the single-bit and double-bit error outputs while reading the dual-port memory. When a bit error is detected, a "sticky" bit indicating bit errors is set in the sequencer's status register on the WB bus. There is one bit like this for single-bit errors, and one for double-bit errors. In addition, the last address where a bit error was detected is available on a dedicated WB register for the sequencer. This allows

bit errors in the sequencer to be detected by the DCS by monitoring the sequencer's status register, and the errors are quickly corrected by rewriting the data at the address in the instruction memory indicated by the error address register. The sequencer is allowed to continue when a single-bit error is detected, but execution is halted in the case of a double-bit error.

3.7.2 Sniffer

The sniffer receives the results of read operations performed on the ALPIDE control bus. Those operations may have been initiated from the sequencer, or via the Alpide Control module's WB interface. As implied by the submodule's name, it passively waits for incoming results and does not initiate anything itself. This is controlled by a relatively simple FSM (fig. 3.15). The sniffer and sequencer operate more or less independently of each other, with a few exceptions; a signal indicating that the sniffer is running is used by the sequencer, which will wait for the sniffer to run before starting execution from its instruction memory [22]; and the sequencer should output a sequence number to the sniffer, which is used by the sniffer to label entries in the result FIFO[23].

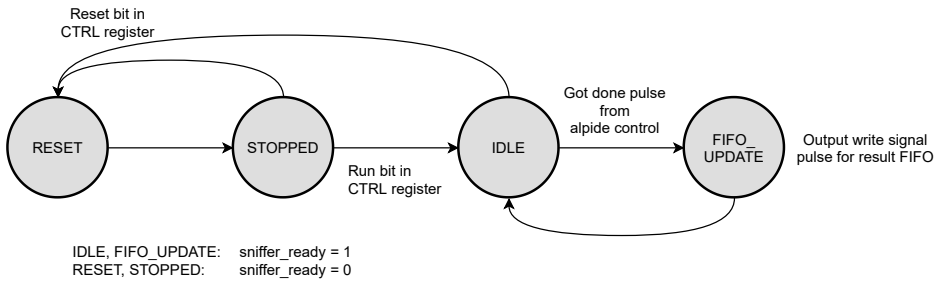


FIGURE 3.15: Alpide Monitor Sniffer FSM Diagram.

Result FIFO and Result Data Words

The result FIFO is implemented using the Xilinx parameterized macro for a synchronous FIFO [52], configured to use BRAM for the internal memory, built-in ECC, and First Word Fall Through (FWFT). The built-in ECC has the same limitations as described for the instruction memory of the sequencer (see the discussion in section 3.7.1), and requires an internal data width of 64 bits. Consequently, the results words are 64 bits wide. The format is shown in fig. 3.16. In principle, 32 bits would

[22] Might be removed, as it may be desirable to run the sequencer without the sniffer, e.g. for configuration of pixel masks.

[23] Planned, but not implemented.

have sufficed for a minimal result entry consisting only of the 16-bit ALPIDE data and 16-bit ALPIDE register address. This leaves an 32 additional bits in a result word, which can be used for status and sequence numbers.

Result words are read out from the FIFO by reading the *RESULT_FIFO_WORDx* registers on the WB bus (see fig. 3.9 and register map in table G.4). Reading the first result register, *RESULT_FIFO_WORD0*, causes the read enable signal of the FIFO to be toggled. A copy of the whole 64-bit result word for that entry is stored temporarily in an internal register, so that the subsequent WB reads of *RESULT_FIFO_WORD1*, *RESULT_FIFO_WORD2*, and *RESULT_FIFO_WORD3* will output data from the same result word[24]. This allows entries to be automatically removed from the FIFO as they are read, without having to use an additional WB transaction. And because the ALPIDE register address and data is located at the lower 32 bits of the result word, it is sufficient to read the *RESULT_FIFO_WORD0* and *RESULT_FIFO_WORD1* registers to obtain the address and data for the ALPIDE register[25]. The entry is still removed from the FIFO, and the additional 32 bits of status data can be ignored.

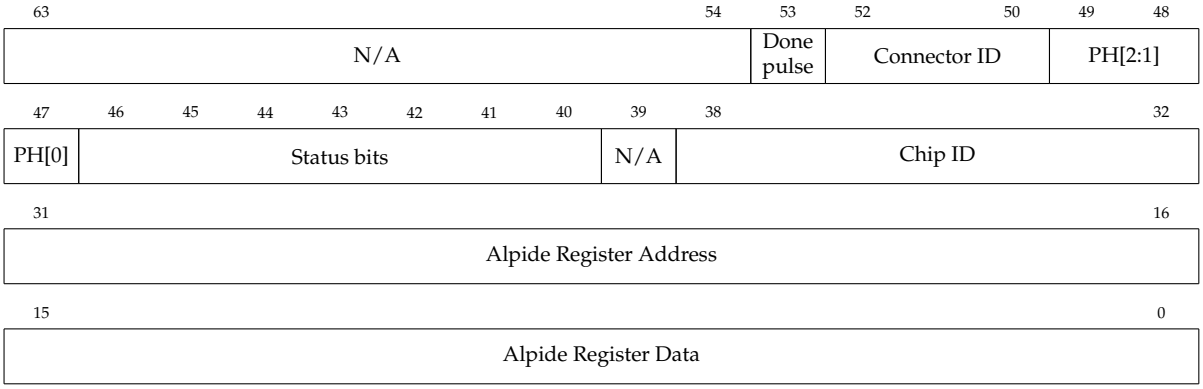


FIGURE 3.16: Sniffer result word.

3.8 Mitigation of Radiation Effects

Triple Modular Redundancy (TMR) is one of the primary mitigation techniques for FPGA designs intended for operation in a radiation environment. But SRAM-based FPGAs, like the Xilinx UltraScale, are not only susceptible to SEUs in the logic elements that constitute the digital design, but also in the configuration memory itself. One important consequence is the need for scrubbing of the configuration memory.

[24]If the FIFO is empty, the *RESULT_FIFO_WORDx* registers are initialized to zero instead, and the FIFO read signal is not toggled.

[25]It is not possible to know the chip ID for the data this way. But if the sequencer is used in single-shot mode the chip ID can be determined based on the order of the sequence and results.

Another is the fact that the majority voters for TMR have an increased radiation cross-section compared to an implementation in a flash-based FPGA, due to the added cross-section of the configuration memory bits [58]. Voting every single register in the design would be very costly in terms of resources, and since the voters themselves have a radiation cross-section it is not certain that such a strategy is the best in terms of protection. Instead, the approach has been to triplicate larger blocks of logic and vote their outputs, also referred to as Block-TMR by some sources [58], but with the critical addition of feedback for correction of important registers and scrubbing.

Memory and registers, such as configuration registers on the WB bus, which retain their value indefinitely should be protected with either ECC or TMR. The same also applies to data streams and FIFOs. Distributed memory (i.e. memory implemented with Look-Up Tables (LUTs)) is not allowed in the main FPGA design in order to simplify the design of the external scrubber[26].

Some of these design guidelines may be relaxed for modules that are not critical during operation of the LHC. However, such modules should not be able interfere with the operation of other modules, so the WB bus for instance has to be protected in any case.

Triplicated Wrappers

A template design for implementation of TMR in the main FPGA design is explained here. Submodules for the FPGA design are developed as one normally would, and the triplication process consists of generating a *TMR wrapper*. TMR can be enabled or disabled by a parameter to the TMR wrapper, and this allows parts or all of the FPGA design to be synthesized and implemented with or without TMR. These wrappers use HDL *generate* statements to either generate an individual (no TMR) instance of a submodule, or generate three instances of the submodule along with the majority voters necessary to vote the outputs from the three instances. Figure 3.17 shows an example TMR wrapper for the FPGA design. It illustrates the different combinations of single and triplicated inputs and outputs one can have on a TMR wrapper. For triplicated inputs, each individual input would be assigned to one individual copy of the logic in the TMR wrapper, whereas a single input to the wrapper would be assigned directly to all copies of the logic. Outputs can be voted with an individual majority voter if an individual output from the wrapper is desired. Or, they can be voted with a triplicated voter, i.e. three voters, which leads to three voted outputs from the wrapper. The single voter approach uses the least amount of resources, but

[26] In principle the *GLUTMASK* bit in a Xilinx device's *Configuration Control Logic* can be used to mask out LUT RAM, allowing LUT RAM to be used with the scrubber. But a side effect of this is that fault injection to LUT RAM would not be possible.

it leads to a single point of failure and is much less reliable than the fully triplicated approach. The general strategy has been to use single voters for less critical signals, and use fully triplicated voters and signals for critical signals, such as the entire WB bus.

Important internal registers of a module, especially the FSM state registers, should also be protected, and this will be discussed in more depth. But counters are a special case. These are generally implemented using dedicated counter modules in the FPGA design which are already fully triplicated. The counter modules are typically instantiated by the wrapper and connected to signals to count up or down from the module to be protected.

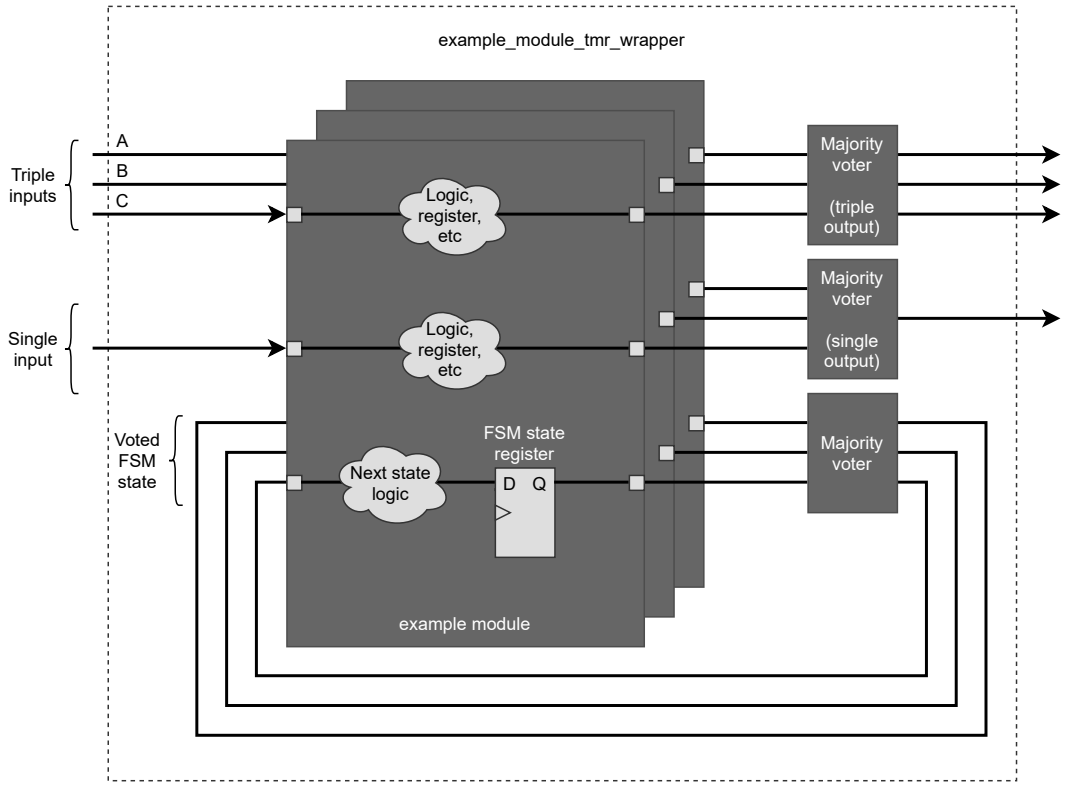


FIGURE 3.17: Example TMR wrapper module for the UltraScale FPGA design.

Generation of the TMR wrappers has been a manual process. However, since the wrappers follow a standard recipe, in principle the generation of the wrappers could probably have been automated.

FSM Protection

Most of the TMR protected submodules with internal FSMs have an output for the current FSM state, and an input for the **voted** FSM state. A majority voter in the submodule's TMR wrapper performs the vote of the FSM state of the three copies of the

submodule, and the voted state is returned on the aforementioned state input. The FSM inside the submodule then uses this voted version of the state register as the current state. This allows the state register to be "repaired" in case it suffered bit-flips due to radiation effects, by using the FSM state of the other two copies of the logic. Lupi et. al. has demonstrated that a 15-fold improvement in Mean Time To Failure (MTTF) may be achieved when the state register is voted using **triplicated voters** [59]. The latter is very important; in the case where a single voter is used, and the cross-section of the voter is equal to the cross-section of the logic it is protecting, then the added cross-section and single-point failure negates the benefit of the triplication. Their results show that there is no improvement in MTTF over the unprotected logic in this case. Consequently, the TMR wrappers in the FPGA design always use triplicated voters for the FSM state registers.

LISTING 3.1: Example of FSM encoding in VHDL for Xilinx FPGA (Note: library use clauses are not shown).

```
1  entity my_module is
2  port (
3    CLK               : in  std_logic;
4    FSM_STATE_O       : out std_logic_vector(C_MY_FSM_STATE_BITSIZE-1 downto 0);
5    FSM_STATE_VOTED_I : in  std_logic_vector(C_MY_FSM_STATE_BITSIZE-1 downto 0));
6  end entity my_module;
7
8  architecture rtl of my_module is
9    signal s_fsm_state_out                : my_fsm_state_t;
10   signal s_fsm_state_voted              : my_fsm_state_t;
11   attribute fsm_encoding                : string;
12   attribute fsm_encoding of s_fsm_state : signal is "sequential";
13 begin
14   -- Convert state register output to std_logic_vector
15   FSM_STATE_O <= std_logic_vector(to_unsigned(my_fsm_state_t'pos(s_fsm_state_out)
         , C_MY_FSM_STATE_BITSIZE));
16   -- Convert voted state register to to my_fsm_state_t
17   s_fsm_state_voted <= my_fsm_state_t'val(to_integer(unsigned(FSM_STATE_VOTED_I))
         );
18
19   p_my_fsm : process(CLK) is begin
20     if rising_edge(CLK) then
21       case s_fsm_state_voted is
22         when STATE1 =>
23           s_fsm_state_out <= STATE2;
24         when STATE2 =>
25           s_fsm_state_out <= STATE1;
26       end case;
27     end if;
28   end process p_my_fsm;
29 end architecture rtl;
```

The FSM register itself is usually encoded using *sequential encoding*, where numbers are assigned to the states in a sequence, i.e. a binary encoding of the state. Representing N states with either sequential (binary) or *gray encoding* requires $ceil(log_2(N))$ bits. Compared to *one-hot encoding* where each state has a dedicated bit, sequential and gray allows the width of the state register to be smaller. This is especially true for FSMs with a large number of states, where the reduced width helps decrease the radiation cross-section of the state register.

LISTING 3.2: Example of type definitions for FSM state register in VHDL for Xilinx FPGA.

```
1  library ieee;
2  use ieee.math_real.all;
3
4  package my_module_pkg is
5  type my_fsm_state_t is (STATE1, STATE2, STATE3);
6  constant C_MY_FSM_STATE_BITSIZE : natural := integer(ceil(log2(1.0+real(
       my_fsm_state_t'pos(my_fsm_state_t'high)))));
7  end package my_module_pkg;
```

Listing 3.2 shows an example of type definitions for the FSM state registers in a VHDL package file, where the states are defined using an enumerated type. These types are used in listing 3.1, which shows an example of an entity with a synchronous FSM. The state register ports are represented as a *std_logic_vector*, which allows them to be voted easily with the previously mentioned majority voter modules. Conversion to and from *std_logic_vector* is shown on lines 15 and 17. The **case** statement of the process uses the voted FSM state register input to determine the current state, as shown on line 21 of listing 3.1. The next state is assigned to the state register output, as seen on lines 23 and 25.

3.9 Radiation Tolerant CAN Controller

To meet the requirements for radiation tolerance for the main FPGA design, a new CAN controller called *Canola* was developed. It was designed with the aim of being a fully featured CAN 2.0B [60] compliant controller with triplicated logic and a simple control interface. The source code for the controller is available in a public GitHub-repository [61].

The implementation of the controller is illustrated in the block diagram in fig. 3.18. The controller is divided into blocks using terminology that is common among CAN controllers, and was designed to be modular and configurable. The blocks drawn

with dashed lines indicate features that are typically part of a CAN controller, such as receive and transmit queues, the Acceptance Filters (ACFs) and a bus interface. These features are optional in the controller (and the blocks in red have not been implemented). In the main FPGA design it is used in a minimal implementation consisting of the blocks drawn with solid lines. In the minimal implementation it has a simple interface to receive and transmit frames, along with some status counters and interrupt signals. A minimal implementation like this may be desirable in some applications, such as for the HLP protocol used in the ITS RU, which the controller was designed for. Since the arbitration ID of received frames is checked by the HLP logic, it was not necessary to include an ACF in the controller. And the WB transactions performed by the HLP logic are already queued in the FIFOs of its WB master, which would make receive and transmit queues in the CAN controller redundant. And finally, the HLP logic was greatly simplified by having a direct interface to the CAN controller, as opposed to communicating with the controller via a bus interface.

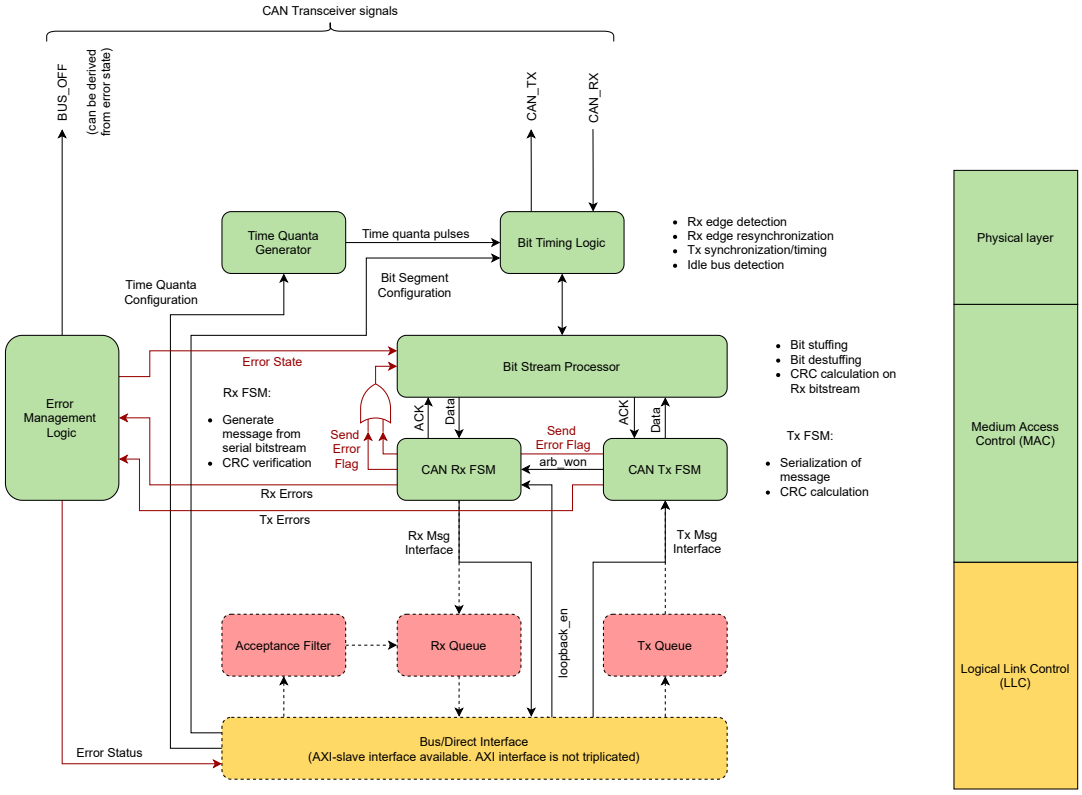


FIGURE 3.18: Block diagram for Canola CAN controller [61].

3.9.1 Bit Timing Logic

The Bit Timing Logic (BTL) is part of the physical layer and is responsible for the timing and sampling of incoming and outgoing bits. As shown in fig. 3.19, the BTL

interfaces directly with an external CAN transceiver[27] via the Tx output and Rx input, and has a simple interface that allows one bit to be sent and received at a time.

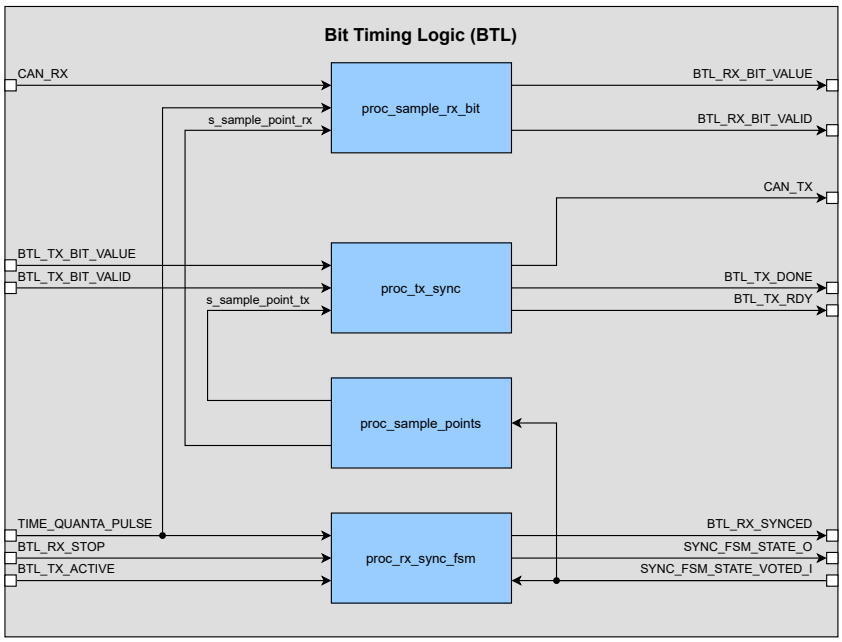


FIGURE 3.19: Block diagram for BTL in Canola CAN controller. The blocks in blue are VHDL processes.

A bit in the CAN protocol consists of four segments: *Sync*, *Prop*, *Phase 1* and *Phase 2* (fig. 3.20). Each segment consists of a number of "time quantas". The sync segment is always one quanta, but the length of the remaining segments is configurable. Received bits are sampled at the beginning of the *Phase 2* segment. Figure 3.19 shows an example of bit timing in CAN, where there is a total of 10 time quantas in a bit.

The logic of the BTL runs at the system clock, but bit timing is synchronized to pulses which the BTL receives from the Time Quanta Generator (TQG), as shown in fig. 3.18. The TQG block generates a pulse per time quanta, and is implemented with a simple counter that essentially divides the system clock by a configurable amount to generate the pulses. The name chosen for the TQG block may be a bit unconventional. CAN controllers typically have a Baud Generator (BG) block for this purpose, but since the TQG block in this controller generates a pulse per time quanta, and not per baud, it was given a different name to reflect that.

Pulses are continuously generated by the BTL for the transmit and receive sample points, even when it is not receiving or transmitting. As per the CAN 2.0B [60] specification, the BTL performs a *hard synchronization* when it receives the Start Of Frame (SOF) of an incoming frame. During a hard synchronization, the BTL is aligned with

[27] A transceiver contains none of the protocol logic and provides only an interface for the differential electrical signals on the CAN bus.

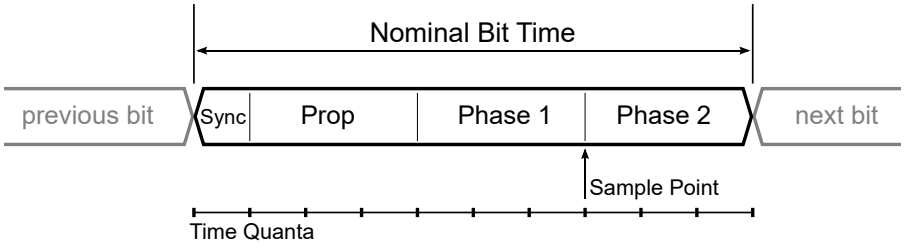


FIGURE 3.20: CAN bus bit timing [62].

the *sync* segment, which puts it in sync with the falling edge of the received bit. On subsequent falling edges in the frame the BTL may perform a *soft synchronization*. A soft sync is limited to jumps by a given number of time quantas specified by the Synchronization Jump Width (SJW), a configurable parameter in CAN controllers. The soft sync kicks in when a falling edge is detected before or after the synchronization segment, and not during the synchronization segment as expected. The phase error, in terms of time quantas, is corrected for by either shortening the first phase segment of the bit, or extending the second phase segment, depending on whether the phase error is negative or positive.

The transmit side of the BTL's interface consists of four signals; *BTL_TX_RDY*, an output to indicate when the BTL is ready to transmit a new bit; *BTL_TX_BIT_VALUE*, an input used to set up the value of the next bit to transmit; *BTL_TX_BIT_VALID*, an input which is pulsed to start transmission of the next bit; and *BTL_TX_BIT_DONE*, which indicates that transmission of the bit is done. Additionally, there is an input to inform the BTL that a transmission is being performed on the bus, *BTL_TX_ACTIVE*, in which case it should not re-synchronize on received bits.

On the receive side of the BTL there is a *BTL_RX_SYNCED* output which indicates when a falling edge on the bus was detected, and that the BTL is synchronized with this edge. The *BTL_RX_BIT_VALID* output is pulsed each time a bit has been received, and the received bit value is output on *BTL_RX_BIT_VALUE* (which is valid when it coincides with *BTL_RX_BIT_VALID*). Finally, the *BTL_RX_STOP* input is used when a message has been received, and indicates to the BTL that it is allowed to perform a hard synchronization the next time a falling edge is detected.

The logic for the synchronization and phase error correction is implemented by the Rx-synchronization FSM. A simplified state diagram for this FSM is shown in fig. 3.21. It consists of four states which corresponds to the segments that a bit is divided into in the CAN specification. As indicated in fig. 3.19, there is a separate process that generates the sample points based on the Rx synchronization FSM state. It outputs a pulse for the Tx sample point when the FSM enters the *ST_SYNC_SEG* state, and a pulse for the Rx sample point in the *ST_PHASE_SEG2* state. The Tx and Rx sample

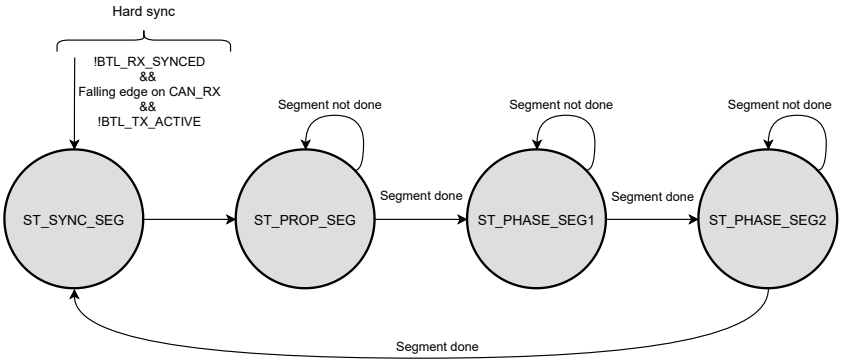


FIGURE 3.21: FSM diagram for Rx-synchronization in the BTL of the Canola CAN controller.

points are used by two dedicated processes, one which outputs the bits to transmit, and the other which samples received data on the CAN bus.

3.9.2 Bit Stream Processor

The Bit Stream Processor (BSP) acts as a layer between the transmit and receive FSMs and the BTL. It interfaces directly to the BTL in order to transmit and receive individual bits. The BSP itself handles transmission and reception of sequences of up to 64 bits[28]. Other responsibilities of the BSP include the CRC calculation of received and transmitted data, bit-stuffing[29] of the CAN message, and transmission and reception of error flags.

Figure 3.22 shows an overview of the BSP. The main functionality of the BSP is to transmit and receive sequences of data, and this is implemented by the two FSMs shown in the figure, and the two corresponding signal interfaces. In addition there are inputs and outputs to signal and detect error conditions, and also detection of mismatch of transmitted versus received bit on the bus. The latter is not necessarily an error. It is also used to detect loss of arbitration, and to detect the acknowledgement bit. And finally there are outputs for the CRC calculations, which are used by the higher-level frame FSMs of the controller, along with the voted CRC inputs which allow for majority voting of the CRC when the TMR-version of the controller is used.

Transmit FSM for BSP. The state diagram for the BSP's Tx FSM is shown in fig. 3.23. From the idle state, the FSM waits for the *BSP_TX_ACTIVE* signal before moving to the *ST_WAIT_TX_DATA* state. While in this state, the vectors *BSP_TX_DATA* and

[28] 64 bits is the maximum length of the payload and is the longest possible field in a CAN message.
[29] Non-data bits that are added as necessary to the data stream to provide enough transitions for synchronization.

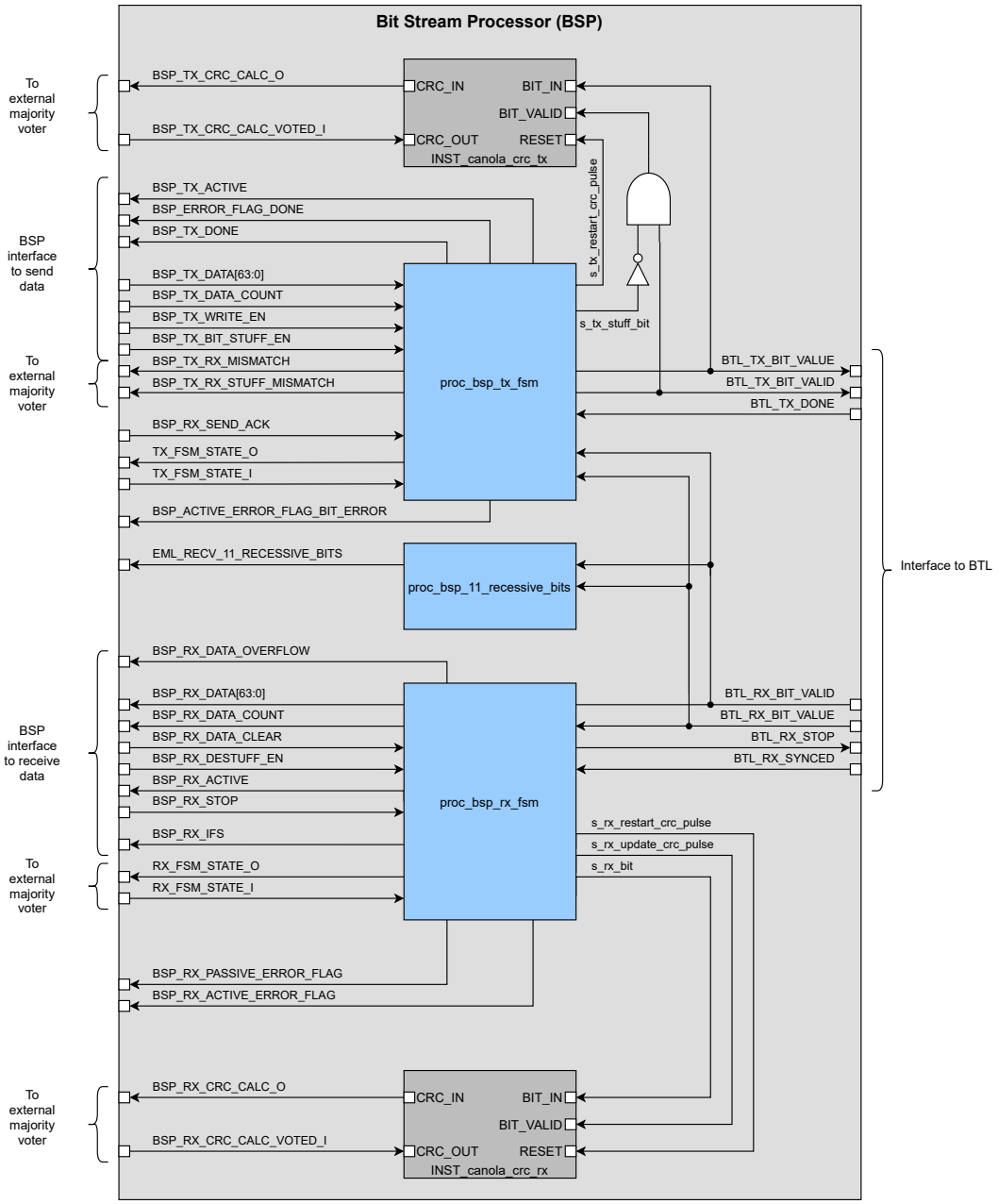


FIGURE 3.22: Block diagram for the BSP in the Canola CAN controller. The blocks in blue are VHDL processes.

BSP_TX_DATA_COUNT should be set up with the data and number of bits to transmit, respectively. The *BSP_TX_WRITE_EN* input should then be pulsed to indicate to the FSM that the data is ready to be transmitted. The FSM will then process one bit at a time, and transmit it using the BTL's Tx interface. When all the data has been transmitted, the FSM returns to the *ST_WAIT_TX_DATA* state, and indicates with the *BSP_TX_DONE* signal that the bits were transmitted.

Sending the acknowledgement bit is also handled by the Tx FSM. This happens

when the *BSP_RX_SEND_ACK*[30] goes high, but since the acknowledgement bit is always a dominant bit it is not necessary to set up the data vectors in this case.

And finally, the *BSP_SEND_ERROR_FLAG* input instructs the FSM to send an error flag immediately, regardless of the current state. The type of error flag (passive or active) sent by the BSP is determined based on the controller's current error state.

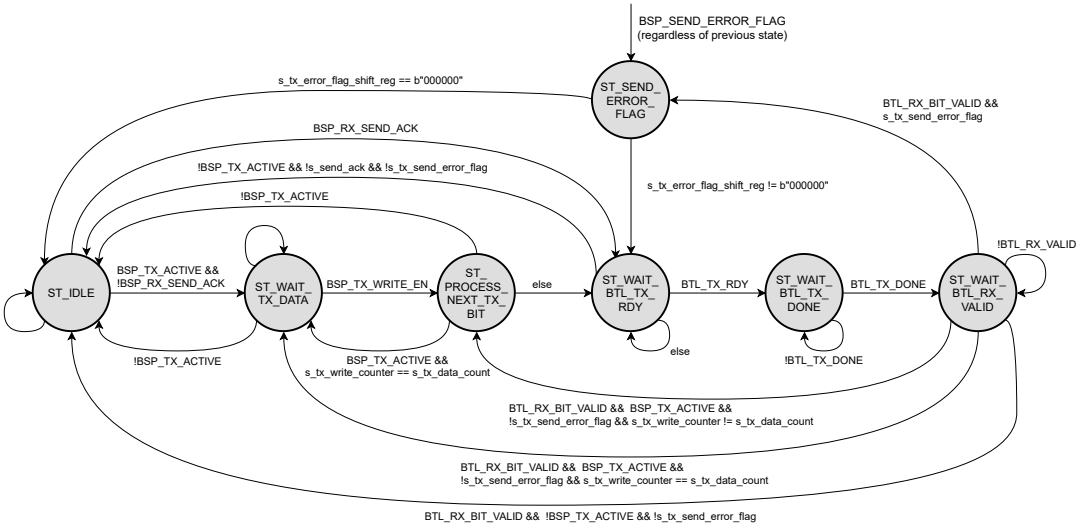


FIGURE 3.23: State diagram for the Tx-FSM of the BSP in the Canola CAN controller.

Receive FSM for BSP. The Rx FSM waits in its idle state for the BTL to synchronize on a falling edge, as shown in fig. 3.24. After synchronization it moves on to a state where it waits for the BTL to sample a bit, indicated by *BTL_RX_BIT_VALID*, and then proceeds to process this bit. Data bits are added to the received data register output *BSP_RX_DATA[63:0]* and *BSP_RX_DATA_COUNT* is increased[31]. If the received data should be de-stuffed[32] then the receive FSM will detect and discard stuff bits, and verify their value. Six consecutive dominant bits is always an error. Six recessive bits is only an error for the parts of a CAN frame where bit-stuffing is enabled. When an error condition is detected, or when *BSP_RX_STOP* goes high to instruct the BSP to stop receiving, the FSM will move on to the states that checks and waits for the bus to be idle. In these states the FSM waits for three recessive bits, which correspond to the Inter-Frame Spacing (IFS), and moves back to the idle state. At this point, the BTL is informed by the *BTL_RX_STOP* signal that the incoming bit stream, which it was synchronized to, has stopped. This prepares the BTL for the next frame by allowing it to do a hard-sync on the next falling edge it detects on the bus.

[30]Since the acknowledgement is sent by a receiving controller, the signal is named _RX_.
[31]*BSP_RX_DATA_COUNT* is reset when the *BSP_RX_DATA_CLEAR* input goes high.
[32]Controlled by the *BSP_BIT_DESTUFF_EN* input. Not all parts of a CAN frame are "stuffed".

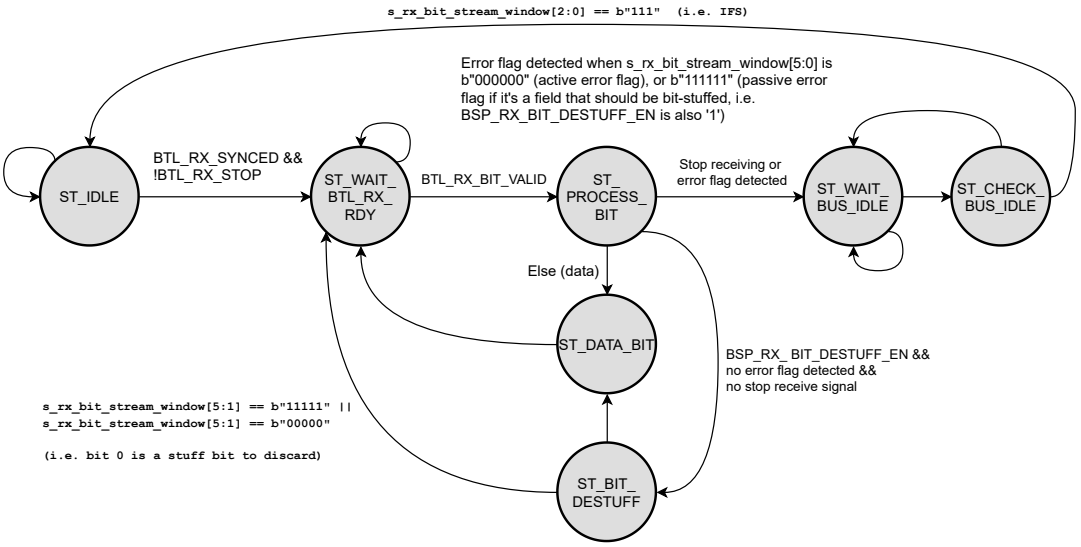


FIGURE 3.24: State diagram for the Rx-FSM of the BSP in the Canola CAN controller.

Simultaneous Transmit and Receive. An important feature of the controller is that the BSP and BTL can receive and transmit bits simultaneously. This is used for read back of the transmitted bits, which is necessary to check for transmit errors, arbitration loss, or to receive the acknowledgement bit when transmitting. It also allows the receive and transmit FSMs for CAN frames (see the next sections) to operate simultaneously. If the transmit frame FSM loses arbitration because a different node was transmitting a higher priority message at the same time, the receive frame FSM is able to continue receiving the frame which originated from the other node.

3.9.3 Transmit FSM for CAN Frames

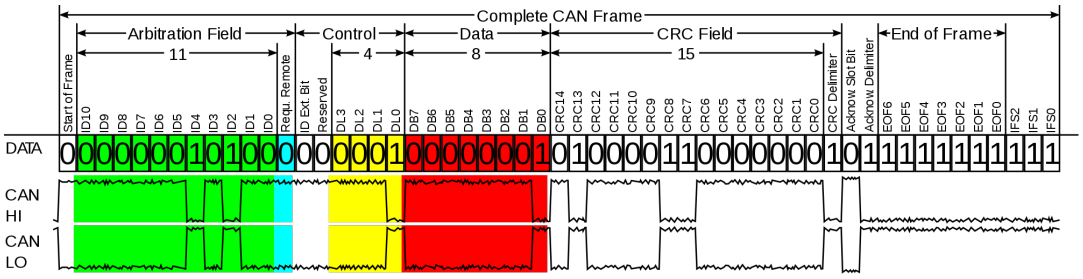


FIGURE 3.25: CAN-frame in base format with electrical levels without stuff-bits [63].

A dedicated FSM is responsible for transmission of CAN frames on the bus, and interfaces directly to the BSP in order to transmit fields of up to several bits (it should not be confused with the transmit FSM of the BSP itself). The state diagram of the

transmit FSM is shown in fig. 3.26. CAN is a complex protocol where the frames consist of a number of different fields of varying widths, which are transmitted in a sequence. An example of a base frame (without the stuff-bits) is shown in fig. 3.25. The base and extended versions of the CAN frames are almost identical, with a few additional fields in the extended frame to support the longer arbitration ID[33]. The FSM has dedicated states for each field in the frame, and supports both base and extended frame formats. Although this implementation leads to a relatively high number of states, 39 in total, the state register is still only $ceil(log_2(39)) = 6$ bits wide, due to the sequential encoding of the state register which was mentioned in section 3.8. And since the flow between the states mostly follow one path, with only a couple of branches where the processing of base and extended frames differ, it should be a comprehensible state diagram that is relatively easy to understand.

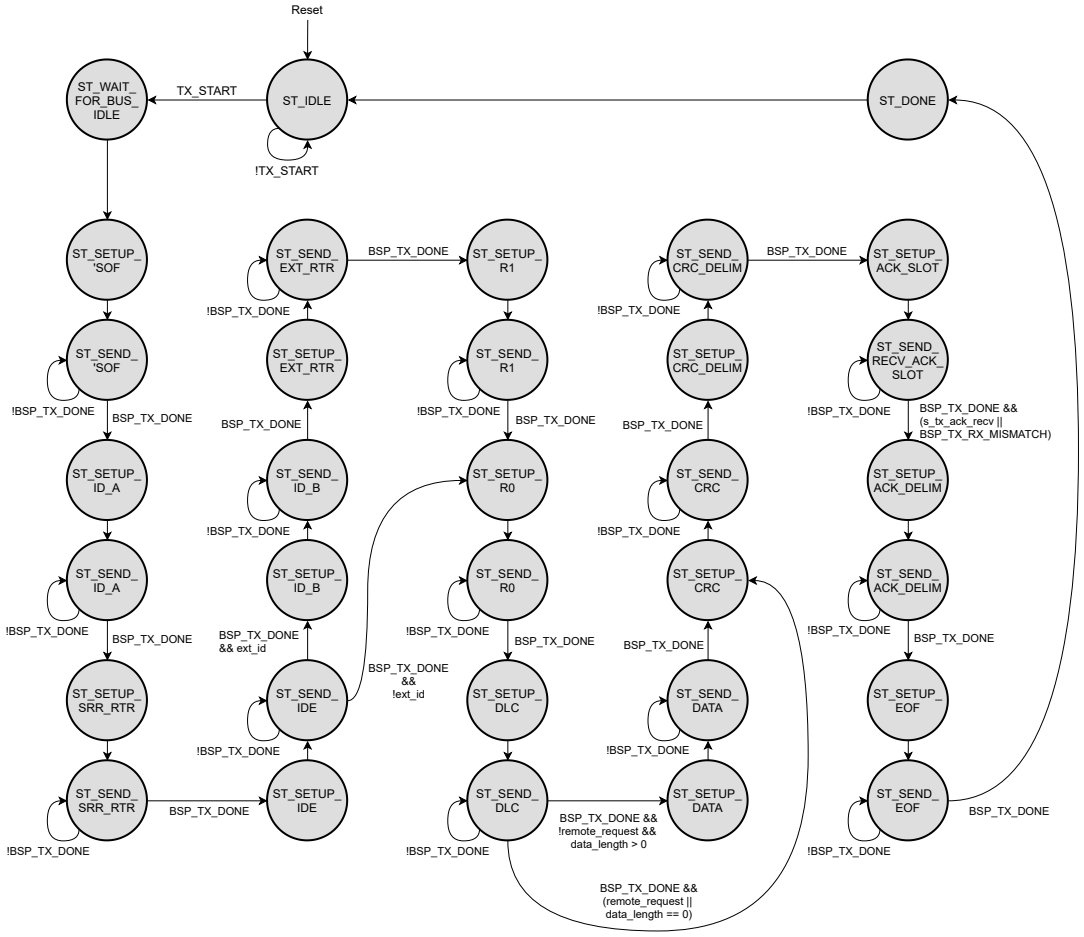


FIGURE 3.26: Simplified state diagram for the Transmit Frame FSM in the Canola CAN controller. Error states and handling not shown.

For each field in a CAN frame, there is a *SETUP* state where the FSM sets up the data for the field in question, before it proceeds to the *SEND* state where it waits

[33]Base frames have 11-bit ID. Extended frames have 29-bit ID.

for the BSP to transmit the data. The transmit FSM indicates to the BSP when the outgoing data should be bit-stuffed, using the *BSP_TX_BIT_STUFF_EN* signal, and the necessary stuff bits are then automatically inserted in the outgoing data stream by the BSP.

During the *SEND* states the FSM will monitor for mismatches between transmitted and received bits, as reported by the BSP. Interpretation of the mismatch signal depends on the field that is being transmitted. For the arbitration fields a mismatch can indicate loss of arbitration (when a recessive bit is being transmitted). For most other fields, and for dominant bits of the arbitration field, a mismatch is an error. And finally, for the acknowledgement slot (in the *ST_SEND_RECV_ACK_SLOT* state), the mismatch signal is use to detect acknowledgements from a listening node[34].

The signals to control the transmit FSM is available at the top level of the CAN controller. From the user's perspective, the data to send and arbitration ID is set up on a *TX_MSG_IN* input, and a *TX_START* input should be pulsed to start transmission. The FSM will indicate if it is already busy with a *TX_BUSY* output. When a frame has been successfully transmitted, a *TX_DONE* output is pulsed. If transmission failed, either because of errors or loss of arbitration, the FSM has a *ST_RETRANSMIT* state which attempts to re-transmit it[35]. If re-transmission fails, then this is indicated on a *TX_FAILED* output and the FSM returns to the *ST_IDLE* state.

3.9.4 Receive FSM for CAN Frames

The implementation of the receive FSM follows the same approach as the transmit FSM. There is one dedicated state for each field in the CAN frames, and both base and extended format frames are supported.

The receive FSM operates in parallel to the transmit FSM. If the transmit FSM should lose arbitration it will notify the receive FSM. In this case the receive FSM will continue to receive the frame that won arbitration, which is being transmitted by a different node on the bus, and it is allowed to acknowledge this frame. But if the transmit FSM should win arbitration, then the receive FSM silently tracks the frame but will not report errors or acknowledge the frame, as it originated from the same node. The frame is not presented as a received frame on the CAN controller's interface in this case. In principle, an internal loopback mechanism could easily be implemented by allowing the receive FSM to acknowledge frames that originate from the transmit FSM.

[34] A sender transmits a recessive bit for the acknowledgement slot, and receivers transmit a dominant bit to indicate successful reception.

[35] Re-transmission is configurable. The controller can retry forever, a specified number of times, or it can be disabled altogether.

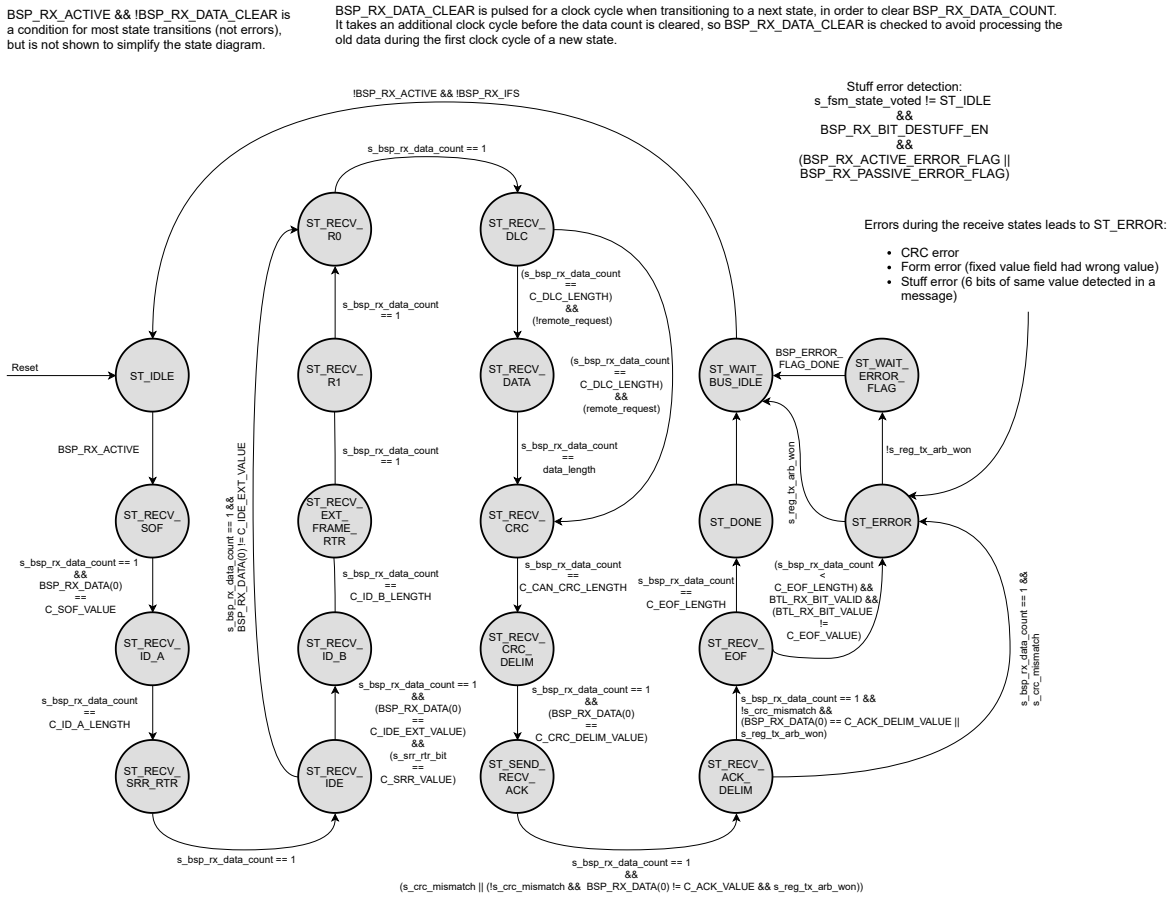


FIGURE 3.27: Simplified state diagram for the Receive Frame FSM in the Canola CAN controller.

The state diagram for the receive FSM is shown in fig. 3.27. For each of the states that correspond to a field in a CAN frame, the receive FSM waits for the BSP to receive the number of bits expected for that field. Fields that should have a fixed value are verified, and so is the CRC field. De-stuffing of bits is handled by the BSP, but when it should be performed is controlled with the *BSP_RX_BIT_DESTUFF_EN* signal by the receive FSM.

At the end of a reception, the FSM will wait for the IFS period before returning to the idle state where it is allowed to receive the next frame. And as for the transmit FSM, a set of signals from the receive FSM is available at the top-level of the CAN controller, so that the received frame can be retrieved.

3.9.5 Error Management Logic

An extensive set of rules for the error management logic in a CAN controller is described in the CAN specification [60], and this is implemented by a dedicated Error Management Logic (EML) block in the Canola CAN controller.

The EML has dedicated inputs for the different types of errors (examples include stuff errors, form errors, and bit errors) that can be detected by the different sub-modules of the design, as well as inputs for successful transmission and reception of CAN frames. These inputs should be pulsed for one cycle, and the EML takes care of increasing or decreasing the Transmit Error Count (TEC) and Receive Error Count (REC) by the right value. The TEC and REC themselves are implemented using external counter modules (with built-in TMR) that can be increased or decreased by an arbitrary value, but they are controlled by the EML.

And finally it is the EML that determines the *error state* for the entire controller, based on the values of the TEC and REC. The error state determines the controller's ability to participate on the bus, to prevent a faulty node from interfering with the communication between good nodes [60].

3.9.6 Radiation Tolerance

The Canola CAN controller achieves radiation tolerance by employing the same techniques as other parts of the FPGA design, but it should be noted that a dedicated set of majority voters and counter modules were implemented for the controller[36].

There are dedicated TMR wrappers for all of the main modules of the Canola controller. The wrappers generally have single inputs and outputs, and the outputs from individual copies inside a wrapper are generally voted with a single–output majority voter (this does not provide the same level of protection as triple voters, but is a compromise made for less critical signals to limit the resource utilization of the controller). The state registers are voted using triple–output majority voters, and uses the "feedback" described in section 3.8 to correct errors. This additional protection is also used for the calculated CRC value and the TQG counter value, since errors here have the potential of accumulating and propagating to the next bit, as well as for the counter module that is used for the TEC and REC, which has this functionality built–in.

ECC has not been employed in the Canola design. Aside from all the registers in the design, which are generally used for short–term storage, there are no memory blocks or buffers. And the CAN protocol itself includes CRC for error detection.

[36]The voters and counters in the main FPGA design rely on mixed SystemVerilog and VHDL support. Since the Canola controller was developed as a standalone project and made available online, it comes with its own set of VHDL-based voters which removes the mixed-language requirement. But a configuration option allows the voters of the main FPGA design to be used.

3.9.7 Resource Utilization

Resource utilization of the Canola CAN controller is shown in table 3.3. The numbers are from a test design for a Xilinx Zynq FPGA featuring four instances of the controller (see section 5.4.1). Instance 0 in the table uses a top-level TMR-wrapper of the controller, with TMR enabled, and requires around five times as many LUTs and three times more registers than the other instances which are not triplicated. Only the utilization of the controllers are shown (the design also included an AXI interface and external counters, but resource count for these were omitted). The design does not use distributed memory, and all LUTs in the table are used for logic. Instance 1 also uses the TMR-wrapper top-level entity, but with the TMR disabled. Its resource utilization is about the same as for instance 2 and 3, which uses a standard top-level entity for the controller with no TMR. This shows that the top-level TMR-wrapper successfully creates singular logic when TMR is not enabled, and does not use more resources in this case.

TABLE 3.3: Canola CAN controller resource utilization in a Zynq test design. Instance 0 is using TMR. The remaining instances have no TMR. Values in percent are relative to the total amount of resources in the *xc7z010clg400-1* FPGA on the Digilent ZYBO Zynq board.

Instance	TMR	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Slice
0	Yes[a]	2701 (15.35%)	1733 (4.92%)	34 (0.39%)	12 (0.25%)	881 (20.02%)
1	No[b]	545 (3.10%)	582 (1.65%)	3 (0.03%)	1 (0.02%)	245 (5.57%)
2	No[c]	540 (3.07%)	582 (1.65%)	5 (0.06%)	0 (0.00%)	268 (6.09%)
3	No[c]	535 (3.04%)	582 (1.65%)	10 (0.11%)	0 (0.00%)	244 (5.55%)

[a] Top-level TMR wrapper with TMR enabled.

[b] Top-level TMR wrapper with TMR disabled.

[c] Standard top-level instance with no option for TMR.

Chapter 4

Auxiliary FPGA Design for the ITS Readout Unit

In this chapter we will look at the FPGA design for the Microsemi PA3 FPGA in the RU, also referred to as the auxiliary FPGA. This design was developed almost entirely at UiB under the lead of Associate Professor Johan Alme, who created the general structure of the design, including the WB bus and register interface, and also the ECC module which decodes ECC encoded data from the external flash memory. The interface to the external flash memory was designed by Attiq Ur Rehman. Magnus Ersland and Gitle Mikkelsen designed and tested the initial versions of the configuration controller which configures the main FPGA with an image from the external flash memory, an important part of the blind scrubber implementation. A fault injector which emulates SEUs in the main FPGA by flipping bits on the data stream for the scrubbing was written by Magnus Rentsch Ersdal. The I^2C slave that communicates with the GBT-SCA was Arild Velure's design, based on an I^2C slave he wrote for the Serialized Analogue-digital Multi Purpose ASIC (SAMPA) chip for the TPC upgrade. My own contributions consists of a UART which acts as a master on the WB bus, the necessary protocol and software to go with it, as well as debugging and testing of the entire design. The UART and software was initially used for testing and configuration of the auxiliary FPGA, and also to transfer configuration images for the main FPGA to the external flash.

The primary purpose of the RU's auxiliary FPGA is configuration and scrubbing of the SRAM-based main FPGA, as discussed in section 2.7.2.

In principle, there are ways to implement configuration and scrubbing that would not require a second FPGA. It is fully possible for the SRAM-based FPGA to configure itself directly from an external flash memory[1]. This could have been paired with Xilinx's proprietary scrubbing solution; the Soft Error Mitigation IP (SEMIP)

[1]Some vendors even offer special configuration Integrated Circuits (ICs) with internal flash memory for this purpose.

core. The SEMIP is essentially implemented with a small microcontroller in the programmable logic of the FPGA, which detects and corrects SEUs in the FPGA. However, the SEMIP core itself is vulnerable to SEUs, and consequently Xilinx's SEMIP solution was deemed not sufficient for the ITS RU.

This is one reason why a second FPGA was used for the scrubber, but there are some other benefits. The auxiliary FPGA offers a lot of flexibility when it comes to configuration of the main FPGA as well as programming of the external flash memory (see section 2.7.2). It also acts as a sort of fail-safe FPGA on the RU, controlling critical systems on the board such as clock switches (see section 2.7.2).

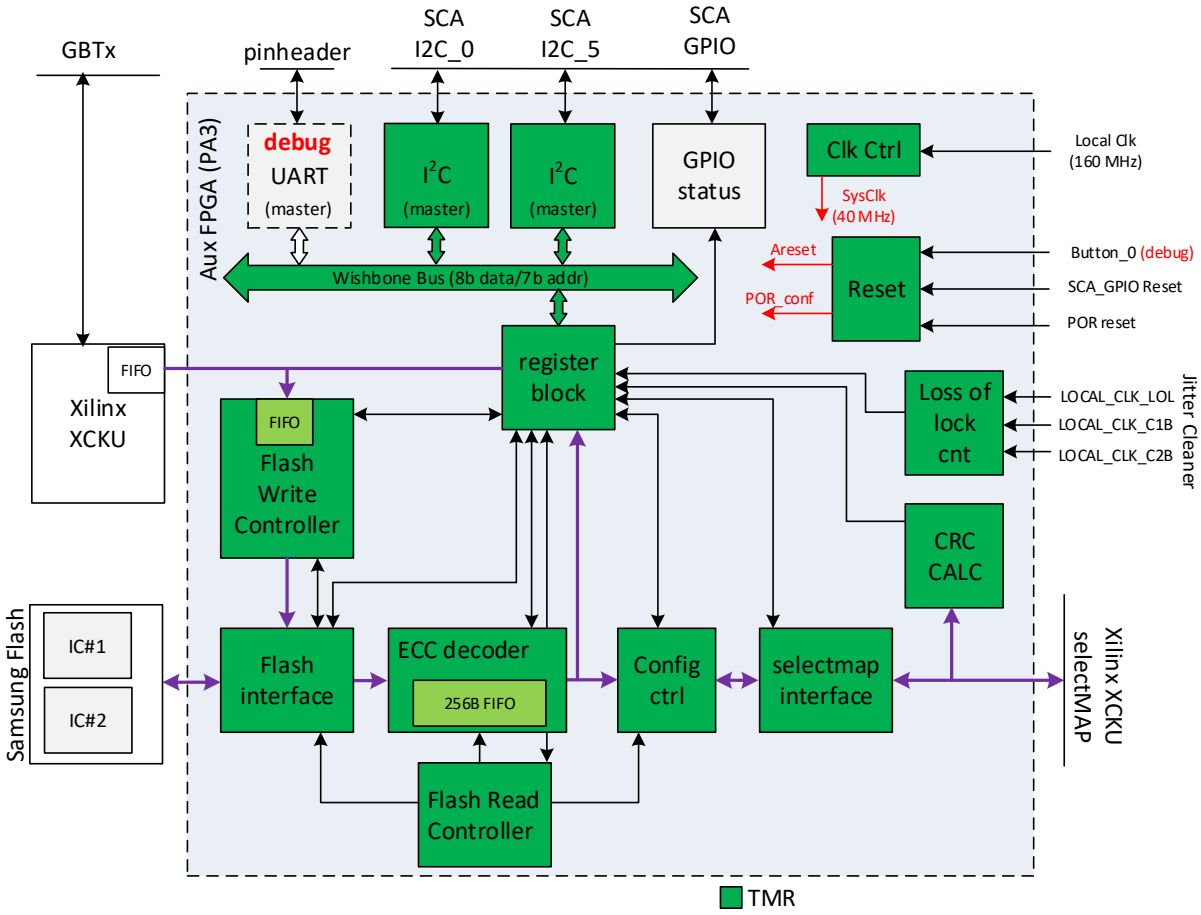


FIGURE 4.1: Block diagram for the Auxiliary FPGA design [64].

4.1 General Structure of Design

The design for the auxiliary FPGA is illustrated in fig. 4.1. The design has a WB bus with 8-bit data width, and 7-bit address width. There is only one slave connected to the bus, and that is the register block. This block contains configuration and status registers for all the other blocks of the design. Access to the WB bus is primarily

TABLE 4.1: Typical I²C transaction with 7-bit address.

Bit number	0	1:7	8	9	10:17	18:25	...		
Field type	Start	Addr[6:0]	R/W	ACK bit	Data byte 0	Data byte 1	...	Data byte N	Stop

performed via one of the two I²C interfaces[2] that connect to the GBT-SCA. A UART with a protocol for WB bus access was also used during development.

Configuration data for the main FPGA is stored in the external *Samsung K9WBG08U1M* flash chip. Configuration images can be downloaded to the external flash via three interfaces; the I²C interface from the GBT-SCA; the UART interface; or via a FIFO interface to the Xilinx FPGA, which allows high-speed transfer via the main GBT-links[3]. The datapath for the configuration data begins with the flash memory at the bottom left of the figure, and goes through the bottom row of modules before reaching the main FPGA.

4.2 Communication Interfaces

4.2.1 I²C Interface

The main control interface to the Auxiliary FPGA is the I²C interface to the GBT-SCA. The GBT-SCA is an I²C bus master, and I²C transactions are initiated via GBT from the FLP by accessing certain registers in the GBT-SCA. The I²C module in the Auxiliary FPGA is itself an I²C-slave, however it is a master on the WB bus.

A typical I²C transaction is shown in table 4.1 for 7-bit device addressing. The implementation of the I²C-slave in the Auxiliary FPGA breaks a bit with the official I²C standard. Normally the device address (or chip address) is sent as the first byte (the *Addr[6:0]* field), and the register address is sent as the second byte (the *Data byte 0* field). But since the auxiliary FPGA is the only I²C slave on the bus, it is redundant to specify a device address. Therefore, the device address field (*Addr[6:0]*) was used to indicate register addresses on the WB bus directly, saving one byte per transaction. The 7-bit width of the WB bus was chosen precisely to match the standard 7-bit device address width of the I²C protocol.

4.2.2 UART Interface

The auxiliary FPGA design features a UART with a protocol that enables access to the internal WB bus of the design. The associated software allows for full control of

[2]The FPGA is an I²C slave. The two I²C buses are separate with only one I²C slave per bus.

[3]This option requires that the Xilinx FPGA has already been configured, and can only be used for updates.

the auxiliary FPGA, upload of Xilinx configuration images to the external flash, and a number of other functions for test and debug. This functionality was not in place for the main GBT and GBT-SCA interfaces during the early stages of the FPGA design. The UART interface played a very important role back then, in particular by offering the only way, at that time, to transfer configuration images to the external flash.

The UART and protocol logic for WB bus access was based on the *UART to Bus* IP available on *OpenCores* [65]. The UART and baud generator were used as they were, but the protocol was modified and the protocol parser rewritten for the auxiliary FPGA design. The modified protocol is described in appendix E. It allows for transfers of up to 65 535 bytes per message with little overhead[4].

The protocol did not include a checksum or CRC code in the message format. But upload of configuration images to the external flash, via the UART, would be followed by read-back and verification. And the configuration data itself includes a checksum and would be rejected by the main FPGA if it was corrupted. With that in mind, the lack of a checksum in the UART protocol was considered an acceptable limitation, since the interface was primarily intended for testing and debugging, and not for use in the experiment.

The baud rate used in the design is 921 600 baud. Each transmitted byte consists of 10 bits: the start bit, stop bit, and eight data bits. This amounts to a transfer rate of $92\,160\,\mathrm{B\,s^{-1}}$. A data length of 4 096 bytes is typically used when transferring configuration images, which allows a full page of the external flash to be transferred per message. The data rate is close to the theoretical maximum of $92\,\mathrm{kB\,s^{-1}}$, since the protocol overhead is practically negligible when transmitting messages that long. The bitstream files for the main FPGA are 24 125 021 bytes, and this is transferred in around five minutes.

UART Software for Auxiliary FPGA

A Graphical User Interface (GUI) and command-line tool was developed for testing and debugging of the design for the auxiliary FPGA. A screenshot of the GUI is shown in fig. 4.2. The software allowed access to any register in the design, as well as transfer of configuration data for the main FPGA, and a number of test features for the *Flash Interface*. The latter included writing known patterns to the external flash, which was used during beam testing to estimate the cross-section of bit-flips from 0 to 1, and from 1 to 0, in the external flash memory.

The software is described in more detail in appendix E.

[4] 4 096 bytes, the size of a page in the flash memory, is generally used for transfer of configuration images.

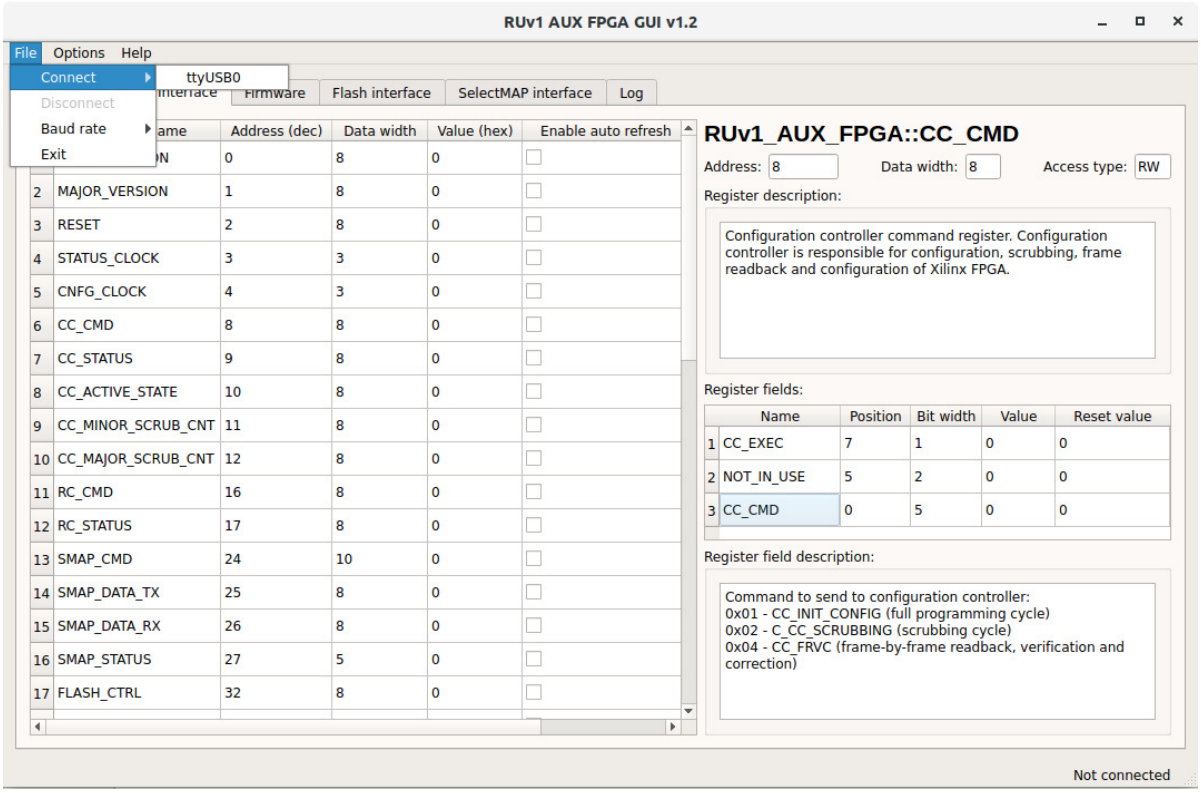


FIGURE 4.2: UART software for testing of Auxiliary FPGA.

4.3 Blind Scrubber Solution

Configuration and scrubbing of the main FPGA is implemented by the chain of modules at the bottom of fig. 4.1. Configuration data stored in the external flash is read by the *Flash Interface*, and is eventually transported to the main FPGA via the *SelectMAP Interface*. The whole process is controlled by the *Configuration Controller*. These three modules form the backbone of the blind scrubber design, and will be discussed in detail later in this section.

4.3.1 Configuration of Xilinx UltraScale FPGAs

There are several options to configure a Xilinx FPGA; the configuration can be initiated and controlled by the Xilinx FPGA itself, in so-called master mode; or an external configuration controller implemented in a microcontroller or FPGA can configure the Xilinx FPGA (slave mode). Master mode is suitable when the FPGA is configured one time on power-on, and the FPGA can read the configuration data directly from external flash memory. However, in cases where a finer degree of control of the configuration process is required, such as partial reconfiguration or external scrubbing, the slave mode is more suitable. Several interfaces are available for configuration,

such as JTAG, Serial Peripheral Interface (SPI) (standard, x4, or x8), serial configuration mode, and SelectMAP. The latter is used between the auxiliary and main FPGAs of the RU board. It is a proprietary configuration interface that Xilinx offers on their FPGAs, such as the UltraScale [66]. SelectMAP uses a parallel data bus, typically eight bits wide[5], and allows for faster configuration of the FPGA than most of the other interfaces.

The actual configuration logic in the Xilinx FPGA is more or less agnostic to the configuration interface choice. The underlying data format is the same in any case, and consists of two package types:

- Type 1: Used to read from or write to a register in the configuration logic
- Type 2: Used to write long blocks of data[6].

There are 20 registers in the configuration logic that are available via the Type 1 packages. Reading and writing of frames, and control of the frame address, is possible via these registers, as well as CRC verification of frames that have been written. The actual configuration frames consist of 123 x 32-bit words (3,936) bits for UltraScale FPGAs and are sent using the type 2 packages.

A bitstream file generated by the Vivado software suite to configure a Xilinx FPGA consists of sequences of register accesses and frames being written to the configuration logic, using the type 1 and type 2 packets. Several options are available when generating the bitstream file, such as the target configuration interface, bit order in a byte, and the so-called "persist" option. The latter is of particular importance for the external scrubber as it allows the SelectMAP to remain active after configuration.

4.3.2 SelectMAP Interface

The Readout Unit uses 8-bit *Slave SelectMAP* for configuration of the main FPGA[7]. The interface for configuration between the main and auxiliary FPGAs consists of the signals shown in table 4.2. The auxiliary FPGA design has a SelectMAP master controller (referred to as the *SelectMAP Interface* in the design, see fig. 4.1), which interfaces directly with the main FPGA via this interface. It is based on an implementation that was used for the FPGA design for the TPC RCU [34]. The controller provides a convenient interface to the rest of the auxiliary FPGA design, which allows bytes of data to be written to or read from the main FPGA, and also implements commands for init,

[5]Larger widths are possible on some FPGAs.

[6]Type 2 packages can not be sent arbitrarily; a type 1 package must first be sent that prepares the logic for the type 2 package.

[7]The configuration interface is selected by the M[2:0] pins on the UltraScale FPGA, and these are hardwired for Slave SelectMAP in the Readout Unit.

startup, abort[8]. The SelectMAP controller is typically controlled by the Configuration Controller, as shown in fig. 4.1. However, it can also be controlled via wishbone registers.

TABLE 4.2: Configuration interface to Xilinx UltraScale FPGA.

Signal	Direction (UltraScale side)	Function
D[0:7]	Bidirectional	Parallel data
CCLK	Input	Configuration clock
DONE	Output (open-drain)	Configuration done
PROGRAM_B	Bidirectional (open-drain)	Reset configuration logic and initiate new configuration sequence
INIT_B	Input	Primarily indicates when the FPGA is resetting the configuration logic or memory, or errors during configuration.
RDWR_B	Input	Controls the direction of the data lines for the SelectMAP interface (write or read to UltraScale FPGA)

4.3.3 External Flash

The *K9WBG08U1M* flash memory chip has an 8-bit parallel IO interface for control. This 8-bit interface is used to setup commands such as read or write, and is used to transfer address and data.

External Flash Structure

Internally, the K9WBG08U1M actually consists of two K9K8G08U0A flash chips[9]. The memory of a K9K8G08U0A chip is organized into 8 192 blocks, and each block consists of 64 pages. A page consists of 4 096+128 bytes, where the additional 128 bytes are intended for ECC data. In total, this adds up to 2 gigabytes per K9K8G08U0A chip (excluding the additional 128 bytes per page), or 4 gigabytes in total for the entire K9WBG08U1M chip.

Figure 4.4 shows how the data is organized in the flash with a 32-bit addressing scheme. The first 13 bits of the address, A_0 to A_{12}, selects the column, and the remaining bits A_{13} to A_{31} is the row address. The column address is equivalent to the byte number in a page, and the row address is essentially the absolute page number. Since there are 64 pages in a block, the first six bits of the row address, A_{13} to A_{18}, specifies the relative page number inside a block. And the remaining bits of the row address, A_{19} to A_{31}, is essentially the block number.

[8]These commands require the *PROGRAM_B*, *INIT_B*, or *RDWR_B* to be toggled for a certain number of clock cycles, and this is handled by the SelectMAP controller in the Auxiliary FPGA design.

[9]The *CE1* and *CE2* chip enable pins are used to select between the two K9K8G08U0A chips.

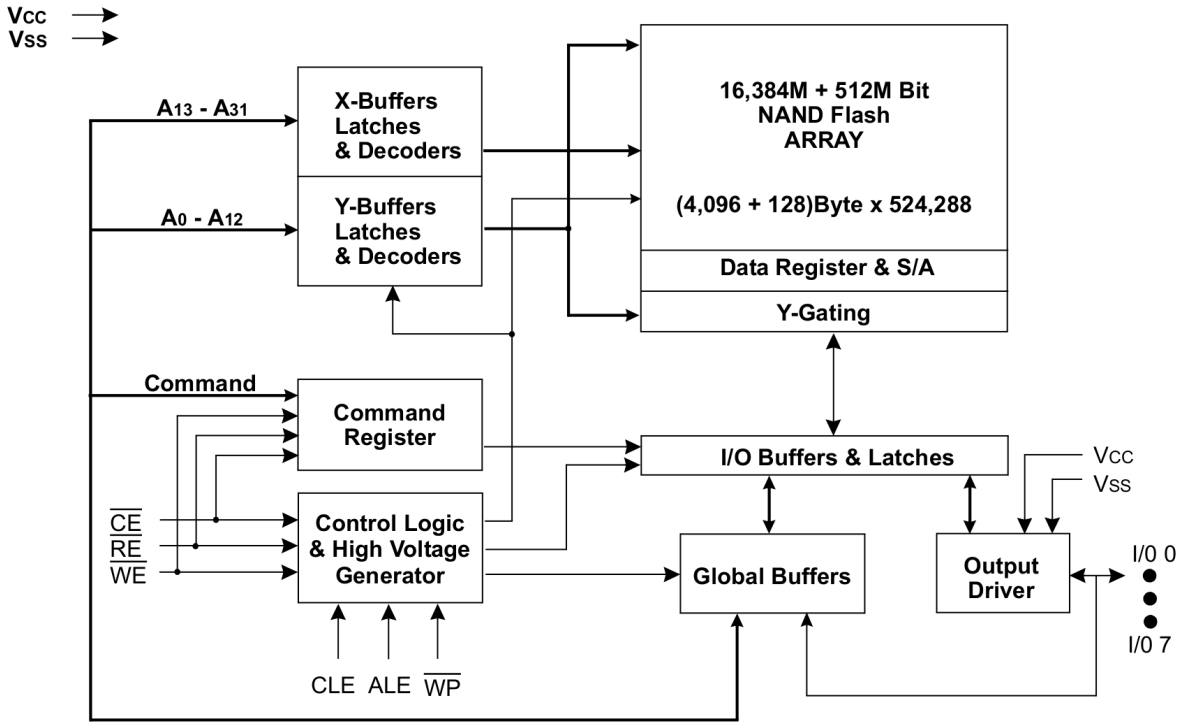


FIGURE 4.3: Functional block diagram for the flash memory chip of the RU [67].

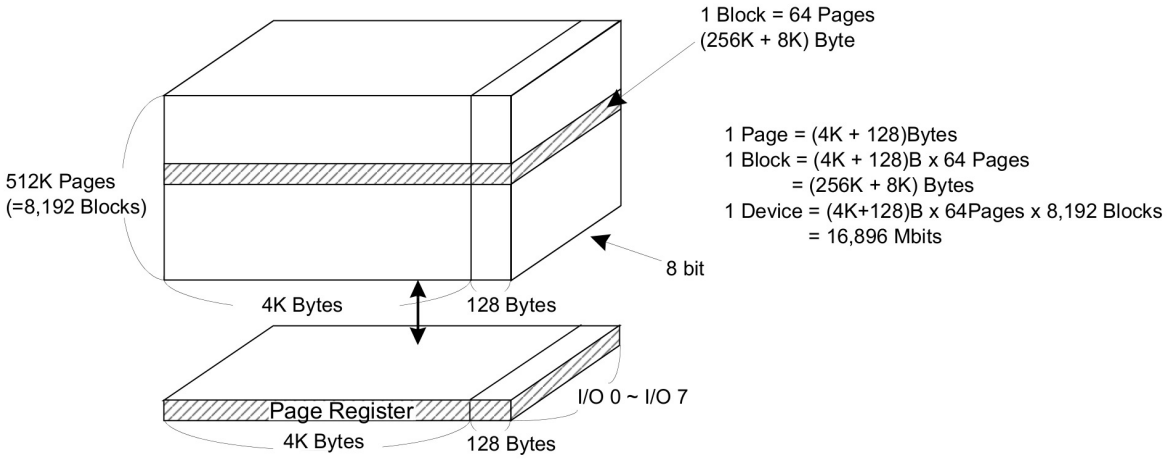


FIGURE 4.4: Flash array organization [67].

Invalid Blocks

The external flash memory may come with several invalid blocks[10] when it is shipped from the manufacturer[11], and additional blocks may become invalid as the device wears. The initial invalid blocks are disconnected from the data lines internally, and

[10] A block that behaves erroneously and should not be used.

[11] IC manufacturers achieve a higher yield for high-density ICs, such as memory chips, by identifying and disabling defective regions of a chip during wafer–testing.

they can be discovered by reading a specific column address in the block. The manufacturer also guarantees that the first block is valid for the first 1 000 program/erase cycles[12] [67].

The procedure for production testing of the RU boards involved generating a table of invalid blocks for each board, and maintaining a database with this information for all the RU boards.

File Format

A simple file format is used in the external flash memory to store the configuration images for the main FPGA. The first page of the flash, which is guaranteed to be valid by the manufacturer, is used to store a table that contains information about the configuration images. This page is referred to as the parameter page, and the format is shown in figs. 4.5 and 4.6. It allows three images to be stored in the flash; two redundant copies of the configuration image for the main FPGA, and a scrubbing image. For each of the images there is a four-byte pattern of 0x665599AA that must be present to indicate that the image is valid. The format also allows the images to be placed in a good section of the external flash based on the information from the database of invalid blocks. And the second copy of the configuration image accounts for the possibility that a block would go bad in the future, or that parts of the primary configuration image should be corrupted due to radiation.

Radiation Cross-section. Beam-testing experiments have shown that the SEU cross-section of a bit in the flash storing a zero is 10^{-16} cm^2/bit, compared to 10^{-21} cm^2/bit for a bit storing a one. Since the configuration files for the Xilinx FPGA have around a 20 to 1 ratio of zeroes to one, the configuration images are stored inverted in the flash which should improve the radiation cross-section by around six orders of magnitude [68].

Error Correction Codes. The configuration images in the flash can be encoded using ECC. *Extended Hamming Codes*[13] are typically used for NAND flash memory [69], and this requires a $2n$ bit code for 2^n bits of data [70], [71]. The data in the external flash are encoded in blocks[14] of 128 bytes, and the ECC code for each block is $2 \times log_2(128 \times 8) = 20$ bits. But for practical reasons the ECC code is stored as three bytes (i.e. 24 bits), and the 128 data bytes are immediately followed by the three ECC bytes in the

[12] Assuming ECC with 1 bit correction per 512 bytes.
[13] Hamming distance of four.
[14] Not to be confused with the blocks of 64 pages in the flash.

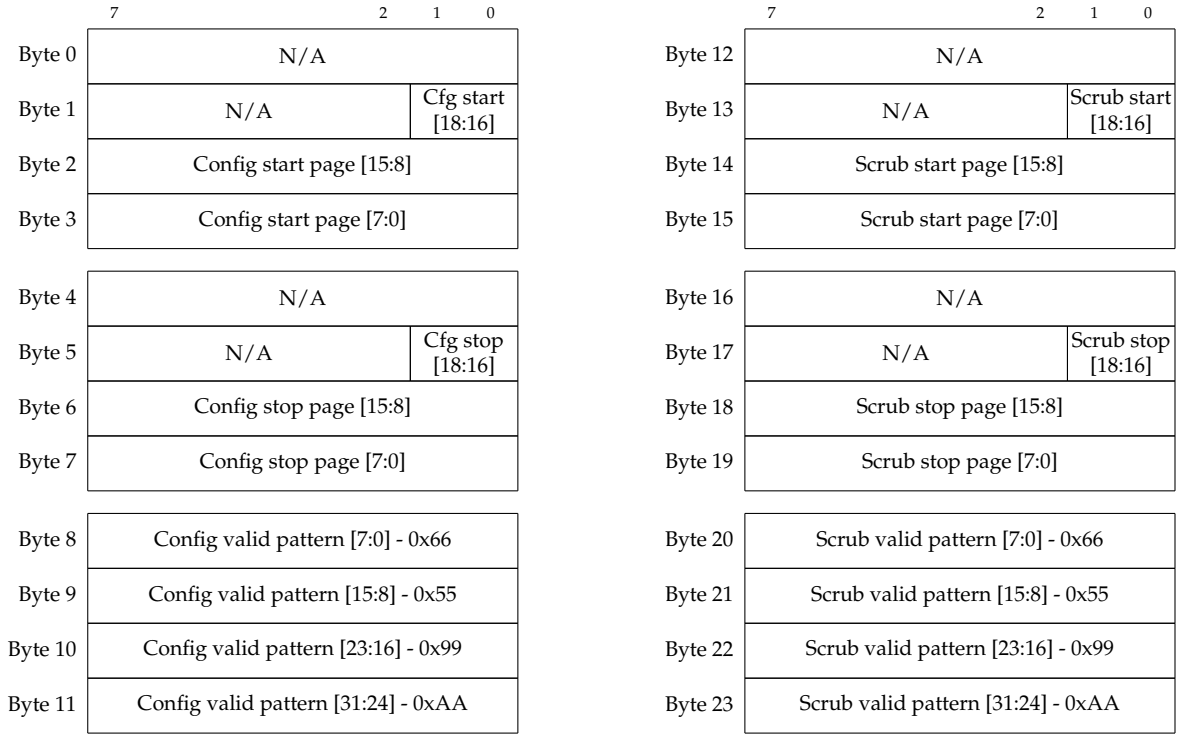


FIGURE 4.5: Parameter page in the external flash memory (continued in fig. 4.6). [64]

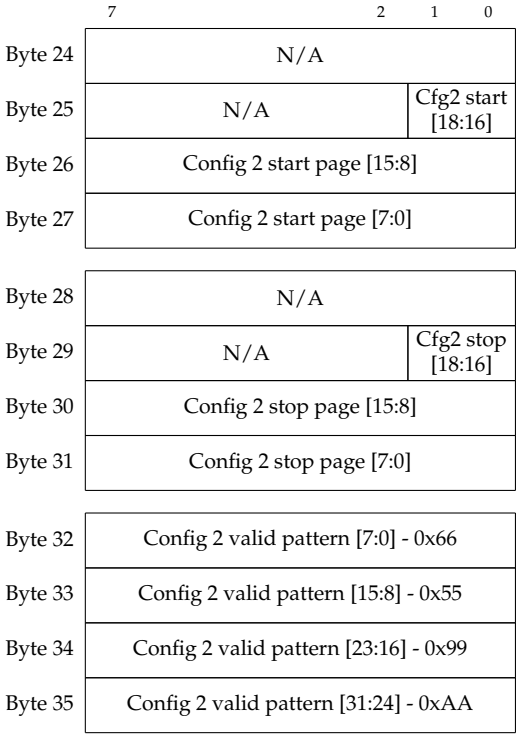


FIGURE 4.6: Parameter page in the external flash memory (continued from fig. 4.5). [64]

flash. This brings the total page size up to 4 192 bytes[15] and utilizes 96 of the 128 "spare bytes" in a page[16].

Flash Interface. The *Flash Interface* module, shown at the bottom left of fig. 4.1, offers a convenient interface that the rest of the FPGA design uses to access the external flash[17].

In a NAND flash memory like the K9WBG08U1M, transactions such as erasing blocks or reading and writing pages is performed by writing predefined sequences of command IDs, row and column address, and data, on the 8-bit IO-interface. The auxiliary FPGA design uses a subset of the available commands; read ID code, read page, write page, and block erase. Sequences for these commands are implemented by the Flash Interface.

The interface to the rest of the FPGA design primarily consists of control inputs for command, row address, and execute, and of course status outputs. For the datapath the Flash Interface has a set of typical FIFO signals[18] and reads or writes data directly to and from FIFOs in the FPGA design.

The Flash Interface also has a simple pattern checker which compares each byte of data that is read to a fixed value (or "pattern"). By reading a range of the flash that had been initialized to this pattern, the number of bits that have flipped can be counted easily without having to transfer the data via the slow I^2C and UART interfaces. This feature was used during beam testing to estimate the cross-sections of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions in the flash.

4.3.4 Read and Write Controllers, and ECC

Writing data to the flash is controlled by the *Write Controller*, shown at the center left of fig. 4.1, which can accept data coming from the main FPGA, and also directly from the WB bus in the auxiliary FPGA design. The Flash Interface reads from this FIFO when it writes a page. Using the Flash Interface, the Write Controller writes data directly to the flash starting at an address configured by a set of WB registers. These registers are used regardless of data source, and must be configured via the I^2C or UART interfaces, so the write process can not be fully controlled from the main FPGA. Neither will

[15] The *Flash Interface* must be configured for the right page size in the *FLASH_TRX_SIZE_LSB/MSB* registers, i.e. 4096 bytes without ECC and 4192 bytes with ECC.

[16] Note that the ECC codes are interwoven with the data, they are not fully stored in the spare section of the flash. This simplified the auxiliary FPGA design and eliminated the need to buffer a full page.

[17] "Flash Interface Controller" might have been a better name to avoid confusion with the IO-interface between the external flash and the FPGA.

[18] *Data in*, *read enable* and *empty* signals for the write FIFO, *data out*, *write enable*, and *full* signals for the read FIFO.

the controller update the parameter page in the flash; the parameter data must be generated externally and written in the same fashion using the Write Controller.

The FIFO for data that is read from the flash resides in the ECC module, also shown in fig. 4.1. The Flash Interface writes data directly to this FIFO when it is reading a page. When ECC decoding is enabled, the ECC module will automatically decode the data stream using the aforementioned Hamming ECC codes, and strip away the three code bytes. The process of reading pages is normally controlled by the *Read Controller*, which also uses the Flash Interface. Access to the interface is multiplexed between the Read and Write Controllers. Four commands are supported by the Read Controller; read configuration image 1, read configuration image 2, read scrub image, and read custom range. For the first three of these commands, the parameter page is read automatically. In this case the Read Controller reads data from the read FIFO in the ECC module, verifies the parameter page and determines where the desired image is located, and proceeds to read data for the actual image. The last command allows for random access to the flash by specifying a range of pages to read via a set of WB registers. The controller itself does not read from the FIFO in the ECC module, with the exception of when the parameter page is read before an image. It simply makes the data available in the FIFO for either the *Configuration Controller* or the WB bus interface.

Originally the auxiliary FPGA design had a 4 096-byte page buffer which was used for both reading and writing, instead of the FIFOs and Read and Write Controllers. But the addition of the FIFOs simplified the design of the ECC decoding, reduced the number of resources required, and consequently reduced the radiation cross-section for the design.

FIFO Interface to Xilinx FPGA for Configuration Data

The previous chapter introduced a FIFO interface between the two FPGAs (see section 3.4). It allows for fast transfer of configuration images (for the main FPGA) to external flash using the GBT links. The actual FIFO is part of a module in the main FPGA, where it is a WB slave. The interface to read this FIFO is exposed on a set of IO pins that connect to the auxiliary FPGA, and the interface is controlled by the Write Controller of the auxiliary FPGA design, as shown on the left side of fig. 4.1.

4.3.5 Configuration Controller

The *Configuration Controller* is at the center of the scrubbing solution for the Xilinx UltraScale main FPGA. As shown in fig. 4.1, it uses the Read Controller to read configuration data from the external flash memory, and writes ECC decoded configuration

data to the main FPGA via the SelectMAP Interface.

A simplified state diagram of the FSM in the Configuration Controller is shown in fig. 4.7. The same FSM is used for both initial configuration and scrubbing. The sequence of state transitions are almost the same in both cases. An initial configuration will reset the SelectMAP Interface and run one time through the process to configure the main FPGA. Initial configuration can be executed automatically when the system is powered up or reset, depending on the position of a DIP-switch on the RU board, or it can be started via a WB register. Scrubbing does not start automatically and must always be triggered a WB register. A continuous scrubbing mode allows the scrubbing to run indefinitely in a loop once it has been started.

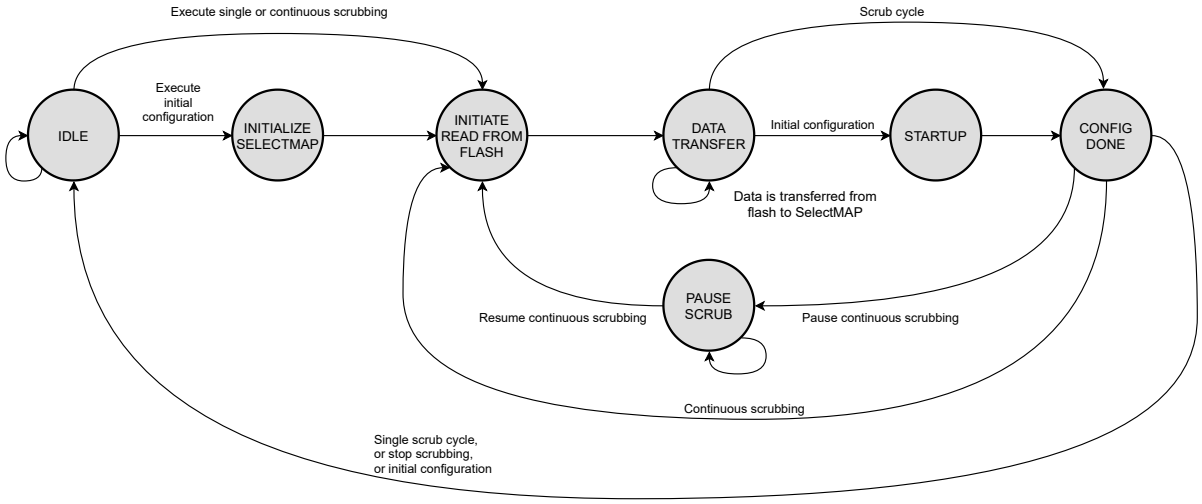


FIGURE 4.7: Simplified FSM diagram for the Configuration Controller.

4.4 Mitigation of Radiation Effects

The mitigation strategies for the design primarily consists of the use of TMR, and the aforementioned ECC encoding of the configuration data in the flash memory. But the strategies for TMR differ from the techniques that were described in section 3.8 for the main FPGA, which were aimed at minimizing the number of voters in the design. Since the flash-based configuration memory of the auxiliary PA3 FPGA is not as susceptible to SEUs [72], an increased number of voters will not add to the cross-section of the design. This makes LTMR, where each individual FF is triplicated and voted, a suitable approach for the auxiliary FPGA design[19]. Manually adding LTMR to an HDL design is a time-consuming and error-prone process. But the *Synplify* toolchain

[19]The design for the auxiliary FPGA is resource constrained and LTMR offered sufficient protection. Better protection could likely have been achieved with Distributed (DTMR) [72], but at the expense of requiring more resources in the FPGA.

that is used for synthesis of the design supports automatic generation of the LTMR [73], and takes place when the attribute shown in listing 4.1 is present in the architecture declaration of a design entity. Almost the entire design is triplicated in this way using the Synplify tools, with the exception of the I^2C module which was manually designed for LTMR.

LISTING 4.1: VHDL attributes for TMR in the auxiliary FPGA design.

```
1 attribute syn_radhardlevel : string;
2 attribute syn_radhardlevel of behave: architecture is "tmr";
```

Chapter 5

FPGA Design Verification and Testing

It goes without saying that a complex design like the ITS detector requires extensive testing before it can be put to use. It has been an ongoing activity involving a large number of people across different sites. A lot of the details are far beyond the scope of this thesis, but this chapter will give an overview of some of the most relevant activities relating to the readout electronics, ranging from verification of the FPGA designs, beam-testing of the readout electronics, and commissioning of the detector. As with the previous chapters, a general overview will be provided, with some extra focus on the work related to this thesis.

5.1 Test Software for the FPGA Designs

Accompanying the main FPGA design is an extensive software suite written in the Python programming language. The software serves several purposes. It is used for verification in the so-called "Python co-simulation" of the main FPGA design; regression testing of the design using the actual RU hardware; and for control and operation of the readout electronics and detector itself.

5.1.1 Board Support Package for the RU and Main FPGA

The Board Support Package (BSP) is a core component of the software suite. It primarily targets the main FPGA, but also includes modules to control the auxiliary FPGA. In broad terms the BSP is structured into a communication stack, software modules for each individual WB module in the design, and an RU board class that ties it all together, as shown in fig. 5.1.

The communication stack is agnostic to the underlying communication interface and supports USB, CAN, and SWT (over GBT via the FLP) using the O^2 software). This is illustrated in fig. 5.2. The stack primarily allows for register access in each

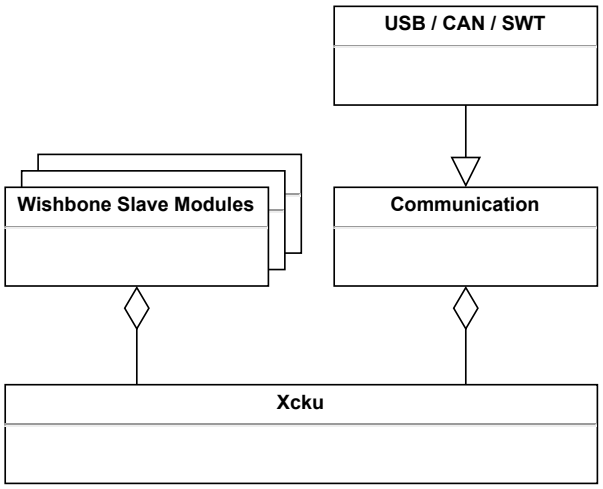


FIGURE 5.1: General structure of the BSP for the RU.

individual WB module in the design, but also includes some more advanced functionality such as sequencing of WB access.

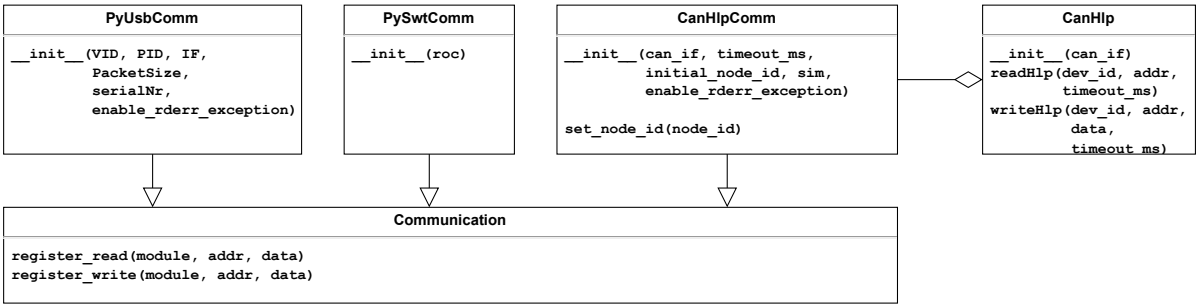


FIGURE 5.2: Overview of Communication classes in the BSP.

The WB slave modules in the BSP follows the pattern shown in fig. 5.3, which specifically shows the implementation of the CAN HLP WB slave. A generic *WishboneModule* has a reference to the *Communication* object and implements read and write in the specific module in the design. The main module for the WB slave, *WsCanHlp* in this case, inherits from the *WishboneModule* and implements functions specific to CAN HLP, such as changing the bit rate. It also has a reference to the corresponding counter monitor module, *WsCanHlpMonitor*, which is derived from the generic *WsCounterMonitor*. Each WB slave module follows this pattern, both in the BSP software and in the HDL design.

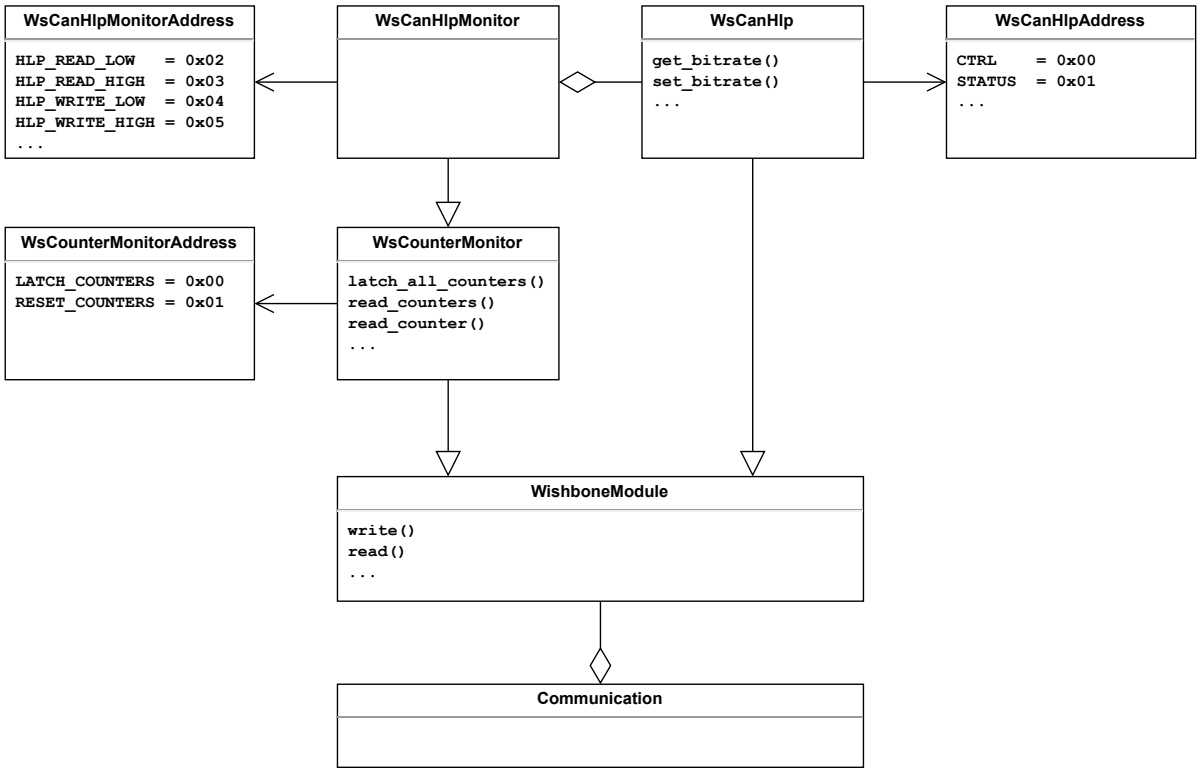


FIGURE 5.3: Overview of Wishbone slave for CAN HLP in the BSP software.

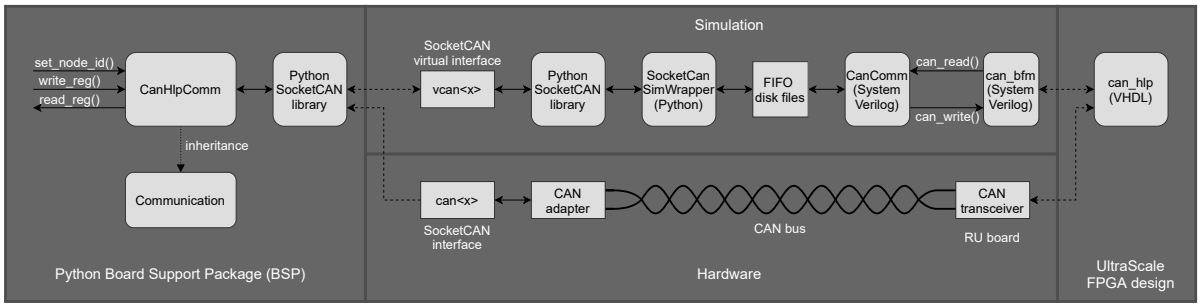


FIGURE 5.4: Software communication stack for CAN.

Software Communication Stack for CAN

As an example of a communication stack in the BSP software, fig. 5.4 shows the implementation of CAN bus communication for both hardware testing and Python co-simulation. *CanHlpComm* is derived from the Communication base class, and implements the CAN communication interface for the BSP software. It talks directly to a *SocketCAN* interface, which can be either a real hardware interface (named *can0*, *can1*, and so on), or a virtual SocketCAN interface (*vcan0*, *vcan1*, and so on). The SocketCAN interface is the end-point when communicating with real hardware, but for the Python co-simulation there are several additional steps. An additional class, the *SocketCanSimWrapper*, acts as a wrapper between the virtual SocketCAN interface and the

"FIFO file"[1] interface to the HDL simulation. In the HDL simulation the *CanComm* class, implemented in *SystemVerilog* communicates with the Python co-simulation via these files, and uses a Bus Functional Model (BFM) for CAN to send requests and receive responses from the CAN HLP module in the FPGA design.

5.1.2 Testbench Software

A dedicated testbench script, also Python-based and using the aforementioned BSP, is available for testing and operation of the RU. Manual access and control of individual WB modules is possible from the script, along with triggering and data readout of the ALPIDE sensor chips.

The testbench has been used extensively during commissioning of the detector.

5.1.3 Regression Test Suite for the Main FPGA

The Python-based regression test suite is heavily based on the BSP and features tests for most aspects of the design, such as:

- Sanity check of each WB module and WB register in the design
- Test of communication interfaces such as GBT and CAN
- Test of control and data interfaces to the ALPIDE chips
- Test of the trigger and readout systems of the FPGA design

The regression test suite is used for hardware testing, but Python co-simulation of the FPGA design is also possible.

5.2 Verification of Main FPGA Design

Verification is perhaps the most critical and challenging step of the entire FPGA design process. In contrast to the world of software, where execution is generally sequential and comprehensible, FPGAs add several dimensions of complexity; the parallel nature of signals, gates, and FFs; routing and distribution of signals across the FPGA, and the associated signal delays; resource limitations, and synchronization between clock domains, to name a few.

Among the most important steps in the verification process is functional verification of the RTL design and static timing analysis. Functional verification comprises

[1] Explained in more detail in section 5.2.1.

the simulation of the HDL code for an RTL design. Test vectors are generated for the inputs of the design, and its outputs are observed to verify its functionality. Verification frameworks like Universal Verification Methodology (UVM) for SystemVerilog, or Universal VHDL Verification Methodology (UVVM) for VHDL, are usually employed for this purpose. Typically, the RTL design is treated as an ideal synchronous circuit in the simulation [74]. But after the design has been synthesized and implemented for an FPGA, it is possible to extract delays and timing information and simulate the design as it is implemented in the FPGA. Static timing analysis is performed by the FPGA vendor's toolchain as the design is synthesized and implemented. By estimating routing delays for signals and propagation delays in combinational logic, the arrival time of signals relative to clock edges is calculated. With this information, the tools can verify that setup and hold times are not violated for each register in the design. However, the static timing analysis relies on user-specified timing constraints (usually specified in a constraint file of the industry-standard Synopsys Design Constraint (SDC) format), and specifying the necessary constraints for a design is a challenge in its own right.

The verification of the main FPGA design follows this classical approach of functional verification and static timing analysis. Testbenches based on the UVM and UVVM frameworks are used for verification of individual modules in the design, and the regression test suite is used for tests of the top-level design in the Python co-simulation. Simulations of the testbenches and regression tests are run automatically in a Continuous Integration (CI) pipeline associated with the project's *Git* repository to ensure that existing features still work as expected as changes are made to the design.

5.2.1 Python Co-simulation

The so-called "Python co-simulation" has been an important part of the development cycle for the FPGA design. It runs automatically when changes are made to the design, as part of a CI pipeline in the project's git server.

The co-simulation uses the Python-based regression test suite to control an HDL simulation[2] of the full FPGA design. Most of the regression test suite can be used with the simulation, including tests that involve communication with the sensor chips; a lightweight HDL model of the ALPIDE in the simulation makes this possible.

On the HDL-side of the simulation there is a BFM for each communication interface in the design. On the Python-side, there is an additional layer for simulation

[2]Simulation is possible using either Mentor Graphics' ModelSim/QuestaSim, or Cadence's Incisive/NCSim.

in the communication stack, which allows for communication with the BFMs. Data is communicated between the HDL simulation and Python-scripts using disk files, which essentially act as FIFOs for data to be written to or received from the BFMs. The Python scripts also run an Remote Procedure Call (RPC) server that allows the HDL-based BFMs some degree of control over the dataflow.

5.2.2 Module Testbenches

The regression test of the Python co-simulation can be likened to an integration test since the whole FPGA design is simulated. The simulation time is quite long (around 1-2 hours), and the scope of individual tests is limited for that reason. It is challenging to test most corner-cases and achieve a high coverage under such circumstances, so it is also necessary to simulate and verify modules in isolation with their own individual testbenches, as part of the CI-pipeline for the FPGA design. Some of the modules associated with the trigger system and datapath are verified with UVM-based testbenches. In addition, there are several UVVM-based testbenches; for the *Alpide Monitor*, which communicates with several instances of the Lightweight-Model of the ALPIDE chips, via an instance of the *Alpide Control* module; the CAN HLP module, which is tested with several instances on one CAN-bus line to test and verify addressing of nodes, as well as broadcasting; and the FIFO module for configuration data.

Canola CAN Controller Testbench

Verification of the Canola CAN controller is also described here, although it is a dedicated project and verification of the controller is technically not part of the main FPGA design project.

Bosch offers a *CAN VHDL Reference* verification suite for CAN controller designs. Ideally, the CAN controller should be verified with this framework. But presently the controller is verified with a set of custom testbenches aimed at verifying conformance to the CAN specification. The testbenches are fully based on the UVVM framework, and feature a custom CAN BFM for UVVM.

The submodules for the BTL, BSP, and EML are verified with dedicated testbenches. It is verified that the BTL is able to synchronize with a falling edge, resynchronize during a bit stream[3], and receive and transmit raw sequences of bits. The BSP is tested by sending and receiving data with the BSP interface, using the BTL. It is verified that the BSP correctly inserts stuff bits when transmitting, and discards stuff bits when receiving. The EML submodule in the Canola design has inputs

[3]Up to 1.5% error in bitrates should be tolerated according to the CAN specification.

for different types of error and success conditions, and it maintains the REC and TEC counters, and error state[4] of the controller. The rules and behavior of these counters is fully described in the CAN specification, and the testbench for the EML verifies that the EML-logic conforms to the specification.

For verification of the full design of the controller there is a top-level testbench. The FSMs for transmitting and receiving CAN frames are tested as part of the full design and do not have dedicated testbenches. The top-level testbench verifies aspects of the CAN specification that are not covered by the other testbenches, such as arbitration loss and retransmission, error state of the controller and detection of errors such as form errors, stuff errors, and CRC errors, which are generated in frames by the BFM. This includes TMR and non-TMR version of the controller. Both the TMR and non-TMR versions of the controller are verified.

And finally, communication between the Canola controller and the CAN controller design from *OpenCores* is tested in a dedicated testbench. The OpenCores design **has** been verified with the Bosch CAN VHDL Reference, so this is an important test of interaction with a proven CAN controller design.

Coverage. Table 5.1 shows reported numbers from the top-level testbench for individual design units in the non-TMR version of the controller, as well as the total coverage for the top-level entity itself (*canola_top*). A total is also shown for the TMR version, but numbers for individual design units are not included here. The voters used in the TMR design are tested with 100% coverage in their own testbenches. The voters are assumed to be fully verified and mismatches in the three paths are not simulated for the TMR version of the controller. Therefore, the reported coverage is lower for the TMR version, and thus the numbers for the non-TMR version are a better representation of coverage for the CAN controller. Coverage numbers from dedicated testbenches for individual design units are reported in table 5.2. Certain submodules, such as the EML, are hard to test in the full design but are verified with higher coverage in their dedicated testbenches.

From table 5.1 the coverage of the full controller (*canola_top*) is 84.73%. This does not take into account the coverage numbers from table 5.2. If these numbers could be combined the total for the controller would quite possibly be higher. A coverage of 100% can be extremely hard to achieve for an advanced design, but a higher number than 84.73% would be desirable. But it is worth noting that the design is not for an ASIC but for FPGAs which are re-programmable. The coverage will likely be improved in the future as the design matures, and it is always possible to fix bugs that are uncovered in the FPGA design.

[4]Error active, error passive, or bus off.

TABLE 5.1: Simulation coverage of Canola CAN controller from top-level testbench.

Design unit	Statement	Branch	FEC Expr.	FEC Cond.	Assertion	Total
canola_frame_tx_fsm	82.70%	83.18%	100.00%	40.00%	N/A	76.47%
canola_frame_rx_fsm	85.18%	83.73%	N/A	68.88%	N/A	79.27%
canola_bsp	96.22%	93.93%	90.00%	80.00%	100.00%	92.03%
canola_btl	96.42%	94.44%	0.00%	78.94%	N/A	67.45%
canola_eml	77.55%	76.92%	N/A	100.00%	N/A	84.82%
receive_error_counter	75.00%	72.72%	N/A	100.00%	100.00%	86.93%
transmit_error_counter	83.33%	81.81%	N/A	100.00%	100.00%	91.28%
recessive_bit_counter	75.00%	62.50%	N/A	100.00%	N/A	79.16%
Total canola_top	88.33%	86.69%	76.47%	72.16%	100.00%	84.73%
Total canola_top TMR	80.85%	76.05%	74.46%	26.90%	100.00%	71.65%

TABLE 5.2: Simulation coverage of individual submodules of the Canola CAN controller from their dedicated testbenches.

Design unit	Statement	Branch	FEC Expr.	FEC Cond.	Assertion	Total
canola_btl	97.32%	95.55%	0.00%	73.68%	N/A	66.64%
canola_bsp	71.69%	67.67%	80.00%	13.33%	100.00%	66.54%
canola_eml	100.00%	100.00%	N/A	100.00%	N/A	100.00%
up_counter[5]	90.00%	87.50%	N/A	100.00%	N/A	92.50%
up_counter[6]	90.00%	87.50%	N/A	100.00%	N/A	92.50%
counter_saturating	100.00%	100.00%	N/A	100.00%	100.00%	100.00%
TMR voters	100.00%	100.00%	N/A	100.00%	100.00%	100.00%

5.3 Verification of Auxiliary FPGA Design

Since the design for the auxiliary FPGA is much simpler than that of the main FPGA, functional verification is performed primarily through the top-level testbench for the design, which is based on the UVVM framework. The tests encompass clock and reset control, communication using the I^2C interfaces, and the entire data path for configuration data for the Xilinx FPGA (which includes the flash and SelectMAP interfaces).

I^2C communication is tested directly using an I^2C BFM, but also using an instance of the I^2C-master from the GBT-SCA HDL design in order to fully simulate interaction between the GBT-SCA chip and the auxiliary FPGA.

A behavioral model of the *Macronix MX30LF1G08AA* flash chip is used for verification of the *Flash Interface*. This model has an interface that is compatible with the *Samsung K9WBG08U1M* flash used on the RU. And verification of the *SelectMAP Interface* is performed using a verification component from Xilinx.

The testbench uses the I^2C interfaces to access WB registers and control the FPGA design. Configuration data from a disk file is written to the flash model using I^2C or

TABLE 5.3: Simulation coverage of the auxiliary FPGA design.

Design unit	Statement	Branch	FEC Expr.	FEC Cond.	Toggle	FSM State	FSM Trans.	Assertion	Total
clock_lol_counters	100.00%	100.00%	100.00%	N/A	66.66%	N/A	N/A	N/A	91.66%
flash_interface	98.46%	97.71%	64.28%	0.00%	78.79%	99.27%	53.30%	N/A	69.25%
selectmap_interface	66.21%	63.24%	100.00%	33.33%	65.78%	62.50%	42.85%	N/A	63.54%
config_controller	57.54%	57.37%	0.00%	15.38%	39.05%	53.84%	22.58%	N/A	34.59%
read_controller	79.26%	75.31%	100.00%	28.57%	21.92%	72.72%	36.00%	N/A	59.90%
write_controller	88.28%	82.19%	1.92%	53.84%	57.32%	100.00%	69.23%	N/A	61.36%
ecc	59.91%	61.80%	4.00%	56.25%	36.37%	20.00%	0.00%	N/A	38.05%
reg_block	86.16%	70.42%	25.00%	25.00%	34.97%	N/A	N/A	N/A	48.31%
checksum	80.00%	80.00%	N/A	50.00%	58.13%	N/A	N/A	N/A	67.03%
top_level total	83.30%	80.36%	21.12%	34.31%	43.09%	87.69%	47.19%	N/A	54.94%
clk_div	100.00%	100.00%	N/A	N/A	100.00%	N/A	N/A	N/A	100.00%
reset_ctrl	93.33%	83.33%	60.00%	N/A	71.87%	N/A	N/A	N/A	77.13%
IO_buffers	N/A	N/A	N/A	N/A	57.57%	N/A	N/A	N/A	57.57%
i2c_wb	90.00%	91.98%	91.66%	71.42%	92.01%	N/A	N/A	0.00%	72.84%
i2c_wb_2	90.00%	91.98%	91.66%	85.71%	73.49%	N/A	N/A	0.00%	72.14%
RU_auxFPGA total	83.55%	83.65%	39.45%	47.22%	53.55%	87.69%	47.19%	0.00%	53.81%

a model of the FIFO interface to the main FPGA, which involves both the *Write Controller* and the *Flash Interface* modules. And this data is then written to SelectMAP using the configuration chain, which allows the *Read Controller*, *Configuration Controller*, ECC module, and *SelectMAP Interface* to be tested.

Coverage

Coverage for the auxiliary FPGA design achieved in the main top-level testbench is summarized in table 5.3. The first nine rows shows the coverage of submodules that are part of the *top_level* entity of the design, followed the total coverage for the *top_level* entity. But it is the *RU_auxFPGA* entity that has the connections to the IO-pins in the FPGA, and this entity has an instance of *top_level* along with I^2C-slaves, clock and reset control. Thus, the last row for *RU_auxFPGA* shows the complete coverage results for the design.

5.4 Hardware Testing of FPGA Designs

A hardware regression test step in the CI-pipeline allows the same regression tests that are used with the Python co-simulation to be run in the actual RU hardware. Additional hardware tests, which are manually triggered, runs on three sub-racks of RUs connected to the actual detector, allowing for more extensive tests with all three stave types.

That briefly summarizes the structured and repeatable tests that are performed for the main FPGA design[7]. Such tests are indispensable for the quality assurance of the design. But it should also be recognized that ad hoc, and unstructured manual tests, *have* played a role during the development of various modules for the design. For example, long-term tests of a module[8], or early smoke-tests and manual testing of a new module in the lab. For instance, the *Alpide Monitor* module has only been tested manually with an IB stave at UiB, as it has not been included in a release of the design yet. But structured test cases are typically developed for the errors that are uncovered by these test activities. A discussion of long-term and stress tests of the CAN controller and HLP logic, for the main FPGA, follows in the next sections.

5.4.1 Canola CAN Controller

The design of the Canola controller was tested using a Digilent ZYBO Zynq-7000 board [75], which features a Xilinx Zynq-7000 FPGA[9].

The *Digilent Pmod CAN* [76] extension boards were used to add a CAN transceiver to the system. These boards are based on the *Microchip MCP25625* chip, which features both a CAN controller and CAN transceiver. The controller and the transceiver are not connected internally in the chip; there is a set of T_{XD} and R_{XD} pins for the transceiver, and *RxCAN* and *TxCAN* pins for the controller. On the *Pmod CAN* boards these pins are connected and an SPI interface to the controller is available on the header (J2) that connects to the ZYBO board. To offer a direct connection to the transceiver for the Canola controllers in the Zynq-design the *Pmod CAN* boards had to be modified. Figure 5.6 shows the modification. The SPI connections to the header were cut, the traces for the Tx/Rx interface between the controller and transceiver were cut, and wire straps were added from the traces for the transceiver's T_{XD} and R_{XD} pins to the header. The *Pmod CAN* boards were also too wide to fit next to each other on the ZYBO board, so the ZYBO board itself was also modified to make two of the Pmod-connections vertical. A PEAK System PCAN-USB [77] adapter was used to connect the system to a computer for testing. The whole setup is shown in fig. 5.5.

Test Design for Zynq FPGA

The Zynq-based FPGA design features four instances of the Canola CAN controller with an Advanced eXtensible Interface (AXI)-slave to interface with the Zynq CPU.

[7] A Python-based test-suite is also available for the auxiliary FPGA design, as well as some functional tests that can be run via the CRU.

[8] There is typically a limited amount of time available for the structured test cases.

[9] Part number *xc7z010clg400-1* is the exact FPGA model on the board. The Z-7010 is the smallest of the Zynq-7000 FPGAs [75].

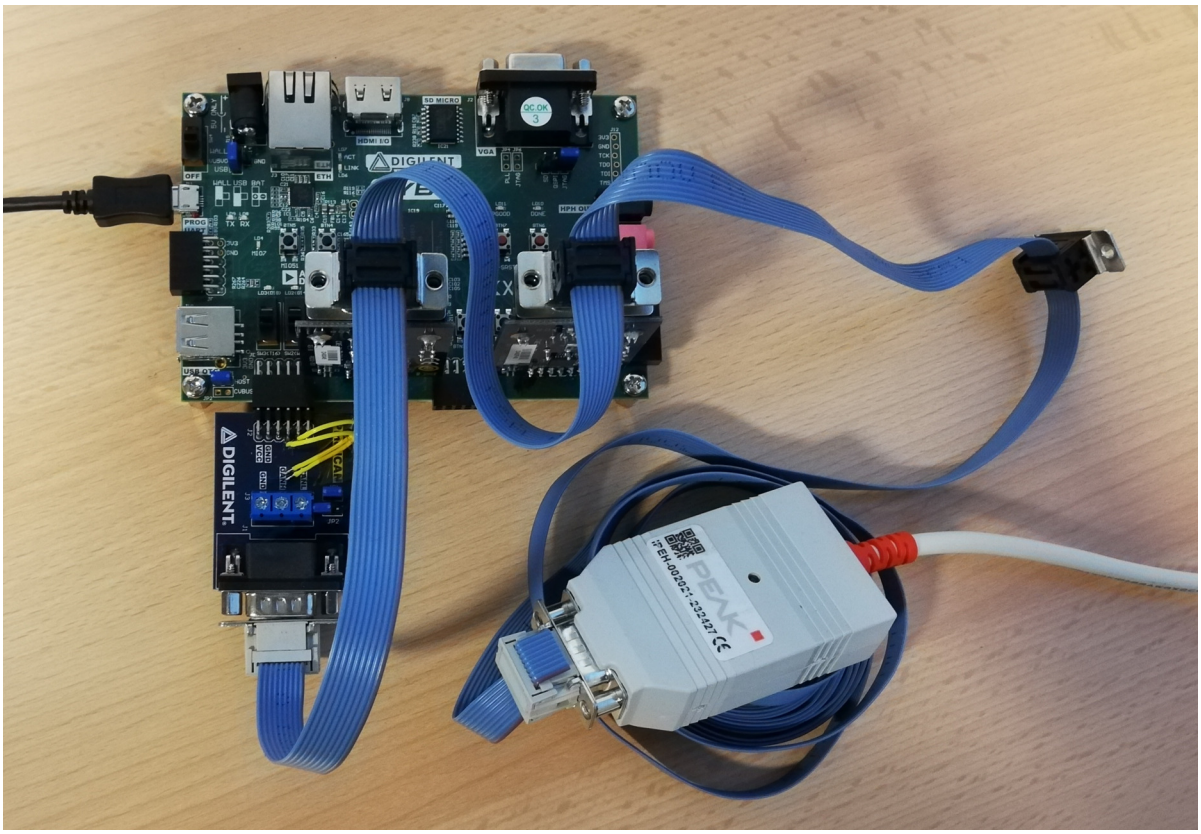
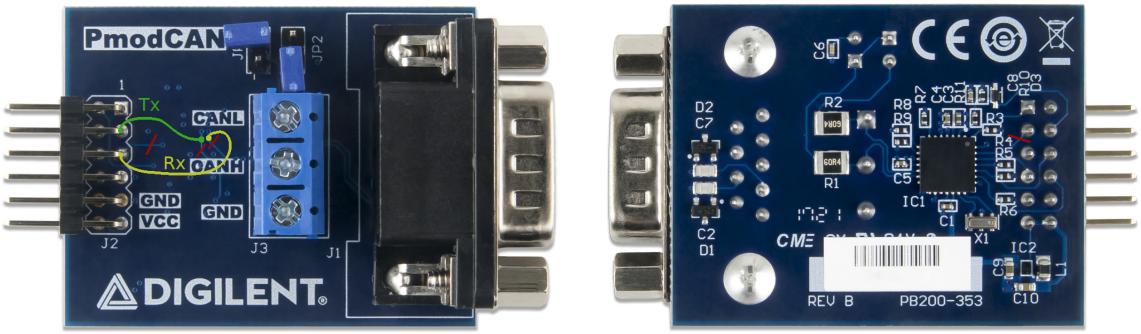


FIGURE 5.5: Digilent ZYBO Zynq and Pmod CAN boards with PEAK CAN to USB adapter for testing of the Canola CAN controller.



(A) (B)

FIGURE 5.6: Modified Pmod-CAN PCB for Canola CAN controller test. Red lines indicate where traces were cut. Green and yellow lines indicate where wire straps were added.

Two instances use the TMR version of the design, one with TMR enabled and one where it is disabled. The other two instances are of the non-TMR version of the controller. The Bus Tool (BUST) project [78] was used to automatically generate the HDL-code for the AXI-slave as well as a C++ module for use in the Zynq firmware. Figure 5.7 shows a block diagram of the Zynq design.

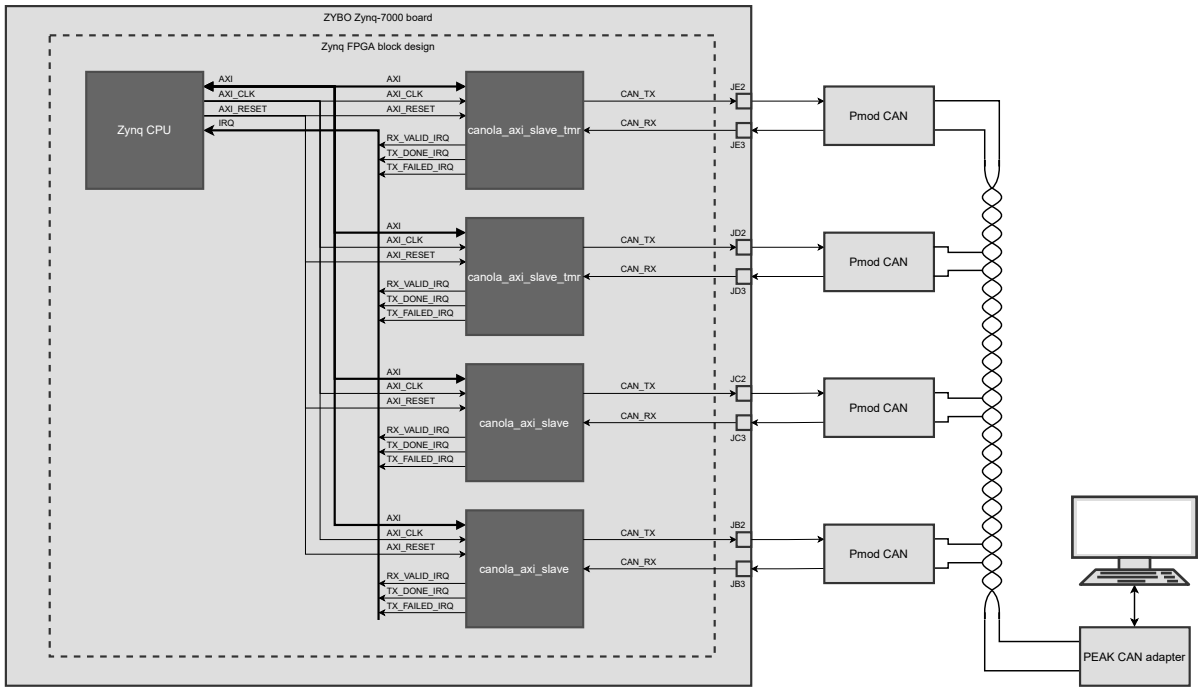


FIGURE 5.7: Zynq system for Canola CAN controller test.

A test was performed where the three controllers alternated between transmitting a message. The messages were generated with random data and parameters, which includes the arbitration ID, extended ID or not, remote request or not, data length and data. A total of 4 070 951 messages were sent and successfully received without errors among the controllers. Testing of simultaneous transmission from the three controllers, as well as at random times, has also been perform to verify that the bus arbitration rules are obeyed by the controllers. At this point the controller appears to be working reliably, but a more long-term test of the controller still needs to be performed.

5.4.2 CAN HLP

Early testing of the HLP for CAN primarily involved writing and reading register values in a loop from the main FPGA design. Testing was performed in Bergen, where two RUs were available for these tests, among the DCS group in Kosice, and at the commissioning site in building 167 at CERN. Testing in Bergen was performed with

a *PEAK System PCAN-USB* [77] adapter. The other test sites used an *AnaGate CAN Quattro* [79] gateway, which is the actual CAN hardware that will be used by DCS in the experiment.

The first version of CAN HLP, which was based on the OpenCores CAN controller, ran for most of the commissioning period[10] at CERN. The system was online most of the time during commissioning, and the CAN bus was actively used by the DCS system for the interlock monitoring of temperatures and voltages. The final version, based on the Canola CAN controller, ran for the last month of the commissioning period. The plots in fig. 5.8 shows the counts of HLP read transactions, received CAN messages, and CAN stuff errors from commissioning runs 603 823 to 603 896. The graphs on the left-hand side are for RUs *L1_00* to *L1_07*, and these RUs are electrically connected on the same CAN bus line. The graphs on the right, for *L1_08* to *L1_15*, are on a different CAN bus line. The counters for the RUs track each other and are normally not reset between runs. However, it appears that RU *L1_11* was reset at one point, as seen from the right hand graphs. This reset appears to coincide with an increase in stuff error counts for RUs *L1_08* to *L1_15*. The cause of some of these stuff errors, or possibly all of them, is due to a missing feature in the Canola CAN controller: when the controller is reset it should wait for the bus to be idle (i.e. 11 consecutive recessive bits) before attempting to receive. Since there is continuous traffic on the bus for the DCS interlock, it is possible that when a controller is reset, it will start receiving mid-message and raise an error flag, which is interpreted as a stuff error from the other controllers. No other types of errors were observed, however there were some instances of arbitration loss. In principle this should not happen since the broadcast feature of the HLP is not used by the interlock monitoring and the RUs are accessed one by one. However, if an RU does not respond to a HLP request in time, the request will time out and the DCS interlock monitoring will move on to the next request. Considering that the AnaGate CAN adapter is accessed via Ethernet, a plausible explanation is that intermittent network delays between the DCS computers and the AnaGate device could cause some HLP transactions to time out before they are delivered on the CAN bus by the AnaGate. If the next request gets bundled up with the one that timed out then the RU that attempts to respond to the first request will possibly lose arbitration to the next request.

Additional data was available from runs 604 000 to 604 247[11]. On average each RU performed 3 185 568 HLP read transactions, and received and transmitted 47 775 615 and 3 185 567 CAN messages, respectively. On average there were 7.7 stuff errors, and 11.4 instances of arbitration lost and retransmitted frames, per RU. This amounts to

[10] See section 5.6.

[11] This was the last run of the commissioning period.

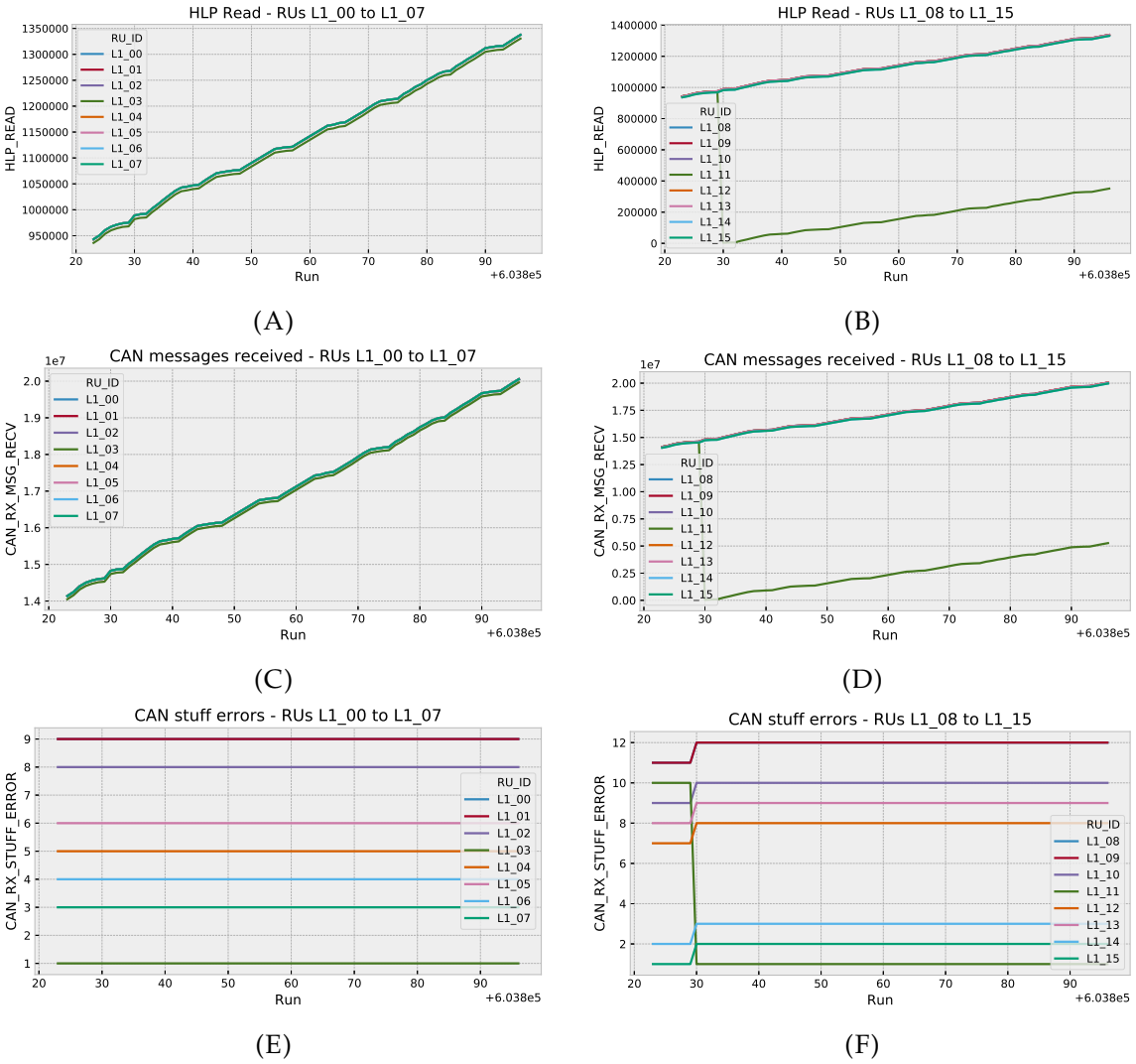


FIGURE 5.8: CAN HLP counter values from commissioning runs 603 823 to 603 896.

a stuff error rate of 2.41×10^{-6} and a retransmit rate of 3.58×10^{-6}. There were no other errors recorded.

5.5 Beam Testing

Several irradiation campaigns were performed during the development of the readout electronics for the ITS. Prior to the development of the final RU board design, and before the Xilinx UltraScale devices were available on the market, a first study was performed by Sielewicz, Rinella, Bonora, *et al.* with the Xilinx Kintex-7 device (XC7K) of the RUv0 prototype [80]. It was irradiated with 22 MeV protons at the *U-120M* cyclotron at the Nuclear Physics Institute of the Academy of Sciences of the Czech Republic, located in Řež near Prague. The objective was to evaluate different TMR test-structures, with and without scrubbing[12], and determine the Configuration RAM (CRAM) cross-section for the different configurations. The cross-section of the test-structures improved by two orders of magnitude with the highest degree of triplication and use of scrubbing, compared to the fully unmitigated case. The strategies for SEU mitigation in the final FPGA designs for the RU were heavily based on this study.

Prague Beam Test of RU Design. The first version of the RU design, the RUv1, underwent several irradiation campaigns using early versions of the FPGA designs. The first of these tests was also performed at the facilities near Prague. The main FPGA and flash memory were irradiated, but the rest of the RU board was shielded from the beam. The objective was to test blind scrubbing of the main FPGA using the JCM tool[13], and to measure the cross-section of SEUs in flash memory bits, σ_{bit}. Unused portions of the flash memory were programmed with patterns of $A5_h$[14] prior to irradiation, using the UART software for the auxiliary FPGA. When the memory was read out after the campaign, the number of flipped bits were counted and σ_{bit} was estimated. For SEUs causing a $0 \rightarrow 1$ transition, σ_{bit} was estimated to $2.62 \cdot 10^{-16}$, compared to only $4.92 \cdot 10^{-21}$ for a $1 \rightarrow 0$ transition [81].

CHARM Beam Test of RU and PU Design. The second irradiation campaign took place at the CERN High energy AcceleRator Mixed field facility (CHARM). The facility is frequently used to evaluate electronics for the LHC and its experiments.

[12]The Configuration Manager (JCM) scrubber was used during proton irradiation. Fault injection was also performed using Xilinx's proprietary SEMIP which produced comparable results.

[13]The JCM tool was not intended for use in the final design, but allowed for testing of blind scrubbing before the scrubbing solution in the auxiliary FPGA design was complete.

[14]Binary 10100101.

24 GeV/c proton beams from the Proton Synchrotron (PS) interact with a metallic target to provide a mixed-field radiation environment for this purpose [82]. The tests at CHARM were intended as a stress test of the readout electronics and FPGA designs, where the RU and a PU were fully exposed to the radiation. The electronics were expected to accumulate a high TID, and some components on the boards were likely to fail.

After four hours of operation the main FPGA was no longer responsive and could not be configured via the auxiliary FPGA. However, communication with the auxiliary FPGA was still possible, so the most likely culprit was SEUs in the external flash memory, and not a catastrophic failure of the main FPGA itself[15].

Oxford Beam Test of RU Design. A final irradiation campaign of the RU was performed at the *ChipIr* facilities at Rutherford Appleton Laboratory near Oxford. One of their facilities has a neutron beam with a base flux of $5 \times 10^6\,\mathrm{cm^2\,s^{-1}}$ (for $E_n > 10\,\mathrm{MeV}$), and features a collimator and moving table that allows for control of the beam size as well as targeting of specific components. The purpose of the tests were to evaluate early versions of the main FPGA design, and also the auxiliary design which now included blind scrubbing of the main FPGA. The main FPGA design was protected with TMR, but the auxiliary one was not at this point. The beam tests were run in different configurations, with the beams targeting different combinations of the two FPGAs and the flash memory of the RU board. CRC errors in the configuration data stream were counted[16]. The previous two beam tests had demonstrated that the cross-section for SEUs causing $0 \rightarrow 1$ transitions in the flash memory were significantly higher than for $1 \rightarrow 0$ transitions. Since the configuration and scrubbing bit-files for the main FPGA had a ratio of 0 to 1 bits of approximately 20:1 and 50:1, respectively, the images were inverted to reduce their cross-sections [83]. Tests with irradiation of the main FPGA alone accumulated a fluence of $1.16 \times 10^{10}\,\mathrm{cm^{-2}}$. No errors were detected indicating that the mitigation in the main FPGA design combined with the scrubber worked quite well, and the main FPGA was operated for the entire duration. The tests with irradiation of both FPGAs had a total fluence of $3.389 \times 10^{10}\,\mathrm{cm^{-2}}$, with 11 CRC errors recorded. And the tests with all three components had a fluence of $1.175\,\mathrm{cm^{-2}}$, with 34 CRC errors recorded. Irradiation of the flash memory did not appear to have an affect on the number of CRC errors, as the relative number of CRC errors per fluence is about the same for both of these tests. The

[15]Due to the nature of the CHARM facility being a parasitic user of the PS' beams, it was not possible to extract the RU for several days. By then it was not possible to conclusively determine which component had failed.

[16]This is a measure of the number of failed configuration attempts of the main FPGA, since the FPGA will reject the configuration if the CRC is bad.

conclusion was that the unmitigated design for the auxiliary FPGA was the source of the CRC errors [81].

Single Event Upset Rate for the Main FPGA. Based on the beam tests, the expected SEU rate for the configuration memory of the main FPGA has been estimated to 0.0004 Hz (every forty minutes on average for **one** RU) [68]. Without mitigation and scrubbing, this would quickly lead to failures in the ITS readout chain of 192 RUs. But a scrubbing rate of 0.58 Hz has been achieved with the external scrubber, which is three orders of magnitude faster than the upset rate [68]. Combined with the TMR of the main FPGA design, the estimated MTTF of the data path ranges from 4 hours for the entire OL, to 23 hours for the entire IL [84]. And sensitive resources in the FPGA (such as clock nets and related circuits) are not expected to be statistically affected by SEUs in a 10 hour run [85].

5.6 Commissioning

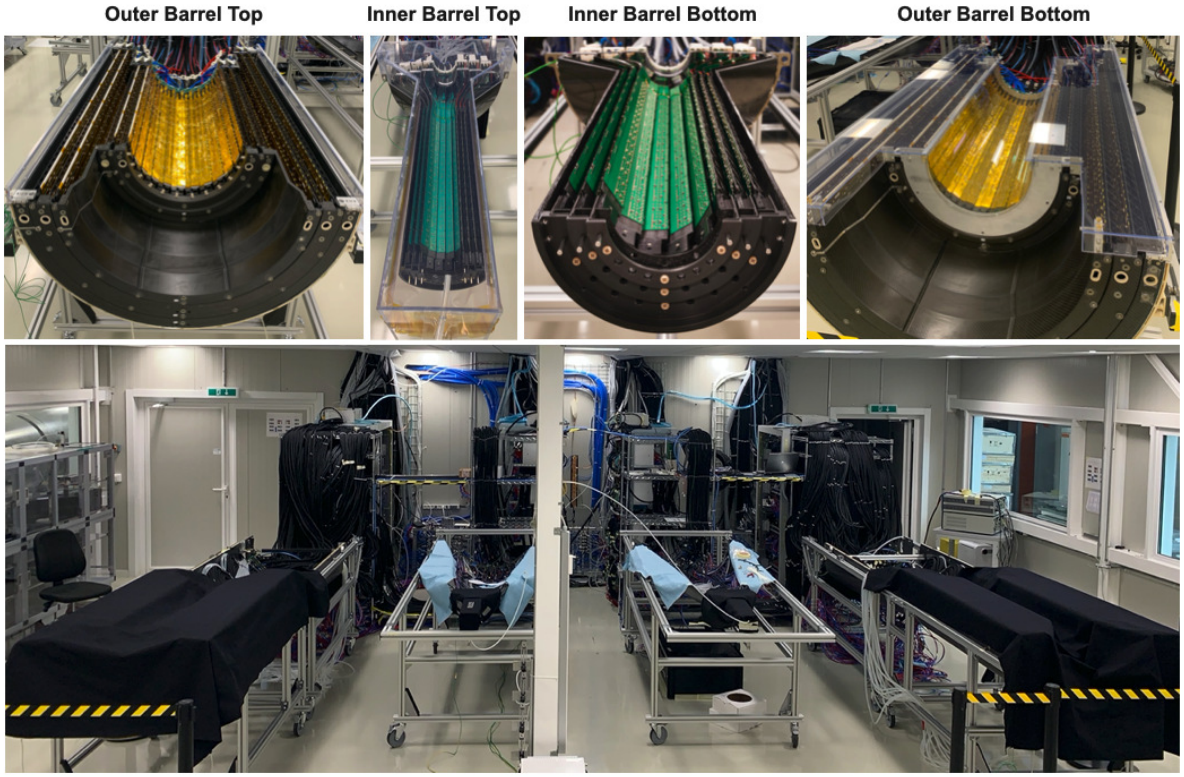


FIGURE 5.9: ITS commissioning in CERN building 167 clean room [86]. The picture at the top shows the racks housing the readout electronics. The pictures at the bottom shows the assembled half-barrels of the detector.

Commissioning of the new ITS detector took place in a clean room in building 167 at the CERN Meyrin site. The full detector was constructed there from pre-assembled modules[17]. Figure 5.9 shows pictures of the assembled half-barrels of the detector in the clean room, as well as the readout electronics. The half-barrels were connected to the readout electronics and were in continuous operation for over a year from May 2019 [86] to December 2020.

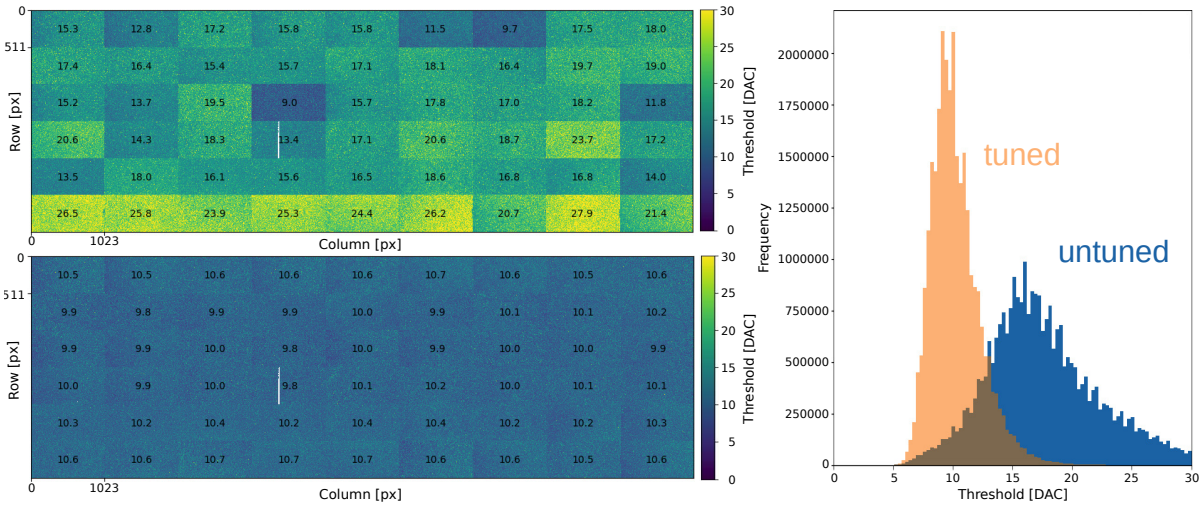


FIGURE 5.10: Threshold tuning for half-layer 0 [88].

The detector was primarily taking cosmic data in this period, but various test and calibration runs were also performed, such as tuning of the threshold for the pixel sensor chips to achieve a uniform threshold for the entire detector. The threshold is tuned by adjusting a DAC in the ALPIDE chip that configures the threshold in terms of the elementary charge e. Figure 5.10 shows a map of the threshold before and after tuning, as well as the threshold distribution. The tuned threshold is around $10e$ with a relatively small deviation.

The commissioning period was also actively used to develop, test, and refine the FPGA designs for the RU, as well as data taking and DCS software for the ITS.

[17] IB-HICs were produced at CERN. OB-HICs at Bari (IT), Liverpool (UK), Pusan/Inha (KR), Strasbourg (FR), and Wuhan (CN). OB staves at Berkeley (US), Daresbury (UK), Frascati (IT), Nikhef (NL), and Torino (IT) [87].

Chapter 6

Simulation Model of the ITS Upgrade and ALPIDE

This chapter describes a simulation model of the ALPIDE chip and ITS detector that was developed in C++ with the SystemC library. The objective of this project was initially to investigate the effects of busy chips on the data collection, and to determine whether a dedicated system to handle busy signals was necessary for the ITS. Eventually the simulation model was used to estimate data rates and readout efficiency of the ITS, and also found applications for other detectors that use the ALPIDE chip.

The ITS RU is a complex device designed to meet a long list of specifications which include operation under a wide range of configurations. Its responsibilities include, but are not limited to, data readout, trigger distribution, and configuration of the sensor chips in one stave. There are three different stave configurations in the ITS, and the number of sensor chips, data and control links, and expected occupancy differ between the different staves and layers. And to complicate things further, the experiment will run in both Pb–Pb and pp, at different interaction rates, and several possible trigger schemes are foreseen. The RU has to work with any of these combinations of staves and trigger schemes.

The specification and design of the RU hardware, as well as the FPGA designs, required detailed knowledge of quantities such as what data rates to expect from the sensor chips. Another question was how the RUs should handle sensor chips which report that they are busy. Should there be mechanisms in the RU's FPGA design to hold back triggers in that case? And should it be coordinated between the RUs with dedicated hardware for a Busy Unit (BU)? And for a given set of simulation parameters, how many events are lost due to busy violations in the triggered mode of the chip, compared to the flush mechanism in continuous mode?

The need to estimate these quantities called for a simulation of the ITS readout

chain. In principle, it may have been possible to model the readout using statistical methods. For example, during the design phase of the ALPIDE a statistical method was used to estimate how many event buffers were necessary [29], [89]. But this was a problem with a rather limited scope. Given the complexity of the entire ITS readout chain and the number of variables involved, a dedicated event-based simulation of the ITS seemed like a more reasonable approach to estimate the numbers used as a basis in the RU design phase.

6.1 Simulation Challenges

A simulation typically involves the following steps:

1. Create a model of the system to simulate
2. Apply input data (stimuli) to the inputs of the model
3. Observe the outputs of the model

How accurate the simulation is depends both on how accurately the model represents the real system, as well as to which degree the input data represents real-world conditions. But simulating a system with arbitrary accuracy is often unnecessary or even impossible to achieve. Higher accuracy typically translates into much longer simulation times which can often be a show-stopper[1]. A large part of the challenge then is making the right trade-offs between accuracy and simulation speed when the model of the system is defined. Typically there are some parts of the system that play a dominant role in shaping the system's outputs. One would want to model those with a higher degree of accuracy, but consider a lower degree of accuracy for parts that play a negligible role, or even omitting them entirely.

6.1.1 Requirements for the Simulation Model

Roughly speaking the simulation must contain a model of the ITS detector itself, the trigger distribution, and the RU. For the trigger distribution it is sufficient to model the delays in the distribution path. The RU can also be modeled in simple terms, since this is not an actual simulation of the readout chain in the RU's FPGA design, but primarily a simulation to quantify data rates from the detector, as well as data loss via the busy mechanisms of the ALPIDE chip.

[1]For example, the full Verilog RTL design of the ALPIDE chip is available, but it was much too slow for use in the simulations presented in this thesis.

A model of the ALPIDE sensor chip is essential for these simulations considering that the ITS detector is made exclusively of those chips in different groupings. And most importantly, the model of the ALPIDE chip must include: the logic for triggering and handling of the MEB; readout from MEB regions into region FIFOs; readout and framing of data from the region FIFOs; clustering of neighboring hits into *DATA LONG* words; data transmission and busy signaling for both IB and OB chips; IB mode, OB master and slave modes. It is also worth noting that the readout logic of the chip is agnostic to what type of particle caused a pixel hit, its energy, and what the track through the detector looks like. This fact can be exploited when the pixel front-end is modeled. The pixel front-end stages before the discriminator can be simplified or omitted since the discriminated pixel data, i.e. a pixel was either hit or not, is essentially all that matters for the readout logic.

Simulation Models of the ALPIDE Chip. A reasonable starting point would be to look at existing models of the ALPIDE. Early estimations of data rates for the ALPIDE chips and RUs were performed by collaborators in the ITS project. They used a *SystemC*-based simulation model of the early prototypes of the ALPIDE chip. SystemC is a library that allows event-driven HDL-like simulations to be modeled with C++[2]. Based on the results of these SystemC-based simulations of the ALPIDE, it was decided that three GBT links would suffice for the RU design. But several important aspects of the ALPIDE were not included in the previous simulation model, which made it not suitable for further use[3].

Another option is to base the simulations on the Verilog RTL design of the ALPIDE chip. Simulations of around 10 000 Pb–Pb events were performed at some point with one instance of the full Verilog design. But these simulations took around 24 hours each, and this long simulation time makes the full design not feasible for simulations with many ALPIDE chips and a greater number of events. A light-weight version of the Verilog design is also available, but it is primarily intended for verification of the RU FPGA design. It features the control bus and registers, the serial data interface and a random data generator. But these features are not relevant for the simulations that are discussed here. And more importantly, it is missing the pixel front-end, MEB and readout logic. Hence, it could not be used for these simulations.

A final possibility would have been to convert the full Verilog design to SystemC using *Verilator* [90]. A Verilator conversion may offer some increase in simulation

[2]SystemC models can be compiled and executed as a stand-alone binary, or used as a module in an HDL simulator like ModelSim. There is also a synthesizable subset of SystemC for FPGA or digital chip design.

[3]Most notably: the readout from regions are not implemented in sufficient detail, busy signaling is not fully implemented, and OB modes are not fully implemented.

speed, but it would probably still be necessary to shave away unnecessary features of the design.

6.1.2 Input Stimuli

To achieve simulation results that reflects the real ITS detector in operation the simulation model must receive input data that resembles real events from collisions in the experiment. This requires an understanding of collisions in the LHC, such as the number of pixel hits, how they are distributed in time, and triggering of the detector. The following paragraphs will discuss the most relevant effects to consider.

LHC Fill Patterns and Collisions

The two rings of the LHC are filled with discrete *bunches* of particles, with each bunch separated in time by 25 ns. An entire LHC ring is divided into a total of 2808 bunches[4]. However, not all bunches are filled; there are specific filling patterns that are used when running the LHC, and some bunches are empty. And for Pb–Pb there are three empty bunches between each Pb bunch, making the effective bunch spacing 100 ns.

In addition to empty bunches, there are also certain **filled** bunches that will not collide in the IPs for ALICE and LHCb, as shown by Hostettler et al [91]. This is due to a combination of the asymmetric position of the IPs, and so-called "PACMAN" effects where the first and last bunch in a train experiences a different net "kick" in the accelerator compared to the bunches in the middle of the train.

The number of collisions during a time frame are approximately *Poisson* distributed [13]. The amount of particles that originate from a collision is known as the *multiplicity* of the event. Most collisions are peripheral and have a low multiplicity. Central collisions[5] can have very high multiplicities but are relatively rare. This can be seen in fig. 6.1, which shows a distribution of charged particle multiplicity in the ALICE TPC for Pb–Pb events. The figure has not been corrected for the acceptance of the TPC[6], so it should be noted that this is not the full event multiplicity. The shape of the distribution will be the same for events recorded in the ITS when accounting for the difference in acceptance. The average hit density per chip, shown earlier in table 2.1 for Pb–Pb, differ between the layers and is significantly higher in the innermost layers due to the proximity to the collision point. The hit density also depends on *pseudorapidity*, as shown in fig. 6.2.

[4] There are 3564 25 ns slots in the orbit, but only 2808 of them are bunch crossings that may contain particle bunches.

[5] "Head-on collisions."

[6] Any detector such as the TPC or ITS will cover a limited range of pseudorapidity, and hence their acceptance will never be 100%.

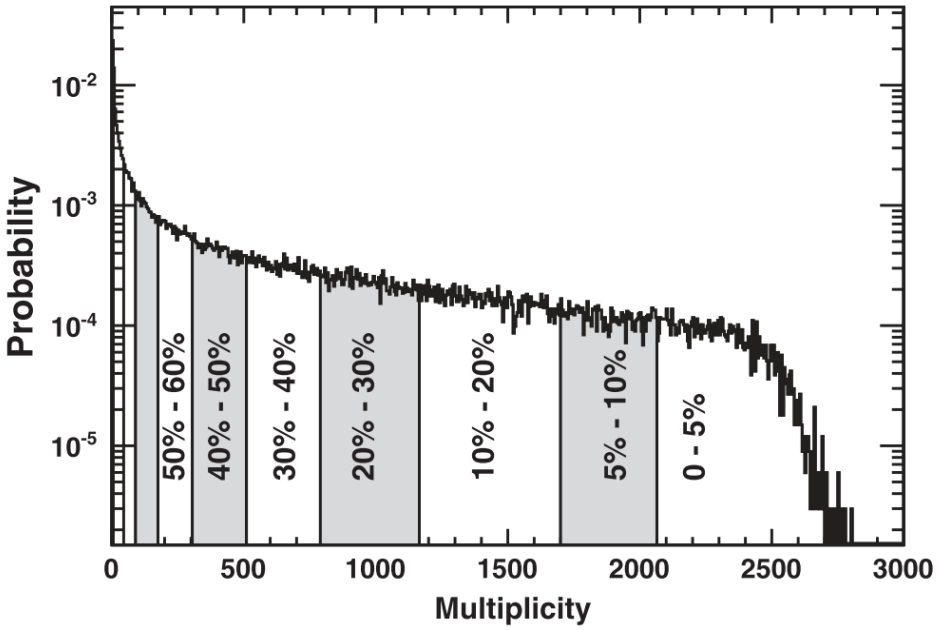


FIGURE 6.1: Uncorrected multiplicity distribution of charged particles in the TPC [92].

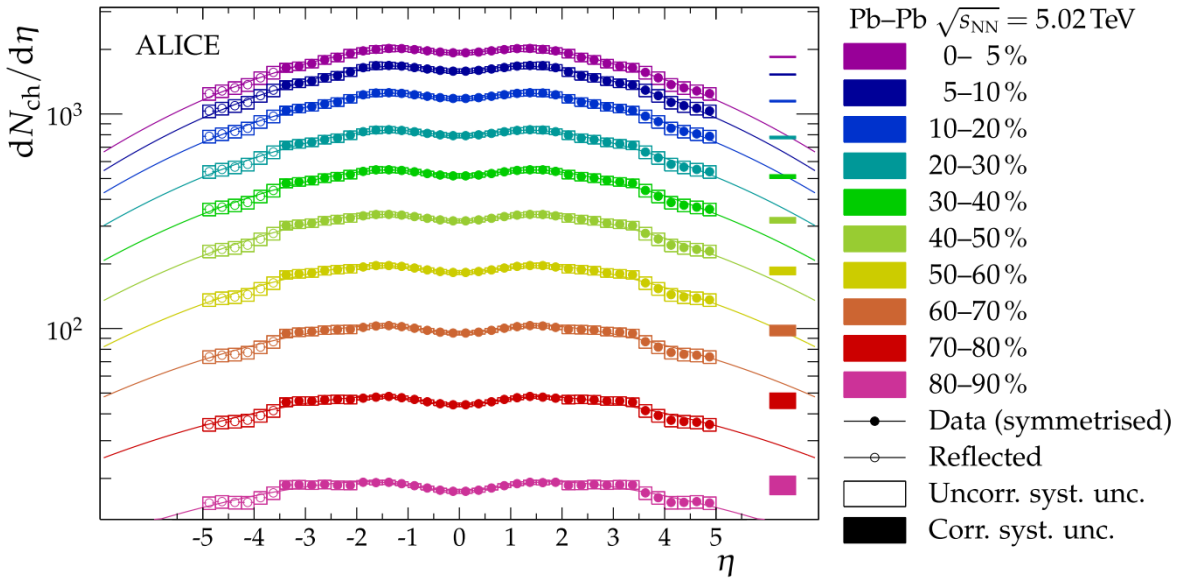
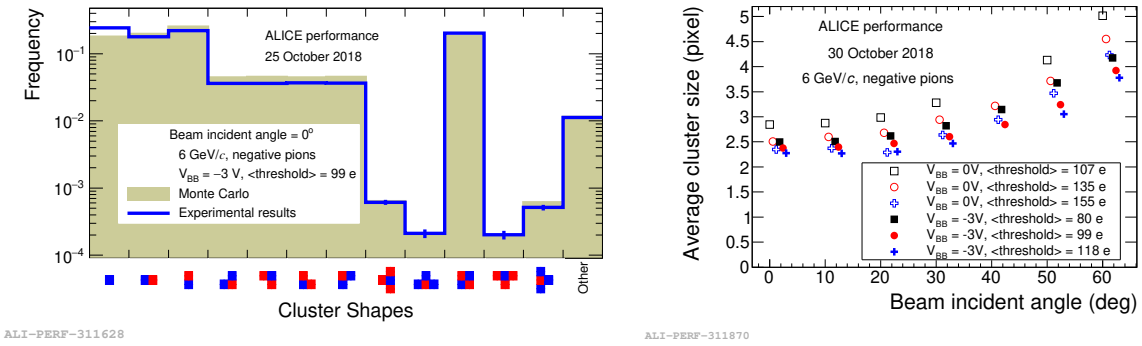


FIGURE 6.2: Charged–particle pseudorapidity density for ten centrality classes over a broad η range in Pb–Pb collisions at $\sqrt{s_{NN}} = 5.02$TeV [93].

Pixel Clusters

The charge that is liberated from particles hitting the sensitive areas of the chip is usually not confined to the geometry of the incident pixel. Typically there are several neighboring pixels that also collect sufficient charge to go over threshold and register a hit. Consequently, there is generally a cluster of pixels associated with each particle hit. Some common cluster shapes and their occurrences are shown in fig. 6.3A. The

size of a cluster depends on a number of factors: the amount of liberated charge, which depends on the mass and energy of the incident particle; the incident angle of the incident particle, as well as the configurable charge threshold in the chip, as shown in fig. 6.3B.



(A) (B)

FIGURE 6.3: ALPIDE cluster shapes and sizes. 6.3A: Relative occurence of different cluster shapes. 6.3B: Average cluster size versus incident angle. Figure credits: Kushpil, Krizek, and Isakov [94]. © 2019 IEEE.

Triggering and Events in the Sensor Chips

The FIT detector, which will be introduced in ALICE in Run 3, is responsible for fast detection of events. It is used for triggered operation of ALICE. The CTP receives signals from the FIT detector, determines if the experiment should trigger on the event, and distributes trigger signals to the other detectors. The ITS readout electronics receives the trigger 980 ns after the event occured and it takes an additional 250 ns for the trigger to reach the sensor chips. In contrast, when the ALICE is running in continuous mode, which is the alternative triggering scheme, trigger signals are sent periodically to the sensor chips.

A *strobe window* starts with each trigger the sensor chip receives. The length of the strobe is configurable, but triggered mode is intended to be used with a short strobe[7], whereas in continuous mode the strobe is typically about the same length as the trigger period[8]. Refer to figs. 2.4 to 2.6, and the corresponding discussion in chapter 2 for more details. From the perspective of the simulation model it is important that all viable triggering schemes can be simulated.

[7] Order of around 100 ns.

[8] Several microseconds typically, with a small gap of around 100 ns between each strobe. The TDR specifies a maximum strobe length of 45 µs at 50 kHz Pb–Pb to avoid a loss of reconstruction efficiency [13].

6.2 Implementation of the Simulations

None of the existing models for the ALPIDE were suitable for the simulations described in this chapter, which led to the development of an entirely new simulation. The new simulation was also implemented in C++ using the SystemC library and features a new model of the ALPIDE, but it and was inspired by several ideas from the older SystemC-based simulation. It was required to simulate a much larger number of events than what could be simulated with the HDL design for the ALPIDE, which implies that it would have to be substantially faster than the HDL design. At the same time, it should provide an accurate representation of the readout logic in the ALPIDE chip.

An overview of the full simulation model is shown in fig. 6.4. Several instances of the ALPIDE model are configured and connected to create staves for the different layers, and the staves connect to RUs. A limited amount of logic is implemented in the RUs; they primarily distribute triggers to their staves and aggregate data from the sensor chips.

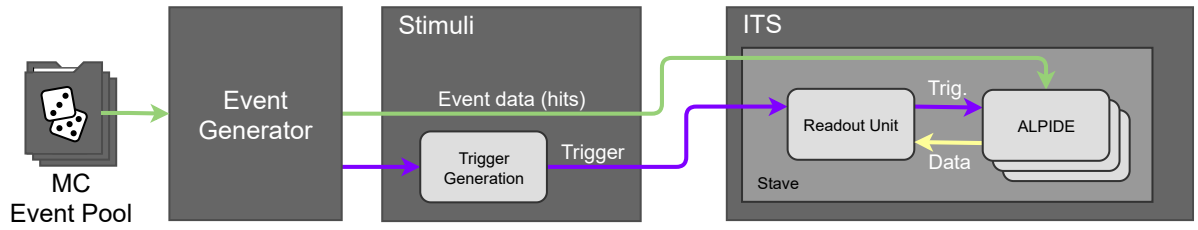


FIGURE 6.4: Overview of SystemC simulation model for ITS.

An event generator picks random events from a pool of Monte Carlo (MC) events and distributes them in time to simulate the collisions in the experiment[9]. The stimuli object is the top-level object in the simulation. It is responsible for the distribution of the pixel hit data from the event generator to the pixel front-end of the ALPIDEs chips, as well as generation of trigger signals.

The simulation runs until the desired number of events have been simulated. At the end of the simulation, quantities such as busy events, data word counts, and trigger information are stored to disk. The data can then be analyzed later and used to estimate performance figures such as readout efficiency and data rates.

Different aspects of the simulation model will be discussed in more detail in the following sections.

[9]It is technically not an event generator when it does not generate its own events. Although, it can generate events with random hits to achieve a certain occupancy, but this is not a proper MC simulation of the detector.

6.2.1 Event Generation

The event generator can generate two types of events: interaction events (collisions) that the simulation can trigger on, and continuous background events. Interaction events are distributed in time with an exponential distribution. As mentioned earlier, the number of collisions within a time frame is Poisson distributed, and the time between events in a Poisson distribution follows an exponential distribution. The mean of the exponential distribution is $1/\lambda$, and λ is configured to achieve the average interaction rate, i.e. $\lambda = 1/(interaction\ rate)$.

A signal from the event generator indicates when a new event is available. The *Stimuli* module will react to this signal by taking the event data and placing it in the front-ends of the pixel chips. The signal can also act as a minimum-bias trigger.

Fill patterns and other bunch crossing effects are not taken into consideration by the event generator. Instead the time of an event is computed and aligned to any 25 ns interval, or bunch crossing, irrespective of fill patterns. This also applies to Pb–Pb simulations, although the effective bunch spacing is larger for Pb–Pb (see section 6.1.2). These effects are not expected to have a large impact on the results considering the relatively low interaction rates in ALICE as well as the long pulse shaping time of the ALPIDE chip's front-end.

Monte Carlo Event Input

To create realistic input events for the sensor chips the event generator can use pregenerated event data from MC simulations. *AliRoot*, the analysis framework of the ALICE experiment, has accurate definitions of the experiment's geometry and integrates several tools for MC simulations: the *Hijing* and *Pythia* event generators, for Pb–Pb and pp, respectively; a custom event generator for the so-called QED background associated with Pb–Pb interactions [20], [95]; and *Geant4* for simulation of the interactions of particles with the detector material. A simulation testbench for the ITS upgrade [96] is also available in the framework. It generates events for pp or Pb–Pb and simulates the ALPIDE sensors' response in the detector. This testbench was used to generate event data for the SystemC simulations. Data were extracted from each simulated event and stored to disk, one file per event. The extracted event data consists solely of discrete pixel hit coordinates. Each coordinate fully specifies the coordinates in the pixel matrix of a chip, as well as chip ID, module ID, and layer ID. The MC simulation also performs clustering for each particle hit, and the event data includes the additional pixel hits per cluster.

When the event generator of the SystemC model is simulating a collision it picks a random event from this pool of MC events for pp or Pb–Pb. The same applies to the

QED background except that input of these events happen at a continuous rate. Pixel noise is not included in the event data and is also not simulated by the event generator. However, in principle the pixel noise of the ALPIDE could have been simulated using the continuous background process of the event generator, in the same fashion as for the QED background.

In principle, it may have been possible to use real ALICE events as input to the simulation. But these events would have been from previous runs and captured with the old ITS, so they would have had to be adapted for the upgraded detector. With that in consideration, and given the fact that a simulation framework was already available for the ITS upgrade, using the data from the MC simulations as input was the most reasonable approach.

Random Event Generation

As an alternative to MC-input, the event generator also includes a "toy event generator" which can generate events with random pixel hits for the sensor chips in the detector. There is no physics behind these events, the pixel hits are uniformly distributed across the sensor chips in a layer. The multiplicity of the generated events is also randomly generated, by picking from a discrete distribution, and the distribution is scaled to achieve the desired average hit density. The distribution of charged particle multiplicity used for Pb–Pb simulations is shown in fig. 6.1. The hit density is specified for each layer, typically with the values specified in table 2.1 for Pb–Pb.

This approach provides the sensor chips in the simulation with a work load which resembles what the detector would experience in the LHC. However, it has several shortcomings, primarily because the pixel hits are distributed uniformly; the hit density should depend on pseudorapidity, as seen in fig. 6.2; and there are no showers of particles, so each sensor chip in the same layer experiences roughly the same occupancy per event. Nevertheless, the "toy event generator" has proved useful for testing and debugging, and was also used for the first estimates of readout efficiency of a single chip, shown in fig. 6.5. Those estimates were in good agreement with estimates from the ITS TDR [13].

Pixel Noise

The hit density data in table 2.1 comes from the Technical Design Report (TDR) for the ITS upgrade, and is a bit dated by now. It was simulated with an expected noise of 10^{-5} fake hits/pixel, which has proven to be a very pessimistic estimate. The fake hit rate depends on the threshold settings, and characterization of the ALPIDE chip that were performed after the TDR was published have shown that a fake hit rate of

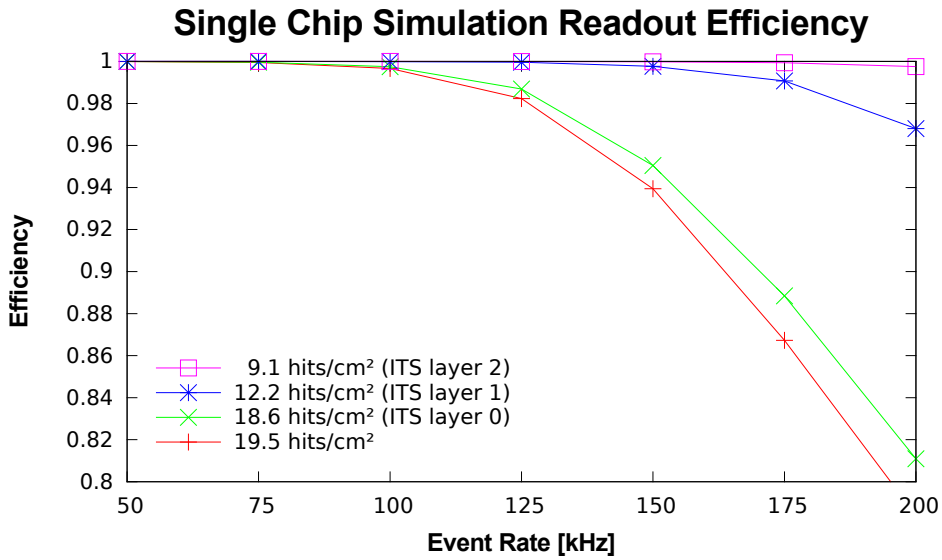


FIGURE 6.5: Single chip readout efficiency for the innermost layer of the ITS. Simulated using the "toy event generator" [97].

around 10^{-9} fake hits/pixel [94] can be achieved without sacrificing detection efficiency. Consequently the hit densities will be significantly lower than indicated in the table, especially in the MLs and OLs where the numbers in table 2.1 are to a large degree dominated by pixel noise. Pixel noise was not included in the simulation model for that reason.

6.2.2 Stimuli and Trigger Distribution

The Stimuli block in fig. 6.4 connects the event generator and detector together and provides the ALPIDE chips of the detector with the event input data. It also generates trigger signals based on the time of the event, when used in minimum-bias mode, or generates the continuous triggers for continuous/periodic mode. The trigger signals are delayed by a configurable amount before reaching the detector. This is typically set up to mimic the trigger delay in the system (see section 2.6.1).

6.2.3 ALPIDE Model

At the heart of the SystemC simulation model is the class for the ALPIDE. The structure of the *Alpide* class was designed to have a structure that resembles the original chip hardware, as shown in the simplified UML class diagram in fig. 6.7. The model has a relatively detailed implementation of the readout logic of the ALPIDE. It comprises most of the modules described in appendix B. Specifically, it includes the FSMs

and logic for the Region Readout Unit (RRU) and Top Readout Unit (TRU), the region FIFOs and frame FIFOs, busy FIFO, and master and slave communication for OB chips. The pixel matrix and MEB are fully modeled, with readout and clustering by the priority encoder and RRUs.

Other less critical aspects were simplified or omitted in its entirety to achieve faster simulation speeds. A simpler version of the Frame and ReadOut Management Unit (FROMU) was implemented. The Data Transmission Unit (DTU) and 8b10 encoding of the data stream was omitted. Instead of serializing the data the model transmits three full bytes per 40 MHz clock cycle to achieve 960 Mbps for IB mode, and one byte per cycle to achieve 320 Mbps for OB mode. The data words transmitted on the data link are fully implemented as shown in fig. B.5[10]. Other features that were not relevant for the simulation, such as the DACs and ADCs, were not implemented. Like the previous simulation model, this work did not include an analog front-end. The pixel hits were modeled as a rectangular pulse, with a fixed dead time and active time[11].

Pixel Hits and Pulse Shaping

Hits in the ALPIDE chip give rise to an analog voltage pulse, which is discriminated with a comparator and configurable threshold, as seen in fig. 2.4. The duration of time that the analog pulse is over the comparator's threshold value is commonly referred to as the ToT, and the time leading up to it is the rise time of the pulse, as shown in the top graph of fig. 6.6.

The analog pulse is not modeled in the SystemC model of the ALPIDE. Instead, only the discriminated digital pulse is modeled, as seen in the bottom graph of fig. 6.6. It is characterized by the ToT and by the rise time, which for the SystemC model is referred to as the active and inactive time of the pixel.

The active and inactive times are configurable settings in the simulation model, but they are constant for all pixel hits during a simulation. The timing information for a pixel hit is actually configured when the hit is created by the event generator. In principle, the simulation model could have been improved with a variable pixel active time (i.e. ToT) to model the "strength" (collected charge) of a hit, and the "fuzzy border" of a pixel cluster could also have been modeled in this way[12]. However, it was considered an acceptable trade-off to use one value for the ToT for all pixel hits, and this also made the implementation simpler.

[10] Comma words are not included since 8b10-encoding is not implemented.
[11] Time over threshold.
[12] Pixels further out from the center of the hit should generally collect less charge, and die out faster.

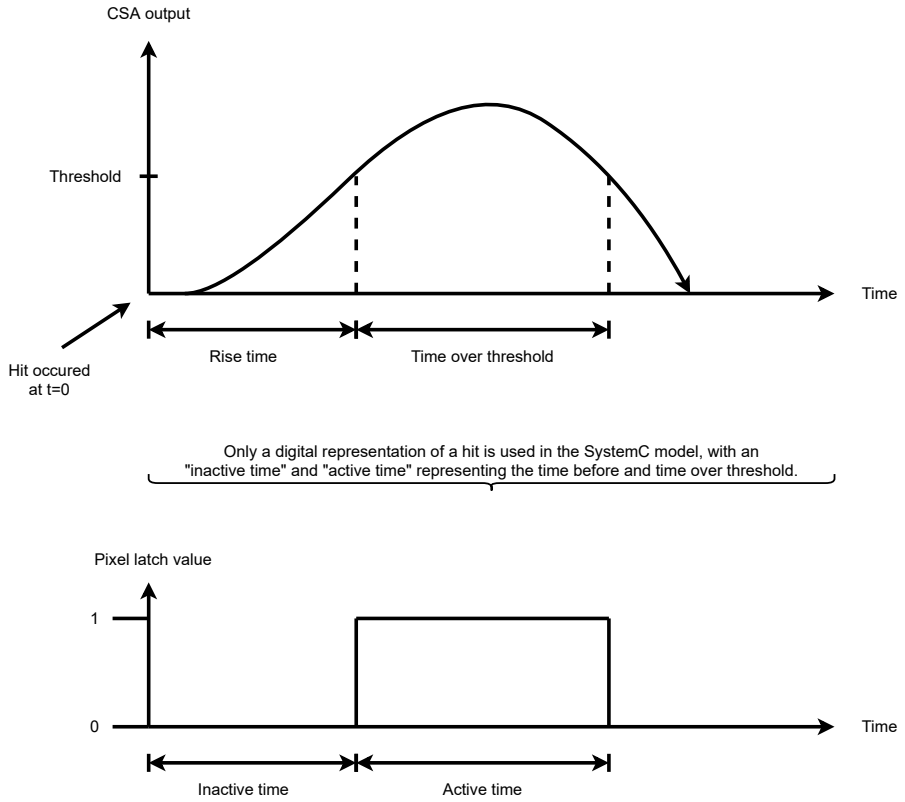


FIGURE 6.6: Illustration of pulse shape output from the preamplifier in the analog front-end of the ALPIDE chip, and the digital pulse output from the comparator. The SystemC model of the chip only models the digital output.

Standard C++ Classes for Pixel Input

A pixel hit is stored in the *PixelHit* class, which contains the coordinates of the hit and the time it occured. The *PixelFrontEnd* class of the model take these pixel hits as inputs. A pixel hit is considered to be "expired" when the simulation time has progressed beyond the active time of the pixel hit. The pixel hits are stored in a queue in the front-end, and the expired pixel hits are removed from the queue as the simulation progresses in time. A simple algorithm removes pixel hits starting with the oldest entry in the queue, and stops at the first hit which has not expired yet. The algorithm requires that all pixel hits have the same active time and dead time. A more advanced implementation of the simulations could include variable active and dead times – if some changes are made to the processing of the pixel hit queue[13].

A strobe window starts when a trigger is received by the ALPIDE model. Pixel hits are retained in the PixelFrontEnd while the strobe is active. Pixels that were active during the strobe enter the *PixelMatrix*. This is performed at the end of the strobe, by "gliding" the strobe window over the PixelFrontEnd's pixel queue. An event entry is

[13]One cannot assume that expired and active pixel hits are ordered in the queue in this case.

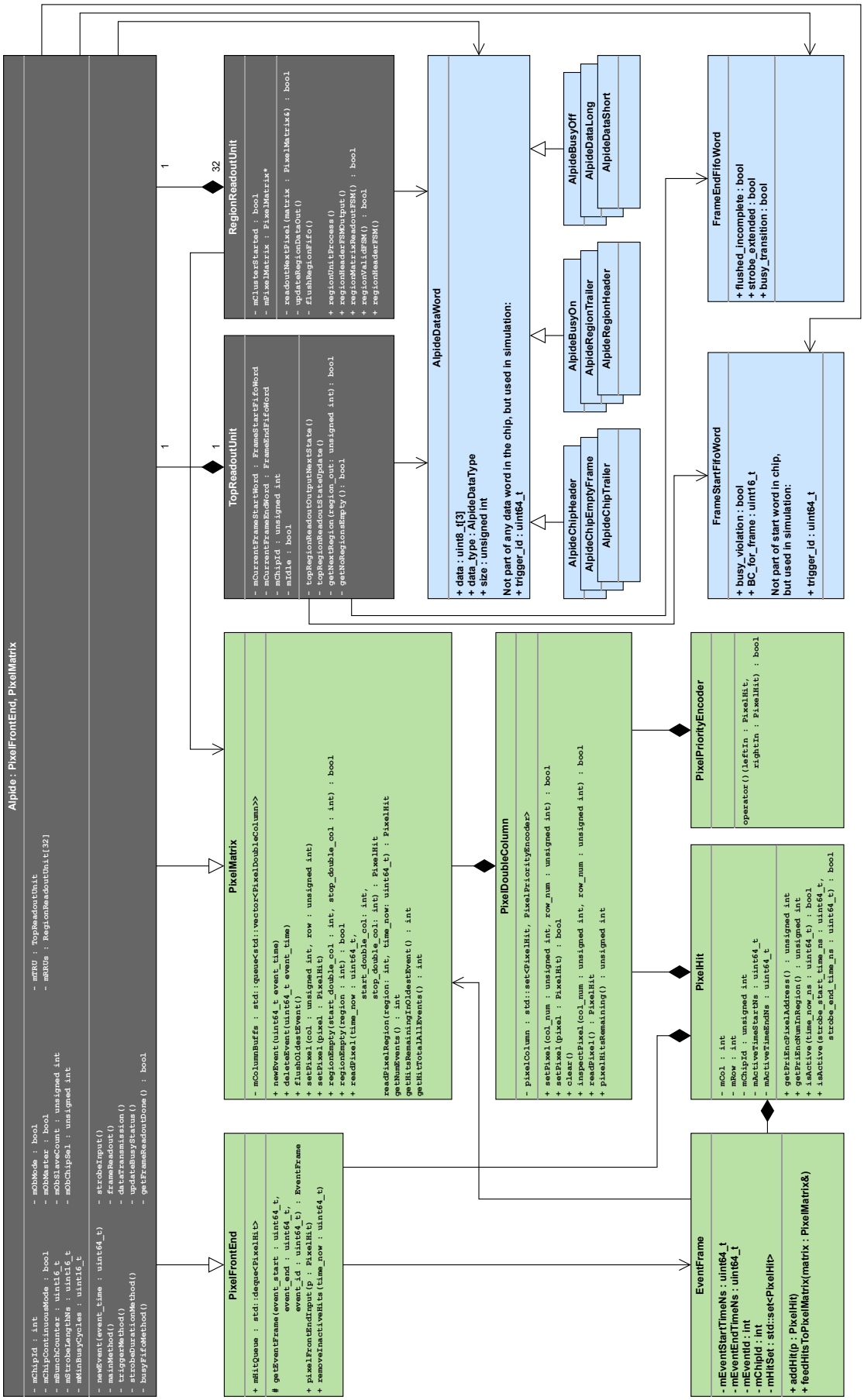


FIGURE 6.7: Simplified Unified Modeling Language (UML) class diagram of Alpide class in SystemC simulation model. The classes in green and blue are pure C++ classes, of which the ones in blue are data classes. The classes in dark gray are SystemC modules.

created in the PixelMatrix using the pixel hits for the strobe. The event entry consists of 512 instances of the *PixelDoubleColumn* class, and the pixel hits are given to the double column that matches its coordinates.

The PixelDoubleColumn stores the pixel hits in a **set** from the C++ standard library, and uses the *PixelPriorityEncoder* class as a sorting algorithm in the set. These two classes were heavily influenced by the implementation in the previous SystemC model for the ALPIDE [40] [14]. When the pixels are read from the set, the sorting algorithm ensures that they are read out in the same order as the Priority Encoder in the ALPIDE, which is shown in fig. B.2 of appendix B. The implementation minimizes the memory footprint of the simulation, since pixels without hits are not stored in the set, and allows the hits to be read out fast in the correct order.

SystemC Modules for Pixel Readout

A SystemC module is a C++ class that inherits from **sc_module**[15]. A SystemC module can use signals, ports, and other interfaces from the SystemC library, as well as declare methods and processes with sensitivity lists. The ports and signals can be used to connect SystemC modules together in a similar way to instances of entities or modules in VHDL and Verilog. The SystemC kernel schedules and coordinates signal updates in the simulation. It also executes methods and processes when the conditions in their sensitivity lists are met, such as, on the rising edge of a clock signal.

The classes of the ALPIDE model that were discussed up till this point were all implemented with standard C++ and did not use any SystemC constructs. The signals and interfaces of SystemC modules add complexity since the classes are not only accessed with standard methods as in normal C++. And extensive use of SystemC modules and signals can increase the execution time significantly when there are a lot of signals and processes that needs to be updated on every simulated clock cycle. As a consequence, the use of SystemC constructs in the ALPIDE model was limited to where they were strictly needed, in order to reduce complexity and achieve a faster simulation.

Figure 6.8 shows a schematic representation of the ALPIDE model, as implemented in the *Alpide* class of fig. 6.7, and outlines the most important connections between the different classes and SystemC modules[16]. The external interface to IB and OB-master instances of the model consists of: pixel input to the front-end's queue

[14] But the sorting algorithm was fixed to implement the "serpent"-like readout pattern, which was not done correctly in the previous model.

[15] Alternatively, the SystemC modules can be declared with the **SC_MODULE("name")** macro.

[16] The 40 MHz system clock was omitted from the schematic.

via the *pixelFrontEndInput* function; the SystemC-interface for *Trigger input*; and a SystemC signal for *Data out*. No other connections are needed for IB chips, but for chips in an OB module there are some additional connections. The OB-slave instances do not use the normal data out signal; the dedicated *OB slave data out* signals of the OB-slaves connect to the *Data from OB slaves* input of the OB-master. And each OB-slave output their internal busy status, and each of these busy status signals connect to the OB-master instance.

Trigger, Framing and Strobe Control. The top-level Alpide class has a simple FSM to represent the FROMU (see appendix B.5.1), shown in fig. 6.9, which handles the triggers and strobes. It signals when readout can start from an event buffer to the 32 instances of the *RegionReadoutUnit*, and also to the one instance of the *TopReadoutUnit* class. The FSMs for the TRU and RRU were implemented based on documentation from the Engineering Design Review (EDR) [98] of the ALPIDE chip, as well as the operations manual [26], and are rather accurate models of the real implementation in the ALPIDE.

RegionReadoutUnit Module. The RegionReadoutUnit implements three FSMs: the *region valid FSM*, shown in fig. 6.10; the *region header FSM* (fig. 6.11); and the *region readout and clustering FSM* (fig. 6.12).

The *region event start* signal issued by the TRU (simultaneously to all RRUs) starts readout of an event from the regions. The region valid FSM indicates to the TRU if there is still data to be read out from this region[17], for the current event. A region is not *valid* when the next word on the FIFO is a *REGION TRAILER* word, which is the data word that is used to delimit data in the region FIFO, and this indicates to the TRU that it should proceed with the next region. The *pop* signal, which is issued by the TRU at the end of an event (simultaneously to all regions), instructs the regions to "pop" the delimiting *REGION TRAILER* word.

The region header FSM is a simple two-state FSM, which essentially multiplexes between the region FIFO and a hardcoded header for the region in question. This saves a FIFO entry for the header words, which are always the same for a given region. The header comes first, assuming that the region has data to be read out (i.e. it is valid), otherwise the region is skipped entirely by the TRU (no header for empty regions).

The actual readout of pixel hits from the priority encoders is performed by the readout and clustering FSM. The hits are packed into *DATA SHORT* words for single

[17] Even if a region's FIFO is empty, it may still be *valid* if there is more data in the MEB for this region to be readout.

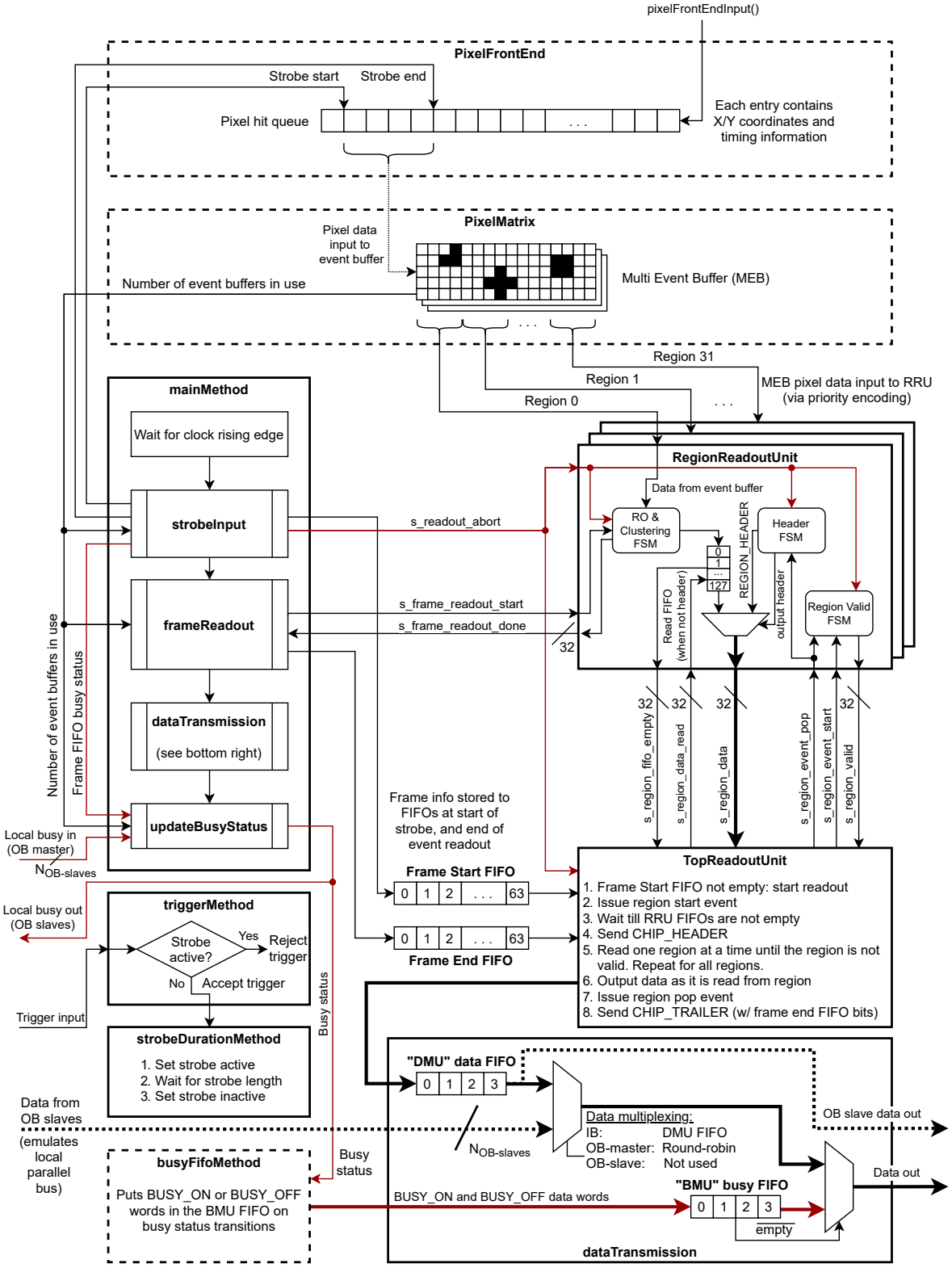


FIGURE 6.8: ALPIDE SystemC model (as implemented in the *Alpide* class).

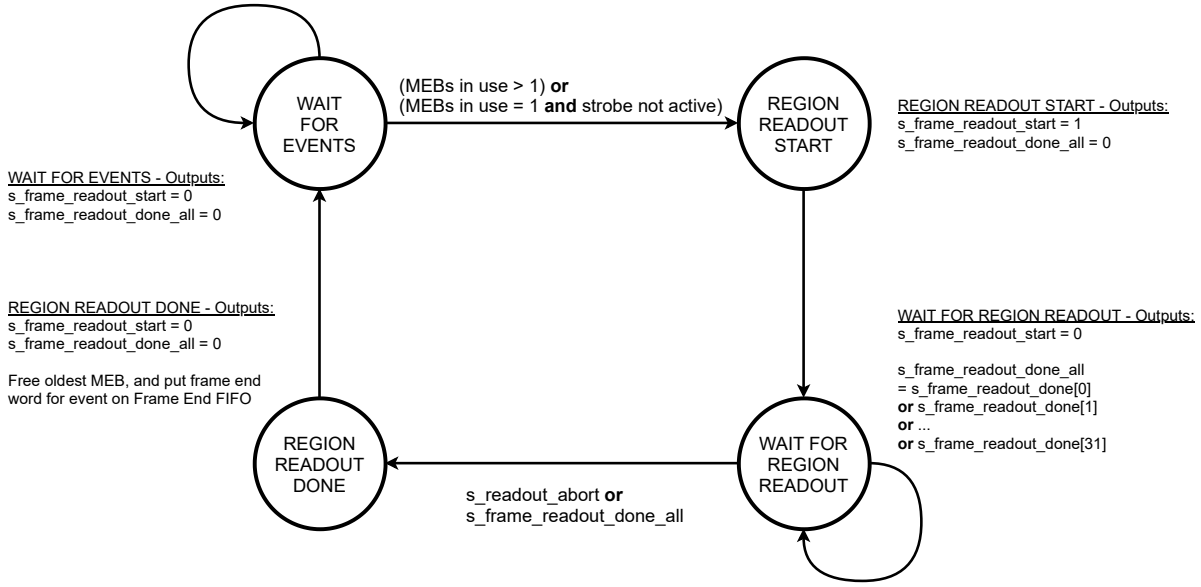


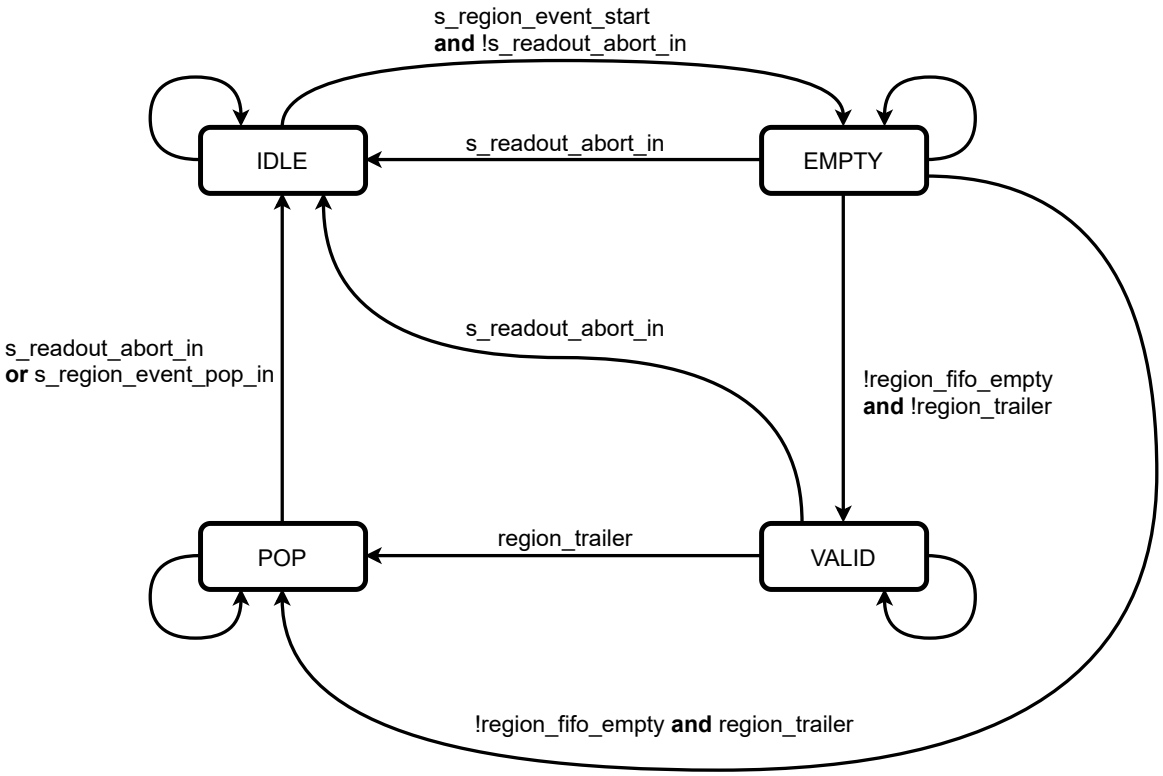
FIGURE 6.9: FROMU FSM in the SystemC simulation model.

hits, or *DATA LONG* for clusters of hits, and put in the region's FIFO. The clustering is implemented with a standard method in the class. A flowchart for the method is shown in fig. 6.13.

TopReadoutUnit Module. The *TopReadoutUnit* class implements the FSM shown in fig. 6.14, and fig. 6.8 shows how it fits into the SystemC model of the ALPIDE. In brief terms, it waits for the *frame start FIFO* to not be empty and issues a region start event, and outputs a *CHIP HEADER* word (or *CHIP EMPTY FRAME* if none of the regions contain hits, i.e. none of the regions are *valid* to begin with). It then proceeds to read out the regions in a *round-robin* fashion, starting with the first region. A region is done when it is not valid anymore, and regions that were not valid to begin with are skipped. When all of the regions have been read out, a region pop event is issued to the regions, instructing them all to pop the delimiting *REGION TRAILER* word. And finally, the event is concluded with the *CHIP TRAILER* word, and the process repeated for the next event (if any).

Clock Gating. To improve simulation time, a clock gating mechanism was implemented for the state machines of the *RegionReadoutUnit* and *TopReadoutUnit*[18]. When it is detected that there are no events to be read out in the chip, the sensitivity lists of the processes that implement the state machines are modified; execution on clock edges stops, and is re-activated when the next event is available.

[18]The real ALPIDE chip also has a clock gating feature that can be enabled for the TRU and RRU to reduce power consumption [26].



OUTPUT / STATE	IDLE	EMPTY	VALID	POP
s_region_valid_out	0	!region_fifo_empty **and** !region_trailer	!region_trailer	0

Note:
region_trailer indicates that the next output from the RRU FIFO is a REGION_TRAILER word. REGION_TRAILER is used to delimit events.

Region also valid in EMPTY state if RRU FIFO empty but Readout&Clustering FSM is just starting readout

FIGURE 6.10: Region valid FSM in the *RegionReadoutUnit* class of the SystemC simulation model.

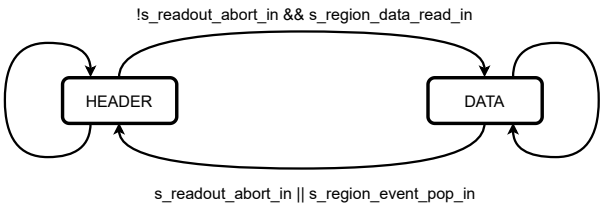
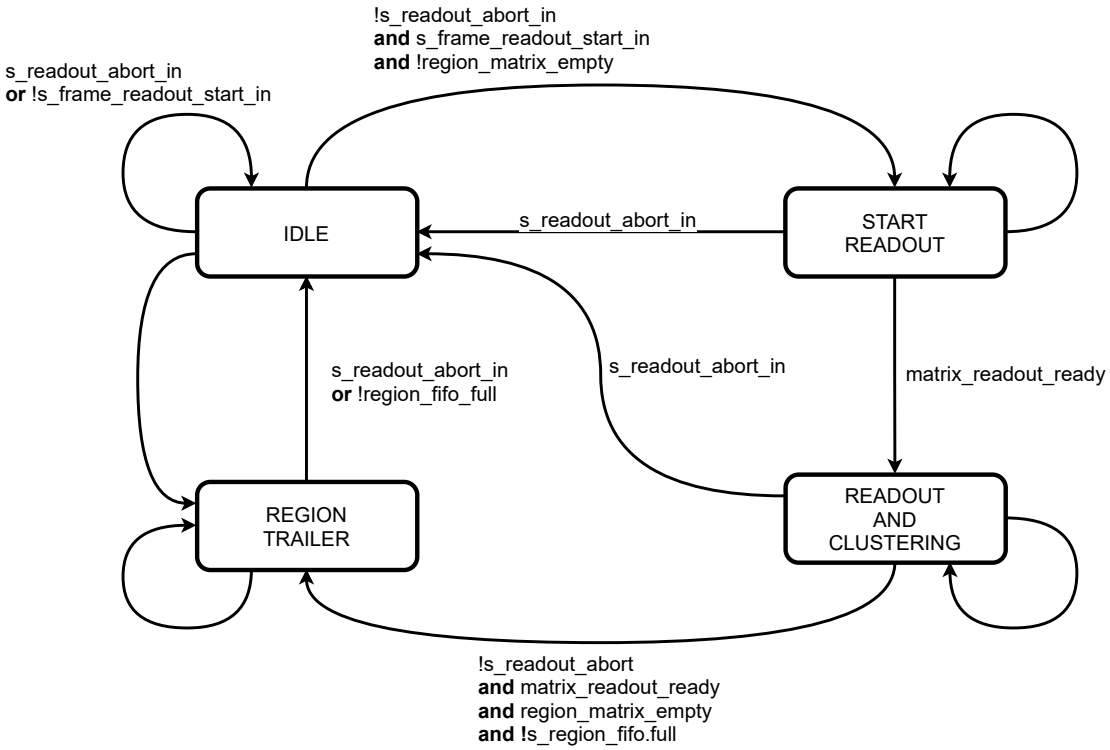


FIGURE 6.11: Region header FSM in the *RegionReadoutUnit* class of the SystemC simulation model.

6.2.4 Readout Unit Model

The main focus of the simulation is to determine data rates and readout efficiency of the ALPIDE chips. This depends solely on the input events and trigger parameters of



OUTPUT	IDLE	START READOUT	RO&CLUSTERING	REGION TRAILER
s_frame_readout_done_out	!s_frame_readout_start_in	0	0	0

The readout start input signal is normally low and pulsed high when readout can start. So the IDLE state normally outputs a high readout done signal, which goes low when readout starts.

Pixels can be read out from the pixel matrix every 2nd or 4th clock cycle, depending on readout speed settings. The matrix_readout_ready signal indicates when readout is possible. Clusters of neighboring pixel hits are collected and put on the RRU FIFO as a cluster (DATA LONG), or as DATA SHORT for single-pixel clusters. Pixel readout is handled by the READOUT AND CLUSTERING state.

FIGURE 6.12: Region readout and clustering FSM in the *RegionReadoutUnit* class of the SystemC simulation model.

the ALPIDE. As a consequence, the model of the RU has a limited scope in the simulation. Its primary function is to distribute triggers to the instances of the ALPIDE and to act as an end-point for the ALPIDE data links.

Trigger Filtering. Section 2.6.1 describes a trigger filtering feature which is used in the main FPGA of the RU. This feature is included in the RU model in the simulation and the length of the trigger filtering window, as well as the trigger delay, is configurable. If t_1 denotes the time of the first trigger, and T_{window} is the length of the window, then the trigger at t_2 is discarded if it falls within the interval $[t_1, t_1 + T_{window}]$. However, it is important that the time-walk (or rise time) of the pulses in the front-end is taken into account when the filtering window is defined. If T_{walk} is the time-walk and the second trigger arrives in the interval $[t_1 + T_{window} - T_{walk}, t_1 + T_{window}]$ then

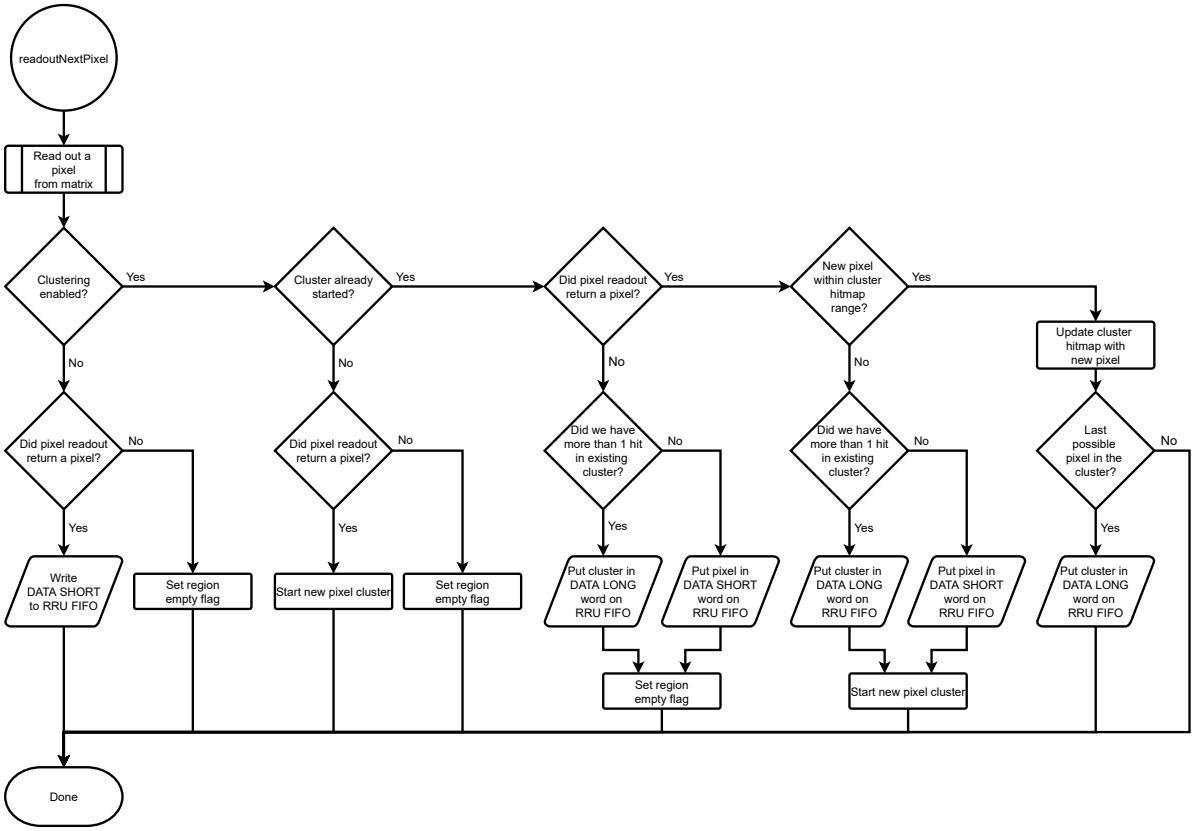


FIGURE 6.13: Flowchart of pixel readout in the *RegionReadoutUnit* class of the SystemC simulation model.

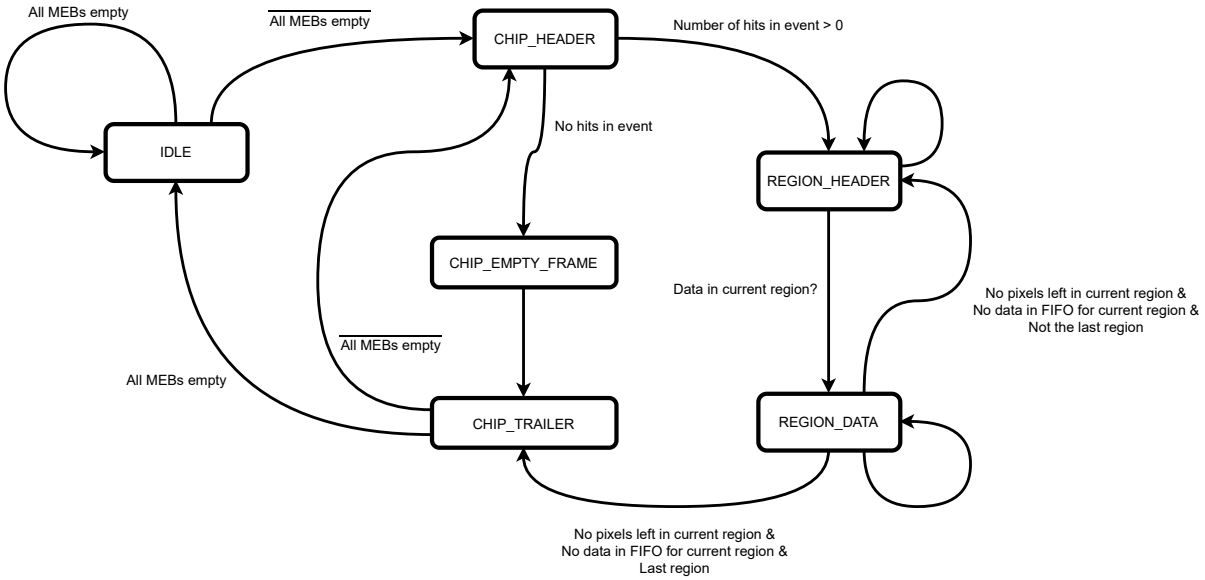
the hits of the second event have not gone over threshold yet and are not included in the strobe for the first trigger (assuming a very short strobe). And since the second trigger is discarded when it falls within this interval the hits of the second event are lost. This is illustrated in fig. 6.15.

To ensure that no data is lost because of the filtering mechanism the filtering window should be defined as $T_{window} \geq T_{delay} - T_{walk}$, where T_{delay} is the total trigger delay.

6.2.5 Top-level Detector Model

The detector model "builds" the detector by creating instances of the *Alpide* class, the stave objects that hold the chips, the instances of the RUs, and connecting them all together. The class hierarchy for simulation of the ITS is shown in fig. 6.16.

The *ITSDetector* class accepts pixel hit input for the events that come from the event generator (*EventGenITS*) via the *StimuliITS* class, as well as trigger input. Each pixel hit object contains a unique chip identifier, and the detector distributes the hits directly to the chip in question. The triggers are distributed to all RUs in the simulation.



Notes on the TRU state machine:

- If the TRU FIFO is full, nothing will be done and the state machine will "pause" until the TRU FIFO is not full again
- A 24-bit word is written to the TRU FIFO in each state (except when TRU FIFO is full)
- The states are named after the data words they write to the TRU FIFO
- The REGION_HEADER state will write IDLE words to TRU FIFO while searching for a region with data. The REGION HEADER word is written one a region with data is found
- The REGION_DATA state will write IDLE words when it is waiting for more data from the Region Readout Unit (RRU) FIFOs.

FIGURE 6.14: TRU FSM in the SystemC simulation model.

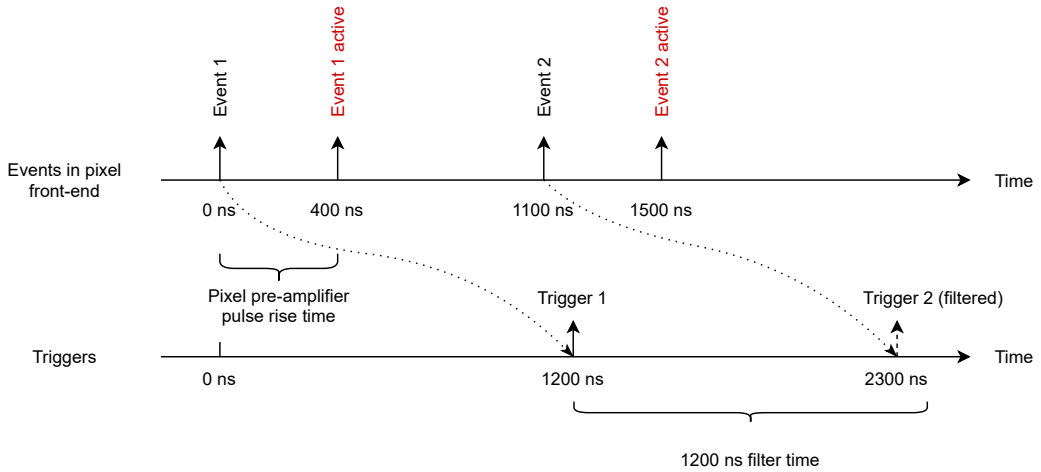


FIGURE 6.15: Example of lost event due to wrongly configured trigger filter. The trigger filter time is too short to account for the pulse shaping time of the second event, which is lost because the pixels' pre-amplifier outputs have not gone over the threshold at the time of the first trigger.

6.2.6 Simulation Settings and Output Data

There are a number of configurable settings in the simulation model, such as: The number of staves to simulate per layer, event generation parameters like the interaction rate and event types, triggering scheme and trigger filtering. A number of

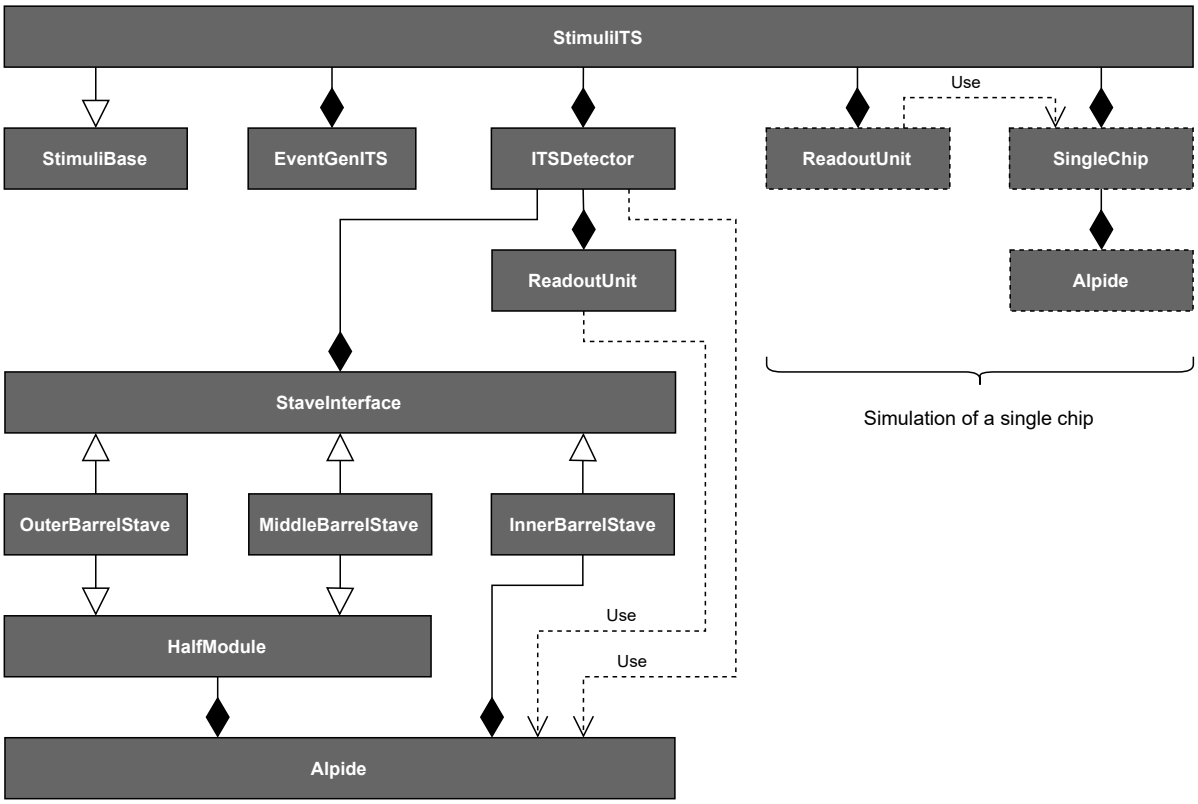


FIGURE 6.16: Simplified UML class diagram of setup for ITS in SystemC simulation model. Full detector simulation uses the *ITSDetector* class. Simulation of a single chip uses the instance of the *ReadoutUnit* and *SingleChip* classes indicated with dashed lines.

settings pertaining to the ALPIDE is also available for all the simulation types. Most importantly the choice between continuous or triggered mode, the strobe length, and inactive and active time for the pixel hits. When combined with MC input for the events (Pb–Pb or pp), the simulation model can be configured to simulate the most likely operating conditions for the ITS (from a readout perspective).

An exhaustive list of settings available in the simulation model can be found in appendix D.1.

6.3 Adaptation of the Simulation Model for FoCal and pCT

While the ALPIDE chip was designed with the ALICE ITS upgrade in mind, the chip was a natural choice for several other pixel detectors, owing to its wide range of configurable settings and high performance, as well as its current status as the state of the art among monolithic pixel sensors. The simulation model was adapted to simulate

two of these projects, namely the planned Forward Calorimeter (FoCal) for ALICE and the pCT project at UiB.

The simulation was adapted by implementing: dedicated Detector and Stimuli classes for FoCal and pCT with a different number and organization of ALPIDE chips and staves; a dedicated event generator for pCT and adaptations to the ITS event generator (*EventGenITS*) to support FoCal; support for new input data formats for FoCal and pCT.

6.3.1 FoCal

The electromagnetic calorimeter of the FoCal detector (FoCal-E) will consist of 20 layers, two of which are pixel layers [99]. The ALPIDE chip was a natural candidate for the pixel layers but simulations were necessary to establish that the ALPIDE chips can capture the data with a high efficiency. Data sets of simulated MC events, for pp and Pb–Pb, were provided by the FoCal collaboration for use with the SystemC simulations. The "Event Generator" used for the ITS simulations could easily be adapted to use this data for FoCal simulations. The pixel hit coordinates from the event data had to be mapped to individual chips in the simulation, which required an adapted Detector class for FoCal in the simulation. But a layout of sensor chips for the FoCal layers had not actually been defined at the time the simulations were performed. As a consequence, it became a part of this work to propose a layout of the sensor chips for the FoCal detector.

Layer and Stave Layout

The FoCal detector plane will cover an area of around 1 m^2. There is a rectangular gap around the beam pipe at the center of the plane with dimensions of around 8 cm $\times$ 8 cm. The suggested layout of ALPIDE chips is shown in fig. 6.17. It consists of *patches* of 15 $\times$ 6 ALPIDE chips. This layout makes the actual gap slightly larger at 8 cm wide and 9 cm tall, since the height of six ALPIDE chips adds up to 9 cm. Ideally the entire detector would be constructed of ALPIDE chips in IB mode. But that requires a dedicated data link for each chip, and it would be hard to realize a stave design with 15 data links. A large number of RUs would also be required to handle all the links. As a compromise, the suggested layout uses a combination of IB and OB chips. The IB chips surround the beam pipe where the expected occupancy is high. The occupancy falls off quickly at higher radii so OB chips can be used further out from the beam pipe.

The layout of the detector can be realized with the two suggested stave designs in fig. 6.18. The patches surrounding the beam pipe can use the *Focal IB/OB* stave design,

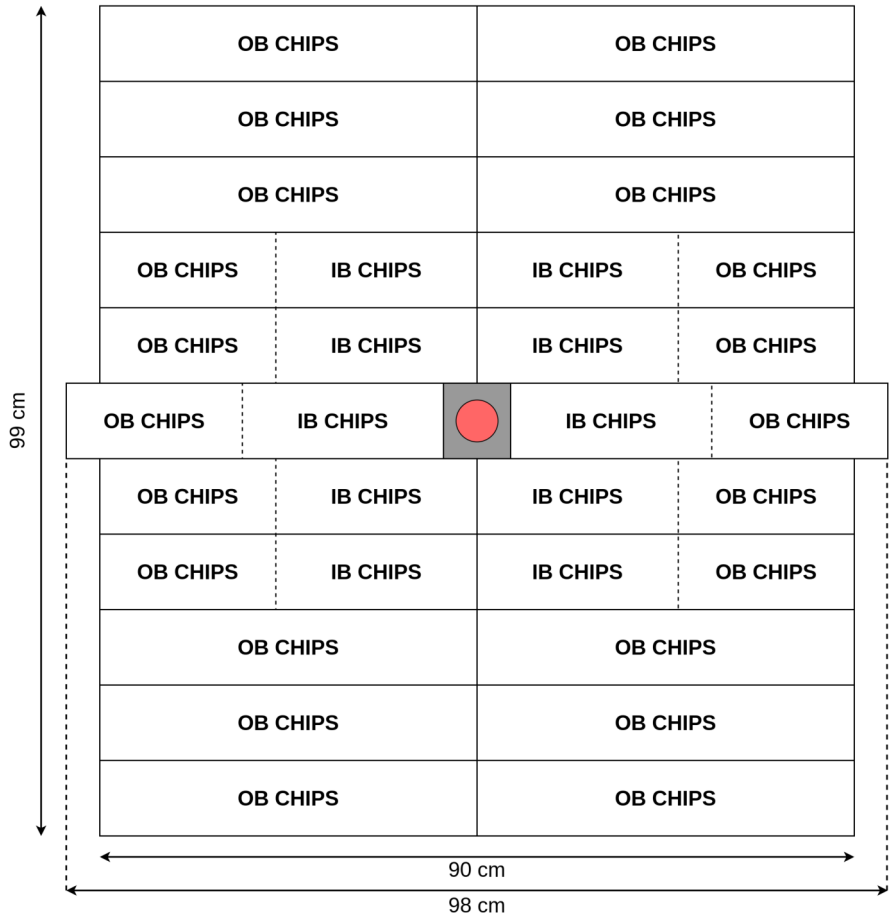


FIGURE 6.17: Proposed layout of ALPIDE chips in the FoCal detector plane. The red circle at the center represents the beam-pipe going through the plane (exaggerated size). The gray rectangle surrounding the beam pipe indicates the 8 cm $\times$ 9 cm gap around the beam-pipe. Each solid rectangle is a *patch* of 15 $\times$ 6 ALPIDE chips. The ten patches around the gap consists of chips in both IB and OB-mode, and the remaining twelve patches consists of chips in OB-mode only.

which consists of eight IB chips and seven OB chips. The addressing of the IB and OB chips can be done with only one the control link. And limiting the number of IB chips to eight allows chip ID 7 to be skipped. This is highly beneficial, since the decoding of ID 7 on the control link does not work properly, and performs broadcast commands to all the chips on the link.

The suggested *Focal OB stave* consists of 3 $\times$ 5 chips in OB mode, where three of them are OB master chips. The addressing on the control link will allow one control link to be shared by two staves, so only three control links are necessary for a *Focal OB patch*. However it is not possible to address more than one group of IB chips, so the *Focal IB patch* will require six control links.

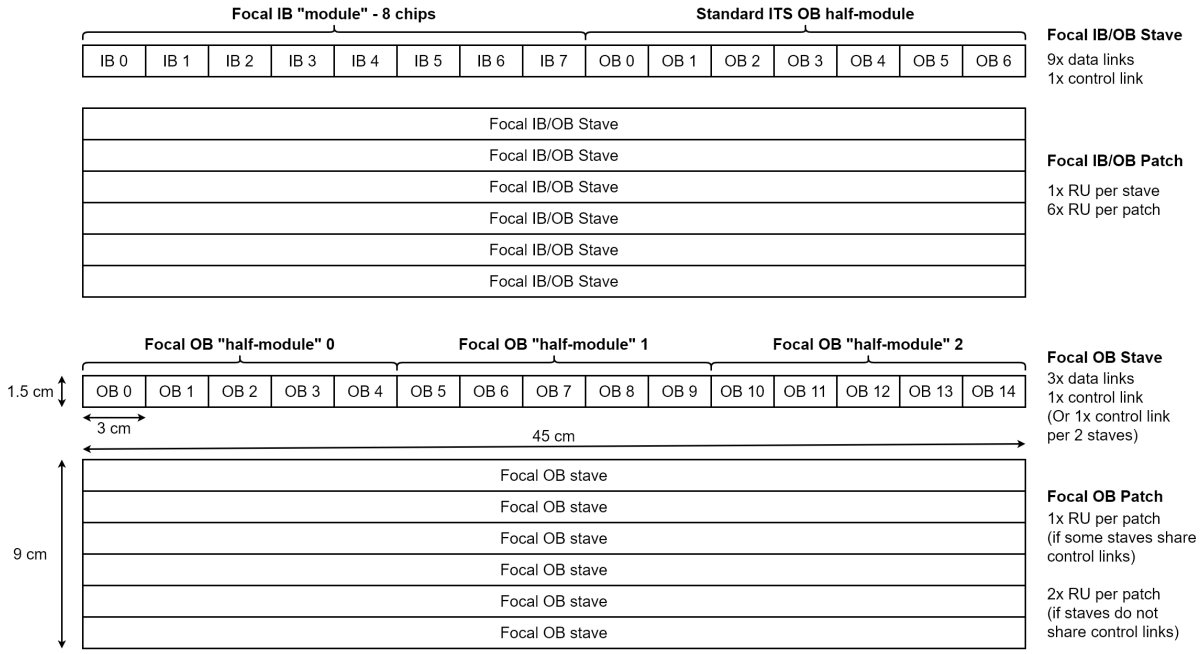


FIGURE 6.18: Proposed layout of ALPIDE chips in staves and patches used in the FoCal detector plane.

Input data

MC simulated events are used as input to the SystemC simulation of the FoCal detector. Pythia was used for the pp-data, and Hijing for Pb–Pb and Pb–p. The simulated data has hits over an area of 1600 mm $\times$ 1600 mm, divided into *macro-pixels* of 0.5 mm. The data contains the number of particle hits within a macro-pixel for a given event, but not the exact hit coordinates within the macro-pixel. The event generator for the SystemC simulation of FoCal generates N random hits within the area of a macro-pixel, where N is the number of hits within the macro-pixel for a given event. Random clusters of pixel hits are created around each random particle hit, using a 2D Gaussian distribution. Occupancy plots of the events from the pp and Pb–Pb data sets are shown in figs. 6.19 and 6.20. The occupancy is plotted for each chip with the detector layout suggested in figs. 6.17 and 6.18. The occupancy is clearly much higher near the center of the detector plane and this is the reason why IB chips with dedicated data links were used near the center.

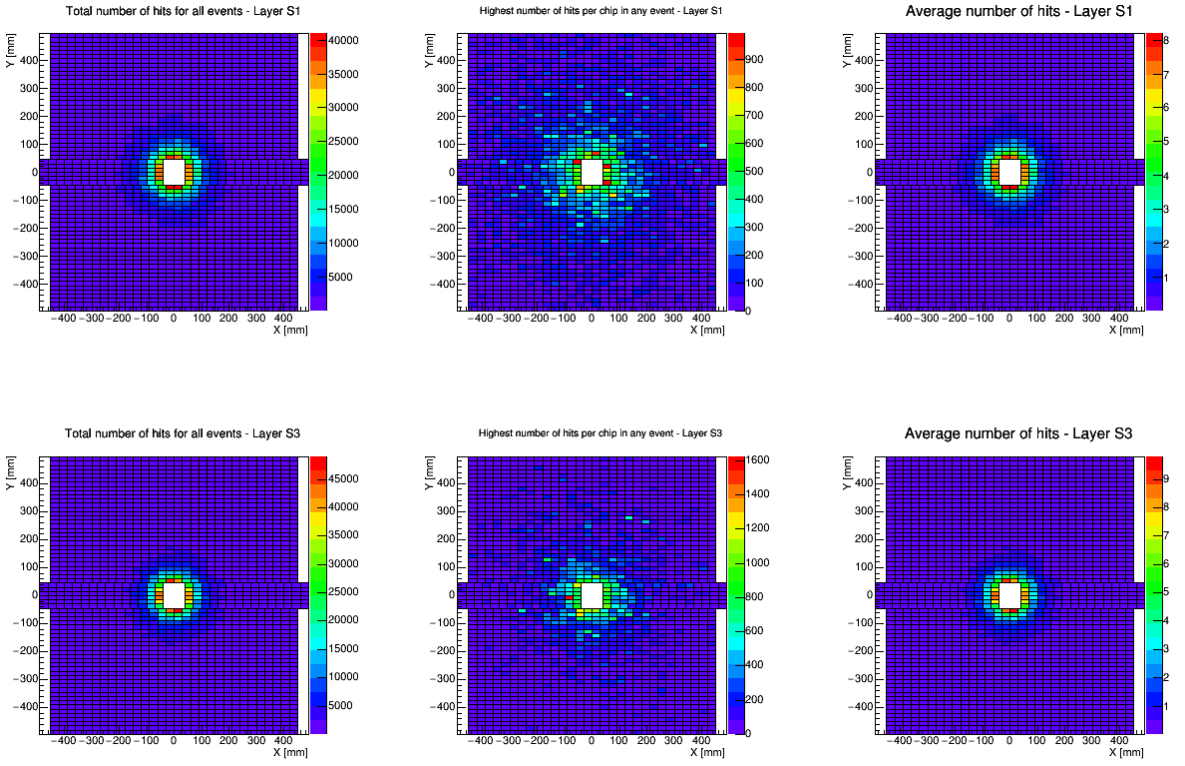


FIGURE 6.19: Occupancy map of pp MC-data for FoCal.

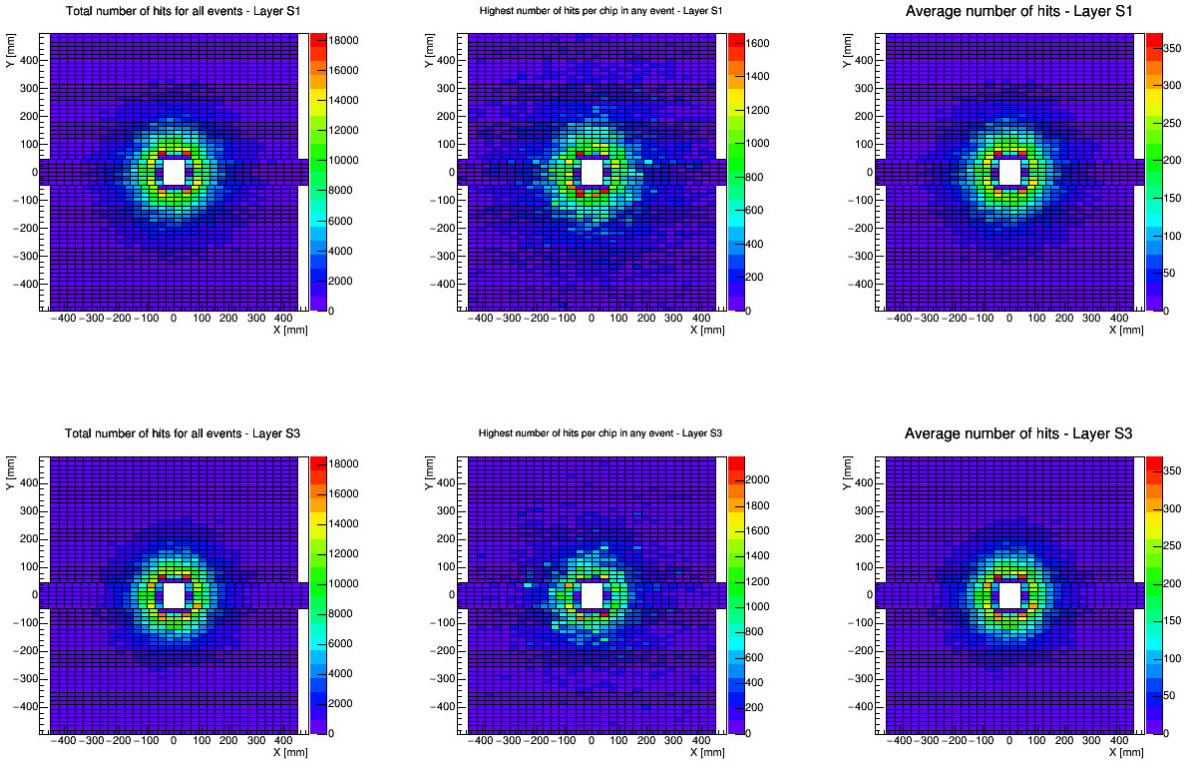


FIGURE 6.20: Occupancy map of Pb–Pb MC-data for FoCal.

6.3.2 Proton CT

The technologies developed at CERN have often found applications outside the fields of nuclear and particle physics. A good example is the ALPIDE pixel sensor developed for ALICE, which is currently being used at UiB to implement imaging technology for use with radiotherapy.

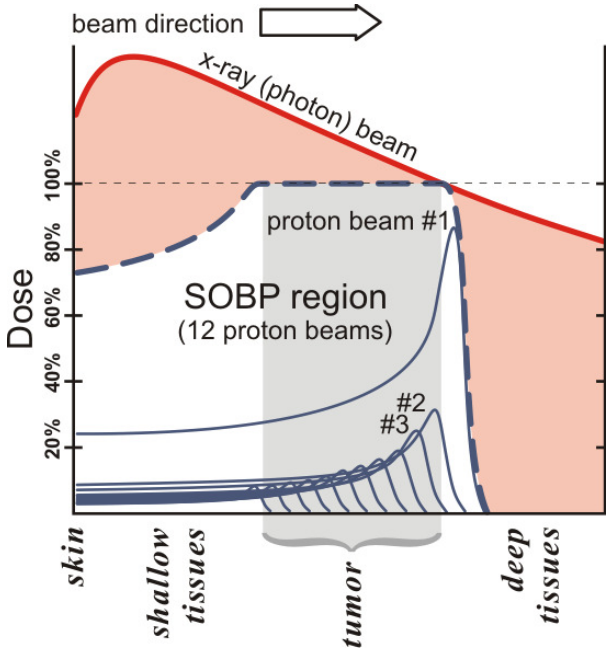


FIGURE 6.21: Comparison of dose-profiles with x-ray and proton treatment. The absorbed dose and Bragg-peak is shown for several proton beams of different energies (#1, #2, #3, ...). By combining several such proton beams a so-called Spread Out Bragg Peak (SOBP) can be achieved where a consistent dose is delivered to the area of the tumor. In contrast, an x-ray beam deposits a much larger dose before and after the tumor which will damage more healthy tissue. [100].

Radiotherapy is a common form of cancer treatment where a cancerous tumor is subjected to radiation. Ideally, the dose of radiation should be perfectly localized to the tumor, but with traditional forms of photon-based radiotherapy the surrounding tissue is also subjected to a considerable dose. This is shown in fig. 6.21. However, with particle therapy using protons and heavier ions, it is in principle possible to target the tumor with much higher accuracy. But there are several technical challenges to overcome. The so-called Bragg-peak[19] has to be placed at the location of the tumor. This requires advanced medical imaging techniques to create a map of tissue and bone in the target organ, which allows the energy of the radiation to be calculated to achieve a certain penetration depth. Traditionally, photon-based Computed Tomography (CT)

[19]The depth at which the stopping power, $-\frac{dE}{dx}$, is at its peak, and most of the particle's energy is deposited.

scanners have been used for the imaging and conversion factors were employed to estimate the Relative Stopping Power (RSP) for charged particles like protons. But these conversions introduce an error on the order of 2-3% [101]. Had the CT scan used protons instead, then these errors could have been avoided which would make the particle therapy more accurate and reduce the amount of damage to healthy tissue.

The pCT Project at UiB

Research and development of a Digital Tracking Calorimeter (DTC) for a pCT prototype is currently in progress at the University of Bergen (UiB) [36], [101]. The DTC will consists of 41 layers of pixel chips. There is no absorber between the first two layers which are used to establish the incident angle of the particles. But the remaining 39 layers have absorbers between them which slow down the particles [36]. A pencil beam (of protons) passes through the organ with the cancerous tumor on its way to the DTC. The beam scans across the surface of the DTC, and the kinetic energy of incoming particles in the DTC is measured by reconstructing their tracks and analyzing cluster sizes. The energy loss in the organ is calculated by comparing the initial energy of the beam, which is a known quantity, with the residual energy of the beam when it reached the DTC. The measurements are repeated at different angles to perform the CT scan.

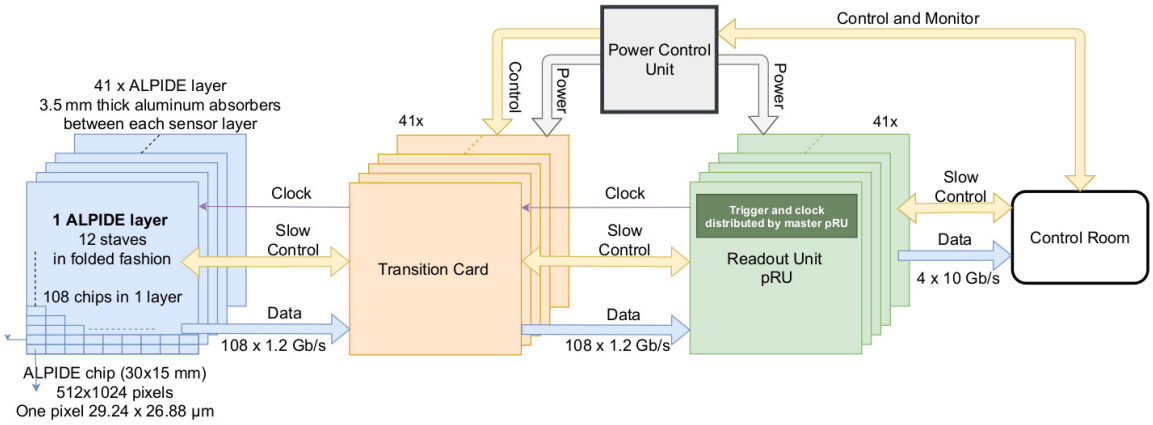


FIGURE 6.22: pCT detector and readout. [37].

Figure 6.22 shows an illustration of the UiB pCT and readout system. The tracking layers of the DTC are based on the ALPIDE chip. Each layer consists of 12 IB-staves identical to those of the ITS, i.e. a total of $9 \times 12 = 108$ ALPIDE chips per layer. Readout is performed using the pCT Readout Unit (pRU) which is based on a Xilinx Kintex UltraScale FPGA. There is one pCT Readout Unit (pRU) per layer responsible

for readout of all 108 data links of the layer[20], and up to four 10 Gb/s Ethernet links are available per pRU to offload data [36].

The amount of data that the pRU will have to cope with, as well as the upstream bandwidth that it requires, were among the questions that had to be answered during the design phase of the pRU which called for simulations of the pCT using the SystemC model.

Input Data

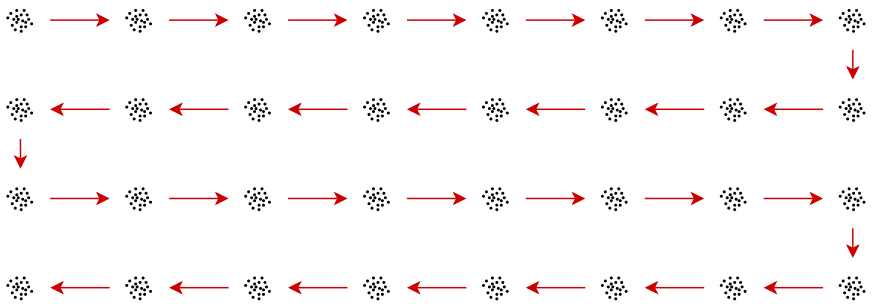


FIGURE 6.23: Pencil beam scan pattern example. The scan starts in the upper left corner and moves in discrete steps as indicated by the arrows.

A set of MC data for the pCT was provided for use with the SystemC simulations. The data had been generated using the *Digital Tracking Calorimeter Toolkit*[21] which was developed by Pettersen et. al. for the pCT [102]. A pencil beam of 230 MeV protons, with an intensity of 1×10^7 protons per second, had been scanned across the DTC plane in those simulations and the coordinates of hits in each layer were recorded. Figure 6.23 illustrates a typical pencil beam scanning pattern. The scan pattern can also be recognized in figs. 6.25A and 6.25B, which shows hit positions in the XY-plane of the first layer of the DTC in the MC data. The XY-plane covers an area of 270 mm $\times$ 135 mm in the data set, a surface equivalent to 9×9 ALPIDE chips[22]. The beam scans across this surface with a speed of the order of 100 m s^{-1} in the X-direction and covers the entire surface in around 6.5 ms.

The energy deposition and slowing of the protons in the sensitive layers and absorbers were simulated, leading to the decreasing occupancy at deeper layers, as shown in fig. 6.24. A simple clustering algorithm based on a 2D-Gaussian distribution had been applied to the primary particles[23], creating random clusters with an

[20] Regular IOs pins are used to read out the 1.2 Gbit s^{-1} ALPIDE data links [36].

[21] The toolkit is based on the Geant4 Application for Tomographic Emission (GATE) MC software.

[22] As mentioned the final DTC design consists of 9×12 chips, but 9×9 is sufficient for the simulations.

[23] The protons, which accounts for almost 99% of the particles in the simulation.

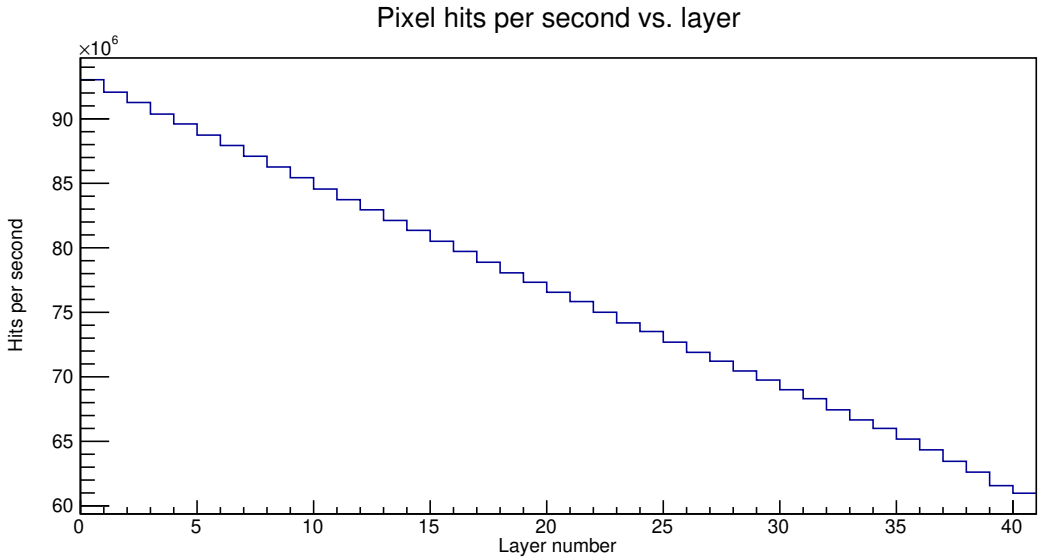
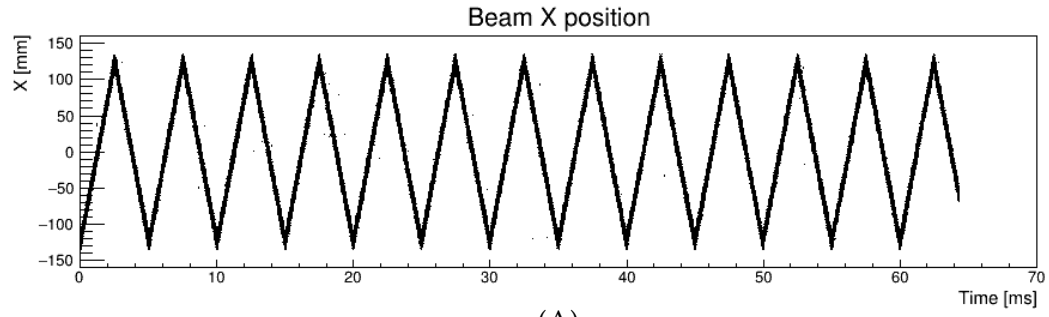
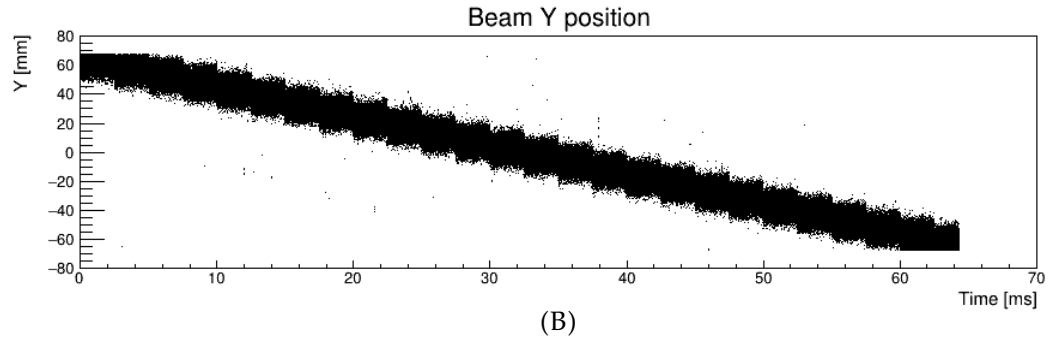


FIGURE 6.24: Hit intensity versus layer in the MC simulated data set for the pCT DTC.



(A)



(B)

FIGURE 6.25: Scan pattern in MC data for the pCT DTC. All hits from the first layer of the DTC are plotted, with the X-position shown in 6.25A and Y-position in 6.25B. Note: The Y-axis scale differs between the two plots.

average of 9 pixels hit per primary [102], which matches typical cluster size distributions from beam tests of the ALPIDE [25]. It should be noted that the actual energy

deposition was not taken into account by the clustering algorithm, and larger clusters are expected in the deeper layer where the energy deposition is higher.

Chapter 7

Simulations and Results

This chapter will discuss simulations and results achieved with the simulation model that was introduced in the previous chapter. The main focus is on simulations for the ITS, but results for the FoCal and pCT detectors are also included. Only a subset of the available simulation data is presented in this chapter. More results can be found in appendix H.

7.1 ITS Simulations

The set of simulations that were run for the ITS is listed in table 7.1. Most of the results and figures presented in this chapter, for the ITS, are based on a prior publication [103] which was part of the work presented in this thesis. The parameters in table 7.1 were chosen not just to address the original concern of a busy detector and data loss, but also to predict how the detector will behave when operated beyond the ALICE Run 3 interaction rates of 50 kHz for Pb–Pb and 200 kHz for pp[1].

The simulations used a pool of MC events as input which had been generated with the AliRoot framework (see section 6.2.1 for more details). There were 10 000 discrete Pb–Pb events with QED background data, and 100 000 pp events. The combinations of parameters in the table adds up to 56 simulations. Each simulation included one full stave per layer, which amounts to 643 ALPIDEs chips in total. Because of time constraints[2], the number of simulated collisions were limited to 100 000 for Pb–Pb, and 1 000 000 for pp. Ideally this number would have been higher.

For Pb–Pb the simulations were run at interaction rates ranging from 50 kHz to 200 kHz, which is twice as high as the specification for ITS. It became apparent early on that the detector performs quite well in the simulations when it is operated at interaction rates within the specifications. This offers limited statistics on the mechanisms

[1] An interaction rate of 200 kHz for pp was originally planned for Run 3. But this was later increased to 500 kHz.

[2] The slowest Pb–Pb simulations ran for around 2-3 days each.

for data loss in the ALPIDE, and it is another reason why simulating higher rates is interesting; a lower readout efficiency is expected at those rates, and this allows for a better comparison between different sets of parameters.

TABLE 7.1: Simulation setup for Pb–Pb and pp simulations of ITS [103].

Parameter	Pb–Pb	pp
Event rates [kHz]	50[a], 100[i], 150[b], 200[b]	400[i], 1000[b], 2000[b], 5000[b]
Number of events simulated	100 000	1 000 000
Strobe length (min-bias trigger) [ns]	100	100
Strobe length (periodic trigger) [ns]	4 900, 9 900, 19 900	4 900, 9 900, 19 900
Period (periodic trigger) [ns]	5 000, 10 000, 20 000	5 000, 10 000, 20 000

[a] ALICE Run 3 requirement [i] ITS specification [b] Beyond specification

Trigger Mode and Strobe Lengths. Each of the interaction rates listed in table 7.1 was simulated with minimum-bias triggers and periodic triggers. Minimum-bias was simulated using the 100 ns strobe length and with the chip in triggered mode. But for the continuous mode simulations with periodic triggers, the three different strobe lengths and corresponding trigger period of the table were all simulated. The strobe length that will be used in the experiment has not been decided yet, but a length of 10 µs to 20 µs is expected. A shorter strobe and period would likely be beneficial, since that should reduce the pileup of events in each frame and make reconstruction easier.

The choice of triggered or continuous mode in the ALPIDE only pertains to the handling of the MEB (see section 2.4). In principle, either mode can be used with a short strobe and LM triggers from the CTP or periodic triggers and long strobes when the experiment operates in continuous mode. The assumption is that the triggered mode in the chip is better suited for the LM triggers, and the continuous mode is better suited for the periodic triggers. To test this assumption, each simulation where the experiment was in continuous mode (i.e. periodic triggers and long strobes) was run with the chips in both continuous and triggered mode. To avoid confusion, the experiment's continuous trigger mode will be referred to as "periodic triggers" throughout this chapter, and the continuous mode setting in the chip will simply be referred to as continuous mode.

Finally, it should be noted that a pixel pulse shaping time of 5 µs was used for all of the simulations.

Pixel Noise. As mentioned in the previous chapter, pixel noise[3] was omitted in the simulations. This leads to a discrepancy compared to some simulations that were

[3] Also called fake hits.

performed at an earlier stage in the development of the ITS [40], which were simulated with a fake-hit rate of 3-4 orders of magnitude higher than measured on the production version of the ALPIDE.

7.2 ITS Simulation Results

7.2.1 Readout Efficiency

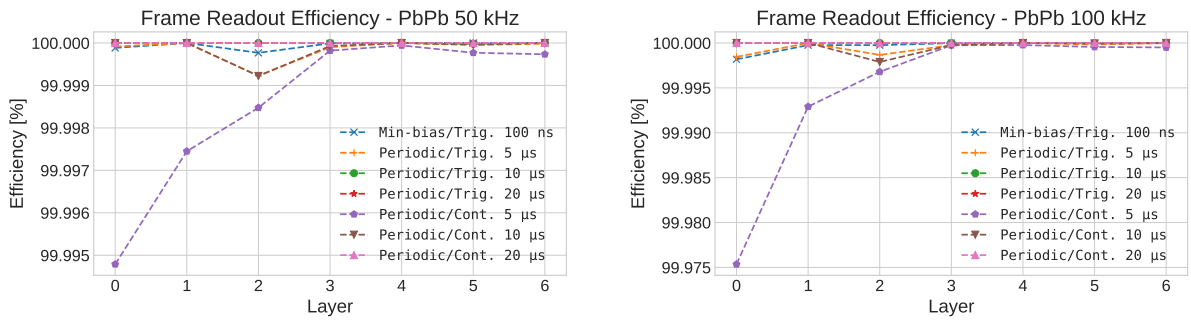
The number of frames (or triggers) is stored by the simulation, as well as how many frames are lost. Let N be the total number of frames (triggers), where $N = N_{OK} + N_{BV} + N_{Flush}$. N_{OK} is the number of frames that were fully read out, N_{BV} the discarded frames due to busy violations, and N_{Flush} the number of frames that were flushed and not fully read out. The readout efficiency in terms of frames is defined as:

$$E_{Frames} = \frac{N_{OK}}{N} = \frac{N_{OK}}{N_{OK} + N_{BV} + N_{Flush}} \tag{7.1}$$

N_{Flush} will always be zero in triggered mode, but there can be both flush and busy violations in continuous mode (but primarily flushes).

Readout Efficiency in Terms of Frames for Pb–Pb

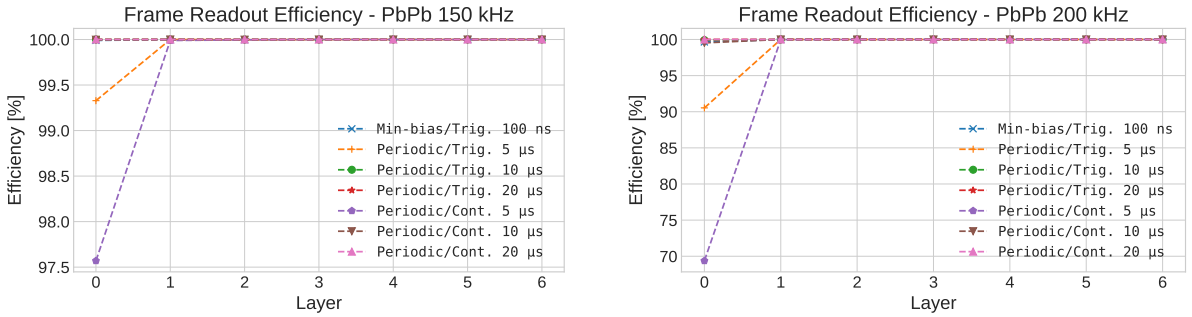
Figures 7.1A and 7.1B shows the readout efficiency, in terms of readout frames, for Pb–Pb at 50 kHz, the ALICE target in Run 3, and 100 kHz which is the specification for the ITS. The efficiency is around 100% regardless of trigger mode and other parameters, though there is an observable drop in efficiency for continuous mode with 5 µs, albeit a small one.



(A) 50 kHz (B) 100 kHz

FIGURE 7.1: Frame readout efficiency for ITS in Pb–Pb simulations at nominal interaction rates [103]. Note: The y-axis has been truncated and the range differs between the two graphs. The efficiency is approximately 100% in every case.

At 150 kHz and 200 kHz, shown in figs. 7.2A and 7.2B, the drop in efficiency with periodic triggers and 5 µs strobe becomes more pronounced. The explanation is likely that the pulse shaping time was also 5 µs; unless the pulse for a hit lines up perfectly within the 5 µs strobe window, the hit will be sampled by two strobes. Most of the events are effectively double sampled with this combination of pulse length and strobe window, which effectively doubles the amount of data to read out.



(A) 150 kHz (B) 200 kHz

FIGURE 7.2: Frame readout efficiency for ITS in Pb–Pb simulations at interaction rates beyond the specifications [103]. Note: The y-axis has been truncated and the range differs between the two graphs.

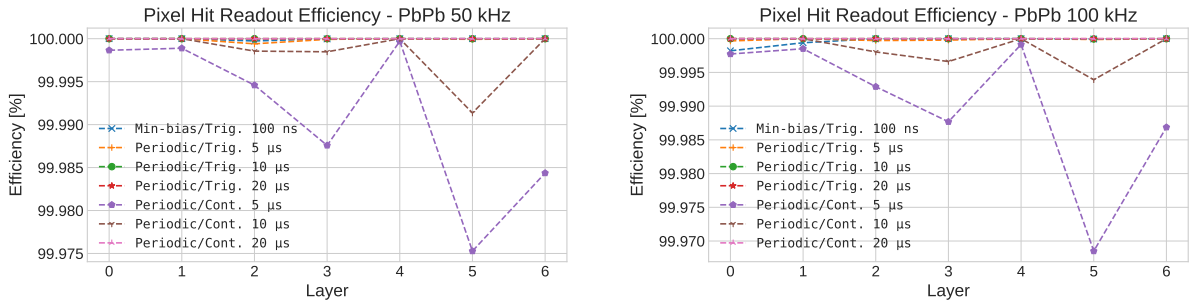
Readout Efficiency in Terms of Pixel Hits for Pb–Pb

The readout efficiency metric that was calculated based on frames is not an entirely fair comparison of triggered and continuous mode, since the lost frames in continuous mode were probably at least partially read out. The simulation also records the number of pixel hits in the simulation, whether they were read out or not, as well as statistics of how many frames they were read out in. Figures 7.3 and 7.4 shows the readout efficiencies for Pb–Pb in terms of pixels, where the efficiency was calculated in this manner:

$$E_{Pixels} = \frac{N_{Pixels\ read\ out}}{N_{Pixels\ not\ read\ out} + N_{Pixels\ read\ out}} \tag{7.2}$$

At 100 kHz there is a tiny loss of data in the innermost layer when using minimum-bias triggers. But the combination of periodic triggers, continuous mode, and a 5 µs strobe appears to be the least efficient at nominal rates (50 kHz and 100 kHz), followed by the same parameters with a 10 µs strobe. It should be noted though that the efficiency is really close to 100% in every case in figs. 7.3A and 7.3B, and that is the main conclusion that can be drawn with the limited statistics.

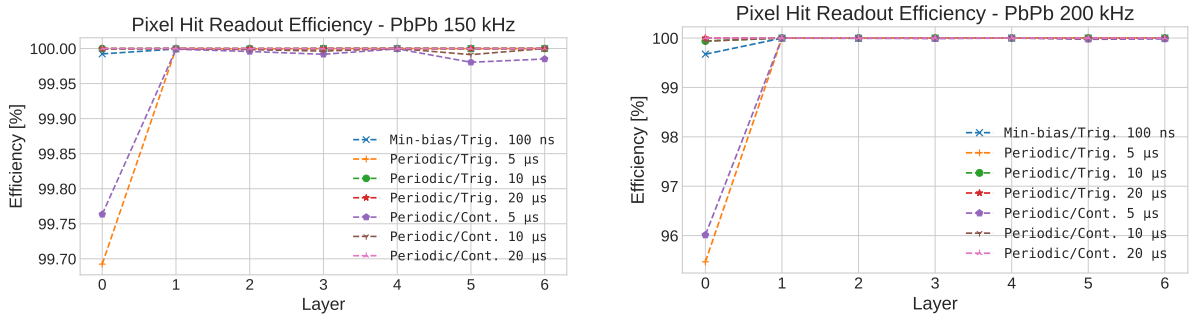
At the higher rates, 150 kHz and 200 kHz, the periodic triggering with 5 µs strobe stands out again with the lowest efficiency. But compared to the frame efficiency in



(A) 50 kHz (B) 100 kHz

FIGURE 7.3: Pixel hit readout efficiency for ITS in Pb–Pb simulations at nominal interaction rates [103]. Note: The y-axis has been truncated and the range differs between the two graphs. The efficiency is approximately 100% in every case.

fig. 7.2, the situation is reversed with the triggered mode performing slightly worse than continuous mode. The combination of periodic triggers and continuous mode also has a small drop in efficiency for layers 3 and 5[4], when using 5 µs or 10 µs strobes. Minimum bias triggering shows some loss of efficiency in the innermost layers again, but it is still well over 99% even at 200 kHz.



(A) 150 kHz (B) 200 kHz

FIGURE 7.4: Pixel hit readout efficiency for ITS in Pb–Pb simulations at interaction rates beyond the specifications [103]. Note: The y-axis has been truncated and the range differs between the two graphs.

As a concluding remark, using minimum-bias triggers has an efficiency of over 99% at any of the simulated interaction rates, and outperforms periodic trigger with 5 µs strobe in any scenario. At higher interaction rates one should consider using periodic triggering with at least a 10 µs strobe, and the continuous mode in the chip appears to perform worse than triggered mode for the outer layers. Figure 7.5, which shows busy counts for the 200 kHz Pb–Pb simulations, may illustrate this better. The ALPIDE chips will frequently report busy in continuous mode, but rarely in triggered mode. From the perspective of the readout electronics, there are only two ways to deal

[4]The first Middle Layer (ML) and the first Outer Layer (OL).

with a busy chip: ignore it and accept that data is lost when the next trigger is sent[5], or withhold the next trigger[6] to allow readout of the current frame to finish. The latter beats the purpose of the flushing mechanism in continuous mode; one would essentially be left with triggered mode with only two event buffers.

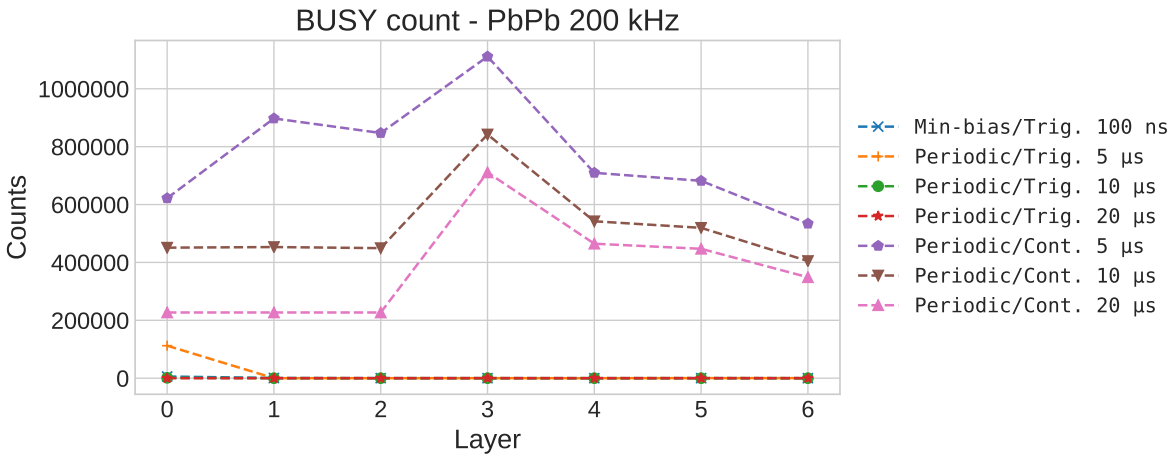
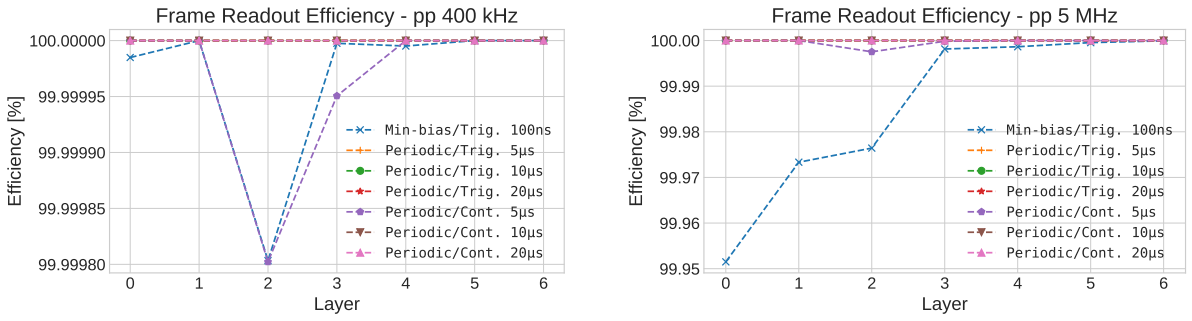


FIGURE 7.5: Busy counts for ITS at 200 kHz Pb–Pb.

Readout Efficiency for pp

Readout efficiency for pp is shown in terms of frames in fig. 7.6, and in terms of pixels in fig. 7.7, for interaction rates of 400 kHz and 5 MHz. Plots for 1 MHz and 2 MHz were omitted since the detector is very efficient in pp at any of the simulated interaction rates[7], regardless of configuration.

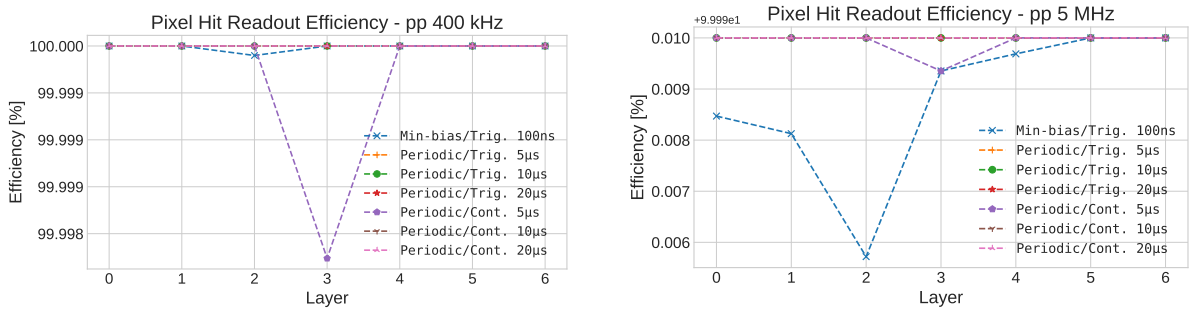


(A) 400 kHz (B) 200 kHz

FIGURE 7.6: Frame readout efficiency for ITS in pp simulations [103]. Note: The y-axis has been truncated and the range differs between the two graphs. The efficiency is approximately 100% in every case.

[5]Unless the busy condition goes away before the next trigger.
[6]Keeping in mind that several chips share a control link, which is used for the trigger.
[7]Because of the significantly lower hit density compared to Pb–Pb.

Minimum-bias triggering was the least efficient in these simulations. The effective trigger rate will be very high for minimum-bias triggering, but is limited to a maximum of around 800 kHz because of the trigger filtering[8] (trigger rate is also limited to $\approx$ 1 MHz by the speed of the control link [26], [98]). Considering that the 5 µs periodic triggers have a trigger rate of 200 kHz, which is constant regardless of interaction rate, it is not surprising that the minimum-bias triggers are less efficient. Besides that, periodic 5 µs triggers show less than 100% efficiency in some cases, with the chips in continuous mode, especially for the third layer as seen in fig. 7.7. But this is not seen with the chip in triggered mode, which strengthens the claim that periodic triggering performs slightly better with the chips in triggered mode, and not in continuous mode.



(A) 400 kHz (B) 5 MHz

FIGURE 7.7: Pixel hit readout efficiency for ITS in pp simulations [103]. Note: The y-axis has been truncated and the range differs between the two graphs. The efficiency is approximately 100% in every case.

There have been discussions in the ALICE Collaboration to run the experiment at interaction rates up to 5 MHz, with a short strobe of 1 µs duration in the ITS to help disentangle piled-up events. In this case, the experiment would mostly rely on the ITS for tracking as the TPC can not cope with these rates. Some additional simulations were performed for this scenario (not listed in table 7.1), and the simulated readout efficiencies (in terms of pixels) are shown in figure 7.8. It indicates that it should be well within the capabilities of the ITS to perform such runs. And again, it is worth noting that the ALPIDE's triggered mode performs better than the continuous mode.

7.2.2 Pileup

Pileup is a term used to describe the situation where a readout frame contains more than one interaction event. From an analysis point of view it is preferable to have a low amount of pileup, since a higher amount would make it harder to distinguish

[8] Trigger filter window is around 1200 ns.

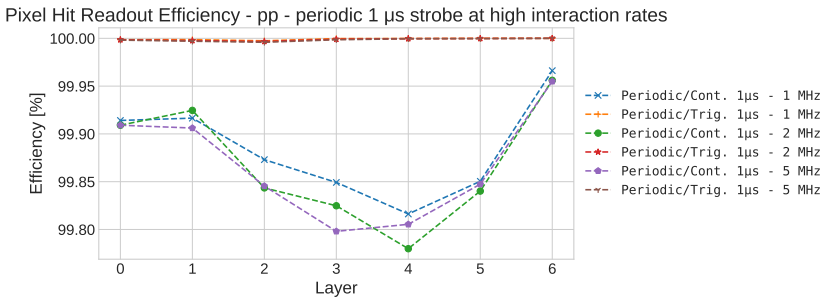


FIGURE 7.8: Pixel hit readout efficiency for ITS in pp simulations at high interaction rates with a 1 µs periodic strobe. Note: The y-axis has been truncated.

between the events in a readout frame. Pileup in Pb–Pb is shown in figs. 7.9 and 7.10[9]. The X-axis indicates the pileup count, i.e. the number of interaction events in a readout frame. Empty frames have a pileup of zero[10].

When minimum-bias triggers are used, most frames contain just the one event that they triggered on, and as expected there are no frames with a pileup of zero. Around 20% of the frames have a pileup of two[11] at 50 kHz and 100 kHz, but very few frames have higher pileup than that. With the periodic triggers, however, there is a significant amount of empty frames when the strobe is short, but not a lot of pileup. The situation reverses with the longer strobes; empty frames are less likely in those cases, but many frames will have higher values of pileup. At 50 kHz, shown in fig. 7.9B, it is rare to see a pileup of more than three, regardless of triggering mode. But at the higher interaction rates a higher value becomes more common, as shown in the remaining figures.

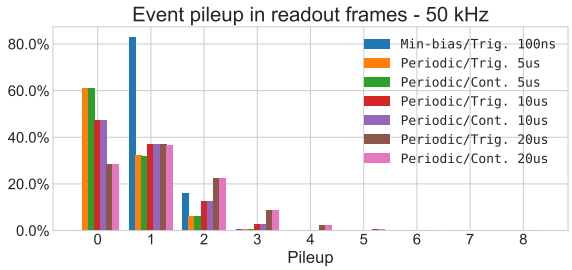
7.2.3 Data Rates

The data produced by the detector is transferred and processed in several steps by the readout electronics, CRU, and FLPs before being stored for later analysis. When operating parameters are chosen for the detector, it might be necessary to consider what data rates to expect to ensure that the experiment can process the data without loss.

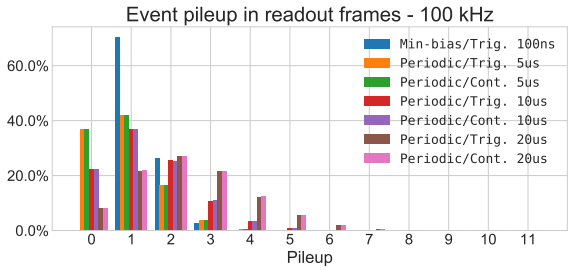
[9]This is based on data from the simulations, although technically the full simulation would not have been necessary to estimate this. It is purely based on the time of an interaction, pulse shaping time, strobe length and the time when the strobe is active.

[10]It should be noted that even though a readout frame for an individual chip may be empty, the frame may be associated with an event for the detector as a whole, if the strobe for the frame happened to coincide with the event.

[11]It might appear as though a pileup value of two is more common for minimum-bias triggers, but the figures are a little misleading because of the high number of zero pileup frames for periodic triggers.

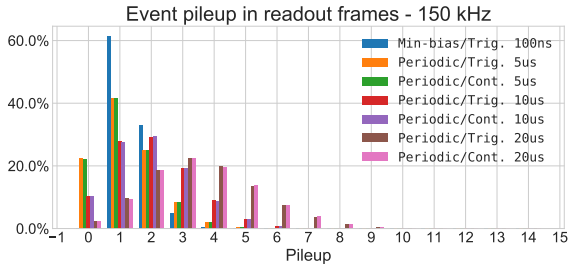


(A) 50 kHz

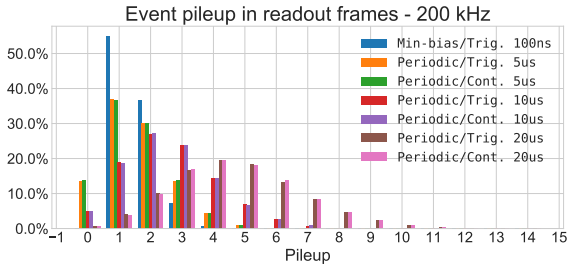


(B) 100 kHz

FIGURE 7.9: Pileup of events in readout frames for ITS in Pb–Pb simulations [103].



(A) 150 kHz

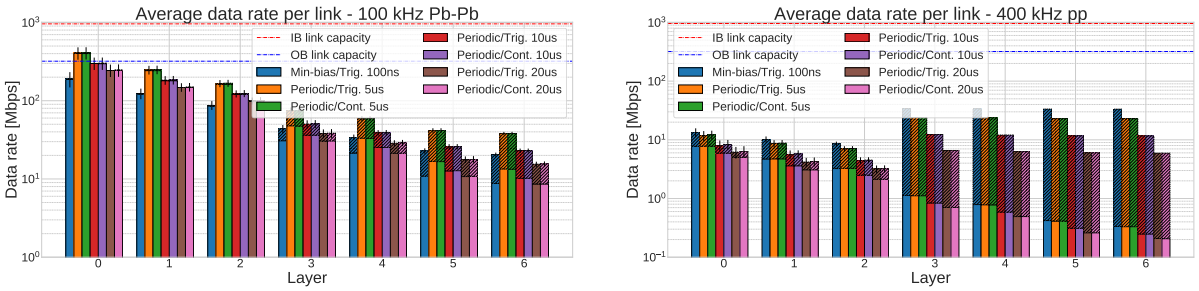


(B) 200 kHz

FIGURE 7.10: Pileup of events in readout frames for ITS in Pb–Pb simulations [103].

Average data rates were estimated based on the simulated data, on a per-link and per stave basis. Data in this context is defined as *CHIP HEADER* and *TRAILER* words, *CHIP EMPTY FRAME*, *REGION HEADER*, *DATA SHORT*, and *DATA LONG* words. *IDLE*, *BUSY ON*, and *BUSY OFF* words are not included as they are stripped away by the RU. The bandwidth of an ALPIDE data link is 960 Mbps or 320 Mbps, for IB or OB, respectively, and the bandwidth of the links gives the theoretical upper limit of the data rate. The margin between bandwidth and actual data rate is also interesting to consider. It is closely tied to the detector's efficiency, as a smaller margin will lead to higher utilization of buffers and FIFOs in the ALPIDE chip.

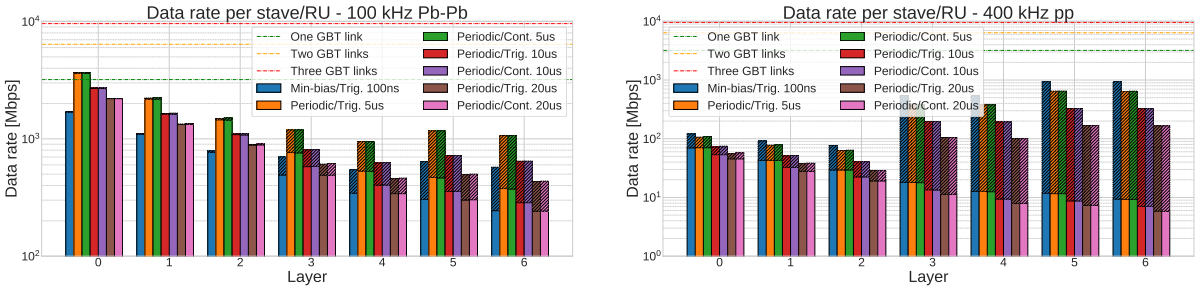
Figure 7.11 shows the average data rate per ALPIDE data link for 100 kHz Pb–Pb and 400 kHz pp, which is what the ITS is specified for. The horizontal red line indicate the bandwidth of an IB data link (layers 0, 1, and 2), and the blue line shows the bandwidth of an OB link (layers 3-6). The shaded area of the bars is the protocol overhead associated with a frame, i.e. *CHIP HEADER*, *CHIP TRAILER*, and *CHIP EMPTY FRAME* words, and the solid part is for for actual pixel hit data. For the outer layers where the hit density is low, the protocol overhead accounts for a significant portion of the data, in fact for pp there is more overhead than pixel hit data. A stave covers ± 2.5 units of pseudorapidity for the innermost layer, compared to only ± 1.3



(A) 100 kHz Pb–Pb (B) 400 kHz pp

FIGURE 7.11: Average data rate per link for ITS simulations. Note: The y-axis is logarithmic and the range differs between the charts.

for the outermost layer [13], and the hit density varies with pseudorapidity as shown in fig. 6.2. Since the data rate has been averaged over all links in a stave, this leads to the error bars in fig. 7.11, which are most pronounced at the inner layers.



(A) 100 kHz Pb–Pb (B) 400 kHz pp

FIGURE 7.12: Total data rate per stave/RU for ITS simulations. Note: The y-axis is logarithmic and the range differs between the charts.

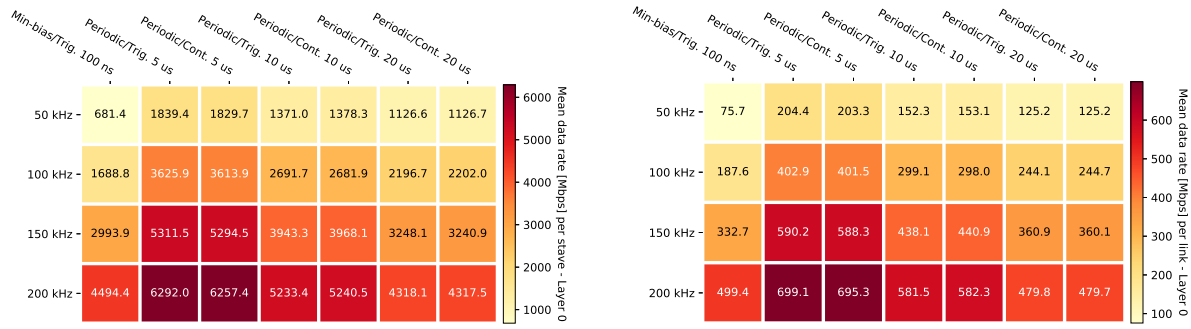
The total data rate per stave is shown in fig. 7.12, and this is interesting to consider because there is one RU per stave to process the data. Each RU has three 3.2 Gbps GBT links that can be used for data transfer, and the plots indicate the horizontal lines indicates the capacity of one, two, or three GBT links. For Pb–Pb at 100 kHz, periodic triggering with 5 µs strobe surpasses the capacity of an individual GBT link for the innermost layer, but in any other case it is sufficient with one. Using minimum-bias triggers has the benefit of producing the least amount of data in most cases, with the exception of periodic triggers with 20 µs strobe in the outer layers.

The results for pp are quite different. First, it should be noted that the amount of data in the inner layers is more than an order of magnitude lower compared to Pb–Pb, and the data rate is well within the capacity of one GBT link, under all circumstances. Minimum-bias triggering actually produces more data than the other triggering modes when running pp. This is because the interaction rate is quite high in pp, and the pulse-shaping time of the ALPIDE is relatively long, around 5 µs, and

as a consequence many events will be sampled more than once when minimum-bias triggers are used.

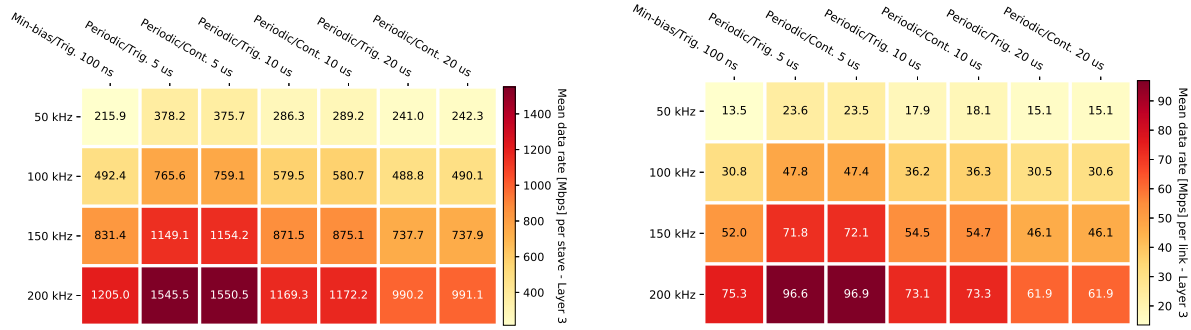
Another key takeaway from all these figures is that the choice of continuous or triggered mode, when used with periodic triggering, has practically no influence on the amount of data that is produced by the detector. But the strobe length with periodic triggers plays an important role; a longer strobe is much more efficient in terms of the amount of data that is produced, since events are often sampled more than once when short strobes are used.

A comparison of simulated data rates for the Pb–Pb simulations, both average per link and stave totals, is shown in figs. 7.13 and 7.14, for the innermost layer and the first of the outer layers.



(A) Per stave (B) Per link

FIGURE 7.13: Simulated data rates for Pb–Pb in layer 0 of the ITS.



(A) Per stave (B) Per link

FIGURE 7.14: Simulated data rates for Pb–Pb in layer 3 of the ITS.

7.3 FoCal Simulations and Results

The FoCal detector is still in the early design stages. The simulations described here were used to determine the feasibility of the ALPIDE chip for the two tracking layers,

TABLE 7.2: Simulation setup for Pb–Pb and pp simulations of FoCal.

Event type	Event rates [kHz]	Number of events	Trigger mode	Trigger period	Chip mode	Strobe
pp	200, 500, 1000	250 000	MB-trig	N/A	Triggered	100 ns
pp	200, 500, 1000	250 000	Periodic	10 µs, 20 µs	Continuous	10 µs, 20 µs
pp	200, 500, 1000	250 000	Periodic	10 µs, 20 µs	Triggered	10 µs, 20 µs
Pb—Pb	50, 100	10 000	MB-trig	N/A	Triggered	100 ns
Pb—Pb	50, 100	10 000	Periodic	10 µs, 20 µs	Continuous	10 µs, 20 µs
Pb—Pb	50, 100	10 000	Periodic	10 µs, 20 µs	Triggered	10 µs, 20 µs

from the perspective of readout capabilities. Concerns about event pileup were also investigated with the simulations, and estimations of data rates should prove valuable for the collaboration to determine the number of FoCal *staves* that can be handled by the three GBT-links of an RU. Some of these results were briefly summarized in the Letter Of Intent (LOI) for the FoCal detector [99].

Table 7.2 summarizes the range of simulations that were performed. The proposed detector layout, which was illustrated in figs. 6.17 and 6.18, consists of 3960 ALPIDE chips in total, and was simulated in its entirety. As for the ITS simulations, the nominal interaction rates for ALICE were simulated for pp and Pb–Pb, as well as higher rates. The simulated triggering modes include minimum bias triggering, with the same trigger filtering parameters as for the ITS, and periodic triggering at different trigger rates for both chip modes (triggered and continuous). Because of limitations of the MC events that were used as input to the model (see section 6.3.1), the event generator of the SystemC model had to generate the actual pixel hits and clusters based on information from the input events. The pixel clusters were generated for each hit in the sensors using a 2D-Gaussian distribution, where the size of the cluster (in terms of pixels) was $\mu = 2.5$ and $\sigma = 1$ pixels. This is representative of measured and simulated cluster sizes[12] in the ALPIDE for Minimum Ionizing Particle (MIP) particles and X-rays [104], [105].

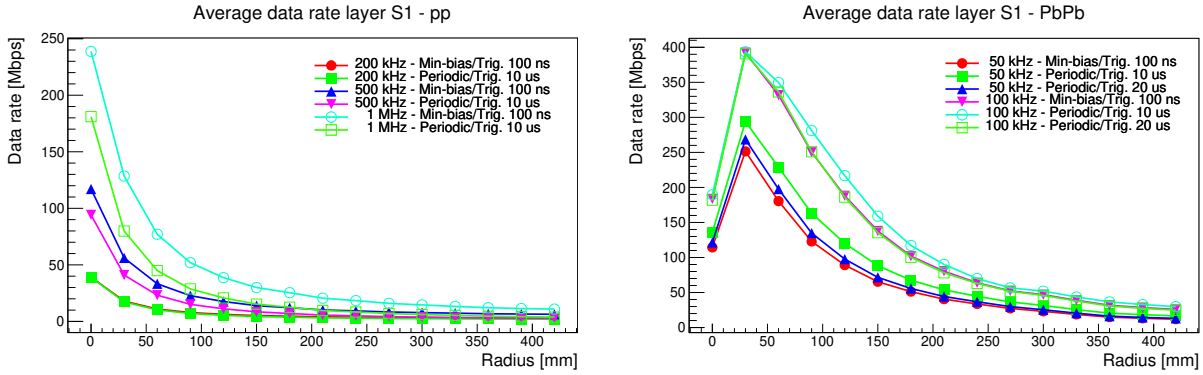
7.3.1 Data Rates and Readout Efficiency

The proposed FoCal layout consists of ALPIDE chips in IB-mode surrounding the beam pipe, and chips in OB-mode are used further out. The first OB chips are located 180 mm vertically from the gap, at the first *patch* consisting purely of OB-mode chips, and horizontally at 240 mm from the gap in the OB-mode portion of the mixed IB/OB-mode patches around the gap (refer to figs. 6.17 and 6.18).

[12]It should be noted that the actual cluster size depends on the threshold for the comparator in the pixel amplifier chain, as well as the back-bias voltage.

Figures 7.15A and 7.15B shows average data rate per chip for a selection of the simulations, both pp and Pb–Pb[13]. For all of the simulated pp cases, the data rate per chip is not higher than 30 Mbit s^{-1} at 180 mm and beyond (keeping in mind that one 320 Mbit data link is shared among 7 OB-mode chips in the mixed IB/OB-mode staves, and shared among 5 chips in the pure OB-mode staves, of the proposed FoCal layout[14]). For the 50 kHz Pb–Pb simulations, however, the data rate is around 70 Mbit s^{-1} at 180 mm, and even higher at 100 kHz. This indicates that the link margin may be insufficient for the first chips in OB-mode. To achieve a higher margin it may be desirable to use fewer OB chips per link for the mixed IB/OB stave.

It should be noted that the MC-simulated Pb–Pb input data for the SystemC model had been generated with a slightly larger gap around the beam pipe, which is also visible in the occupancy maps of fig. 6.20. This explains the dip in data rate for the Pb–Pb simulations at 0 mm. But this discontinuity would presumably not be there if the input data was generated with the correct gap size.



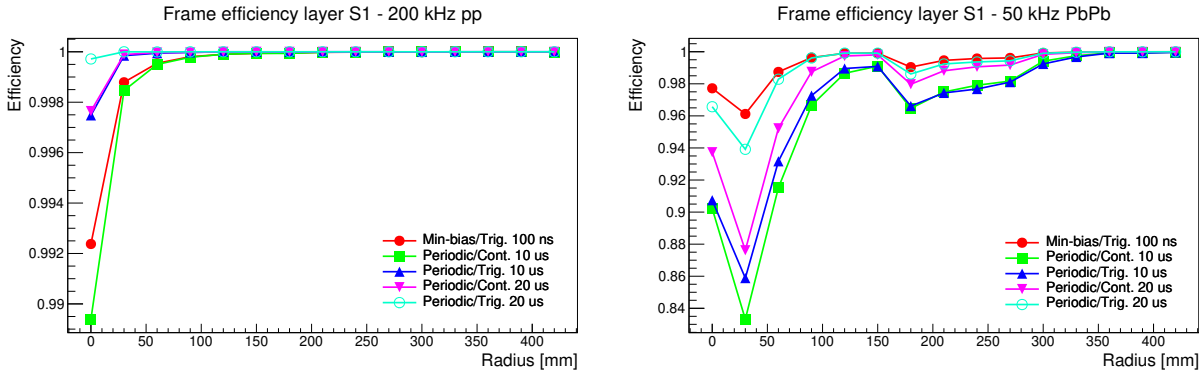
(A) pp (B) Pb–Pb

FIGURE 7.15: Average data rate (per chip) versus radius simulated for FoCal. There is 30 mm between each data point (the width of the chip), and each point indicates the average data rate for the chips between that point and the next point. Note: The y-axis range differs between the graphs.

The readout efficiency, in terms of readout frames, is shown for nominal interaction rates in figs. 7.16A and 7.16B. These plots tell a similar story to those of the ITS simulations. For Pb–Pb events it appears that the minimum-bias triggering scheme has the best performance, but with pp events it appears to be the most inefficient (for the innermost chips at least). This is even more evident in fig. 7.17A, which compares minimum-bias and periodic triggering at different interaction rates for pp. And it is interesting to note that the combination of periodic triggers and triggered mode in

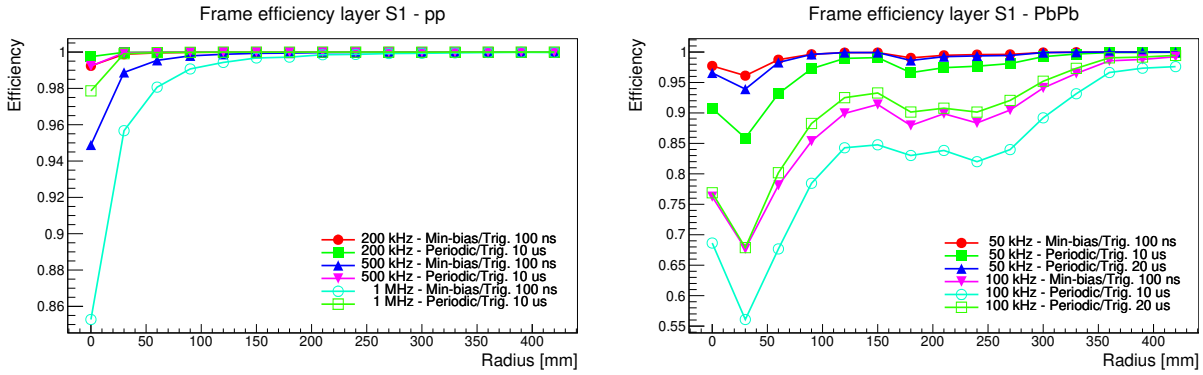
[13] 20 µs strobe was left out for pp, since those have the lowest data rate.

[14] On average the available capacity is 46 Mbit s^{-1} or 64 Mbit s^{-1} per OB-chip, depending on the stave type.



(A) pp @ 200 kHz (B) Pb–Pb @ 50 kHz

FIGURE 7.16: Frame readout efficiency versus radius simulated for FoCal at nominal interaction rates for ALICE. Note: The y-axis has been truncated and the range differs between the two graphs.



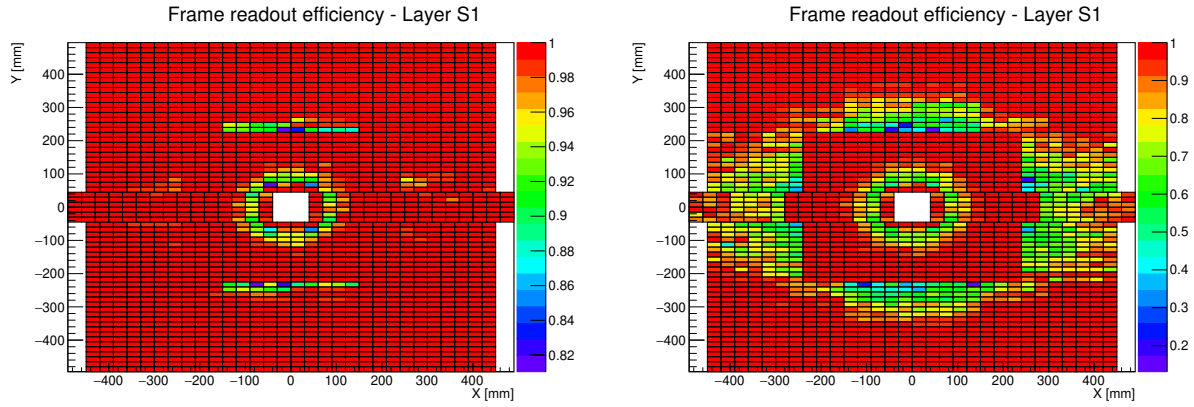
(A) pp (B) Pb–Pb

FIGURE 7.17: Frame readout efficiency versus radius simulated for FoCal. Note: The y-axis has been truncated and the range differs between the two graphs.

the ALPIDE performs better than the combination of periodic triggers and continuous mode. This can be seen for all the simulated cases in figs. 7.16 and 7.17, for both pp and Pb–Pb. There were hints of this in the simulation results for the ITS, but in the case of FoCal it is much more evident. In fact, for 200 kHz pp, triggered mode in the ALPIDE with a 10 µs strobe performs about as well as continuous mode with a 20 µs strobe (see fig. 7.16A).

The readout efficiency suffered a lot more in the Pb–Pb simulations. The worst efficiency is seen for chips at 30 mm from the gap, but as mentioned before the input data was generated with a too large gap, and one can assume that the chips at 0 mm have even worse efficiency. As high as 96% is achieved at 30 mm with minimum-bias triggering (but the efficiency for the innermost chips would likely drop below 90% with the right gap size). Also worth noting is the decrease in efficiency at 180 mm where the first chips in OB-mode are located. This is much better illustrated in fig. 7.18,

which clearly shows that the efficiency decreases where the OB-mode chips begin. It may be necessary to consider if more ALPIDE sensors need to operate in IB-mode in the proposed FoCal layout. But it should be noted that the efficiency for the first OB-chips is still better than the worst case among the innermost IB-chips.



(A) 50 kHz (B) 100 kHz

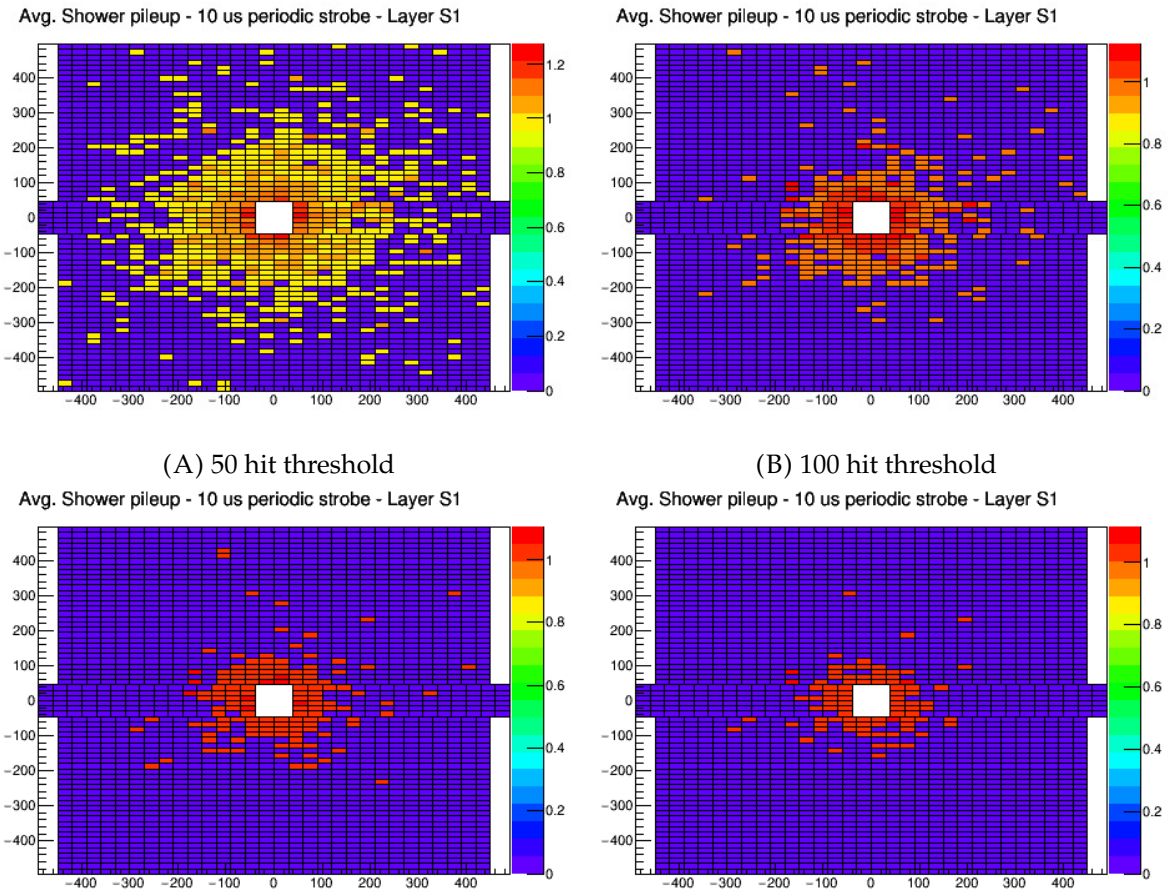
FIGURE 7.18: Map of frame readout efficiency for Pb–Pb simulations of FoCal.

7.3.2 Pileup of Showers

Pileup of events in a readout frame causes challenges for analysis and reconstruction of the events. For the range of pseudorapidity covered by the ITS, the tracks in the data are traced back to a primary vertex[15]. In the case of pileup, the tracking and vertexing algorithms in the experiment is able to locate secondary vertices for the additional events, which makes it possible to distinguish between more than one event in a readout frame.

The main purpose of the two layers of pixel sensor in the FoCal-E detector is to provide improved position resolution to the energy measurements performed by the low-granularity pad layers of FoCal-E and the detection of photon pairs from π^0 decays, which are suppressed to improve the measurement of direct photons [99]. The pixel layers do not provide the same degree of tracking and vertexing as the ITS, so pileup events can not be separated by looking for secondary vertices. Fortunately, the pad layers of FoCal-E offer timing information accurate to 25 ns (compared to the long integration time of the ALPIDE of $\approx 5\,\mu s$), which can help distinguish between events in a pileup [99]. However, the pileup of *shower events*[16] is still a concern. To

[15] An estimated origin point of the collision.
[16] An event with a large number of hits.



(A) 50 hit threshold

(B) 100 hit threshold

(C) 150 hit threshold

(D) 200 hit threshold

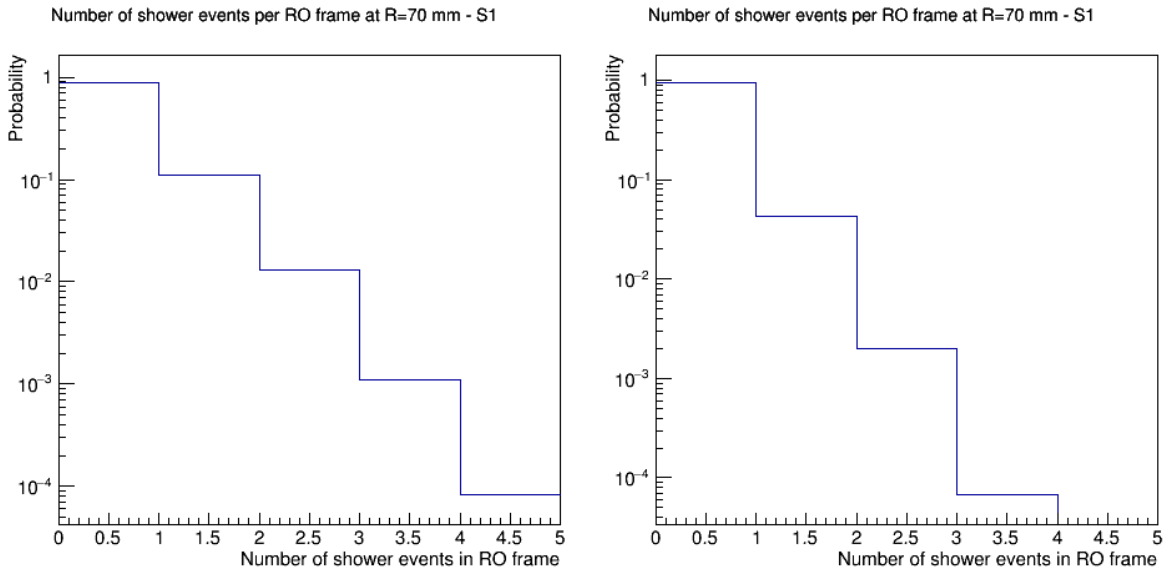
FIGURE 7.19: Average pileup of shower events per readout frame in FoCal (for frames that contain at least one shower). Simulated for 500 kHz pp.

resolve those it may be necessary to improve the timing accuracy of the pixel layers by the use of a shorter strobe, e.g. 1 µs [99].

To estimate the pileup of showers, a definition of what constitutes a shower is necessary. Since this information was not available in the input data to the SystemC model, a best effort approach was to define the shower by a threshold in terms of the number of hits per chip per event. Figure 7.19 shows the pileup of showers in the detector, simulated[17] for thresholds ranging from 50 to 200 hits with a 10 µs strobe length, for 500 kHz pp. Note that events that were below the threshold were excluded

[17]This did not require a full simulation of the readout of the ALPIDE, and a simplified simulation developed in ROOT was used instead of the SystemC model.

from the calculation[18], and the calculation was performed for each chip individually. Corresponding to fig. 7.19, the histograms of fig. 7.20 shows the distribution of shower pileup of the chips at a distance of 70 mm from the beam pipe [99]. Assuming the threshold of 50 hits, around 10% of the shower events would pileup in the same readout frame when a 10 µs strobe is used. This is a relatively high ratio, and further measures will be necessary to be able to distinguish between these events.



(A) 50 hit threshold

(B) 100 hit threshold

FIGURE 7.20: Pileup of showers per readout frame in FoCal for 1 MHz pp with a 10 µs strobe [99].

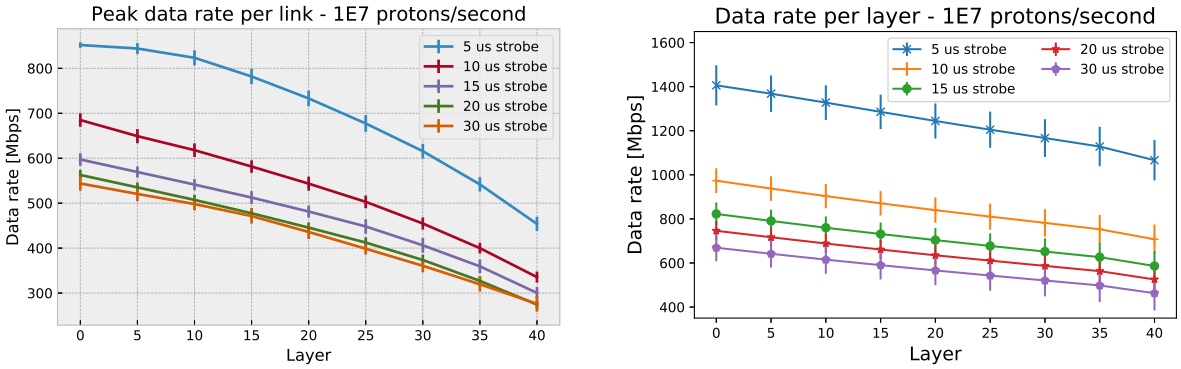
7.4 pCT Simulations and Results

The pCT detector will be used with a pencil beam which scans across the detector plane in a raster pattern. With this continuous beam of particles there are no events to trigger on and the data must be captured continuously. The triggering scheme foreseen for the detector is based on periodic triggers distributed simultaneously to the entire detector.

The simulations runs through a set of data from a MC simulation of a 230 MeV pencil beam scanning across the detector.

[18] A clarification may be necessary: A readout frame containing one shower event has a pileup of one in this calculation. If it contains two shower events, the pileup is two. Readout frames containing no showers were excluded from the calculation. What is left is the average number of shower events per readout frame, but only among the readout frames that contain at least one shower.

In the simulation, the detector plane consisted of twelve IB-staves[19] per layer. Layers 0, 5, 10, 15, 20, 25, 30, 35, and 40, were simulated. Trigger rates of 200 kHz (5 µs), 100 kHz (10 µs), 66.7 kHz (15 µs), 50 kHz (20 µs), and 33.3 kHz (30 µs) were simulated. The strobe length was 100 ns shorter than the trigger period in each case, and each trigger rate was simulated for both the triggered and continuous modes of the ALPIDE chip.



(A) Peak data rate per link

(B) Average data rate per layer

FIGURE 7.21: Simulated data rates for the pCT detector in continuous mode. The peak data rate (A) is the highest rate during a 10 µs interval, and the average rate (B) is the average for the whole duration of the simulation. In both cases, the average of all links in a layer is shown, with the error bar indicating the spread in values among the links. Note: The y-axis has been truncated. [36]

As with the simulations for the ITS and FoCal, the goal was to determine data rate and readout efficiency. But the pCT differs from those detectors in that the pRU handles a much larger number of ALPIDE chips, and has a much lower bandwidth than that of the ALPIDE data links combined. So, in the case of the pCT, an additional requirement was to record how the data rate varies over time, not just the average and total. But the data rate must be estimated over an interval, which was set to 100 µs for all of these simulations.

Figure 7.21 shows the data rates that were estimated with the simulations, for the different trigger rates (and strobe lengths) and at each layer[20] that was simulated. Only continuous mode is shown (triggered mode was omitted since the data rates were almost identical). Interestingly, the average data rate per layer is not significantly higher than the peak data rate per link[21].

[19] 9 ALPIDE chips per IB-stave.

[20] For each layer of the pCT there will be only one pRU, which connects to all the ALPIDE chips of its respective layer. But the term RU is used in the simulation and results, since the simulation model was originally designed for the ITS.

[21] An average rate was calculated over each 10 µs interval per link, and the interval with the highest average was chosen as the peak for that link. In the plot, the peak rates have been averaged again over

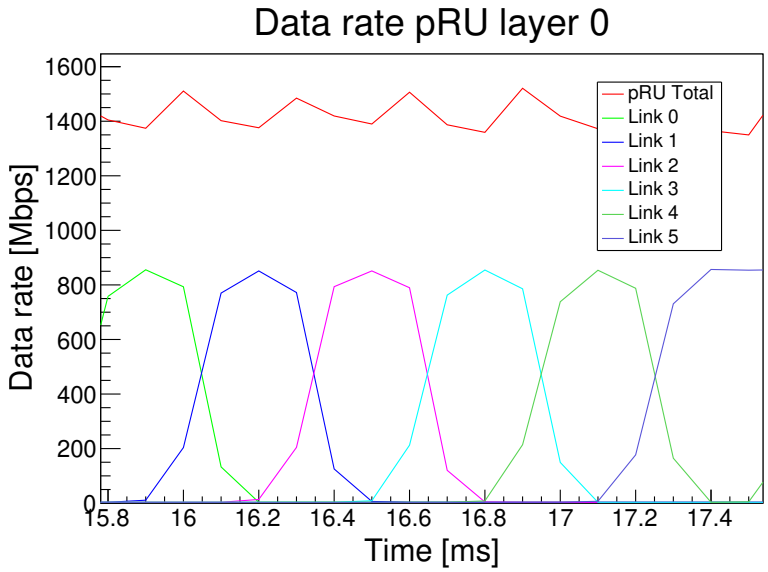


FIGURE 7.22: Data rate shown over time for the first layer of the pCT detector, simulated in continuous mode with 5 µs strobe (only a part of the full simulation is shown). A selection of ALPIDE data links are shown, corresponding to the chips that the beam was focused on. The total for the layer (RU total) is also shown, which is the sum of the individual ALPIDE data links for the layer (there are more links than shown in the figure that contribute to the total). [36]

But the result is not surprising. The pencil beam has a diameter of around 25 mm and is typically focused on *one* (or a few) ALPIDE chips at a time. While that would trigger a large number of pixels in that particular sensor chip, the other sensor chips of the same layer would register few or no hits. Figure 7.22 shows this effect in action for a select number of data links, as well as the total, for the first layer. In principle, the beam should become less focused at deeper layers, because of straggling effects and secondaries, but the simulated results in fig. 7.21 still showed a reduction in data rate at deeper layers.

And finally, the readout efficiency is shown in fig. 7.23. In the case of the pCT, this efficiency is reported in terms of actual pixel hits, instead of readout frames. For each trigger, the vast majority of sensor chips will report an empty frame, with the exception of the few chips that are in the vicinity of where the beam is focused. The efficiency in terms of readout frames would be heavily skewed towards a high efficiency, unless the frames were weighted based on the amount of data they contain. Instead, this measure in terms of pixel hits gives an exact representation of how much of the data is successfully read out.

all available links in the layer, and the error bars illustrate the variation in peak data rate between the links of that layer.

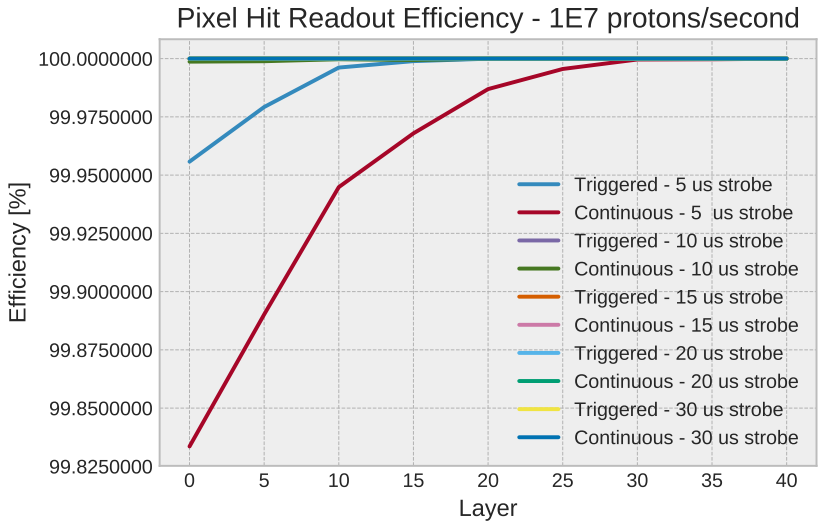


FIGURE 7.23: Readout efficiency for the pCT in terms of pixel hits. Note: The y-axis has been truncated and the range differs between the two graphs. The efficiency is approximately 100% in every case.

To briefly summarize the results, it appears that the pCT should be able to operate with a high efficiency when scanned with a pencil beam with an intensity of 1×10^7 particles per second and a scanning speed of around $100\,\mathrm{m\,s^{-1}}$ in the X-direction. And while there is an observable difference in efficiency between the simulated parameters, the overall efficiency is quite good in either case; the worst case is 99.8%, for 5 µs trigger period and strobe, with the chip in continuous mode. The peak data rate per layer is constrained by the capabilities of an individual ALPIDE chip, and the pRUs should easily be able to cope with the incoming data. The bottleneck for this detector will be the ALPIDE sensor itself, and not the external readout system.

As final remark for the pCT, it should be noted that the scanning direction in the MC data (see figs. 6.23 and 6.25) runs perpendicular to the columns of the ALPIDE chips in the SystemC-based simulation. An alternative would have been to run the beam in parallel with the columns of the chip. This is more favorable for the Priority Encoders and readout of pixel clusters in the chip (see appendix B.2), which operate on a pairs of columns, and could lead to a reduction in the amount of data to read out. However, it would not be a good utilization of the parallel architecture of the readout circuit in the ALPIDE (see fig. B.8), and would likely lead to a reduction in the overall readout speed and increase the chance of data loss (e.g. due to busy violations). The parallel scanning direction has not been simulated.

Chapter 8

Conclusions

ALICE is an experiment at the LHC which main objective is to study QGP and other open questions in QCD, primarily through collisions of lead nuclei (Pb–Pb). The performance and capabilities of the experiment will be vastly expanded for Run 3 of the LHC by the detector upgrades that are under installation in LS2. One of these upgrades is of the ITS, which will be replaced in its entirety by a new system. The new ITS, formally referred to as the ITS upgrade, is the topic of this thesis, and the main focus is on the readout electronics for this new detector.

The new ITS is fully based on pixel sensors and is currently the largest detector of its kind, with its $\approx 13 \times 10^9$ pixels spread over an active area of $10\,\mathrm{m}^2$ [106]. Compared to its predecessor it has improved tracking resolution and reduced material budget. And it can handle a vast increase in interaction rates, making it ready for the planned increase in luminosity of runs 3 and 4, which leads to a great increase in data collection and improved statistics for the experiment. The core component of the ITS, which makes this all possible, is the ALPIDE pixel sensor chip; a custom MAPS chip implemented in the 180 nm TowerJazz CMOS imaging process, and developed at CERN for the ALICE collaboration.

The combined bandwidth of the data links from the detector amounts to 1 497,6 Gbps[1]. The readout electronics for the ITS consists of 192 RUs, featuring high-speed FPGAs, with the power and bandwidth to process and transmit the detector data over optical links to a server farm. But the role of the RUs is not limited to readout; they are responsible for **all** communication with the detector, which includes control and trigger distribution. The RUs are located a few meters from the IP in the experiment and will be exposed to radiation. Consequently, radiation tolerance has been an important concern during the design of the RU circuit board, and in the FPGA designs for the RU.

Full-scale testing of the ITS detector upgrade started with the commissioning period in May 2019. At that point the detector was being gradually assembled in a clean

[1] This is also the peak throughput of the detector, but the average throughput is significantly lower.

room at CERN's Meyrin site. In December 2020, it was disassembled and moved for installation in the ALICE experiment at LHC point 2, which was completed on May 12, 2021.

8.1 Readout Electronics and FPGA Designs

The detector and readout electronics, along with the associated systems for power, triggering, control and data readout, have been in continuous operation with data taking[2] throughout the commissioning period[3]. At this point the readout electronics and FPGA designs allow for full operation of the detector, and meet the specifications for data readout, trigger rates, and other requirements. The FPGA designs have been under active development during the commissioning, and these activities will continue even after Run 3 of the LHC begins[4].

Several irradiation campaigns of the readout electronics were performed to test and characterize the radiation tolerance of the electronics, as well as to test and develop mitigation strategies for the FPGA designs. These strategies are based primarily on: TMR, with correction of important registers such as FSM state or WB configuration registers; the use of ECC for memories and FIFOs; and the implementation of configuration and blind scrubbing of the SRAM-based main FPGA, by the use of a flash-based auxiliary FPGA and an external flash memory. Remote updates of the configuration of the FPGAs is possible, with two redundant paths for the main FPGA: a slow "fail-safe" path, and a fast path via the main FPGA itself[5].

Control of the main FPGA, and indirectly the detector's sensor chips and the systems that power them, is performed using the optical links of the RUs. But redundancy is also offered here: a CAN interface allows for full control of the RU, and this interface is actively used by the DCS system to monitor certain temperatures and voltages. Also discussed in this thesis is a custom CAN controller and high-level protocol, which were developed specifically for the main FPGA design of the RU.

The DCS system can also monitor internal temperatures and voltages in the sensor chips. This is performed indirectly using the RUs' GBT or CAN interfaces, in order to access the control links to the sensor chips. But there is a critical limitation; the control links are also used for triggering, which the control transactions for monitoring must not interfere with. A solution to the problem is presented in this thesis; the *Alpide*

[2]Cosmic and test patterns as there are no beams available before LHC resumes operation.
[3]Partial operation of the detector started before it was fully assembled.
[4]Scheduled for March 2022.
[5]Requires the main FPGA to be configured and running already.

Monitor[6] module in the main FPGA design allows for autonomous monitoring of the sensor chips, and synchronizes the control transactions with abort gaps in the LHC fill pattern to guarantee that triggers are not blocked by its operation.

Radiation-tolerant CAN Controller

The CAN controller of the main FPGA design was designed to be fully CAN 2.0B compliant. An extensive simulation test suite verifies the compliance, and also includes steps to test it against an existing CAN 2.0B compliant controller. And naturally, hardware implementations of the controller in FPGAs have been tested with commercial CAN hardware. It should be mentioned, however, that it has not yet been verified against the *Bosch VHDL Reference CAN* framework, and the radiation tolerant design has not been proven with a beam-test. But hopefully, these two issues will be addressed in the future.

The project has been made publicly available, and is relatively well documented and easy to use. The controller has several configurable options; for instance, not all applications require radiation tolerance, and the TMR can be disabled to save resources. The controller can be controlled via a simple direct interface, but an AXI-lite bus interface is also available. Support for other bus-interfaces can easily be implemented. Anyone who seeks to implement a CAN controller in the fabric of an FPGA may find this project useful, and it should be especially suited for applications in radiation environments, such as other physics experiments or space applications.

8.2 Simulations

The commissioning period has given valuable experience operating the detector with test patterns and cosmic runs, but it has been without the beams of the LHC. Data taking of collisions in the experiment will subject the detector to a much higher load and occupancy. But we are not left entirely in the blind. Extensive simulations have been performed for the ITS, such as those described in this thesis. Several combinations of running modes and parameters were simulated to predict readout efficiencies, data rates, and expected pileup per frame. Our results predict that the detector should be able to operate with a high efficiency at the LHC. This is especially true for pp-runs, where the detector can run at much higher interaction rates than it was designed for. Rates beyond the specifications should also be possible for Pb–Pb runs, although it depends on some parameters such as the strobe length in this case.

[6] A proof of concept implementation is available for this module, but it has not been put into use in the FPGA design yet.

Furthermore, a dedicated busy system was deemed unnecessary for the detector, based on the simulations and results described in this thesis. A clear benefit is a reduction in the overall complexity of the system, as well as a lower cost of design and manufacturing. Nevertheless, a concept for a possible implementation of a busy system is introduced in appendix F. The BU in this system is based on the RU design, and can be implemented with relative ease if the need should arise in the future.

Although the simulation model described in this thesis was intended for the ITS originally, it has been successfully adapted and used for simulations of two other ALPIDE-based detectors: the FoCal detector at ALICE and the pCT at UiB.

While FoCal was in the early conceptual stages, the simulation model was used to provide early insights into the possibility of using the ALPIDE sensors for the silicon tracking planes of the detector, and some of the layout and specification of the tracking planes has been based on this work. The FoCal project was subsequently approved by the LHC Experiments Committee (LHCC), and its development is in progress. Additional adaptations and developments of the simulation model is underway at UiB, to be able to facilitate the FoCal project with further simulations.

Similarly, for the pCT, the simulations demonstrated that the focused pencil beam would primarily trigger pixels in one ALPIDE chip at a time. This effectively limits the amount of the data the readout electronics has to cope with, and the simulations also predict what beam intensities are suitable for the detector. In summary, the SystemC-based simulation model of the ALPIDE has provided simulated performance that has aided the design and development of three different detectors.

8.3 Outlook

The second long shutdown of the LHC has finally come to a conclusion after delays due to the Covid-19 pandemic. Run 3 is scheduled for the spring of 2022, and the experiments are now gearing up for data taking when operation of the LHC resumes. At the end of October 2021, when the first stable beams of protons were circulating in the LHC since the shutdown, data of the pp collisions in ALICE was captured by the ITS and the events were successfully reconstructed.

Development of the next upgrades of the experiment are already in progress. A planned replacement for the IB of the ITS is in the works. It is referred to as the ITS3, and will have the individual pixel chips replaced with a large pixel sensor die of unprecedented size, which has been thinned down to the point that it is flexible. The sensor will be folded into a cylindrical shape around the beam pipe, allowing for an even shorter distance to the collision point.

The FoCal detector, which has been discussed in this thesis, is planned for Run 4. Some of the development activities are taking place at UiB, such as design of the sensor staves and readout electronics. This also includes further simulations based on the work presented in this thesis.

Appendix A

List of Publications

A.1 Papers Published as First Author

1. "System simulations for the ALICE ITS detector upgrade"

   Voigt Nesbo, Simon, Alme, Johan, Bonora, Matthias, *et al.*

   EPJ Web Conf. vol. 245, 2020

   DOI: *10.1051/epjconf/202024502011*

2. "Implementation of a CANbus interface for the Detector Control System in the ALICE ITS Upgrade"

   Nesbo, Alme, Bonora, *et al.*

   PoS, vol. TWEPP2019, 2020

   DOI: *10.22323/1.370.0083*

3. "Simulations of busy probabilities in the ALPIDE chip and the upgraded ALICE ITS detector"

   Nesbo, Alme, Bonora, *et al.*

   PoS, vol. TWEPP-17, 2017

   DOI: *10.22323/1.313.0147*

A.2 Papers Published as Co-Author

1. "A High-Granularity Digital Tracking Calorimeter Optimized for Proton CT"

   Alme, Barnafoldi, Barthel, *et al.*

   Frontiers in Physics, vol. 8, 2020

   DOI: *10.3389/fphy.2020.568243*

2. "External scrubber implementation for the ALICE ITS Readout Unit"

   Ersdal[1]

   PoS, vol. TWEPP2019, 2020

   DOI: *10.22323/1.370.0136*

3. "Development of Readout Electronics for a Digital Tracking Calorimeter"

   Groettvik, Alme, Barthel, *et al.*

   PoS, vol. TWEPP2019, 2020

   DOI: *10.22323/1.370.0090*

4. "A Radiation-Tolerant Readout System for the ALICE Inner Tracking System Upgrade"

   Schambach, Alme, Bonora, *et al.*

   2018 IEEE Nuclear Science Symposium and Medical Imaging Conference Proceedings (NSS/MIC), 2018

   DOI: *10.1109/NSSMIC.2018.8824419*

A.3 ALICE Collaboration Papers

196 ALICE collaboration publications with co-authorship as a collaboration member[2].

[1] Because of a registration error only the first author is listed in the publication's entry on the website of the proceedings, although the co-authors were included in the header of the paper.

[2] Source: `https://inspirehep.net/`

Appendix B

Internal Readout Logic of the ALPIDE

B.1 Pixel Front-End and Multi Event Buffer

The block diagram in fig. B.1 shows the content of a pixel cell in the ALPIDE. It consists of the pixel diode, the analog front-end, and the digital Multi Event Buffer (MEB). The *STROBE* signal is used to latch a hit into the current *Hit Storage Latch*. It is actually a 3-line signal, *STROBE[2:0]*, which acts as a one-hot *write select* signal for the three pixel latches when the strobe is active. Likewise there is a 3-line *MEMSEL[2:0]* signal which is used as to read from the latches. The *STROBE[2:0]* and *MEMSEL[2:0]* are global signals which are distributed to all pixels in the ALPIDE, so the MEB of the pixels operate in concert across the chip, essentially combining the 3-bit MEB of each pixel into a large 3-event deep MEB for the entire pixel matrix of 1024 × 512 pixels.

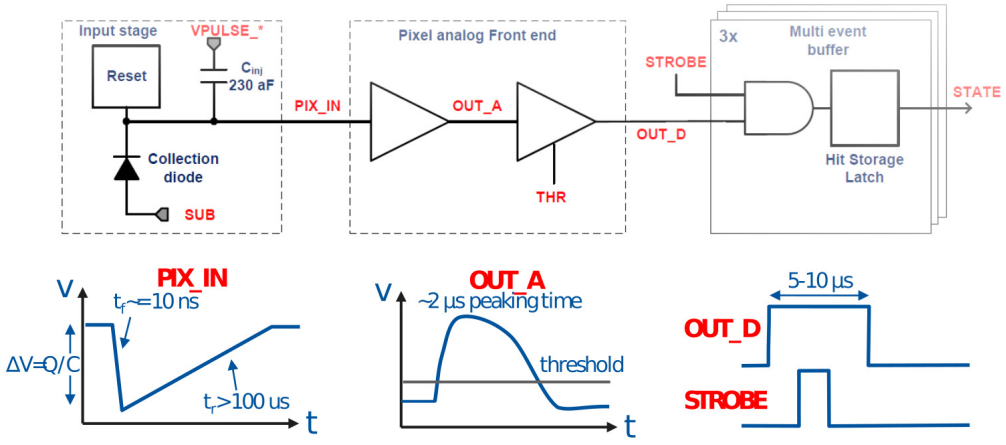
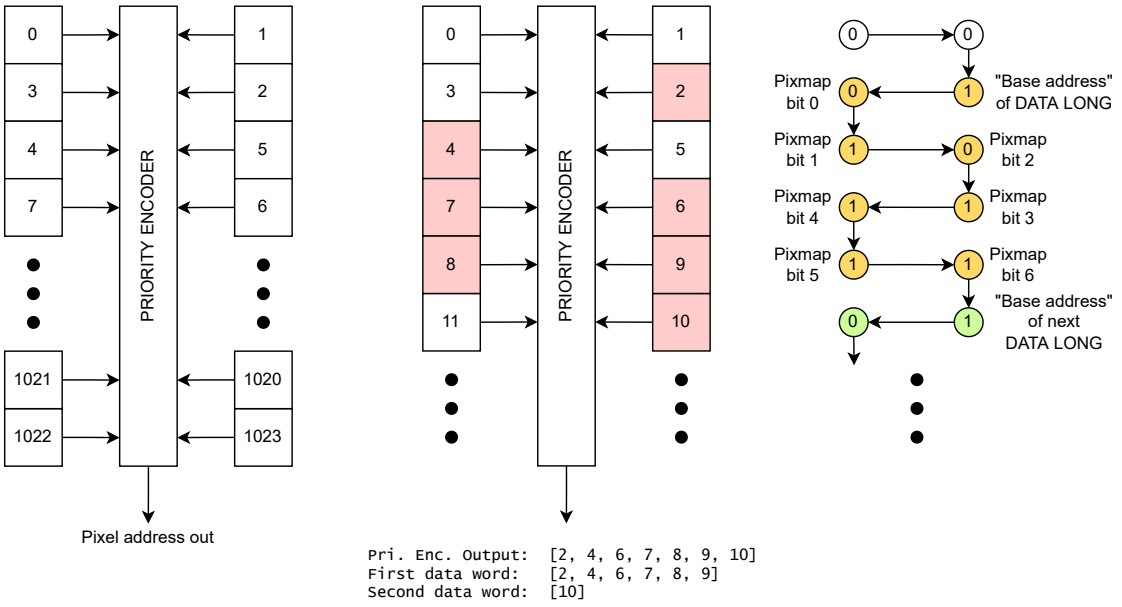


FIGURE B.1: Block diagram of ALPIDE pixel cell [26].

B.2 Priority Encoder

The priority encoder is a large asynchronous circuit of combinational logic. The current pixel latch of each pixel in a double-column connects to the inputs of the encoder.

The output of the encoder is the first pixel address within the double column that has a hit, or in other words, the first pixel that has a high or logical one stored in its register. The addressing of pixels in the priority encoder is illustrated in fig. B.2A. Readout and clustering of these pixels, as shown in fig. B.2B, is explained later in B.5.3. As an address is read out from the priority encoder by the digital readout logic, the pixel register that was read is cleared, and the priority encoder moves on to the next pixel with a hit. The digital readout logic that reads addresses from the priority encoder runs at 40 MHz, and hence has a clock period of 25 ns. Large combinational circuits have long path delays, and 25 ns may not be sufficient time for the priority encoder to update its output. Therefore the digital readout logic allows a minimum of two clock periods, 50 ns, before reading out the priority encoder. The readout speed of the priority encoder is configurable, with a fast setting of 50 ns, and a slow setting of 100 ns.



(A) Addressing of double columns. Pixels with hits are read out by the priority encoder in the specified order.

(B) Readout and clustering. Red pixels indicate hits. Pixel 2 is the first hit, and forms the base address of a (DATA LONG) data word (in yellow). A bitmap of the next seven pixels (i.e. pixels 3 to 9) is included in the data word, and indicates which of those pixels had hits. A new data word (in green) is started for pixel 10 since it comes after the bitmap.

FIGURE B.2: Priority encoder and double column readout.

B.3 Data Link

Data is transported off the chip with an LVDS interface provided by the *HSDATA* pins[1]. The interface is referred to as pseudo-LVDS in the ALPIDE manual, because the common mode voltage is 0.9 V [2] as opposed to 1.2 V which is the LVDS standard. However the LVDS interface on the *HSDATA* pins is still compatible with commercial LVDS receivers. The data is 10b8 encoded and transmitted with the LSB, and three different bit rates[3] are supported: 1200 Mbps, 600 Mbps, and 400 Mbps.

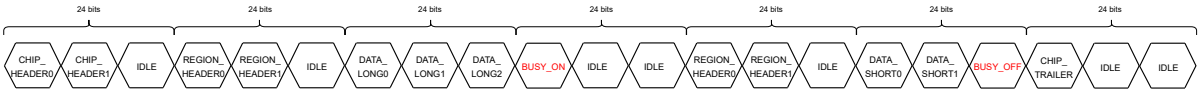


FIGURE B.3: Alpide data stream example (IB), with IDLE padding illustrated.

The data format is listed in fig. B.5. *IDLE* words are transmitted when the chip is idle or the next data word is not ready to transmit. After the chip has received a trigger and is ready to transmit data, it will start with the *CHIP HEADER* data word. It will then follow with a *REGION HEADER* word for the first region that has data, followed by *DATA SHORT* and *DATA LONG* words until all the hits in that region has been transmitted. This process is then repeated for every region that has data, and finally a *CHIP TRAILER* word marks the end of the data frame. If no pixel hits were registered for the event, the chip will transmit a *CHIP EMPTY FRAME* data word. The *CHIP EMPTY FRAME* word replaces the *CHIP HEADER* and *CHIP TRAILER* combination.

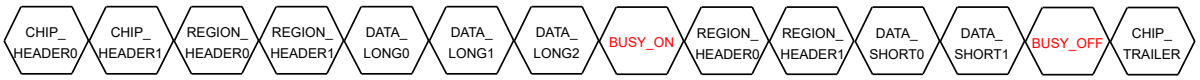


FIGURE B.4: Alpide data stream example (OB).

The *IDLE* words can be inserted anywhere during a data frame, with the limitation that 16-bit and 24-bit data have to be transmitted in full and can not be broken up by an IDLE word. IDLE words are inserted when the chip does not have another data word ready, and is also used as "padding" [4], as shown in fig. B.3. In the same fashion the chip may transmit *BUSY ON* and *BUSY OFF* data words during the data frame to communicate changes in busy status as fast as possible.

[1] See the ALPIDE operations manual for a complete pin diagram [26].

[2] According to the manual the 0.9 V common mode voltage was chosen to lower power consumption, and it simplified the design since it is half the supply voltage of 1.8 V.

[3] The bit rates listed here are with 8b10 encoding. Corresponding bit rates without encoding: 960 Mbps, 480 Mbps, 320 Mbps.

[4] When operating at 1200 Mbps and 600 Mbps, the chip will process 24 bits at a time. Data words smaller than 24 bits will be padded with the necessary number of IDLE words to add up to 24 bits.

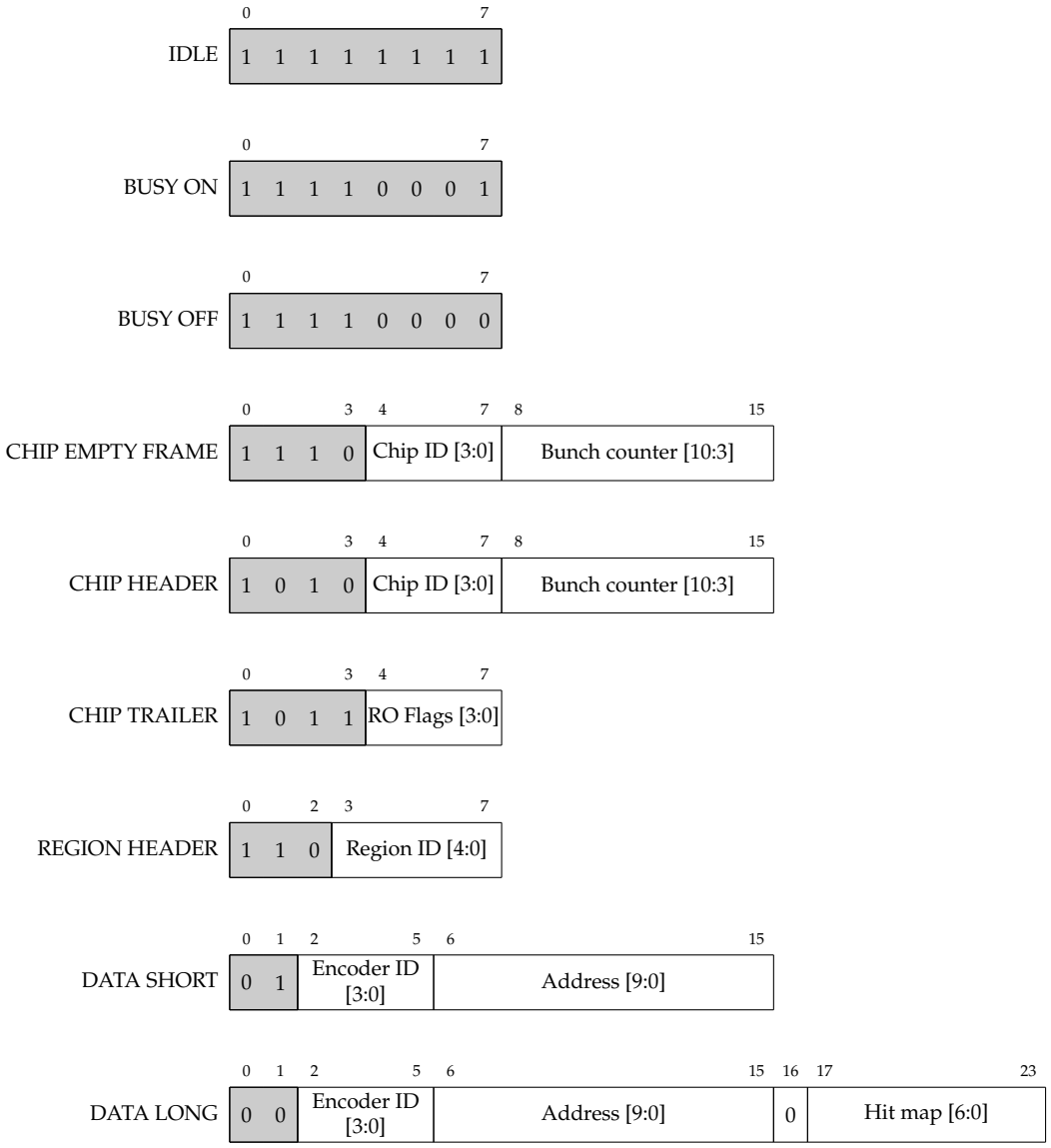


FIGURE B.5: ALPIDE data words. The data word type identifier field is shaded.

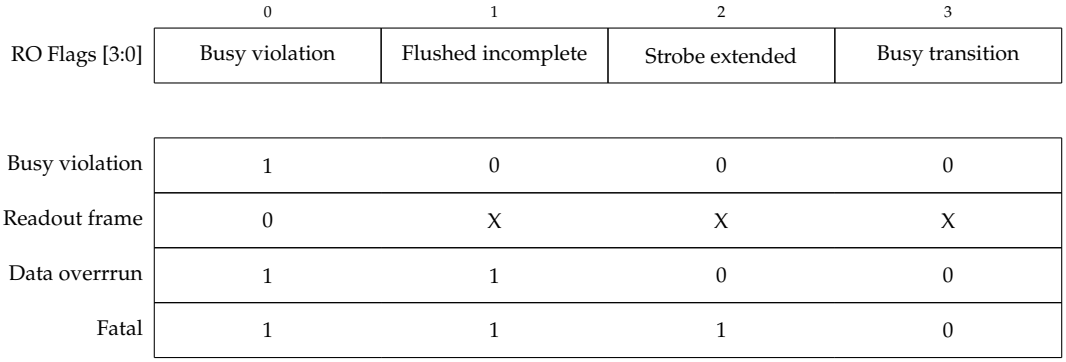


FIGURE B.6: ALPIDE CHIP TRAILER readout flag definitions, and special combinations of flags.

B.4 Control Link and Trigger Input

The ALPIDE chips are controlled using the differential DCTRL port[5]. The control interface is a half-duplex, bidirectional multi-drop bus which operates at 40 MHz, synchronous to the chips' clock input. The chips use Manchester encoding by default in order to maintain DC balance, and also support Manchester encoded input from the RU.

The idle state of the bus is logic 1, and a low start bit initiates a transaction. Transactions are composed of sequences of 10-bit characters, where each character consists of the low start bit, followed by eight data bits, and ending with a high stop bit. Sequences of characters can be transmitted with no gap between the characters, but a gap of up to 42 cycles is possible before the chip interprets the transaction as aborted.

B.4.1 Control Protocol

At the protocol level, transactions are initiated by transmission of one of the possible opcodes listed in table B.1. The opcodes are one character long, and may take additional arguments.

A read command takes four characters to initiate: *RDOP*, *Chip ID*, and two characters for the 16-bit address of the register to read. After a bus turnaround phase, the chip responds with three characters: the Chip ID, followed by two characters for the 16-bit data.

A write command takes six characters to initiate; *WROP*; *Chip ID*; two characters for the 16-bit register address; and finally two characters for the 16-bit data to write. The chip does respond to write commands. The write commands can be unicast or multicast. The same *WROP* opcode is used in either case, but a special *MULTICAST Chip ID* is used for multicast. Two multicast Chip IDs are defined; *GLOBAL BROADCAST*, binary 00001111, which is received by all chips; *OB MULTICAST*, where the 3 LSBs of the ID is 111, and the remaining bits address a specific module in an OB stave, allowing the write command to be broadcast to all the chips in that module only [6].

[5]The RU controls the IB and OB-master chips using the differential DCTRL port, and the OB-master forwards control transactions to the OB-slave chips on a single-ended CTRL port/pin.

[6]In the late development stages of the ITS, after production of the ALPIDE chips and staves had begun, it was discovered that the chips would erroneously interpret IB chip ID 7 as a broadcast message. Any write to chip number seven in an IB stave would effectively be broadcasted to the whole stave. This is rather unforunate, since any chip-specific settings (such as the pixel mask) will then have to be written to chip ID 7 first, and then these "broadcast" writes have to be undone individually for the remaining eight chips in the IB stave.

TABLE B.1: ALPIDE control bus opcodes. Reproduced from Alpide Operations Manual [26].

Opcode	Hex value	Purpose
TRIGGER	B1	Trigger command
TRIGGER	55	Trigger command
TRIGGER	C9	Trigger command
TRIGGER	2D	Trigger command
GRST	D2	Chip global reset
PRST	E4	Pixel matrix reset
PULSE	78	Pixel matrix pulse
BCRST	36	Bunch Counter reset
DEBUG	AA	Sample state in shadow registers
RORST	63	Readout (RRU/TRU/DMU) reset
WROP	9C	Start Unicast or Multicast Write
RDOP	4E	Start Unicast Read

The remaining opcodes are all broadcasted. The function performed by the opcodes *GRST*, *PRST*, *PULSE*, *BCRST*, *DEBUG*, and *RORST*, can be performed individually for a chip as well by a unicast write to the chip's command register.

The control signal from the RU connects to every chip in the ITS IB. But as we shall see later, the OB consists of a number of half-modules of seven chips, where one chip acts as a master, and the remaining six are slaves. The control signal from the RU only connects to DCTRL on the master chips in the OB. The master chip then forwards communication to and from the RU to the slave chips via the single-ended *CTRL* port.

B.4.2 Trigger Input

The ALPIDE chip does not have a dedicated IO pin for trigger input; it is communicated serially over the DCTRL input. At a first glance this may appear like a very slow way of distributing the trigger, since a standard six character multicast write takes 50 clock cycles (1250 ns) to execute. Which, of course, would beat the purpose of the new low-latency LM trigger signal used for ITS. Fortunately this is not the case. There are dedicated *OPCODEs* for triggers, and the chip features a fast trigger decoding logic in the Control Management Unit (CMU). The trigger word is detected after the first two LSBs of the OPCODE. This adds up to three clock cycles, or 75 ns, to decode the trigger OPCODE; one cycle for the start bit, and then the two LSBs.

B.5 Digital Readout Circuitry

Framing and readout of events is performed by several submodules in the ALPIDE chip, as illustrated in fig. B.7.

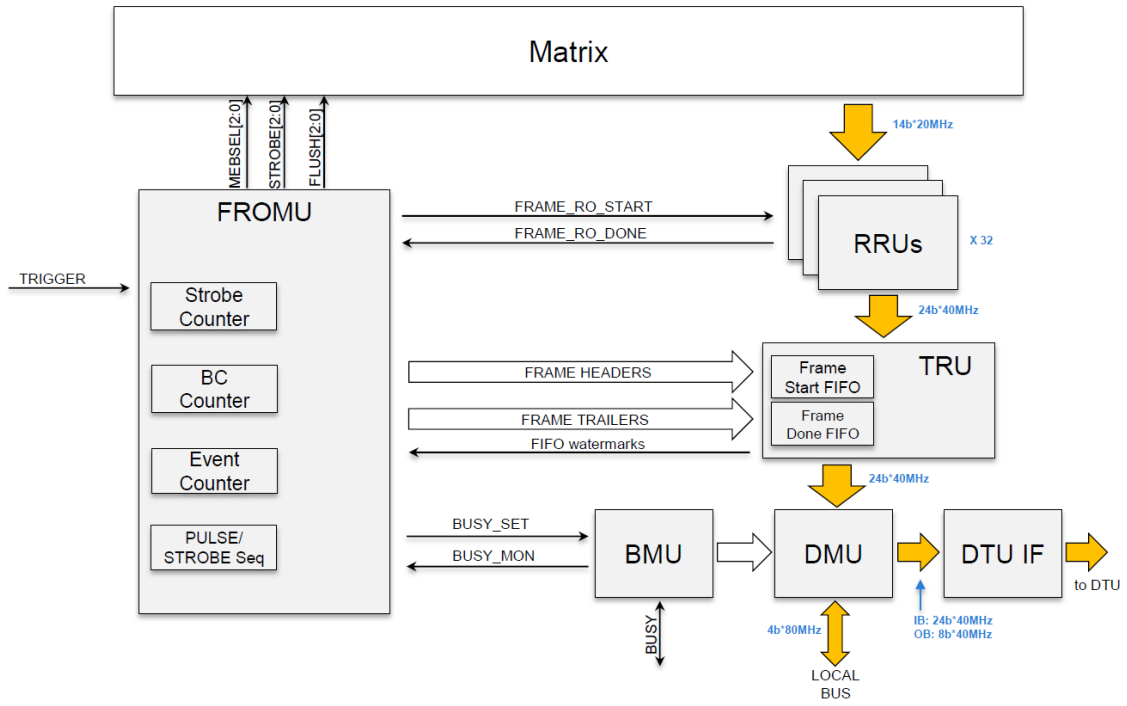


FIGURE B.7: Overview of readout architecture in the ALPIDE chip [98].

B.5.1 Frame and ReadOut Management Unit (FROMU)

The FROMU is the main module that controls the framing and readout in the chip, and keeps track of which buffers of the MEB are in use. When it receives a trigger input it initiates a strobe interval, and controls which MEB the hits are latched into. The chip has a 12-bit Bunch Crossing (BC) which wraps around at the value 3563, which corresponds to the number of bunches in one LHC orbit. Information about each trigger, such as the eight MSBs of the aforementioned BC counter, and flags to indicate busy status during the strobe interval, are stored in the two frame FIFOs[7]. These FIFOs are later used by the TRU to create frames of pixel hit data. It is also the responsibility of the FROMU to initiate readout of the MEB.

[7]One FIFO stores the BC counter when the strobe window opens, and the other FIFO stores the busy flags when the strobe window closes.

B.5.2 Busy Management Unit (BMU)

The chip will be in a busy state depending on MEB occupancy, or if the frame FIFO reaches a high level. The Busy Management Unit (BMU) receives input from the FROMU on changes in busy status, and communicates changes in busy status via the chip's serial data output. The *BUSY ON* and *BUSY OFF* data words are used for this, and these data words are placed by the BMU in a special busy FIFO which has priority over the data FIFO. For a chip in IB mode the busy input comes solely from the FROMU, but this is not the case for chips in OB mode. It was mentioned earlier that the OB slave chips share one local data bus to the OB master chip. Busy status should be communicated quickly, so the slave chips should not have to wait for their turn on the local data bus. Instead there is a shared busy signal with pull-up in the OB half-module, which the OB slave chips can pull low to indicate when they are busy. The BMU of the OB master chip monitor this line and determine a busy status for the entire OB half-module, and transmits the changes in busy status.

B.5.3 Region Readout Unit (RRU)

The RRU is responsible for readout of pixels from its region in the pixel matrix into a region FIFO, *clustering* of pixel hits into DATA LONG words, and control of readout from its own region FIFO. These tasks are performed by three FSMs:

- Region readout and clustering FSM
- Region valid FSM
- Region header FSM

Readout and Clustering

The readout and clustering FSM receives a signal from the FROMU when an event buffer is ready to be read out. It then reads out pixel hits from the priority encoders in its region in a round robin fashion, and creates clusters for neighboring hits. The pixel hit data that is read out is placed in a region FIFO which is 128 entries deep, and has a width of 24 bits. As pixel hits are read out they are stored as 24-bit *DATA LONG* words in the region FIFO. The first pixel hit sets the base pixel address in the data word, and if the following pixel hits are among the next seven addresses of the first hit, they are included in the same *DATA LONG* word with corresponding bits in the hitmap. This is a simple clustering algorithm that allows the region FIFO to be used more efficiently, and also reduces the amount of data to transmit off the chip. DATA LONG words that only contain one hit are translated into 16-bit *DATA SHORT* words

when they are read from the FIFO. For OB chips this will further reduce the data to transmit on the local bus between the chips, and on the differential data output of the master chip. For IB chips it does not make a difference because the chip processes 24 bits to transmit as a time, and does not break up data words. 8-bit IDLE words are transmitted for the remaining bits.

When there is no more data to be read out from the priority encoders, the FSM places a *REGION TRAILER* [8] data word in the FIFO. This is a special data word that serves to delimit events in the region FIFO, and its presence at the FIFO's output is also used to indicate when an event is empty or has been fully read out from a region. The REGION TRAILER word is only 8 bits, but the whole data word is triplicated in the FIFO to fill the entire 24 bits. This particular data word is then triple-voted to protect it from SEU effects. It may seem counterintuitive to only have SEU protection for this delimiter word and not the data words. However it should be possible to recover from corrupted data with protocol checkers in the RU and further upstream. But it is extra important to avoid corruption of the REGION TRAILER word since it is used as a delimiter. Errors in this data word could cause the chip to lose track of which trigger the data in the region FIFO belongs to.

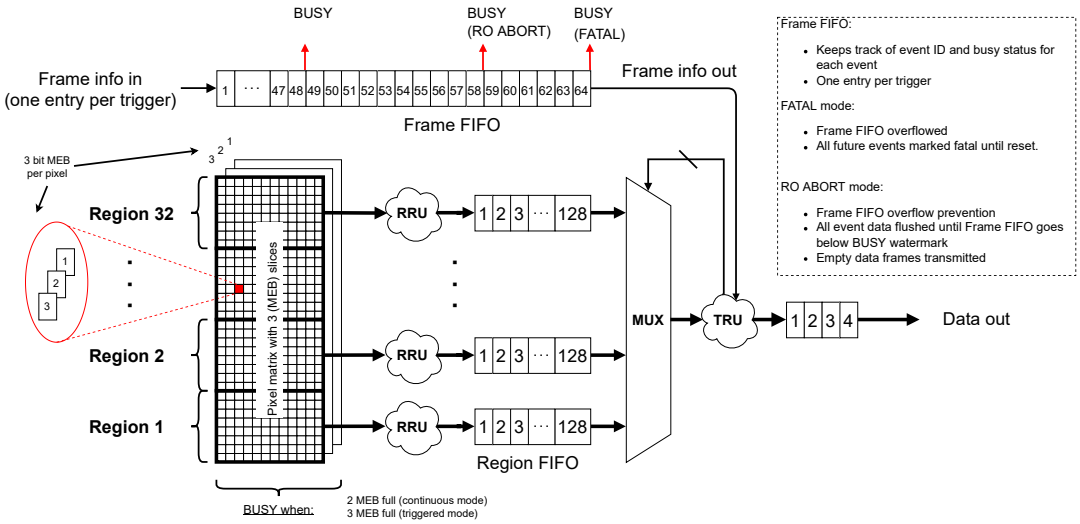


FIGURE B.8: Simplified illustration of the readout from the MEB and region FIFOs in the ALPIDE chip [97].

[8] Note that this data word is only used internally in the RRU, and will never appear on the output data stream from the chip.

Region Valid

A region valid signal indicates that the region has more data to be read out for the current event. The status of this signal is controlled by a dedicated FSM which monitors the output of the region FIFO, and outputs the valid signal as long as the next word is not a REGION TRAILER. Eventually as an event has been fully read out, the REGION TRAILER word should be the next word in the FIFO in all the regions. At this point all the RRUs will be instructed to pop the REGION TRAILER word, and prepare for the next event.

Region Header

A simple two-state FSM controls whether the RRU should output data from its FIFO, or a REGION HEADER word. In principle the REGION HEADER word could have been placed as entries in the region FIFO. But since the REGION HEADER word is essentially constant for a given region, ie. it does not differ between events, precious storage space in the region FIFO is saved by outputting it this way. One could also argue that this approach protects the REGION HEADER word from corruption due to SEEs.

B.5.4 Top Readout Unit (TRU)

Readout from the FIFOs of the RRUs and framing of the data is performed by the TRU. The frame FIFOs are part of the TRU. As mentioned earlier, the FROMU places an entry with the eight MSBs of the internal BC counter into the *frame start FIFO*[9], at the beginning of a strobe interval. When the strobe interval comes to an end, a corresponding entry with some status bits for the strobe is placed in the *frame end FIFO*. The TRU waits for the end FIFO to contain an entry, which indicates that there is an event to read out. Then it proceeds to pop a word from the start FIFO, and outputs the BC ID with either a CHIP HEADER or CHIP EMPTY FRAME data word, depending on whether the event is empty or not. If there is data to read out, the TRU will start reading from the RRUs, one at a time, and finally end with a CHIP TRAILER data word. At the end of the frame, the end FIFO is popped, and the status bits from this FIFO is included in the CHIP TRAILER or CHIP EMPTY FRAME words.

[9]There is a frame start FIFO and a frame end FIFO, collectively referred to as the frame FIFO in this text.

B.5.5 Data Management Unit (DMU)

The primary responsibility of the Data Management Unit (DMU) is to multiplex between different sources of data. The connections to and from the DMU are indicated in fig. B.7.

For the OB master and IB chips the data is forwarded to the DTU for transmission off the stave. The different sources of data include: the busy FIFO from the BMU; data coming from the TRU in the case of OB master and IB chips; data received by OB master chips on the local parallel bus.

Slave chips in the OB output their data on the local parallel bus using the DMU, and do not use the DTU and serial link. The OB master chip receives the data from the slaves via the DMU, and forwards this data to the DTU. The chips have internal pullup resistors on the *DATA[7:0]* pins for the local bus, and access to the bus is coordinated by a "virtual token". The order in which the chips access the bus has to be configured in the chips, and the chips monitor the bus to know when it is their turn, by inspecting the Chip ID which is included in CHIP HEADER and CHIP EMPTY FRAME words (fig. B.5). In this way the token order can be set up to skip a chip if necessary. This may be useful in the event that one chip has died and became unresponsive on the local bus, because in this scenario the next chip would wait forever for the previous chip to finish its access.

B.5.6 Data Transmission Unit (DTU)

The DTU is responsible for serialization of data and transmission off the chip on the *HSDATA* pins for the high-speed differential serial interface. The DTU is a custom design for the ALPIDE chip, and consists of a Double Data Rate (DDR) serializer, a Phase Locked Loop (PLL), and differential output driver. The PLL generates a 600 MHz clock from the 40 MHz reference clock, which is used to drive the shift registers for the serializer and the output stage. 30 bits of 8b10-encoded data are received by the DTU every 40 MHz clock cycle. Even and odd bits from the sequence are run through two shift registers at 600 MHz, and bits from the two shift registers are multiplexed on rising and falling clock edges to achieve the DDR output. This adds up to the bit rate of 2 bits $\times$ 600 MHz = 30 bits $\times$ 40 MHz = 1200 Mbps.

Data output at 600 Mbps and 400 Mbps is achieved by simply repeating each bit two times, or three times, respectively. The DTU clock still runs at 600 MHz in any case.

There is a separate module for control of the DTU, the Data Transmission Unit Logic (DTUL), which is not clearly indicated in fig. B.7. The DTUL works as a layer between the DTU and DMU, as illustrated in fig. B.10 from the ALPIDE manual. The

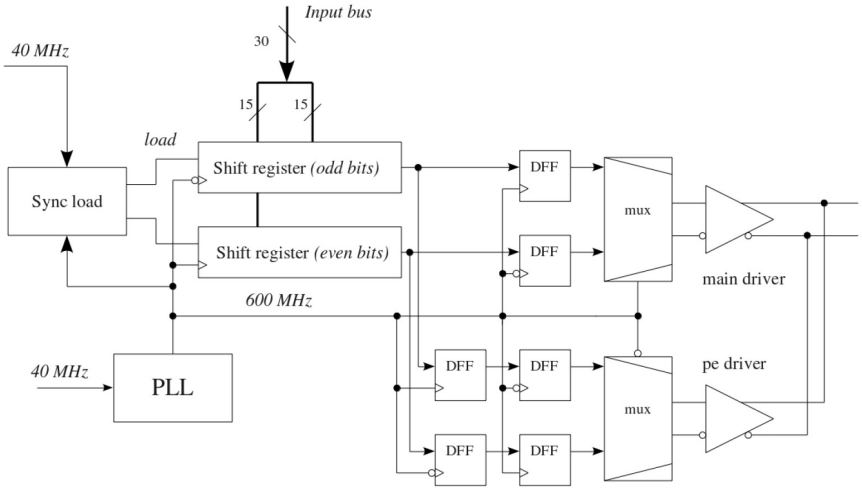


FIGURE B.9: Simplified schematics of the DTU in the ALPIDE [108].

DTUL implements the 8b10 encoding, and feeds the DTU with the 30 bits of 8b10 encoded data every 40 MHz clock cycle.

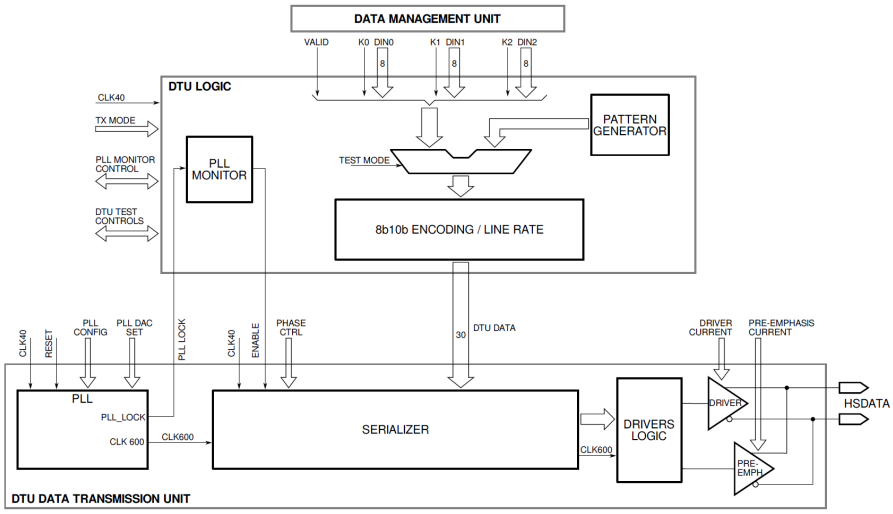


FIGURE B.10: Block diagram of the DTU and DTUL in the ALPIDE [26].

Appendix C

Protocols for Trigger, Readout, and Control over GBT

This appendix outlines the different protocols used by the RUs on the GBT links. This includes the protocols that allow for communication between the RUs and CRUs, and also the trigger protocol between the CTP/LTUs and the RUs. This includes protocol words that are used for control, as well as the protocol words and data formats to implement transmission of sensor data from the FEEs to the FLPs via the CRUs.

C.1 GBT Frames

The base GBT frame (table C.1) consists of 120 bits, of which 80 bits are for payload data. A 32-bit Focused Error Correction (FEC) accounts for most of the other 40 bits. Of the remaining bits there are four bits dedicated for *slow control*, of which two are for communication with the GBT-SCA chip, and a four bit header field. The header bits are used for synchronization and for identification of a frame as either an idle frame or a data frame. The latter is indicated with a *DATA_VALID* bit which is supplied with the 80 bits of payload data [44].

TABLE C.1: GBT frame format [44].

B range	Field Name	Field Width	Description
119:116	H	4	Header
115:114	IC	2	Internal Control
113:112	EC	2	External Control (for GBT-SCA
111:32	D	80	Payload data
31:0	FEC	32	Focused Error Correction (FEC)

C.2 Heartbeat Triggers and Frames

The LS2 upgrades of the ALICE trigger system introduces the concept of HB triggers and frames. The time of a heartbeat corresponds to the orbit time in the LHC, which is 88.924 µs [28]. The HBs are sent periodically from the CTP, either "directly" to the detector front-ends via the LTU, or more indirectly via the FLPs and CRUs. The heartbeats serve a dual purpose:

- Framing of data into HeartBeat Frames (HBFs) by the CRUs. The HBF consists of data from triggers delimited by two HB.
- The periodic triggers in continuous mode are derived from the HBs. For the ITS the triggers to the ALPIDE chips are generated by the readout electronics based on the HBs, as shown in fig. C.1. The HB period should be a multiple of the trigger period.

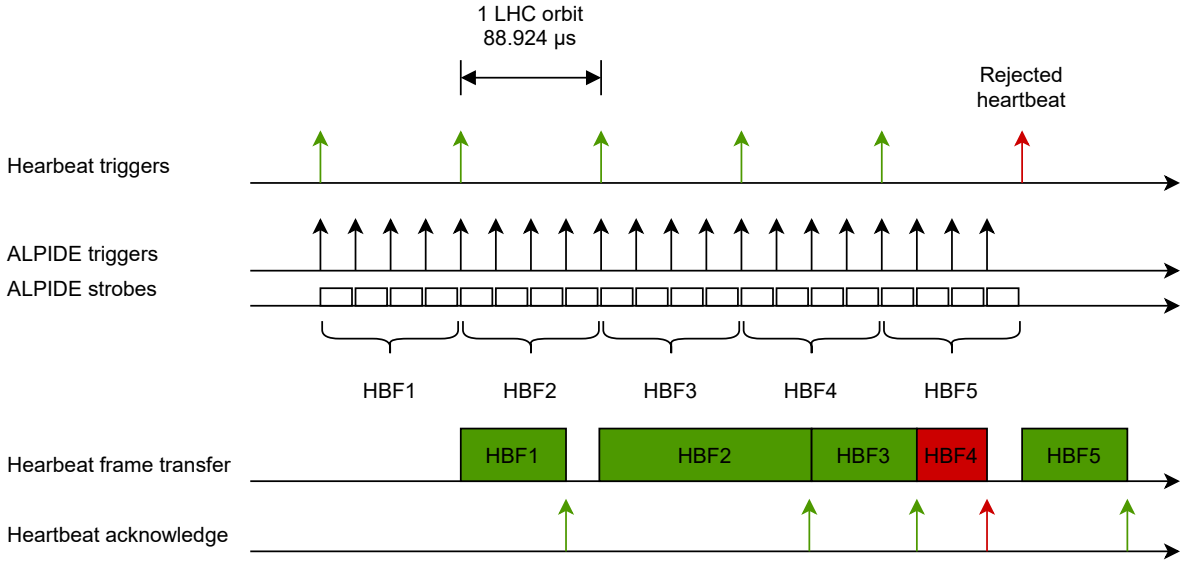


FIGURE C.1: Heartbeat triggers and frames, and continuous triggers for ITS.

The HBs are sent in both continuous and triggered mode, and HBFs are used to delimit data regardless of mode. In triggered mode, however, the CTP issues physics triggers to the RUs, which forward the triggers to the sensor chips with a minimal delay (only HB triggers are shown in fig. C.1). These triggers are independent of HBs, but as mentioned they are still associated with a HBF based on which two HBs they fall in-between.

For each HBF, the CRUs sends a HB acknowledge to the CTP to indicate if the HBF was successfully read out. The CTP pieces together this information in a *HB map*

for the entire detector. There is a basic throttling mechanism in place to reduce the trigger rate in the case where the overall data quality suffers from too many missing HBFs. Based on the quality of the HB map, the CTP makes a decision of whether the detector should capture data for the next HB or not. A HB trigger is always sent from the CTP on each orbit, but a HB accept/reject flag in the HB trigger message indicates if the front-end electronics should accept the HB and capture data [28]. This is also illustrated in fig. C.1.

C.3 CTP/LTU Protocols

Table C.2 shows the format of the trigger messages that the LTU or CTP transmits over GBT. The additional *DATA_VALID* bit of a GBT message is referred to as *TTValid* in this context, and indicates if the *TType* (trigger type) field is valid [109]. The orbit and BC fields are always valid, and the trigger level field is used by detectors which employ a two-level trigger scheme (the ITS uses a one-level trigger scheme, i.e. the LM trigger) [110].

The trigger type field (TType) is 32 bits wide, and the meaning of the different TType bits is explained in table C.3. Note that the different trigger bits are not mutually exclusive; combinations of bits are possible.

TABLE C.2: Trigger message over GBT. [110]

GBT Payload bit range	Field Name	Field Width
79:48	ORBIT	32
47:44	Trigger level / spare	4
43:32	BCID	12
31:0	TType (Trigger type)	32

TABLE C.3: Trigger Type (TType) bits. [110]

Bit	Name	Comment
0	ORBIT	ORBIT
1	HB	Heart Beat flag
2	HBr	Heart Beat reject flag
3	HC	Health Check
4	PhT	Physics Trigger
5	PP	Pre-Pulse for calibration
6	Cal	Calibration trigger
7	SOT	Start of Triggered data
8	EOT	End of Triggered data
9	SOC	Start of Continuous data
10	EOC	End of Continuous data
...	...	Spare
29	TPCsync	TPC synchronization
30	TPCrst	TPC reset
31	TOF	TOF special trigger

C.4 CRU Control Words

The CRU protocol uses the *DATA_VALID* field of the GBT frame to distinguish between "control word" frames (*DATA_VALID=0*) and data frames (*DATA_VALID=1*). The four MSBs of the payload data are used to denote the type of control word when *DATA_VALID* is zero [109]. The control words that have been implemented are listed in tables tables C.4 to C.7.

Idle words are transmitted over GBT when there are no other control words or data words to send. The SWT transactions are used by the ITS RU for control. And finally, the SOP and EOP are used to delimit data frames that are transmitted from the RU to the CRU. Sensor data is transmitted with *DATA_VALID=1* in GBT frames between the SOP and EOP control words using the RDH format [109].

Additionally, the protocol allows control words to be transmitted between the SOP and EOP words, since their *DATA_VALID* bit is zero they can easily be distinguished from the sensor data.

C.4.1 Idle Control Word

TABLE C.4: CRU Idle Control Word. [109]

Bit range	Field name	Description
79:76	Control code	"0000" for IDLE
75:0	Reserved	Not used

C.4.2 Start Of Packet (SOP) Control Word

TABLE C.5: CRU SOP Control Word. [109]

Bit range	Field name	Description
79:76	Control code	"0001" for SOP
75:60	Length	Length of packet
59:44	TTS busy	Busy information bits
43:0	Reserved	Not used

C.4.3 End Of Packet (EOP) Control Word

TABLE C.6: CRU EOP Control Word. [109]

Bit range	Field name	Description
79:76	Control code	"0010" for EOP
75:60	Length	Length of packet
59:28	Checksum	Checksum of packet
27:27	End flag	End of packet flag ('1' = yes, '0' = no)
43:0	Reserved	Not used

C.4.4 Single Word Transaction (SWT) Control Word

TABLE C.7: CRU SWT Control Word. [109]

Bit range	Field name	Description
79:76	Control code	"0011" for SWT
75:0	Parameters	Detector/implementation specific parameters

C.5 CRU Data Words

Blocks of data from the FEEs are transmitted using the RDH format defined for ALICE in Run 3. The format consists of RDH itself followed by a block of data. The RDH contains information such as: FEE ID; trigger, orbit and bunch crossing IDs; and size

of the data block. The data size can be zero, in which case there is no data block following the RDH.

RDHv6 Format over GBT from FEE to CRU

For FEEs that communicate to the CRU via GBT, such as the ITS RU, the RDH is transmitted as a sequence of four 80-bit GBT words.

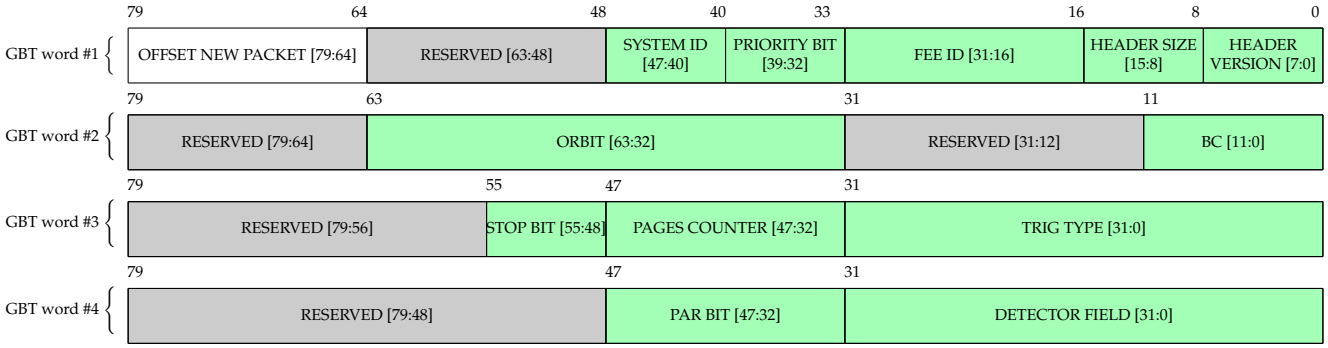


FIGURE C.2: RDH version 6 for GBT, sent from FEE to CRU. [111]

Each 80-bit GBT word maps to a 64-bit RDH word in the FLP. The upper sixteen bits of the GBT words are reserved and should always be zero, as they are not used.

RDH Format from CRU to FLP

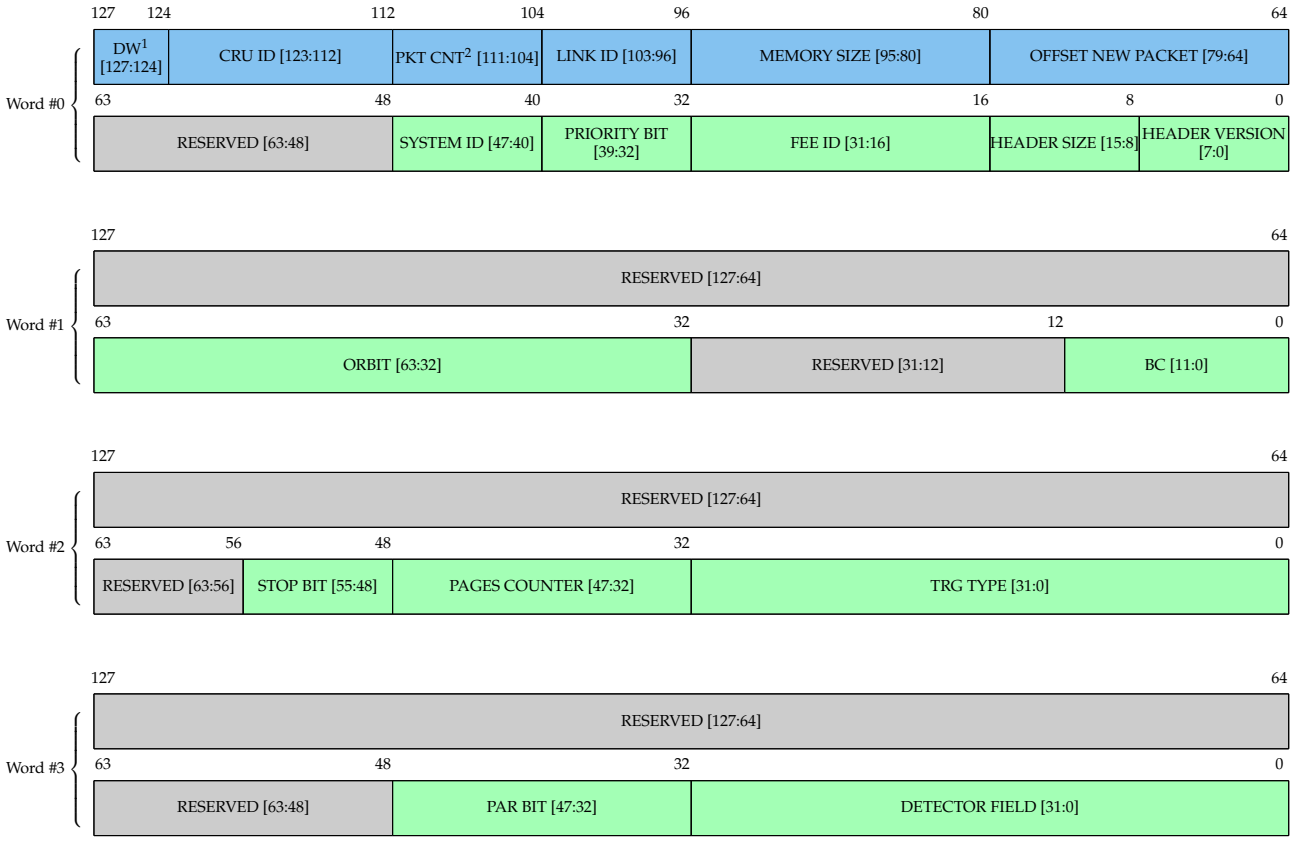


FIGURE C.3: RDH version 6 in the CRU, padded and converted to fit 128-bit words. [111]

RDH Format in FLP Memory

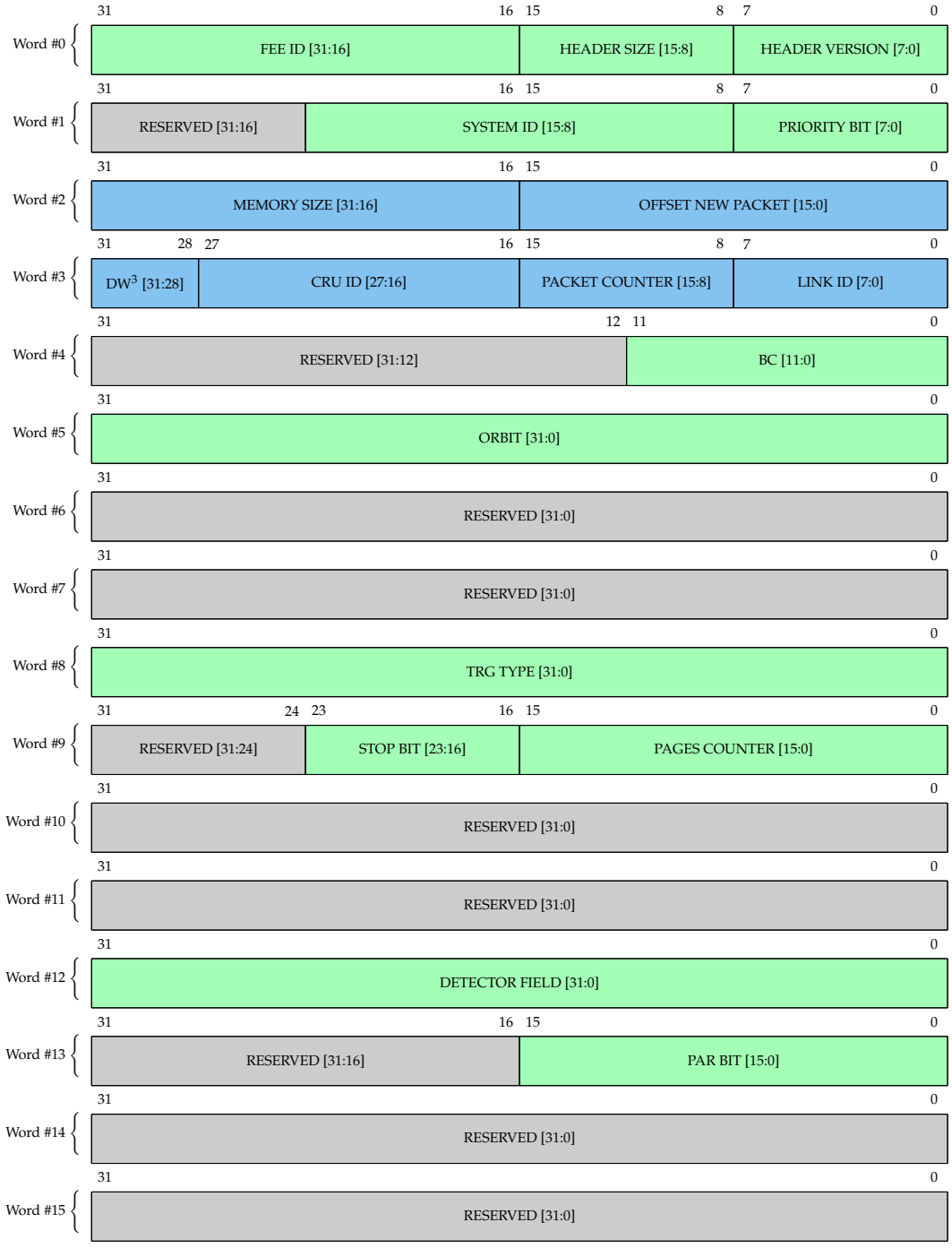


FIGURE C.4: RDH version 6 in the FLP memory, stored as 32-bit words. [111]

Appendix D

SystemC-based Simulation Model for ALPIDE and ITS

The source for the simulation is available in this git repository:
`https://github.com/svnesbo/alpide_its_sim.git`

The readme file has instructions on how to build project and how it is used. The compiled binary can be run directly from the top-level directory of the project:

```
$ bin/alpide_its_sim --help
```

Running the program with the *–help* or *-h* parameter will list available command-line parameters. If no parameters are supplied the simulation will start automatically using settings from *config/settings.txt* (if the file does not exists it is created with default settings), and will create a *run_X* directory under *sim_output/* (where X is increased for each simulation run). A copy of the settings that were used for a simulation run is stored in its directory.

A custom settings file can be specified with the *–config* or *–cfg* parameter, and the output directory can be changed with the *–output_dir_prefix* or *-o* parameter:

```
$ bin/alpide_its_sim -c my_settings.txt -o my_sim_output_dir/
```

Many of the available settings (listed in the next section) can be overridden on the command-line. Refer to the help options for more details.

D.1 Configurable Settings

Simulation settings for the SystemC simulation model are configured in a settings file. The file uses the standard .ini file format. An excerpt of a example settings file for the simulation is shown in listing D.1. The tables at the end of this appendix contains a full list of available settings.

LISTING D.1: Excerpt of example settings file for the simulation.

```
[alpide]
chip_continuous_mode=false
data_long_enable=true
dtu_delay=10
matrix_readout_speed_fast=true
minimum_busy_cycles=8
pixel_shaping_active_time_ns=5000
pixel_shaping_dead_time_ns=200
strobe_extension_enable=false

[data_output]
data_rate_interval_ns=100000
write_event_csv=true
write_vcd=false
write_vcd_clock=false
```

TABLE D.1: ALPIDE simulation settings.

Group	Setting	Default value	Comment
alpide	chip_continuous_mode	false	Enable continuous mode in ALPIDE (true), or use triggered mode (false)
alpide	data_long_enable	true	Enable clustering of adjacent pixel hits (and use of DATA LONG words)
alpide	dmu_fifo_size	64	Size of Data Management Unit (DMU) FIFO (the output "bottleneck" FIFO)
alpide	dtu_delay	10	Simulate delay (in clock cycles) introduced by serializing and encoding in DTU.
alpide	matrix_readout_speed_fast	true	Matrix priority encoder readout clock speed. True = 20MHz, false = 10MHz.
alpide	minimum_busy_cycles	8	Minimum number of clock cycles the chip must be busy before reporting BUSY ON
alpide	pixel_shaping_active_time_ns	6000	Equivalent to Time over Threshold (ToT)
alpide	pixel_shaping_dead_time_ns	200	Equivalent to rise time before ToT
alpide	strobe_extension_enable	false	Not implemented

TABLE D.2: Data output simulation settings.

Group	Setting	Default value	Comment
data_output	write_event_csv	true	Enable writing of event data (delta_t and multiplicity) to CSV file
data_output	write_vcd	false	Enable writing SystemC signals to Value Change Dump(VCD) file (requires lots of disk space for many events)
data_output	write_vcd_clock	false	Enable writing clock to VCD file (requires even more disk space)
data_output	data_rate_interval_ns	10000	Time intervals that data rate should be calculated over

TABLE D.3: General simulation settings. Settings in yellow are for ITS and FoCal only. The remaining settings are applicable to all simulations.

Group	Setting	Default value	Comment
simulation	n_events	10000	Number of interaction events to simulate
simulation	random_seed	0	Random seed. Setting to 0 will initialize random generators with a high entropy random seed.
simulation	single_chip	true	Simulate a single chip (true) or detector (false)
simulation	system_continuous_mode	false	Use continuous mode at system level, i.e. periodic triggers. Must be true for pCT simulation.
simulation	system_continuous_period_ns	false	Trigger period when system_continuous_mode is enabled
simulation	type	"its"	Simulation type (its/pct/focal)

TABLE D.4: Event generation simulation settings. Settings in yellow are for ITS and FoCal only. Red is for ITS and pCT. Green is for ITS. The remaining settings are applicable to all simulations.

Group	Setting	Default value	Comment
event	average_event_rate_ns	2500	Average interaction rate in nanoseconds
event	bunch_crossing_rate_ns	25	Bunch crossing rate/period in nanoseconds
event	monte_carlo_file_type	"xml"	MC input file type. Possible values: xml (ITS), binary (ITS), root (Focal/pCT)
event	qed_noise_path		Path to files with QED or noise events
event	qed_noise_input	false	Enable use of QED/Noise input files
event	qed_noise_feed_rate	false	Scale up/down feed rate of QED/noise input. Allows same QED files to be used at different interaction rates
event	qed_noise_event_rate	false	The time duration which the QED/noise in an input file is integrated over
event	random_hit_generation	true	Generate random events (don't use MC data files)
event	random_cluster_generation	false	Generate a random cluster of pixel hits around each hit
event	random_cluster_size_mean	4	Mean number of pixels in a random cluster
event	random_cluster_size_stddev	2	Standard deviation of number of pixels in a random cluster
event	strobe_active_length_ns	100	Strobe active time in nanoseconds
event	strobe_inactive_length_ns	100	Strobe inactive time in nanoseconds
event	trigger_delay_ns	1000	Total trigger delay in nanoseconds
event	trigger_filter_enable	true	Trigger filtering enable/disable (triggered mode only)
event	trigger_filter_time_ns	10000	Trigger filter time in nanoseconds. If filtering is enabled, and twotriggers fall within this filter time, the last trigger(s) will be filtered(removed).

TABLE D.5: FoCal-specific simulation settings.

Group	Setting	Default value	Comment
focal	monte_carlo_file_path		File with MC event data for Focal simulations
focal	staves_per_quadrant	3	Number of staves per quadrant to simulate

TABLE D.6: ITS-specific simulation settings.

Group	Setting	Default value	Comment
its	bunch_crossing_rate_ns	25	Bunch crossing rate in nanoseconds
its	layer0_num_staves	12	Number of staves to simulate in layer 0 (maximum: 12)
its	layer1_num_staves	16	Number of staves to simulate in layer 1 (maximum: 16)
its	layer2_num_staves	20	Number of staves to simulate in layer 2 (maximum: 20)
its	layer3_num_staves	0	Number of staves to simulate in layer 3 (maximum: 24)
its	layer4_num_staves	0	Number of staves to simulate in layer 4 (maximum: 30)
its	layer5_num_staves	0	Number of staves to simulate in layer 5 (maximum: 42)
its	layer6_num_staves	0	Number of staves to simulate in layer 6 (maximum: 48)
its	hit_density_layer0	18.6	Average hit density in layer 0 (hits per cm^2). Used when event/random_hit_generation is true
its	hit_density_layer1	12.2	Average hit density in layer 1 (hits per cm^2). Used when event/random_hit_generation is true
its	hit_density_layer2	9.1	Average hit density in layer 2 (hits per cm^2). Used when event/random_hit_generation is true
its	hit_density_layer3	2.8	Average hit density in layer 3 (hits per cm^2). Used when event/random_hit_generation is true
its	hit_density_layer4	2.7	Average hit density in layer 4 (hits per cm^2). Used when event/random_hit_generation is true
its	hit_density_layer5	2.6	Average hit density in layer 5 (hits per cm^2). Used when event/random_hit_generation is true
its	hit_density_layer6	2.6	Average hit density in layer 6 (hits per cm^2). Used when event/random_hit_generation is true
its	hit_multiplicity_distribution_file		File with discrete distribution for multiplicity to use when event/random_hit_generation is true
its	monte_carlo_dir_path		File with discrete distribution for multiplicity to use when event/random_hit_generation is true

TABLE D.7: pCT-specific simulation settings. The last nine settings in the table are for generation of a random beam (not used for the simulations in the thesis).

Group	Setting	Default value	Comment
pct	layers		Semicolon separated string that configures which layers to be included in pCT simulation. E.g. "0;5;10" for layer 0, 5, and 10.
pct	num_staves_per_layer	1	Number of staves to simulate per layer (maximum: 12)
pct	monte_carlo_file_path		Path to file with MC data for pCT simulation
pct	time_frame_length_ns	200	Should be set to the integration time used in the MC event file, e.g. 10 µs. All hits within such a time frame (integration time) has the same timestamp, and the simulation will spread the hits out in time randomly over the specified time frame.
pct	random_particles_per_s_mean	1E9	Mean particles per second for random beam
pct	random_particles_per_s_stddev	0.2E9	Standard deviation for random beam
pct	random_beam_stddev_mm	0.1	Beam radius is specified with this parameter
pct	random_start_coord_x_mm	-0.5	Start coordinate (X) of beam
pct	random_start_coord_y_mm	-0.5	Start coordinate (Y) of beam
pct	random_end_coord_x_mm	27.5	End coordinate (X) of beam
pct	random_end_coord_y_mm	2.0	End coordinate (Y) of beam
pct	beam_step_mm	3.0	Beam will move in steps of this size
pct	beam_time_per_step_us	125	Beam will move a step at these time intervals (in microseconds)

D.2 Output Data and Data Formats

A number of statistics and quantities are recorded during the simulation and stored to file in the output directory for the simulation run. This includes the time of triggers and size of event data, utilization of the MEB as well as counts of busy events etc. in the ALPIDEs chips. For each RU the time and duration of busy events (including busy violations etc.) per ALPIDE data link is stored, and also the data rate (for configurable time intervals) and statistics for the utilization of the ALPIDE data links with counts of each type of data word.

D.2.1 Simulation Output Files

An exhaustive list of output files is shown below:

- Alpide_MEB_histograms.csv
- Alpide_stats.csv
- physics_events_data.csv (ITS and FoCal simulation)
- pct_events_data.csv (pCT simulation)
- For each RU in the simulation:
  - RU_X_Y_busy_events.dat
  - RU_X_Y_busyv_events.dat
  - RU_X_Y_flush_events.dat
  - RU_X_Y_ro_abort_events.dat
  - RU_X_Y_fatal_events.dat
  - RU_X_Y_Data_rate.csv
  - RU_X_Y_Link_utilization.csv
  - RU_X_Y_Trigger_actions.dat
  - RU_X_Y_Trigger_summary.csv
  - Where X is the layer number, and Y is the stave number.
- simulation_info.txt
- timestamp.txt
- triggered_readout_stats.csv
- untriggered_readout_stats.csv

It would have been beneficial to use a standard data format, like the *.root* files of CERN's ROOT framework, but the simulation model was originally designed to not have a ROOT-dependency.

The comma-separated files should have a self-explanatory format (some details are still provided below). But some simple custom binary formats were used for the

.dat files in order to limit the file size. For the *trigger actions* the format is shown in table D.8. A value of either 0x00, 0x01, or 0x02, is stored per trigger, to indicate if the trigger was sent from the RU to the ALPIDE chips (on a specific control link), not sent, or filtered out. Table D.9 shows the format used for the *RU_X_Y_busy_events.dat* files which stores the time (and trigger ID) at the time of *BUSY ON* and *BUSY OFF* for each data link. And finally, table D.10 explains the format used for the *busyv*, *flush*, *ro_abort*, and *fatal* event files.

TABLE D.8: Data format for trigger files.

Field size	**Description**	**Comment**
64-bit unsigned	Number of triggers	Header
8-bit unsigned	Number of control links	
8-bit unsigned x *N*	Trigger action control link 1 Trigger action control link 2 ... Trigger action control link *N*	Trigger 0
8-bit unsigned x *N*	Trigger action control link 1 Trigger action control link 2 ... Trigger action control link *N*	Trigger 1
8-bit unsigned x *N*	Trigger action control link 1 Trigger action control link 2 ... Trigger action control link *N*	...
8-bit unsigned x *N*	Trigger action control link 1 Trigger action control link 2 ... Trigger action control link *N*	Trigger *M*

TABLE D.9: Data format for busy event files.

Field size	**Description**	**Comment**
8-bit unsigned	Number of data links	Header
64-bit unsigned	Number of events for data link 1	Header for data link 1
64-bit unsigned 64-bit unsigned 64-bit unsigned 64-bit unsigned	Time of BUSY ON Time of BUSY OFF Trigger ID at time of BUSY ON Trigger ID at time of BUSY OFF	Busy event 1 for data link 1
...		
64-bit unsigned 64-bit unsigned 64-bit unsigned 64-bit unsigned	Time of BUSY ON Time of BUSY OFF Trigger ID at time of BUSY ON Trigger ID at time of BUSY OFF	Busy event N for data link 1
Repeats for each data link		

TABLE D.10: Data format for files storing busy violations, flush, data overrun, and fatal events.

Field size	**Description**	**Comment**
8-bit unsigned	Number of data links	Header
For each data link:		
8-bit unsigned 64-bit unsigned	Chip ID Number of events	Header for chip 1
64-bit unsigned 64-bit unsigned 64-bit unsigned 64-bit unsigned	Trigger ID for event Trigger ID for event Trigger ID for event Trigger ID for event	Event 1 Event 2 ... Event *N*
Repeats for each chip with data for a data link		

D.2.2 Pixel Readout Statistics

The files *triggered_readout_stats.csv* and *untriggered_readout_stats.csv* contain pixel readout statistics.

The meaning of triggered and untriggered in this context indicates whether a pixel hit was associated with an event one would trigger on, or whether it was associated with a background event that one would not trigger on. For instance, statistics for every pixel hit in a collision event in the ITS simulation will go in the *triggered_readout_stats.csv* file. Noise, QED background from ultra-peripheral collisions, etc. will go in the *untriggered_readout_stats.csv*. But note that in the case of pCT simulation, all the pixel hits associated with the pencil beam will actually go in the *untriggered_readout_stats.csv* file.

The two files list counts of how many times a pixel hit was read out, for each chip ID in the simulation. Table D.11 shows an example of what the readout statistics could look like in this file. The table shows that there were 131 pixels that were never read out for chip ID 0, and for chip ID 1 and 2 there were 45 and 12 pixels that were never read out, respectively. For chip ID one there were 57234 pixels that were read out exactly one time, 6984 that were read out exactly two times, and so on.

Obviously a high count in column zero for readout count is bad, as it indicates loss of data. Readout counts of two and up is not ideal, because it indicates that the same data is read out more than once, which is a waste of bandwidth. Ideally as much data as possible is read out exactly one time.

For a chip c, we can define the number of pixel hits read out as:

$$\sum_{n=1}^{N} readout\_count(c, n) \tag{D.1}$$

TABLE D.11: Counts of how many times individual pixel hits were read out (i.e. oversampled when larger than 1), versus chip ID.

		Readout count					
		0	1	2	3	4	5
Chip ID	0	131	57234	6984	486	32	2
	1	45	43256	4896	385	21	0
	2	12	31811	3214	125	0	0

Where $readout\_count(c,n)$ is the content of the cell for chip c and readout count n, and N denotes the highest pixel readout count value. The number of pixel hits that were not readout out for chip c is:

$$readout\_count(c,0) \tag{D.2}$$

For all chips, the total number of pixels read out is:

$$A = \sum_{c=0}^{M-1} \sum_{n=1}^{N} readout\_count(c,n) \tag{D.3}$$

Where M is the number of chips. The total number of pixel hits not read out:

$$B = \sum_{c=0}^{M-1} readout\_count(c,0) \tag{D.4}$$

Based on these equations, we can define the pixel hit readout efficiency for all chips in the simulation as:

$$E = \frac{A}{A+B} = \frac{1}{1+\frac{B}{A}} = \frac{1}{1+\frac{\sum_{c=0}^{M-1} readout\_count(c,0)}{\sum_{c=0}^{M-1}\sum_{n=1}^{N} readout\_count(c,n)}} \tag{D.5}$$

And the pixel hit loss for all chips in the simulation as:

$$L = \frac{B}{A+B} = \frac{1}{1+\frac{A}{B}} = \frac{1}{1+\frac{\sum_{c=0}^{M-1}\sum_{n=1}^{N} readout\_count(c,n)}{\sum_{c=0}^{M-1} readout\_count(c,0)}} \tag{D.6}$$

Duplicate Pixel Hits

In principle the readout count should in many cases have a maximum theoretical value. For example, if we are running in continuous mode with a 5 µs strobe, and the time over threshold for the pixel hits is also set to 5 µs, then it is impossible to read out a pixel hit more than two times. For a readout count of three to be possible, the

time over threshold would have to be longer than the strobe, otherwise the time the pixel is active can not coincide with three strobe windows.

However, from the pixel readout statistics one may occasionally see higher readout counts that should be impossible to each. The reason for this is how the simulation model handles duplicate pixel hits. When the chip gets a hit in a pixel where a hit was already registered, and if both of these two pixel hits are active (over threshold) at the same time, then they are considered a duplicate pixel hit. In principle it would make the most sense to extend the active time (time over threshold) of the original hit, but for practical reasons this was tricky to implement in the simulation. Instead the simulation marks the additional hits as duplicates of the first one, and when the original hit is read out the readout counters for the duplicate pixel hits are increased as well. But as a result of this, we may see an increase in readout count that goes beyond what should theoretically be possible for an individual hit.

For the ITS simulations this does not occur frequently, since the hit coordinates are completely random in the collisions generated at the LHC. However for pCT simulations one will frequently see these duplicate hits, because the pCT uses a small pencil beam where the flux of protons is focused to a diameter of around 5 mm, and as a result one will quite frequently have protons hitting the same pixels.

D.3 Monte Carlo Simulated Events for ITS in the SystemC Model

MC simulations of the ITS were performed to generate discrete events for use with the SystemC-based simulation model described in chapter 6. The MC simulations were performed using the "ITSU[1] testbench for simulation and reconstruction" [96], which is part of the AliRoot framework. The testbench uses Pythia for generation of pp events in ALICE. And the Hijing event generator is used for Pb–Pb, along with an additional event generator of the AliRoot framework for the so-called QED events [95] (the background of electron-positron pairs associated with peripheral collisions in Pb–Pb [20]). In either case, pp or Pb–Pb, Geant is used to simulate the trajectory of the generated particles in the magnetic field of the experiment and interactions with matter and decays as they traverse through the detectors.

Configuration of the testbench is done in the Config.C file. This includes which event generators to use, and also which parts of the experiment to simulate. In principle, the entire experiment can be included in the simulation. But since the ITS is

[1] ITS Upgrade.

situated directly around the beam pipe, it is sufficient to simulate only the beam pipe itself and the ITS.

Code snippets of the Config.C files used for Pb–Pb, pp, and QED events, are shown in listings D.2 to D.4. Only the relevant parts of the Config.C file is included.

LISTING D.2: itsuTestBench setup for Pb–Pb.

```
Int_t generatorFlag = 2;

/* $Id: Config.C 47147 2011-02-07 11:46:44Z amastros $ */
enum PprTrigConf_t
{
  kDefaultPPTrig, kDefaultPbPbTrig
};

const char * pprTrigConfName[] = {
  "p-p","Pb-Pb"
};

static PprTrigConf_t strig = kDefaultPPTrig;// default PP trigger configuration

void Config()
{
  // ...
  AliSimulation::Instance()->SetTriggerConfig(pprTrigConfName[strig]);
  // ...
  else if (generatorFlag==2) {
    // Pure HiJing generator adapted to ~2000dNdy at highest energy
    AliGenHijing *generHijing = new AliGenHijing(-1);
    generHijing->SetEnergyCMS(5500.); // GeV
    generHijing->SetImpactParameterRange(0,15); // MinBias, set to 0,5 for central
    generHijing->SetReferenceFrame("CMS");
    generHijing->SetProjectile("A", 208, 82);
    generHijing->SetTarget    ("A", 208, 82);
    generHijing->KeepFullEvent();
    generHijing->SetJetQuenching(1);
    generHijing->SetShadowing(1);
    generHijing->SetSpectators(0);
    generHijing->SetSelectAll(0);
    generHijing->SetPtHardMin(4.5);
    Float_t thmin = EtaToTheta( 2.5);   // theta min. <---> eta max
    Float_t thmax = EtaToTheta(-2.5);  // theta max. <---> eta min
    generHijing->SetThetaRange(thmin,thmax);

    AliGenerator*  gener = generHijing;
    gener->SetSigma(50e-4, 50e-4, 5.0);   //Sigma in (X,Y,Z) (cm) on IP position
    gener->SetVertexSmear(kPerEvent);
    gener->Init();
  }
  TGeoGlobalMagField::Instance()->SetField(new AliMagF("Maps","Maps", -1., -1.,
      AliMagF::k5kG));
  // ...
  // Only ITS and PIPE simulated
  Int_t   iITS   =  1;
  Int_t   iPIPE  =  1;
```

```
48    // ...
49    if (iPIPE) {
50      //================== PIPE parameters ============================
51      AliPIPE *PIPE = new AliPIPEupgrade("PIPE", "Beam Pipe");
52    }
53    if (iITS) {
54      //================== ITS parameters ============================
55      gROOT->ProcessLine(".x CreateITSUv2ALP3.C");
56    }
57  }
```

LISTING D.3: itsuTestBench setup for pp.

```
1   Int_t generatorFlag = 4;
2
3   /* $Id: Config.C 47147 2011-02-07 11:46:44Z amastros $ */
4   enum PprTrigConf_t
5   {
6     kDefaultPPTrig, kDefaultPbPbTrig
7   };
8
9   const char * pprTrigConfName[] = {
10    "p-p","Pb-Pb"
11  };
12
13
14  static PprTrigConf_t strig = kDefaultPPTrig;// default PP trigger configuration
15
16  void Config()
17  {
18    // ...
19    AliSimulation::Instance()->SetTriggerConfig(pprTrigConfName[strig]);
20    // ...
21    else if (generatorFlag==4) {
22      gSystem->Load("libpythia8.so");
23      gSystem->Load("libAliPythia8.so");
24      gSystem->Setenv("PYTHIA8DATA", gSystem->ExpandPathName("$ALICE_ROOT/PYTHIA8/
            pythia8/xmldoc"));
25      gSystem->Setenv("LHAPDF",      gSystem->ExpandPathName("$ALICE_ROOT/LHAPDF"));
26      gSystem->Setenv("LHAPATH",     gSystem->ExpandPathName("$ALICE_ROOT/LHAPDF/
            PDFsets"));
27
28      // pp Pythia Monach-tune generator
29      int tune = 14; // kPythia8Tune_Monash2013
30      AliGenPythiaPlus *pythia = new AliGenPythiaPlus(AliPythia8::Instance());
31      pythia->SetMomentumRange(0, 999999.);
32      pythia->SetThetaRange(0., 180.);
33      pythia->SetYRange(-2.5.,2.5);
34      pythia->SetPtRange(0,1000.);
35      printf("Setting pythis8 processe to %d, tune %d\n",kPyMbDefault, tune);
36      pythia->SetProcess(kPyMbDefault);
37      pythia->SetEnergyCMS(5500);
38      pythia->SetEventListRange(-1, 2);
39      pythia->SetTune(tune);
40      pythia->Init();
```

```
41   }
42   TGeoGlobalMagField::Instance()->SetField(new AliMagF("Maps","Maps", -1., -1.,
         AliMagF::k5kG));
43   // ...
44   // Only ITS and PIPE simulated
45   Int_t   iITS    =  1;
46   Int_t   iPIPE   =  1;
47   // ...
48   if (iPIPE) {
49     //================== PIPE parameters ============================
50     AliPIPE *PIPE = new AliPIPEupgrade("PIPE", "Beam Pipe");
51   }
52
53   if (iITS) {
54     //================== ITS parameters ============================
55     gROOT->ProcessLine(".x CreateITSUv2ALP3.C");
56   }
57 }
```

LISTING D.4: itsuTestBench setup for QED.

```
1  Int_t generatorFlag = 5;
2
3  /* $Id: Config.C 47147 2011-02-07 11:46:44Z amastros $ */
4  enum PprTrigConf_t
5  {
6    kDefaultPPTrig, kDefaultPbPbTrig
7  };
8
9  const char * pprTrigConfName[] = {
10   "p-p","Pb-Pb"
11 };
12
13 static PprTrigConf_t strig = kDefaultPPTrig;// default PP trigger configuration
14
15 void Config()
16 {
17   // ...
18   AliSimulation::Instance()->SetTriggerConfig(pprTrigConfName[strig]);
19   // ...
20   else if (generatorFlag==5) {
21     // QED electrons
22     AliGenCocktail *cocktail = new AliGenCocktail();
23     cocktail->SetProjectile("A", 208, 82);
24     cocktail->SetTarget     ("A", 208, 82);
25     cocktail->SetEnergyCMS(5500.);
26     cocktail->SetSigma(50e-4, 50e-4, 5.0);    //Sigma in (X,Y,Z) (cm) on IP position
27     cocktail->SetVertexSmear(kPerEvent);
28     Float_t thmin = EtaToTheta( 2.5);    // theta min. <---> eta max
29     Float_t thmax = EtaToTheta(-2.5);  // theta max. <---> eta min
30     cocktail->SetThetaRange(thmin,thmax);
31     cocktail->SetName("QEDelectrons");
32
33     //--------------------- QED (Ruben) --------------------
34     AliGenQEDBg *genBg = new AliGenQEDBg();
```

```
35      genBg->SetEnergyCMS(5500.);
36      genBg->SetProjectile("A", 208, 82);
37      genBg->SetTarget     ("A", 208, 82);
38      genBg->SetYRange(-6.,3);
39      genBg->SetPtRange(1.e-3,1.0);        // Set pt limits (GeV) for e+-: 1MeV
            corresponds to max R=13.3mm at 5kGaus
40      genBg->SetLumiIntTime(6.e27,250e-9);  // luminosity and integration time
41      cocktail->AddGenerator(genBg,"QEDep",1);
42      genBg->SetVertexSource(kInternal);
43      cocktail->Init();
44    }
45    TGeoGlobalMagField::Instance()->SetField(new AliMagF("Maps","Maps", -1., -1.,
          AliMagF::k5kG));
46    // ...
47    // Only ITS and PIPE simulated
48    Int_t    iITS    =  1;
49    Int_t    iPIPE   =  1;
50    // ...
51    if (iPIPE) {
52      //=================== PIPE parameters ============================
53      AliPIPE *PIPE = new AliPIPEupgrade("PIPE", "Beam Pipe");
54    }
55    if (iITS) {
56      //=================== ITS parameters ============================
57      gROOT->ProcessLine(".x CreateITSUv2ALP3.C");
58    }
59  }
```

D.3.1 File Formats for Events in the SystemC Simulations

Output files from the AliRoot simulations are not used directly in the SystemC-based simulation of the ITS. Instead, pixel hits (or "digits") were extracted from the events generated with the AliRoot simulations, and stored in an Extensible Markup Language (XML)-based files used by the SystemC-based simulation. One file per event. The format, which is shown in listing D.5, was largely based on a format used in a previous simulation model of the ITS [40].

Later versions of the SystemC-based simulation also implemented custom binary format as an alternative, in order to reduce the file size and which were quicker to parse. Table D.12 shows the data words used in the file. The data words themselves are one byte. Some of them are immediately followed by one or more parameters, in the order listed in the table. The format follows the same general recipe as the XML file. The file starts and ends with the *DETECTOR_START* and *DETECTOR_END* words. Layers are delimited by the *LAYER_START*, along with the layer number, and *LAYER_END* ends the layer. The same goes for staves, modules and chips. An arbitrary number of pixel hits (the *DIGIT* word) can be located between a *CHIP_START* and *CHIP_END*.

LISTING D.5: XML file format for events in ITS simulation.

```
<its_detector>
  <lay id=0>
    <sta id=0>
      <ssta id=0>
        <mod id=0>
          <chip id=0>
            <dig>123:64</dig>
            <dig>234:12</dig>
            <dig>10:54</dig>
          </chip>
        </mod>
      </ssta>
    </sta>
  </lay>
  <lay id=1>
     ...
  </lay>
</its_detector>
```

TABLE D.12: Binary file format for events in ITS simulation.

Data word	Value	Parameters	
DETECTOR_START	0x20	N/A	
DETECTOR_END	0x40	N/A	
LAYER_START	0x01	layer_id (8-bit unsigned integer)	
LAYER_END	0x11	N/A	
STAVE_START	0x02	stave_id (8-bit unsigned integer)	
STAVE_END	0x12	N/A	
MODULE_START	0x03	module_id (8-bit unsigned integer)	
MODULE_END	0x13	N/A	
CHIP_START	0x05	chip_id (8-bit unsigned integer)	
CHIP_END	0x15	chip_id (8-bit unsigned integer)	
DIGIT	0x06	X-coord (16-bit unsigned integer)	Y-coord (16-bit unsigned integer)

D.3.2 Monte Carlo Events

Figures D.1 to D.3 shows the multiplicity, in terms of pixel hits and for the range of pseudorapidity covered by the respective layers of the ITS, for the MC event pools generated for Pb–Pb, QED, and pp. It should be noted that the so-called QED events were generated by integrating the QED background at a luminosity corresponding to 50 kHz Pb–Pb over 250 ns, and each "event" corresponds to 250 ns of this background. This background is continuously fed into the pixel front-ends of the ALPIDE chips in the SystemC simulation, but the rate is scaled with the simulated interaction rate (since interaction rate is proportional to luminosity).

Table D.13 summarizes the average number of pixels per event for each layer in the event pools, along with the pixel hit density[2]. A similar trend is seen compared to tables in the TDR of the ITS [13], but the numbers differ as the TDR specified maximum hit densities and in terms of particle hits (not pixel hits).

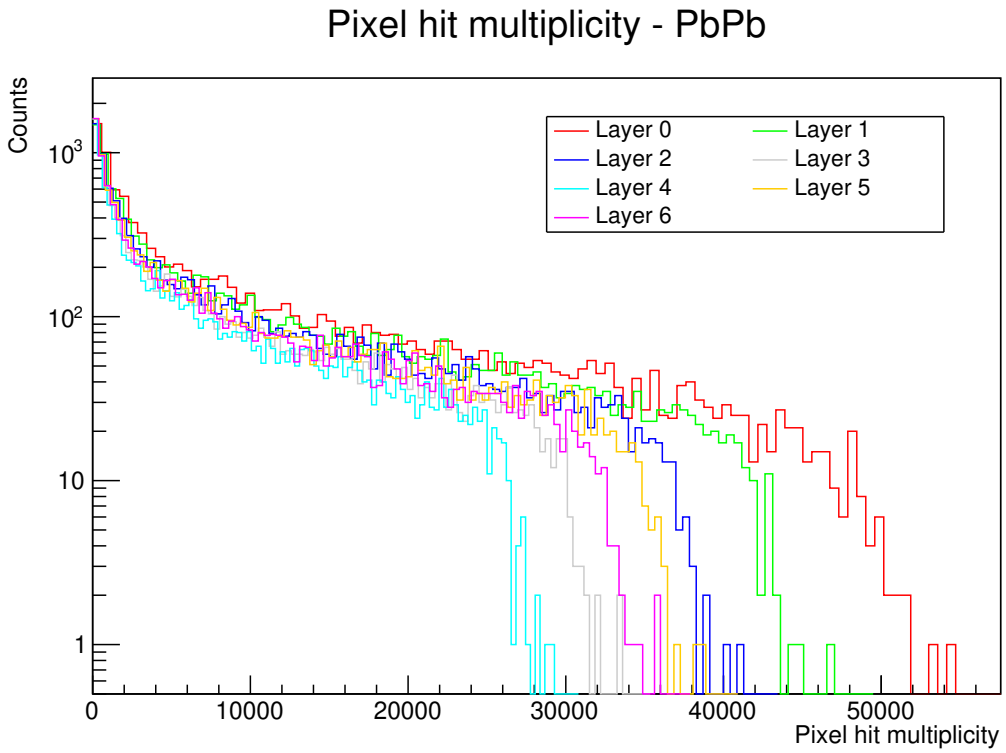


FIGURE D.1: Pixel hit multiplicity of Pb-Pb event pool for ITS simulations.

[2]Calculated based on the surface area of all sensors chips in a layer.

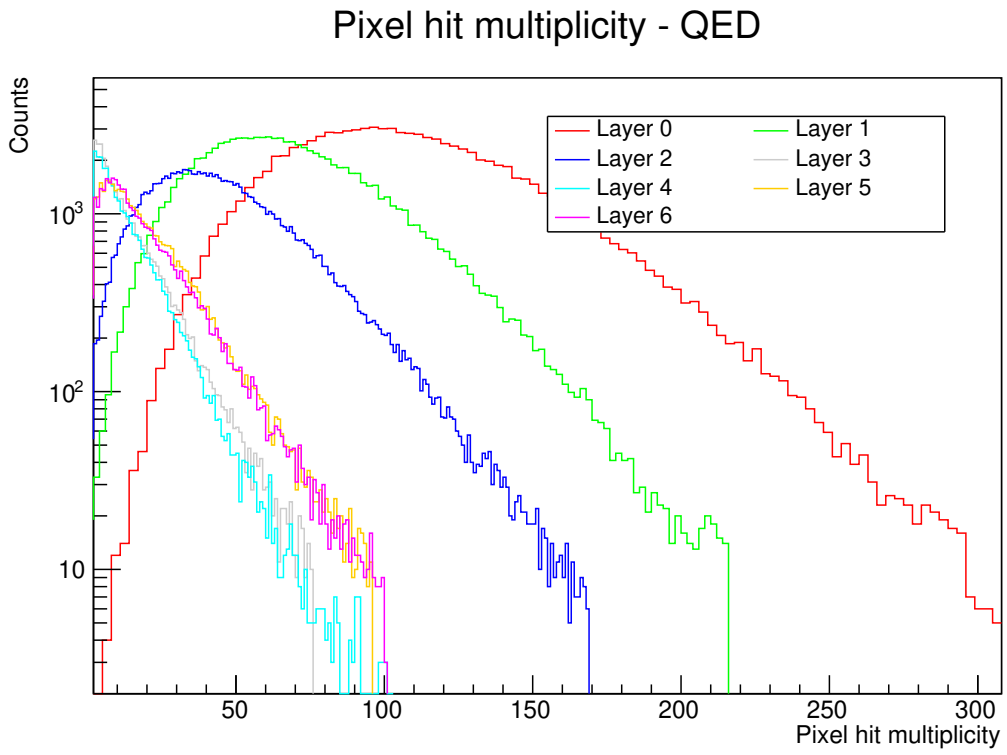


FIGURE D.2: Pixel hit multiplicity of QED event pool for ITS simulations.

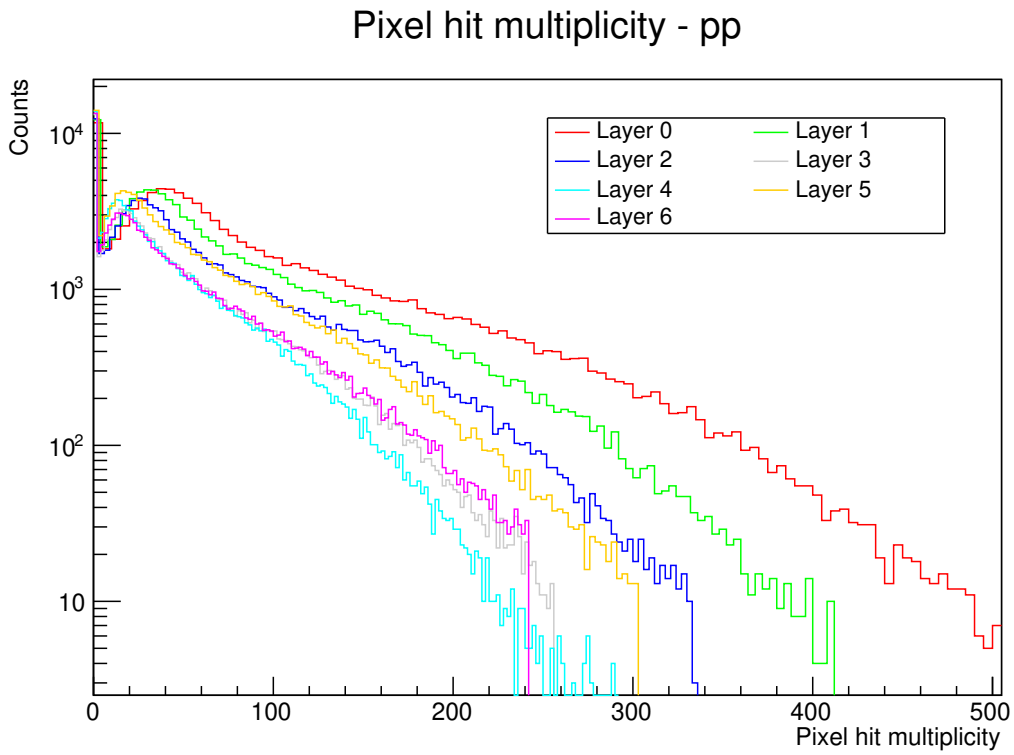


FIGURE D.3: Pixel hit multiplicity of pp event pool for ITS simulations.

TABLE D.13: Average event size (in terms of pixels) and average pixel hit density for the MC event pool that was generated for the SystemC simulations of the ITS.

Layer	**Pb–Pb**		**QED**		**pp**	
	Total	cm^{-2}	Total	cm^{-2}	Total	cm^{-2}
0	10 190	2.097×10^{1}	110	2.263×10^{-1}	88.15	1.814×10^{-1}
1	9017	1.392×10^{1}	69.67	1.075×10^{-1}	70.08	1.081×10^{-1}
2	7966	9.835	46.56	5.748×10^{-2}	59.37	7.330×10^{-2}
3	6485	5.849×10^{-1}	4.819	4.346×10^{-4}	44.51	4.014×10^{-3}
4	5684	4.028×10^{-1}	4.146	2.938×10^{-4}	38.23	2.709×10^{-3}
5	7503	2.127×10^{-1}	7.85	2.225×10^{-4}	50.84	1.441×10^{-3}
6	6936	1.710×10^{-1}	7.37	1.817×10^{-4}	46.48	1.146×10^{-3}

Appendix E

UART Protocol and Debug Software for Auxiliary FPGA

The protocol[1] for communication with the auxiliary FPGA via UART enables full access to the internal WB bus of the FPGA design. The associated GUI software allows for full control of the FPGA design and its features via the WB bus.

E.1 Connections to the Readout Unit

The RU board does not feature a dedicated connector for the UART on the auxiliary FPGA. The connection has to be made via pin-header J14 using a 3.3V USB to serial adapter that supports 1 Mbaud or higher data rates[2]. Tx connects to pin 1 on J14, Rx on pin 3, and GND on any of the even numbered pins[3].

E.2 Protocol

Commands sent to the FPGA start with the header shown in fig. E.1. A fixed start byte is sent first, which has the value 0xA5. The following byte specifies the command to execute: read, write, or no operation, together with two bits to indicate whether the WB bus address should increase for each byte, and whether the receiver should acknowledge the message.

The message ends here for a no operation command. However, if the *Ack enable* bit was set in the command byte, the receiver should acknowledge the no operation command. This can be used to verify that there is a connection.

For read and write operations, the command byte is followed by the WB address. When the *Address increment* bit is disabled, the bytes in the message are all written to

[1] Based on the "UART to Bus" IP available on OpenCores [65].
[2] 921 600 bps is the baud rate in the Auxiliary FPGA design.
[3] Tx/Rx is specified relative to the host computer.

	7	6	5	4	3	2	1	0
Start command	1	0	1	0	0	1	0	1
Command	Not used		Command type [1:0]		Not used		Address increment	Ack enable

FIGURE E.1: Header format in the modified UART to Bus protocol.

	7	6	5	4	3	2	1	0
Start command	1	0	1	0	0	1	0	1
Command	Not used		0	0	Not used		Address increment	Ack enable

FIGURE E.2: NOP command in the modified UART to Bus protocol.

this address. This is useful for writing to registers that are mapped to the input of a FIFO. When address is enabled, the address in the message denotes the start address[4]. The length of the data to read or write is transmitted in the two bytes following the address byte. The 16-bit data length allows for transfers of up to 65535 bytes[5].

E.2.1 No Operation Command

A *NOP* command performs no operation and the message concludes after message header. The full *NOP* message is shown in fig. E.2. The FPGA will send an acknowledgement if the *Ack enable* bit was set, however.

E.2.2 Read Command

A read command message extends the header with a byte for the WB address to read, and two bytes for the number of bytes to read. The full message is shown in fig. E.3. If acknowledgement is enabled, the FPGA will respond with the acknowledgement byte first. After that it responds with the requested number of bytes read from the register(s).

E.2.3 Write Command

The write command message is shown in fig. E.4. It has the same format as the read message, but concludes with the payload data for the bytes to write. When enabled, the FPGA will respond with the acknowledgement byte after the full message, including payload data, has been received. There is no response from the FPGA if acknowledgement was not enabled.

[4]This was a useful feature in previous versions of the FPGA design which had a large page buffer for the flash.

[5]Messages of zero length are allowed.

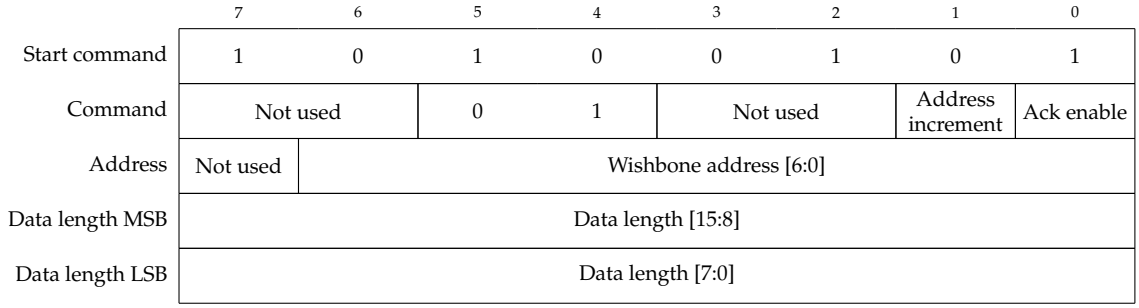


FIGURE E.3: Read command in the modified UART to Bus protocol.

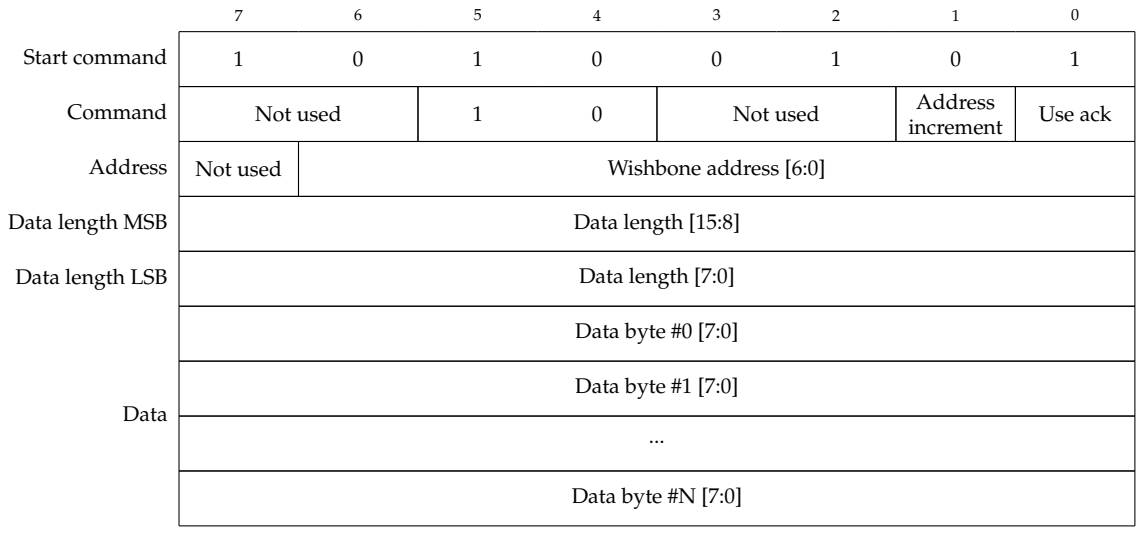


FIGURE E.4: Write command in the modified UART to Bus protocol.

E.3 Software

The Qt-based GUI software allows for: monitoring of all registers; read and write to individual registers and register fields; access to SelectMAP registers; debug functionality for the *Flash interface*; and most importantly, firmware upload of the configuration images for the main FPGA to the external flash.

E.3.1 Connecting to Auxiliary FPGA

The default baud rate for the software is the same as for the FPGA design; 921600 bauds. However, it can be changed via the **File**→ **Baud rate** menu if necessary.

A connection to the auxiliary FPGA is established by connecting to the serial port from the **File**→ **Connect** menu, as shown in fig. E.5. "Connected to *COM-port* @ *baud rate*" should be displayed on the status bar.

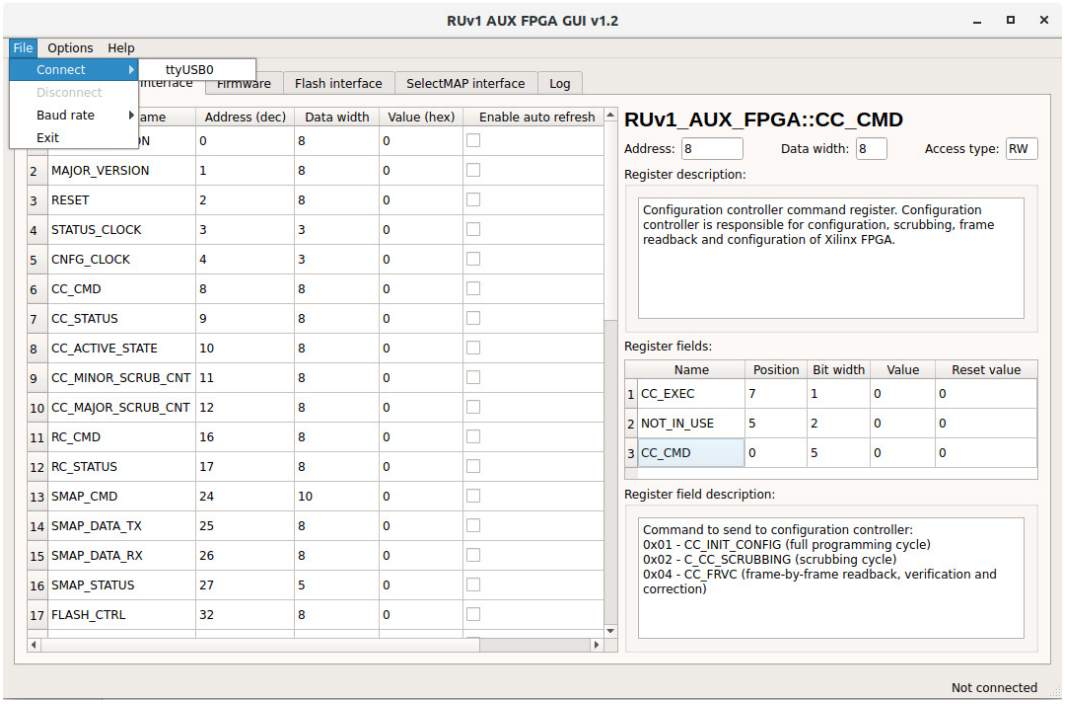


FIGURE E.5: Auxiliary FPGA debug software. File menu and sub-menus.

E.3.2 Direct Access and Monitoring of Wishbone Registers

The default tab in the software, *Wishbone Register Interface*, displays all WB registers available in the FPGA design. Selecting a register will display more information on the right side of the GUI, such as a description and information about each register field. This is also shown in fig. E.5. Further information about the register fields can

be displayed by selecting them, and the register fields can written individually via the register field table.

By checking the boxes in the *Enable auto refresh* column of the register table, the software will periodically poll and refresh the values of the chosen registers.

E.3.3 Uploading Firmware to External FLASH

Configuration and scrubbing images can be uploaded to the external flash from the *Firmware*[6] tab. The images are selected with a file system browser popup when the **Start** button is clicked.

Options include reading back the images to verify that they were correctly written, and the start block in the flash for both images. The software will automatically update the parameter section of the flash after the new images have been written.

The other tabs of the software, and register polling, is disabled while the upload is in progress (except for the log tab). It takes around 8 minutes to upload the 24.1 MB bitstream file for the UltraScale FPGA at the baud rate of 921 600 (without read-back verification).

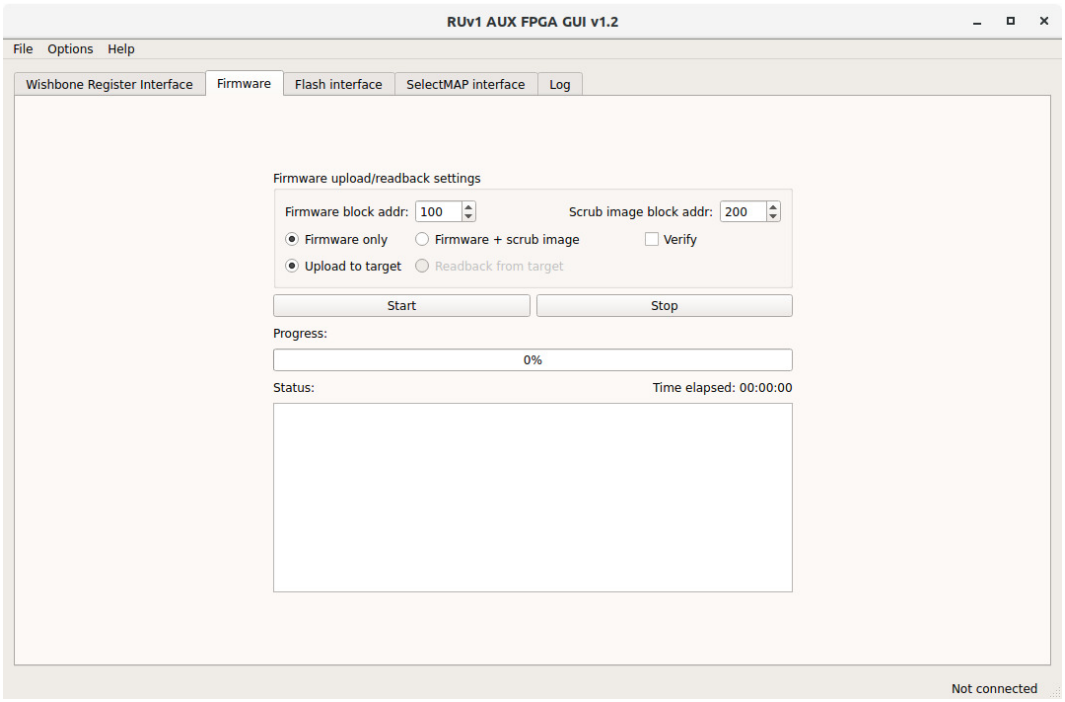


FIGURE E.6: Firmware upload.

[6]The term "firmware" has been common jargon in the project for the configuration image. This usage is technically incorrect – typically it refers to firmware for an embedded CPU, not an FPGA design.

E.3.4 Flash Interface Testing

The *Flash interface* tab, shown in fig. E.7, has a variety of features for testing the flash interface. On the left side, there are functions for reading the Flash ID Code, erasing the Flash, writing a hex value to all bytes in a selected block range, and read flash contents and storing them to disk. The latter two functions were used during beam tests of the RU; parts of the flash were initialized to a known pattern prior to the radiation test, and after testing the data was read out and the number of bits that had flipped were counted.

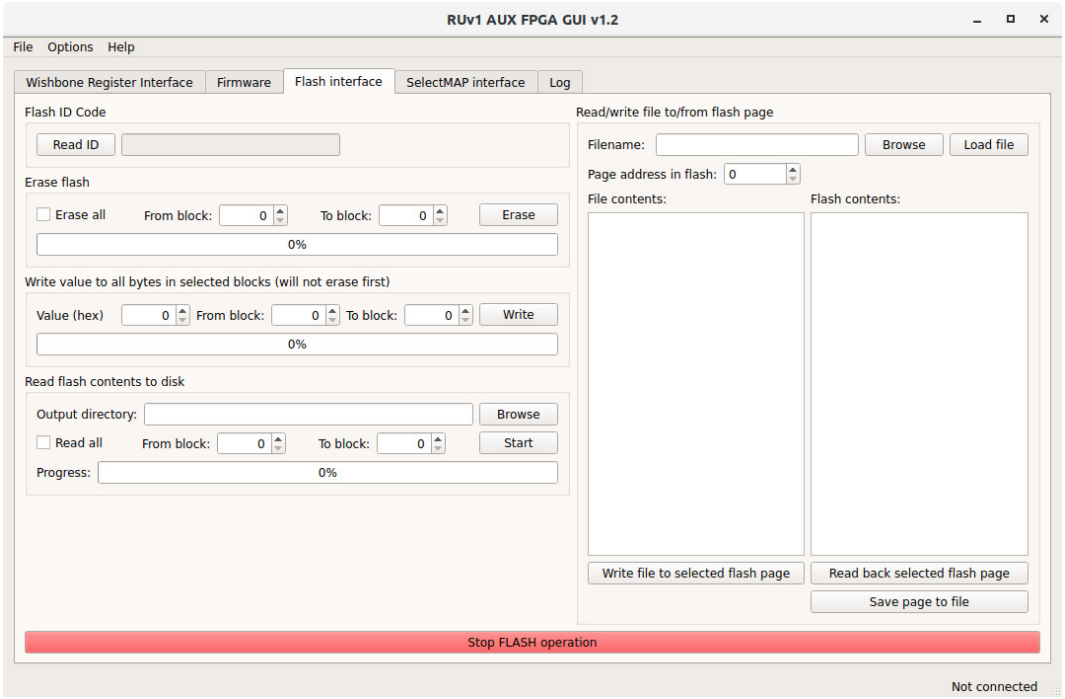


FIGURE E.7: Flash interface tab with a variety of test features.

And finally, the right side has functions to write a disk file to a single page in the flash, and also to read out a single page and store it to file.

E.3.5 SelectMAP Interface Testing

The *SelectMAP interface* tab has a table listing the available registers in the SelectMAP interface. Clicking refresh on a register will refresh/read it, and editing the value and pressing enter will trigger a write to the register with the new value. This allows for limited testing and debugging of the SelectMAP interface to the UltraScale FPGA.

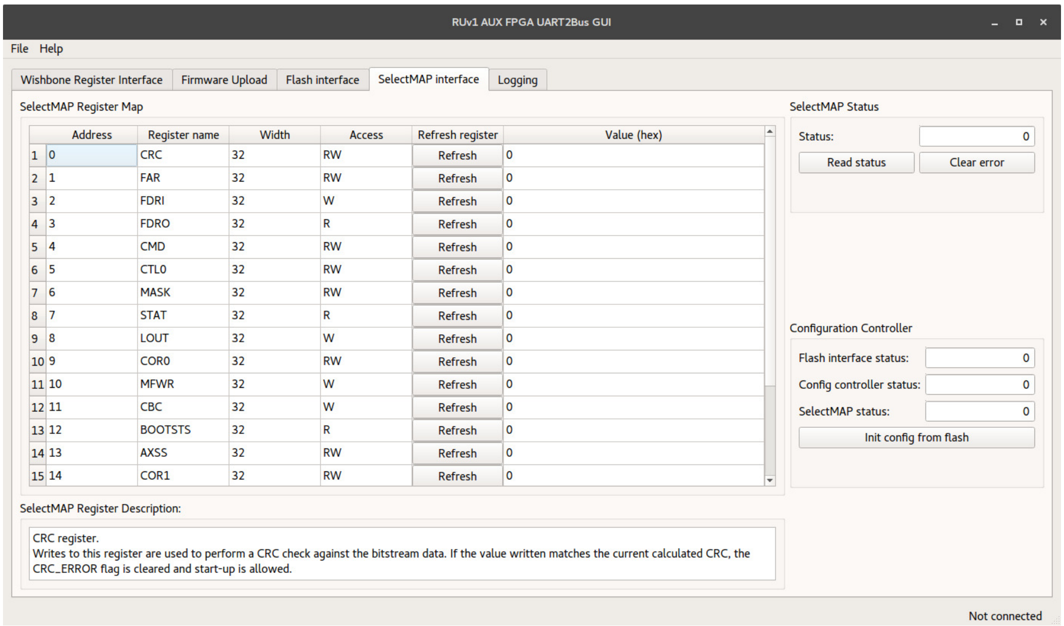


FIGURE E.8: SelectMAP tab.

E.3.6 Logging

The program logs many of the most important events, fig. E.9, especially errors. The newest entries are displayed at the top in the log.

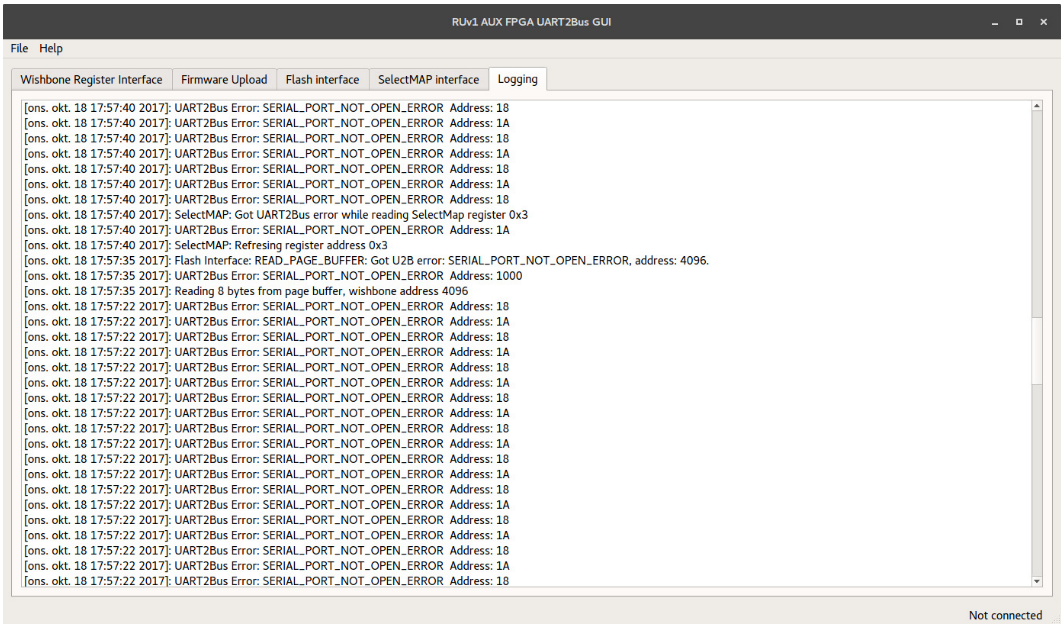


FIGURE E.9: The log interface.

Appendix F

Concept for Busy Unit

The ALPIDE signals a change in busy status on the data link using the *BUSY ON* and *BUSY OFF* words, which are prioritized over normal data. Typically the busy status occurs when all buffers of the MEB are in use (see appendices B.3 and B.5.2 for more details).

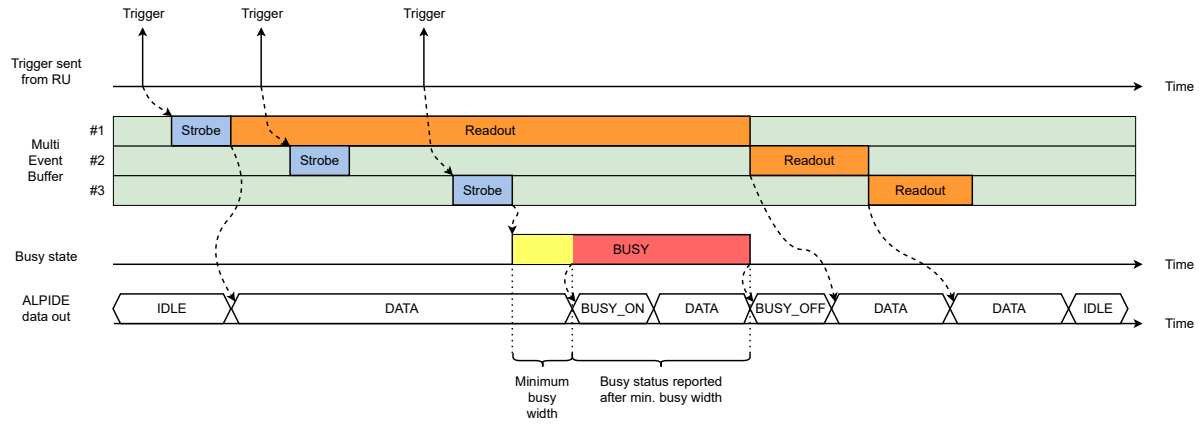


FIGURE F.1: Illustration of busy signals from ALPIDE.

The waveforms in fig. F.1 illustrates the relationship between triggers, strobes, the MEB, data readout, and busy signaling. When the third buffer of the MEB is occupied at the end of the third strobe (assuming triggered mode), the chip is in a busy status. But it has to be busy for at least the minimum busy width period, a configurable parameter in the chip, before the chip issues the BUSY ON word on the data link to indicate the busy status. When readout of the first event from the MEB has concluded, and a buffer is available for new events, the chip goes out of busy and issues the BUSY OFF word to indicate the new busy status.

Figure F.2 illustrates that the chip may be busy between triggers, but it does not affect readout/triggering as long as the busy status ends before the next trigger. However, a busy violation (assuming triggered mode in the ALPIDE) occurs when a trigger is received while the chip is busy. The effect of a busy violation in an ALPIDE chip is the complete loss of the event associated with that trigger. Or in the case of

continuous mode, the equivalent flushing mechanism results in partial loss of data for the oldest event in the MEB.

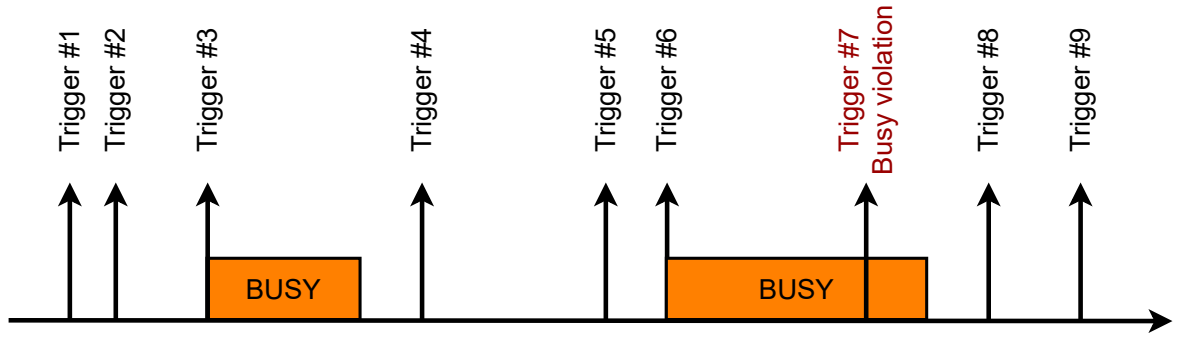


FIGURE F.2: Illustration of busy and busy violations.

F.1 Impact on Readout Data

Ideally the ALPIDE should be operated at a trigger rate and with a pixel hit occupancy that is well within the readout capabilities of the chip. But since events happen at random times and with varying size (in terms of pixel hits), it would not be unexpected to see a chip report the busy status occasionally, or even the odd busy violation.

But with higher trigger rates the readout capabilities will be exhausted more often, to the point where busy violations (or flushed events in the case of continuous mode) happen so frequently that many events have an intolerable amount of "holes" due to "missing chips".

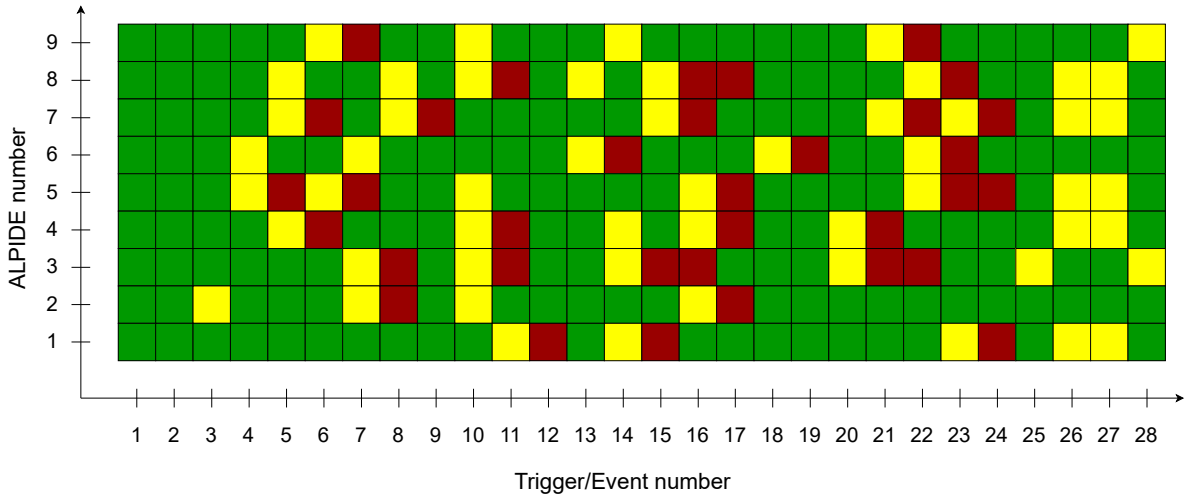


FIGURE F.3: Illustration of poor quality events due to busy violations ("swiss cheese events").

Figure F.3 shows an illustration to demonstrate this scenario (it is not based on real or simulated data). Each box represents a readout frame associated with a trigger/event for an individual ALPIDE chip. Both green and yellow boxes are successfully read out, but the chip is reporting busy status when the box is yellow. Red boxes are busy violations where the data is lost, because a trigger was received when the chip was busy. In this contrived example, there are sixteen events that have "holes" due to busy violations, i.e. there is at least one red box for a specific trigger/event number. Only twelve events are complete (no busy violations).

F.2 Busy Handling

When it comes to handling the busy signals from the sensor chips, the options that are available to the readout electronics are limited. Besides monitoring of the busy status, the only active option is to interfere in the triggering of the sensor chips by withholding the next trigger when the chips are busy. But, it is important to note that the sensor chips are triggered simultaneously by broadcasting the trigger word on the shared multi-drop control link. Withholding the trigger affects all sensor chips on a control link; it can not be done for an individual chip. The most reasonable approach would be to set a threshold for the number of chips that must report a busy status before the trigger is not issued.

In the previous example there were nine ALPIDE chips, which is equivalent to an IB-stave. Based on the events of fig. F.3, fig. F.4 tries to illustrate the effect of not issuing the next trigger when a threshold of four busy chips is reached. The grey boxes indicate the triggers that were not issued. It is assumed that omitting a trigger gives the detector some "time to breathe" and has a positive effect on the next two triggers; flipping some of the busy violations and busy events for those triggers.

In this example, which is also a bit contrived, six events are missing in full since they are not triggered on. However, compared to the previous example, there are now only four events that have "holes" due to busy violations, and eighteen events are complete. In other words, it may be possible to improve the quality of the events that are read out, at the expense of sacrificing a few events in full (which likely have holes and are of low quality anyway).

Busy Processing

The 26^{th} and 27^{th} events in fig. F.3 illustrate an interesting case where busy is reported after several triggers in a row, but without any busy violations occuring. This happens when a chip goes busy after receiving a trigger, but goes out of busy before the next

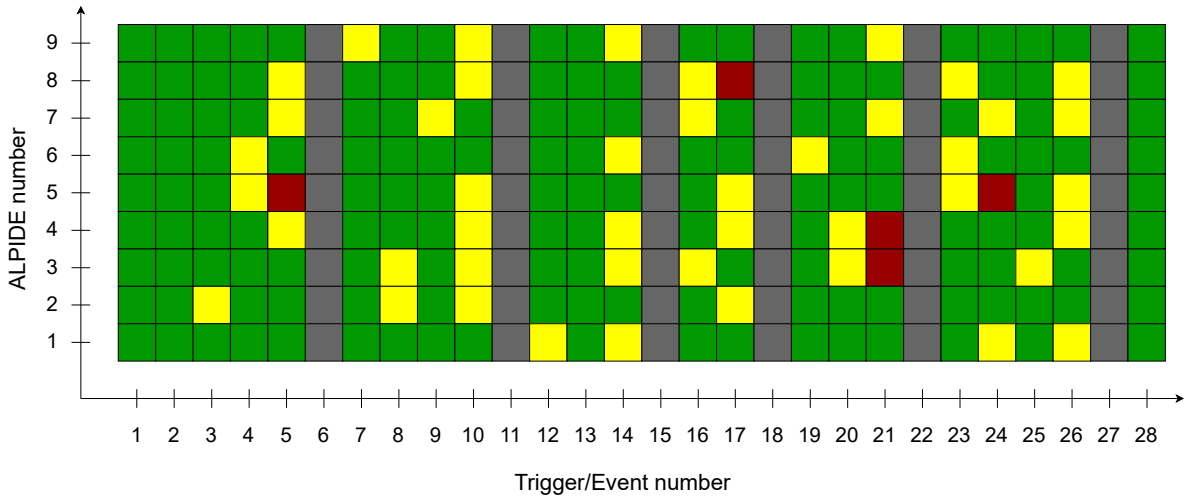


FIGURE F.4: Illustration of improved quality events with busy handling. The next trigger is skipped when a threshold of four busy chips has been reached, giving the detector some "time to breathe".

trigger because it finished reading out the oldest event in the time in between. The chips do signal this change of busy status (with the BUSY OFF data word). But if it happens right before the next trigger, and the RU did not have enough time to process this information, it is possible that the RU acts based on an outdated busy status. As a result, the RU did not issue the 27^{th} trigger in fig. F.4, leading to the unnecessary loss of an entire event that the sensor chips would have been able to process.

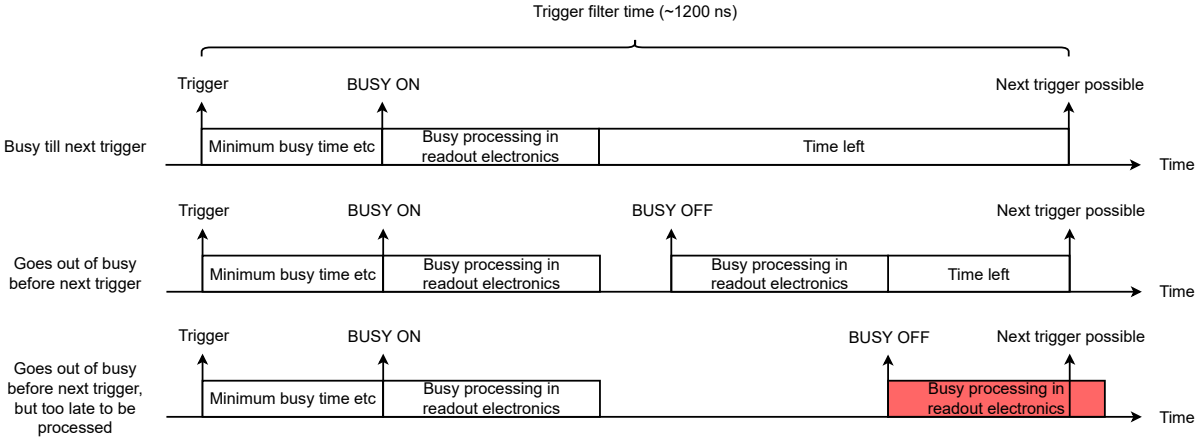


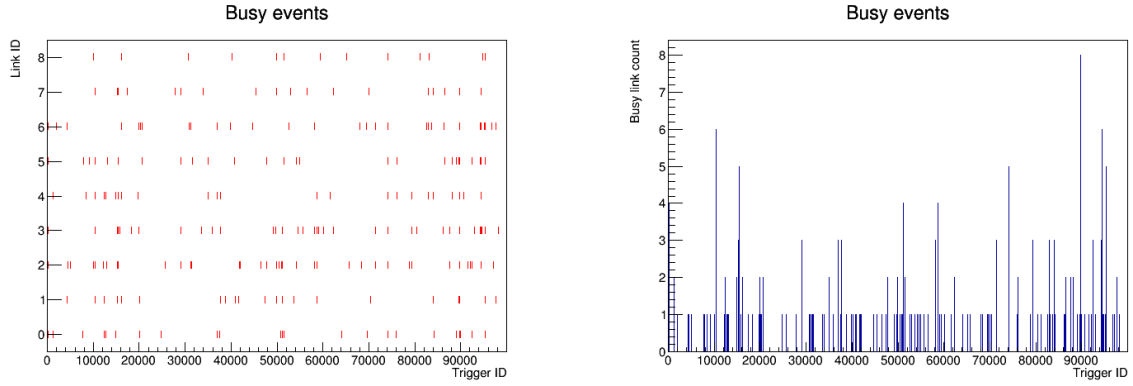
FIGURE F.5: Illustration of busy processing.

The shortest possible distance between two triggers is around 1200 ns, which is defined by the trigger filtering window (see section 2.6.1).

Figure F.5 shows some examples with two triggers separated by this distance, and where the chip goes busy after the first trigger. The time from trigger to the BUSY ON word, and processing of BUSY ON in the readout electronics, is always constant. The example at the top is the simplest; the chip remains busy until the next trigger, which

will cause a busy violation. As long as long as the readout electronics are capable of receiving and processing the BUSY ON word within the 1200 ns, it should be able to make a timely decision about whether the next trigger should be sent or not.

The next two examples shows the chip going out of busy in time before the next trigger, by issuing the BUSY OFF word. Hence, the chip is ready for the next trigger, and the readout electronics is able to account for this if it is able to process the BUSY OFF word in time. But if the BUSY OFF word is received too late, then the readout electronics will still think the chip is busy. And this can lead to the situation that was explained earlier for triggers 26^{th} and 27^{th} in fig. F.3.



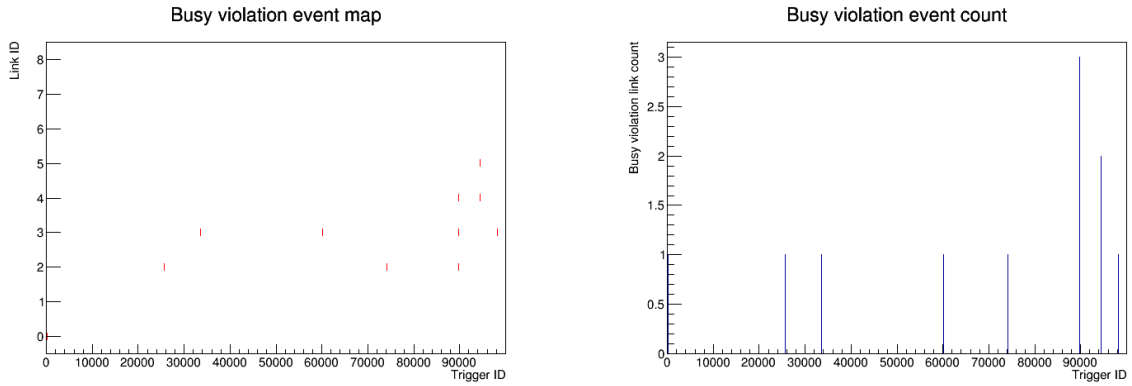
(A) (B)

FIGURE F.6: Busy link map (F.6A) and counts (F.6B) versus trigger number for an RU in the innermost ITS layer at 100 kHz Pb–Pb with minimum-bias triggers. Simulated with SystemC model of the ALPIDE and ITS.

Clearly, time to process the BUSY OFF word should be as short as possible, as it plays an important role on the performance of a busy system in the readout electronics. To further emphasize this, consider figs. F.6 and F.7. Figure F.6 shows how often the ALPIDEs in an IB stave were busy in a simulation of the ITS for 100 kHz Pb–Pb. Figure F.7 shows the busy violations from the same simulation. There is a large number of busy reported, most of which are "false alarms", since there are only a few busy violations.

F.3 Busy Unit

The previous examples showed how data quality could be improved by strategically withholding triggers. But the example was for an individual IB-stave only. The detector has a total of 48 IB-stave and 144 staves in the OB (middle and outer layers). That amounts to $48 + 4 \times 144 = 624$ control links (4 control links per stave in the OB), in other words, 624 sections of the detector that can be triggered independently.



(A) (B)

FIGURE F.7: Busy violation map (F.7A) and counts (F.7B) versus trigger number for an RU in the innermost ITS layer at 100 kHz Pb–Pb with minimum-bias triggers. Simulated with SystemC model of the ALPIDE and ITS.

Keeping in mind that the objective is to achieve more complete events, by omitting triggers when a large part of the detector is busy to give the sensor chips "time to breathe", it is evident that these trigger decisions must be coordinated for the whole detector.

The RU has dedicated transceiver connections for busy input and output to report busy status among the readout electronics. This allows for several possible topologies to communicate and handle busy status in the readout electronics. The most feasible option is to have the RUs transmit their busy status to a dedicated Busy Unit (BU). The BU then determines the busy status for the entire detector, and communicates this back to the RUs.

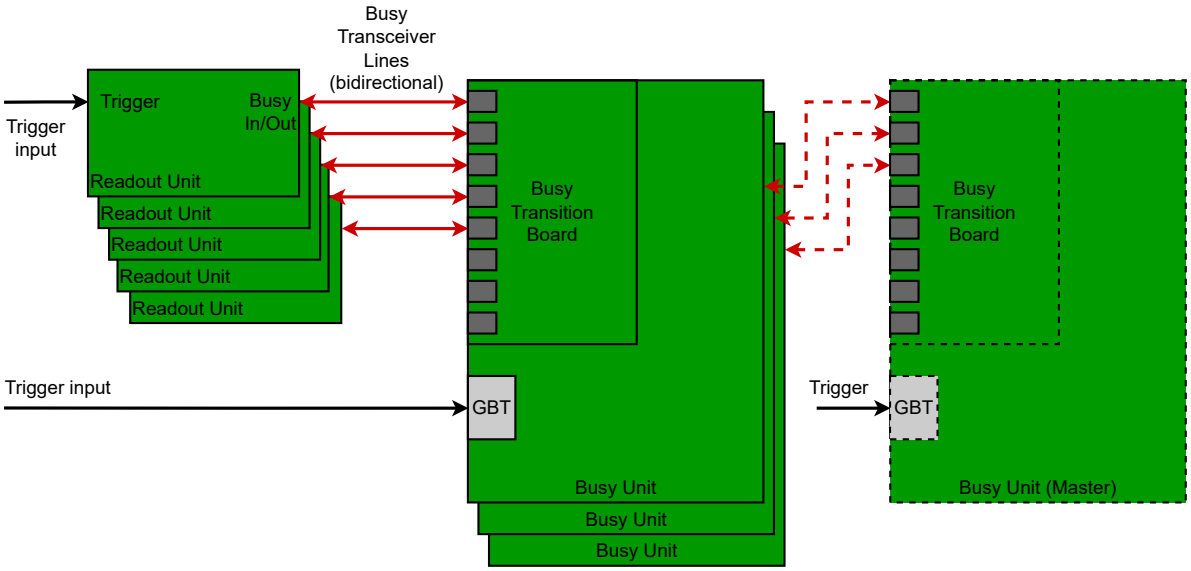


FIGURE F.8: BU and connections to RUs.

The concept is illustrated in fig. F.8. Presumably, it will be necessary to have more than one BU, as it is unlikely that a BU could be designed to cater for all 192 RUs. And with an additional "master" BU, the busy status can be coordinated and synchronized for all RUs in the detector.

The BU itself can be based almost entirely on the RU design, with a dedicated *Busy Transition Board* for the busy links. Since this transition board would require both transceiver inputs and outputs, the current RU design can not be re-used without modification as a BU, since the RU design primarily has transceiver inputs routed to the connectors for the transition board. But the UltraScale FPGA has free transceiver outputs available, so it is a matter of a small modification (routing transceiver outputs to the transition board connectors) to realize a BU design. Most of the main FPGA design for the RU, as well as the auxiliary design for the scrubbing solution, can be re-used for the BU, and it is probably not necessary to perform beam testing etc. of the BU when it is almost identical to the RU.

Alternatives to the Busy Unit

Connecting the busy inputs and outputs of the RUs in a daisy-chain, as shown in fig. F.9, was previously discussed as an alternative to a dedicated BU.

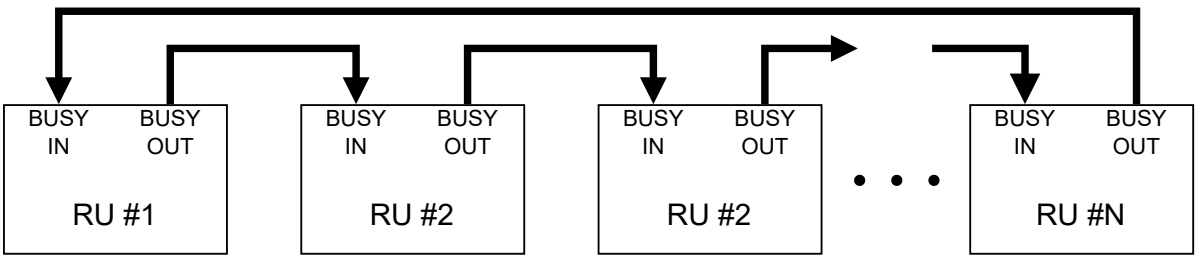


FIGURE F.9: Daisy-chained busy signals between RUs.

In very simple terms, the implementation of a busy module in the RU FPGA design would look something like in fig. F.10. An RU transmits its (local) count of busy ALPIDE links over the busy link daisy-chain, and receives counts for each of the remote RUs. The detector busy status must be determined locally in the busy module of each RU, since there is no central device that determines this status with the daisy-chain approach. The simplest way to determine the busy status is when the sum of busy ALPIDE links reaches a threshold, as shown in the figure. But it would probably make sense to set a threshold for each layer, and possibly build more complex logic into this decision.

A challenge with the daisy-chain approach is the synchronization of the detector busy status, since a change in busy ALPIDE links reported by one RU would have

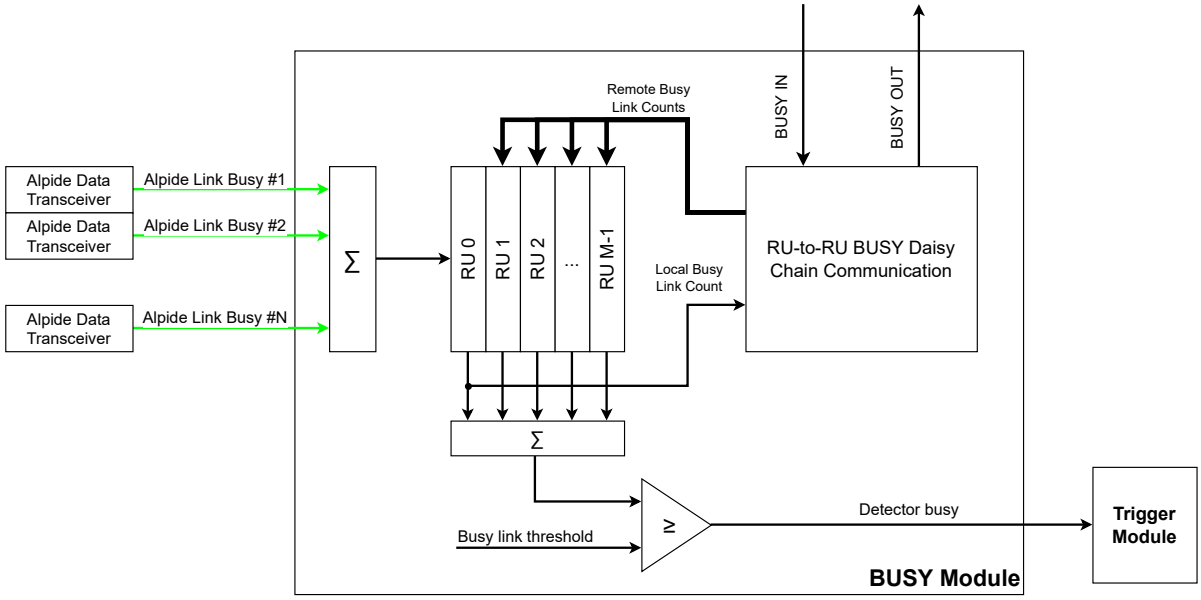


FIGURE F.10: Busy module for daisy-chained RUs.

to propagate through the daisy-chain, and will not reach the other RUs simultaneously. But more critical is the total delay of transmitting busy via a daisy-chain of 192 RUs. Assuming that an RU can receive and forward another RU's busy count in one 160 MHz clock cycle, then the total delay to propagate through the daisy chain is $192 \times 6.25\,\text{ns} = 1.2\,\mu\text{s}$. This is roughly the same as the trigger filter window, i.e. the minimum time between two triggers when the experiment runs in triggered mode. Referring to the previous discussion of the importance of quickly processing the BUSY OFF data words from the ALPIDE chips, it should be evident that a delay of this order is intolerable. The daisy-chain approach has been abandoned for that reason, leaving the BU as the only feasible option.

As a final remark, it should be mentioned that transmitting busy status from the RUs to the CTP/LTU is also not an option. There is no direct path from the RUs to transmit signals to the CTP/LTU, since the optical links for triggers from the LTU are downlinks only[1]. The only possible path would be via the CRU, with substantial latency considering that the latency from LTU to RU is already on the order of 1 µs on the trigger downlinks. The latency by itself makes this a futile venture, as the CTP would be making decisions based outdated busy information.

[1]Passive optical splitters are also used to split the downlink signals from the LTU.

Appendix G

Register Maps

This appendix provides register maps for the main and auxiliary FPGA designs. For the main FPGA the register maps were limited to the modules associated with this work, because of the large extent of that particular design. Complete register maps and further details for all modules can be found in the manuals for the main and auxiliary FPGA designs.

G.1 Main FPGA Design

G.1.1 Alpide Monitor - Sequencer

TABLE G.1: Main FPGA Register Map - Alpide Monitor Sequencer Registers.

Address	Width	Name	Description
0x00	4	CTRL	Control register
0x01	4	STATUS	Status register
0x02	16	ADDR	Address in instruction mem that the DATA[0..3]_WR registers writes to
0x03	16	DATA0_WR	Set up word 0 (bits 15:0) of instruction word to be written
0x04	16	DATA1_WR	Set up word 1 (bits 31:16) of instruction word to be written
0x05	16	DATA2_WR	Set up word 2 (bits 47:32) of instruction word to be written
0x06	16	DATA3_WR	Set up word 3 (bits 63:48) of instruction word to and toggle writting of entire data (bits 63:0) to instruction memory
0x07	16	INSTR_MEM_SBIT_ERR_ADDR	Address in instruction memory where single bit error was detected
0x08	16	INSTR_MEM_DBIT_ERR_ADDR	Address in instruction memory where double bit error was detected

TABLE G.2: Sequencer Control Register Fields.

Bit range	Field name	Description
0	Run single-shot bit	Start sequencer, run sequence one time
1	Run continuous bit	Start sequencer, run sequence continuously
2	Clear single-bit error	Clear single bit error status
3	Clear single-bit error	Clear double bit error status

TABLE G.3: Sequencer Status Register Fields.

Bit range	Field name	Description
0	Running bit	Sequencer is running
1	Unknown Op	Unknown Opcode encountered in instruction memory
2	Single-bit error	Single bit error encountered in instruction memory
3	Double-bit error	Double bit error encountered in instruction memory

G.1.2 Alpide Monitor - Sniffer

TABLE G.4: Main FPGA Register Map - Alpide Monitor Sniffer Registers.

Address	Width	Name	Description
0x00	3	STATUS	Status register
0x01	3	CTRL	Control register
0x02	16	RESULT_FIFO_WORD0	Read word 0 (bits 15:0) of result FIFO word (also toggles FIFO read enable)
0x03	16	RESULT_FIFO_WORD1	Read word 1 (bits 31:16) of result FIFO word
0x04	16	RESULT_FIFO_WORD2	Read word 2 (bits 47:32) of result FIFO word
0x05	16	RESULT_FIFO_WORD3	Read word 3 (bits 63:48) of result FIFO word

TABLE G.5: Sniffer Status Register Fields.

Bit range	Field name	Description
0	Running bit	Indicates if sniffer is running
1	FIFO empty bit	Indicates if result FIFO is empty
2	FIFO full bit	Indicates if result FIFO is full

TABLE G.6: Sniffer Control Register Fields.

Bit range	Field name	Description
0	Start bit	Start sniffer (pulsed bit)
1	Stop bit	Stop sniffer (pulsed bit)
2	Reset bit	Reset sniffer FSM (pulsed bit)

G.1.3 CAN HLP

TABLE G.7: Main FPGA Register Map - CAN HLP Registers.

Address	Width	Access	Name	Description
0x00	3	RW	CTRL	Control register
0x01	5	R	STATUS	Status register
0x02	6	RW	CAN_PROP_SEG	CAN controller propagation segment length
0x03	6	RW	CAN_PHASE_SEG1	CAN controller phase segment 1 length
0x04	6	RW	CAN_PHASE_SEG2	CAN controller phase segment 2 length
0x05	4	RW	CAN_SJW	CAN controller Synchronization Jump Width (SJW)
0x06	8	RW	CAN_CLK_SCALE	CAN controller clock scale
0x07	9	R	CAN_TEC	CAN Transmit Error Counter (TEC)
0x08	9	R	CAN_REC	CAN Receive Error Counter (REC)
0x09	16	RW	TEST_REG	Test/dummy register
0x0A	15	R	FSM_STATES	Top-level FSM states

TABLE G.8: CAN HLP Control Register Fields.

Bit range	Field name	Description
0	TRIPLE_SAMPLING_EN	Enable triple sampling of CAN bits
1	RETRANSMIT_EN	Enable retransmission of CAN frames
2	TEST_MODE_EN	Enable HLP test mode

TABLE G.9: CAN HLP Status Register Fields.

Bit range	Field name	Description
0	ERROR_ACTIVE	CAN controller in Error Active mode
1	ERROR_PASSIVE	CAN controller in Error Passive mode
2	BUS_OFF	CAN controller in Bus Off mode
3	CAN_RX	CAN controller Rx input value
4	CAN_TX	CAN controller Tx output value

TABLE G.10: CAN HLP FSM State Register Fields.

Bit range	Field name	Description
3:0	HLP_FSM_STATE	CAN HLP FSM state
8:4	CAN_FRAME_RX_FSM_STATE	CAN controller Rx Frame FSM state
14:9	CAN_FRAME_TX_FSM_STATE	CAN controller Tx Frame FSM state

G.1.4 FIFO Interface to PA3 FPGA for Configuration Data

TABLE G.11: Main FPGA Register Map - PA3 FIFO Interface Registers.

Address	Width	Access	Name	Description
0x00	16	W	WRITE_FIFO_DATA	Data to write to PA3 FIFO
0x09	1	W	TEST_REG	Test/dummy register

G.2 Auxiliary FPGA Design

TABLE G.12: Auxiliary FPGA Register Map – Version Registers. [64]

Address	Name	Width	Access	Reset	Description
0x00	MINOR_VERSION	8	R	0x03	[7:0]: Minor num
0x01	MAJOR_VERSION	8	R	0xA2	[7:4]: A = alpha, B = Beta, 0 = final release [3:0]: Major Num

TABLE G.13: Auxiliary FPGA Register Map – Git-hash Registers. [64]

Address	Name	Width	Access	Reset	Description
0x50	HASH_0	8	R	0x0	Git-hash (short version) of the commit the design was generated from
0x51	HASH_1	8	R	0x0	
0x52	HASH_2	8	R	0x0	
0x53	HASH_3	8	R	0x0	

TABLE G.14: Auxiliary FPGA Register Map – Debug Registers. [64]

Address	Name	Width	Access	Reset	Description
0x40	DIPSWITCH1	8	R	0x–	Status of connected dipswitches [9:0] Value of Dipswitches
0x41	DIPSWITCH2	2	R	0x–	
0x42	PUSHBUTTON	4	R	0x0	Pushbutton Values. Always 0 unless pressed down. *Note: Pushbutton[0] is used as master reset so this will always show up as 0.*
0x43	CTRL_LEDs	2	RW	0x0	Value to put out on LEDs of PA3

TABLE G.15: Auxiliary FPGA Register Map – Clock Registers. [64]

Address	Name	Width	Access	Reset	Description
0x02	CLR_CLK_LOL_CNTS	1	W(T)	0x00	[0]: Clear clock Loss of Lock (LoL) counters
0x03	STATUS_CLOCK	3	R	0x00	[0]: lcl_clk_lol [1]: lcl_clk_c1b [2]: lcl_clk_c2b
0x04	CNFG_CLOCK	3	RW	0x00	[0]: lcl_clk_in_sel [1]: lcl_clk_cs [2]: lcl_clk_rst
0x05	LOCAL_CLK_LOL_CNT	8	R	0x00	Loss of Lock counter, lol
0x06	LOCAL_CLK_C1B_CNT	8	R	0x00	Loss of Lock counter, c1b
0x07	LOCAL_CLK_C2B_CNT	8	R	0x00	Loss of Lock counter, c2b

TABLE G.16: Auxiliary FPGA Register Map - Config Controller Registers. [64]

Address	Name	Width	Access	Reset	Description
0x08	CC_CMD	8	RW	0x00	[6:0]: Config Controller Command Register h01: Init_config h02: continuous scrubbing h04: single scrubbing cycle h08: Stop continuous scrubbing h10: Clear Scrubbing Counter h20: Init config 2 [7]: EXECUTE command = '1' (T)
0x09	CC_STATUS	3	R	0x00	Config Controller Status Register [0]: busy [1]: config done [2]: scrubbing active [3]: scrubbing done [4]: config done, golden image [5]: error
0x0A	CC_ACTIVE_STATE	4	R	0x00	Active state of CC state machine h0: idle h1: init_sm_wait h2: init_flash_read h3: fifo_status h4: transfer_data h5: transfer_data_done h6: startup_wait h7: pause_scrub hF: others
0x0B 0x0C	CC_SCRUB_CNT_LSB CC_SCRUB_CNT_MSB	8 8	R R	0x00 0x00	Number of scrubbing cycles executed
0x0D	CC_CTRL	1	RW	0x00	[0]: Set high to pause continuous scrubbing

TABLE G.17: Auxiliary FPGA Register Map – Read Controller Registers. [64]

Address	Name	Width	Access	Reset	Description
0x10	RC_CMD	8	RW	0x00	[6:0]: Read Controller Command register H01: READ_PARAMETER h02: READ_CONFIG h04: READ_SCRUB h08: START_PAGE h10: STOP_PAGE h20: READ_CONFIG2 [7] EXECUTE command = '1' (T)
0x11	RC_STATUS	8	RW	0x00	Read Controller Status Register: [0]: RC_busy [1]: Waiting_for_flash [2]: Reading bitfile [3]: Reading parameters [4]: Config parameter ok [5]: BS parameter ok [6]: Parameter error [7]: Config 2 parameter ok
0x12 0x13 0x14	RC_FLASHPAGE1 RC_FLASHPAGE2 RC_FLASHPAGE3	8 8 3	R R R	0x00 0x00 0x00	Page address read from flash [18:0]

TABLE G.18: Auxiliary FPGA Register Map – SelectMap Registers. [64]

Address	Name	Width	Access	Reset	Description
0x18	SMAP_CMD	8	RW	0x00	[5:0]: SelectMAP Command register h01: Init Xilinx h02: Startup h04: write one byte h08: read one byte h10: abort h20: read/write finished [6]: Clears any error in SelectMap interface. (T) [7]: EXECUTE command = '1' (T)
0x19	SMAP_DATA_TX	8	R	0x00	Byte read from Xilinx SelectMap interface
0x1A	SMAP_DATA_RX	8	RW	0x00	Byte to write to Xilinx SelectMap interface
0x1B	SMAP_STATUS	6	R	0x00	SelectMap Interface Status register [0]: init_b does not respond to prog_b = 0 [1]: init_b does not go high after prog_b [2]: Done never goes high during startup [3]: Done status from Xilinx (high if device is configured) [4]: Configuration done successfully [5]: Interface busy

TABLE G.19: Auxiliary FPGA Register Map – Read FIFO Registers. [64]

Address	Name	Width	Access	Reset	Description
0x30	FIFO_DATA_RD	8	R	0x00	Return data from ECC FIFO. *Note: Must not be read during init config or scrubbing*
0x31	FIFO_DATA_WR	8	(R)W	0x00	Data to write to Flash. *Note: When writing to Flash the number of bytes written to the Flash must match page size given by FLASH_TRX_SIZE_LSB/MSB*
0x32	FIFO_STATUS	5	R	0x1A	Status of both Write FIFO and Read FIFO [0]: Read FIFO full [1]: Read FIFO empty [2]: Write FIFO full [3]: Write FIFO empty [4]: Xilinx FIFO Empty ('1' if Xilinx is configured, '0' if not)

TABLE G.20: Auxiliary FPGA Register Map – Write Controller Registers. [64]

Address	Name	Width	Access	Reset	Description
0x33	FIFO_WRITER_CMD	8	RW	0x0	FIFO Write controller command register [6:0]: Command h01: Write to flash via Xilinx FIFO h02: Write to Flash via I2C/UART h04: Stop writing. Executed after all data has been written h08: Clear error flags [7]: Execute (T)
0x34	FIFO_WRITER_STATUS	5	R	0x01	Status register of FIFO write controller [0]: Ready to accept data [1]: Active input ('0' = I2C/UART, '1' = Xilinx) [2]: Ending data transmission active [3]: Error [4]: Command not recognized [5]: TMR Error in FIFO
0x35	FIFO_WRITER_TMR_ERR	8	R	0x00	Number of TMR errors in triplicated Write FIFO

TABLE G.21: Auxiliary FPGA Register Map – Flash Interface Registers. [64]

Address	Name	Width	Access	Reset	Description
0x20	FLASH_CTRL	8	RW	0x00	Flash Interface Command Register [5:0]: Command h01: PAGE_WRITE h02: PAGE_READ h04: READ_ID h08: READ_STATUS h10: RESET_FLASH h20: BLOCK_ERASE [6]: Verify Pattern [7]: Flash Interface Execute (T)
0x21	FLASH_STATUS	7	R	0x00	Flash Interface Status Register [0]: Done with command (Read/Write/Erase/Read ID) [1]: FIFO status (Write FIFO EMPTY or Read FIFO FULL) [2]: Status bit from Flash after ended command. '1' when an error has occured. [3]: Write Active [4]: Read Active [5]: Command active (ReadID, Erase, Reset) [6]: Trx_done (sticky bit)
0x22	FLASH_PATTERN	8	RW	0x00	Debug feature When the 'Verify Pattern' bit in FLASH_CTRL is set, each byte in a page that is read back is verified against this pattern.
0x23	FLASH_MIS_CNT	8	R	0x00	Debug feature Number of pattern recognition errors when 'Verify Pattern' is enabled.
0x24	FLASH_ROW_ADDR1	8	RW	0x00	[5:0]: Page address [7:6]: Block address[1:0]
0x25	FLASH_ROW_ADDR2	8	RW	0x00	[7:0]: Block address[9:2]
0x26	FLASH_ROW_ADDR3	5	RW	0x00	[3:0] Block address[13:10] [7:4]: Not used
0x27	FLASH_TRX_SIZE_LSB	8	RW	0x60	Page size on Flash: 4096 (0x1000): Not using spare section (default) 4192 (0x1060): Page size for ECC encoding (32x3 bytes hamming codes) 4224 (0x1080): Page size incl. full spare section
0x28	FLASH_TRX_SIZE_MSB	5	RW	0x10	
0x29	FLASH_STATUS_WORD	8	R	0x00	Status word transferred from Flash memory after erase command – See table 3 on page 55 of data sheet
0x2A	FLASH_SELECT_IC	2	RW	0x01	Select which Flash IC to use. Default is chip 1. b00: Not legal value. Default Flash b01 (IC 1) is selected b01: Flash IC 1 (FLASH_CE1/FLASH_R_B1_n) b10: Flash IC 2 (FLASH_CE2/FLASH_R_B2_n) b11: Both IC1 and IC2 active (*NOT LEGAL FOR READ PAGE OR READ ID OPERATION - WILL READ CORRUPT DATA. Writing and Erasing is OK*)
0x2B	FLASH_TRX_CNT_MSB	5	R	0x00	Number of bytes written or read by Flash Interface
0x2C	FLASH_TRX_CNT_LSB	8	R	0x00	
0x46	FLASH_STATE_0	8	R	0x00	Debug registers showing active states of all FSMs in Flash Interface. [3:0]: NAND Flash Controller FSM [9:4]: Page Write Control FSM [15:10]: Page Read Control FSM [21:16]: Block Erase Control FSM [25:22]: Read ID Control FSM
0x47	FLASH_STATE_1	8	R	0x00	
0x48	FLASH_STATE_2	8	R	0x00	
0x4D	FLASH_STATE_3	2	R	0x00	
0x4E	FLASH_ST_WRD	8	R	0x00	Status word from Flash after a transaction is finished. [0]: Error flag (positive polarity) [7:1]: Don't care

TABLE G.22: Auxiliary FPGA Register Map – ECC Registers. [64]

Address	Name	Width	Access	Reset	Description
0x38	ECC_COMMAND	2	RW	0x0	ECC Command and Control register [0]: Enable ECC (positive polarity) [1]: Clear Status and Counters (T)
0x39	ECC_STATUS	3	R	0x0	ECC Status register [0]: Double bit error (uncorrectable) [1]: Single bit error [2]: ECC error (single bit error in ECC code) [3]: TMR error in Read FIFO Writing any value to this register clears it
0x3A	ECC_SB_CNT	8	R	0x00	Single Bit Error Counter
0x3B	ECC_FIFOTMR_ERR_CNT	8	R	0x00	Number of TMR errors in triplicated Read FIFO

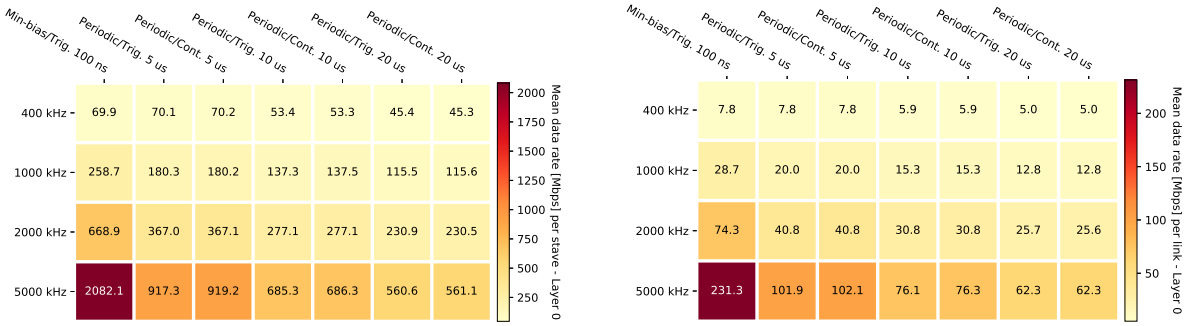
TABLE G.23: Auxiliary FPGA Register Map – CRC Registers. [64]

Address	Name	Width	Access	Reset	Description
0x49	CRC_0	8	R	0x0	Calculated CRC-32 value from the CRC checker
0x4A	CRC_1	8	R	0x0	
0x4B	CRC_2	8	R	0x0	
0x4C	CRC_3	8	R	0x0	

Appendix H

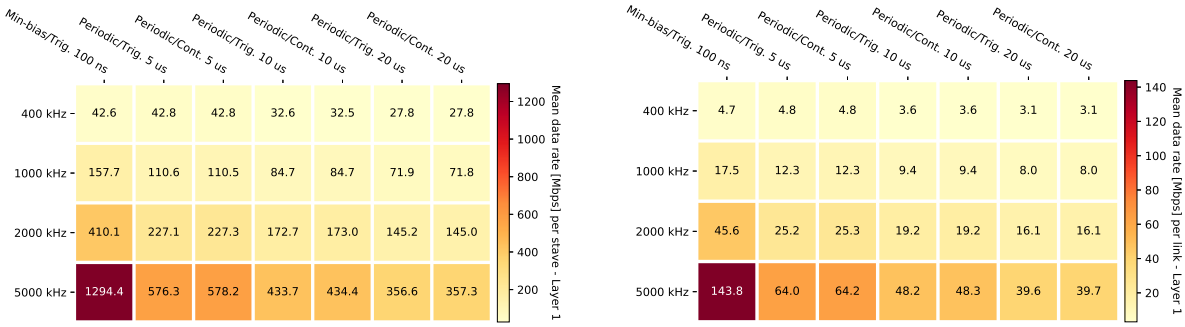
Simulation Results

H.1 ITS - pp



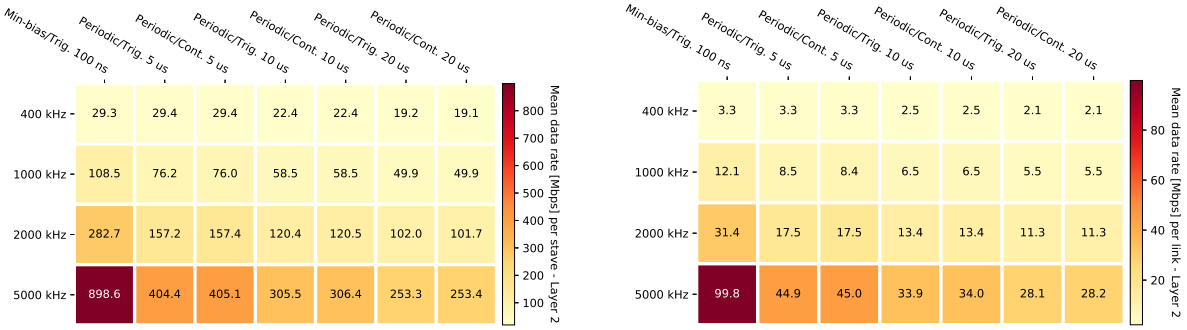
(A) Per stave (B) Per link

FIGURE H.1: Average data rate per stave/RU (A) and per link (B), simulated for pp in layer 0 of the ITS.



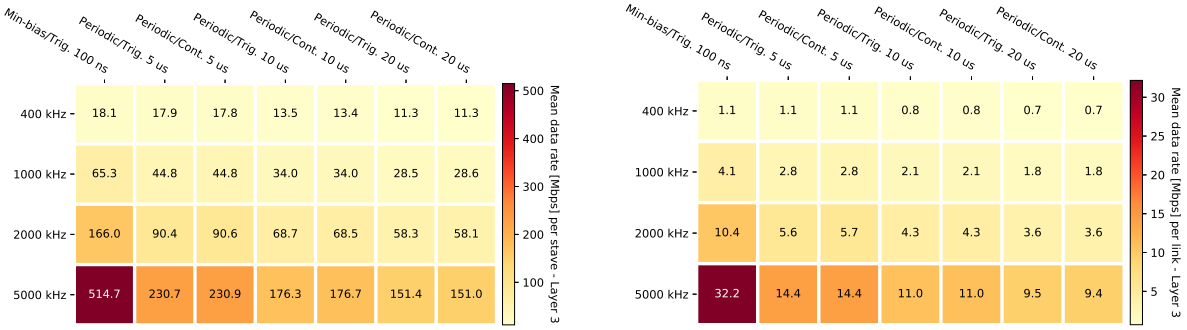
(A) Per stave (B) Per link

FIGURE H.2: Average data rate per stave/RU (A) and per link (B), simulated for pp in layer 1 of the ITS.



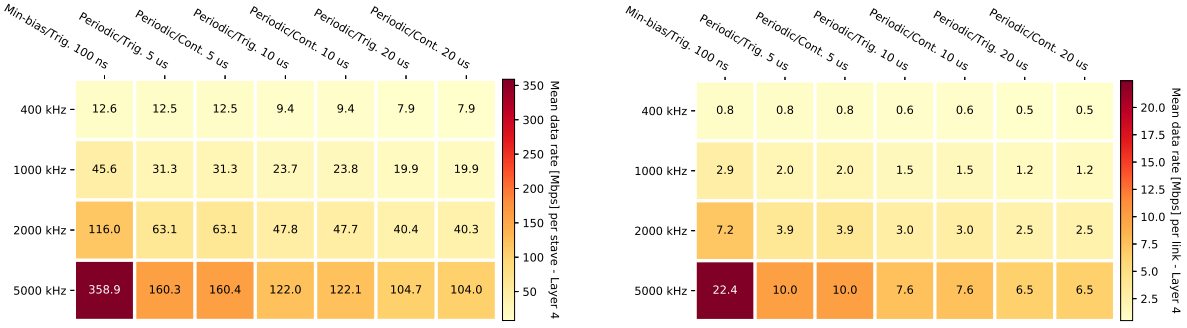
(A) Per stave (B) Per link

FIGURE H.3: Average data rate per stave/RU (A) and per link (B), simulated for pp in layer 2 of the ITS.



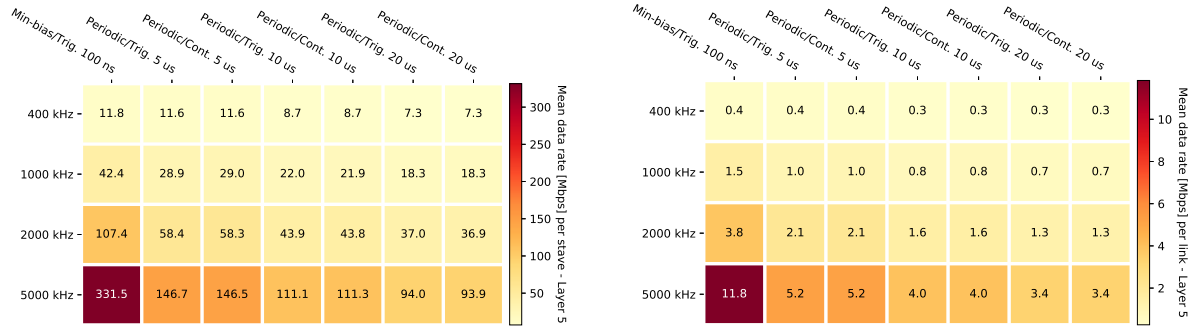
(A) Per stave (B) Per link

FIGURE H.4: Average data rate per stave/RU (A) and per link (B), simulated for pp in layer 3 of the ITS.



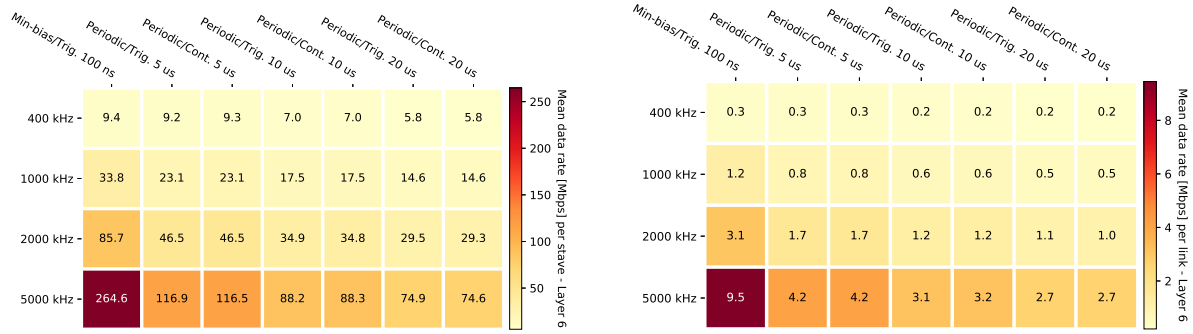
(A) Per stave (B) Per link

FIGURE H.5: Average data rate per stave/RU (A) and per link (B), simulated for pp in layer 4 of the ITS.



(A) Per stave (B) Per link

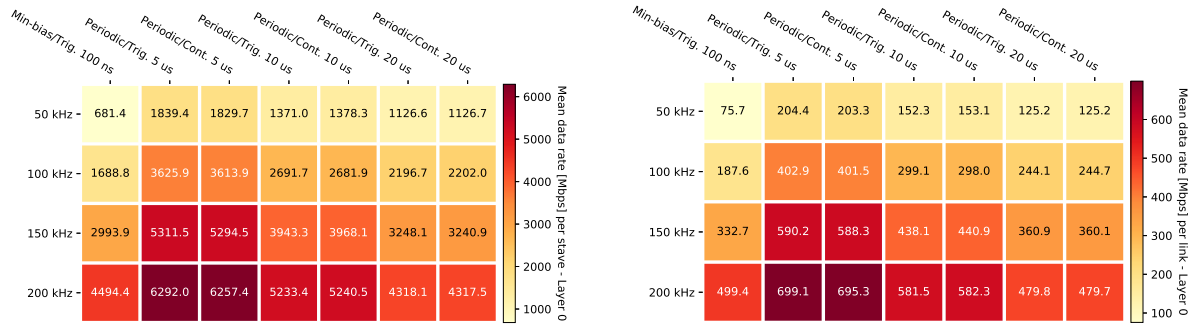
FIGURE H.6: Average data rate per stave/RU (A) and per link (B), simulated for pp in layer 5 of the ITS.



(A) Per stave (B) Per link

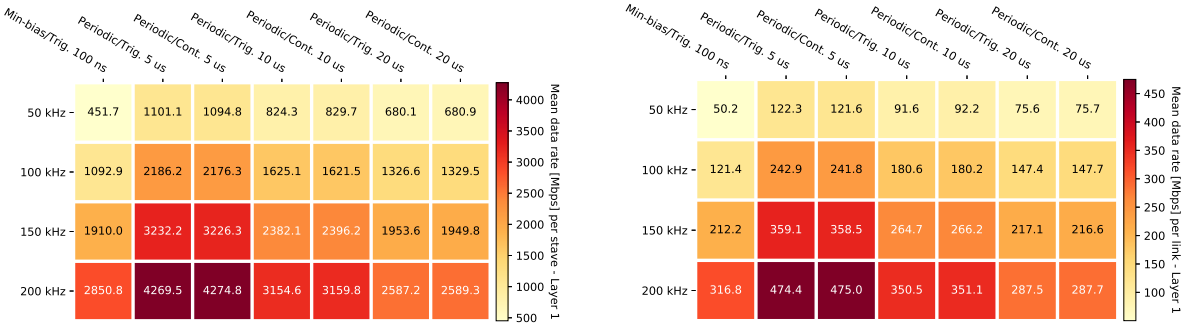
FIGURE H.7: Average data rate per stave/RU (A) and per link (B), simulated for pp in layer 6 of the ITS.

H.2 ITS - Pb–Pb



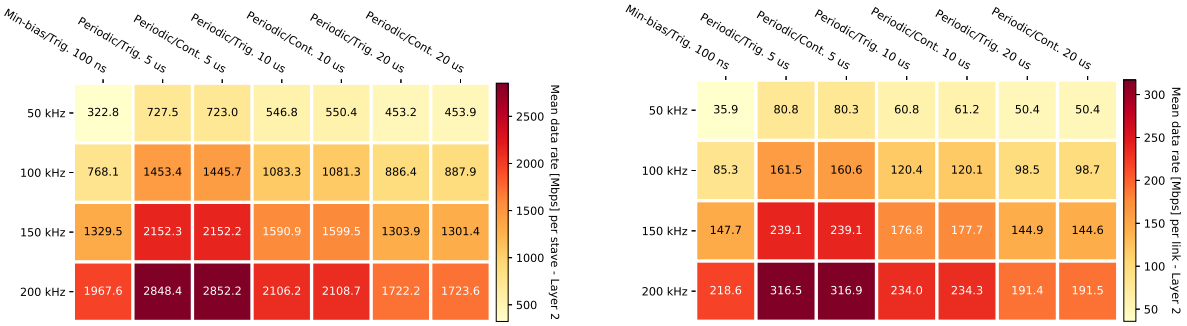
(A) Per stave (B) Per link

FIGURE H.8: Average data rate per stave/RU (A) and per link (B), simulated for Pb–Pb in layer 0 of the ITS.



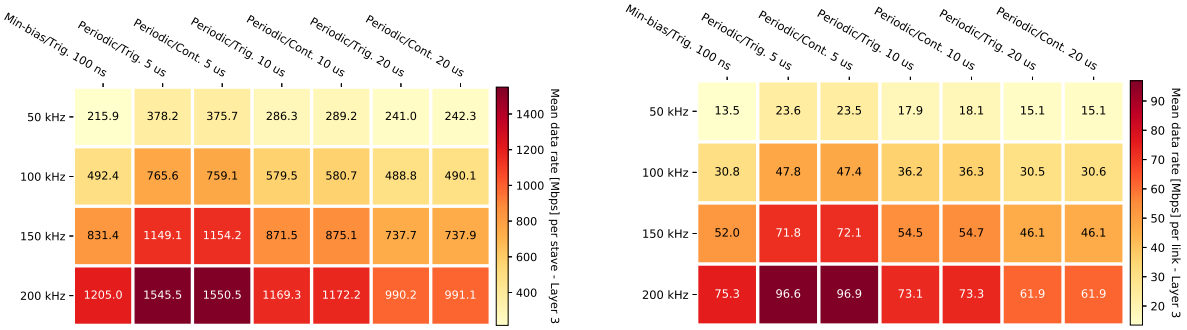
(A) Per stave (B) Per link

FIGURE H.9: Average data rate per stave/RU (A) and per link (B), simulated for Pb–Pb in layer 1 of the ITS.



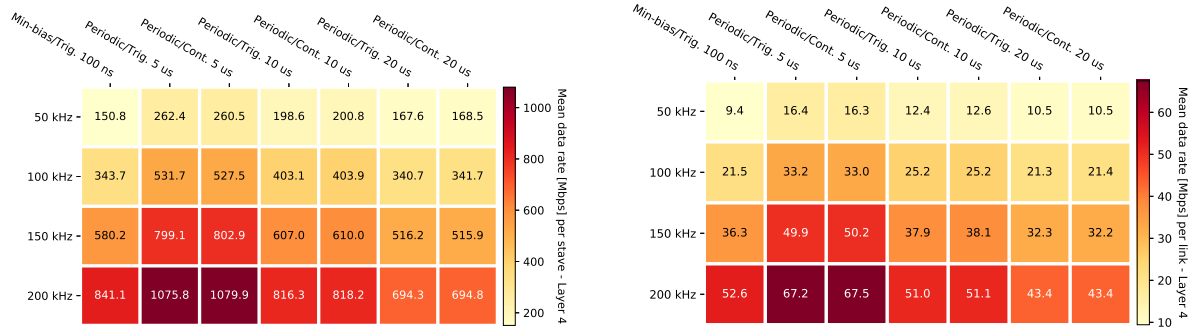
(A) Per stave (B) Per link

FIGURE H.10: Average data rate per stave/RU (A) and per link (B), simulated for Pb–Pb in layer 2 of the ITS.



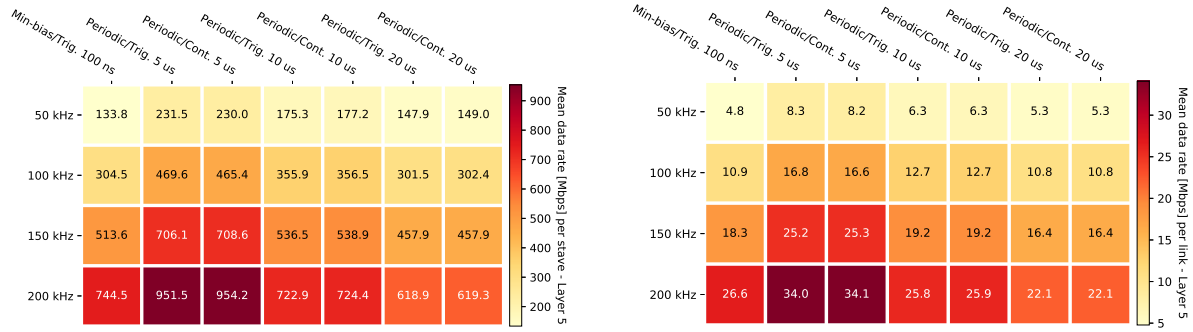
(A) Per stave (B) Per link

FIGURE H.11: Average data rate per stave/RU (A) and per link (B), simulated for Pb–Pb in layer 3 of the ITS.



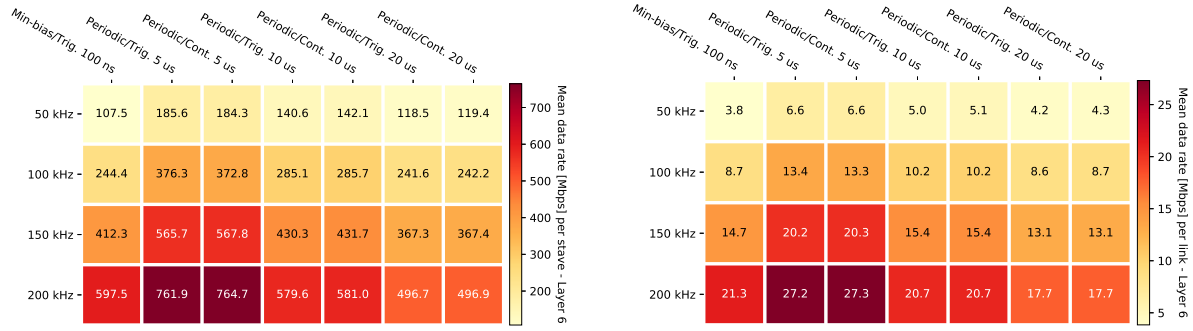
(A) Per stave (B) Per link

FIGURE H.12: Average data rate per stave/RU (A) and per link (B), simulated for Pb–Pb in layer 4 of the ITS.



(A) Per stave (B) Per link

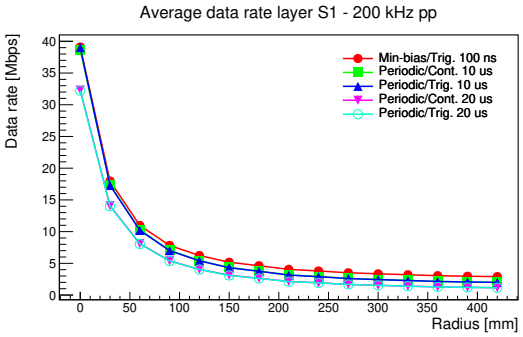
FIGURE H.13: Average data rate per stave/RU (A) and per link (B), simulated for Pb–Pb in layer 5 of the ITS.



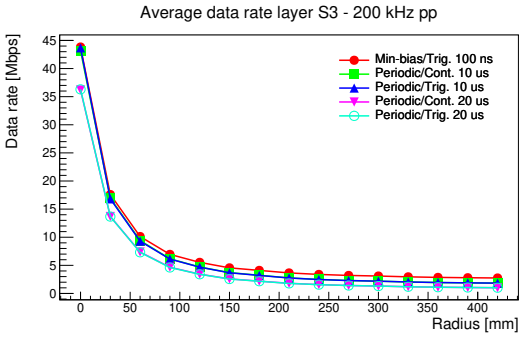
(A) Per stave (B) Per link

FIGURE H.14: Average data rate per stave/RU (A) and per link (B), simulated for Pb–Pb in layer 6 of the ITS.

H.3 FoCal - pp

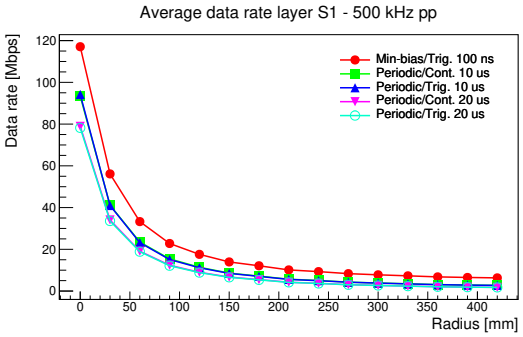


(A) Layer S1

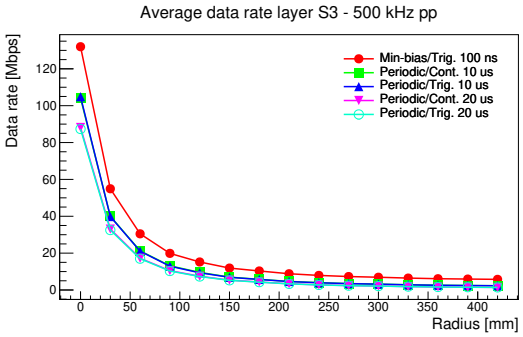


(B) Layer S3

FIGURE H.15: Average data rate versus radius for different trigger and strobe settings, simulated for 200 kHz pp in layer S1 (A) and layer S3 (B) of the FoCal detector.

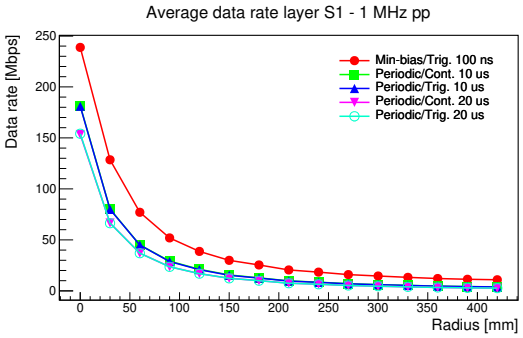


(A) Layer S1

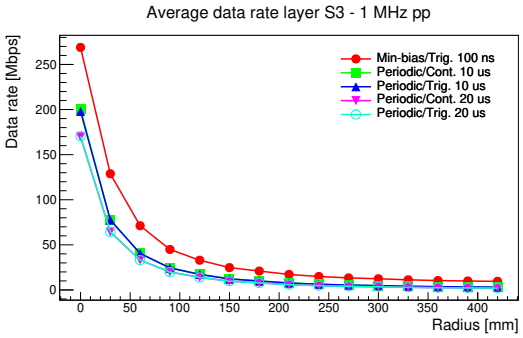


(B) Layer S3

FIGURE H.16: Average data rate versus radius for different trigger and strobe settings, simulated for 500 kHz pp in layer S1 (A) and layer S3 (B) of the FoCal detector.

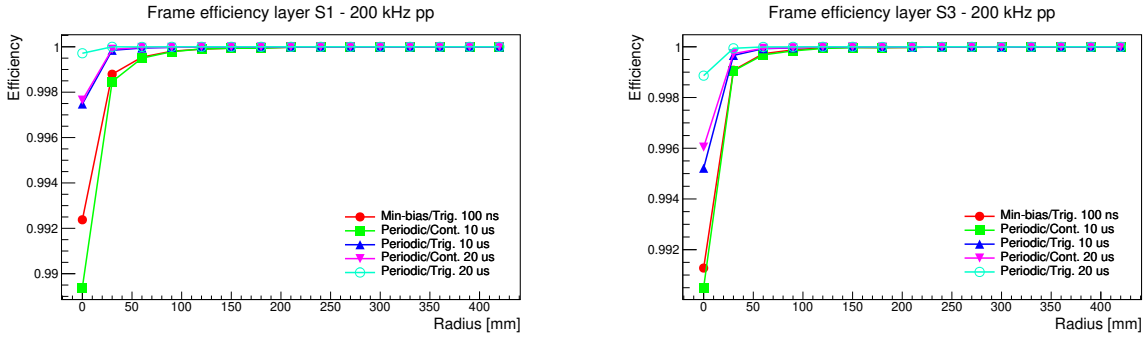


(A) Layer S1



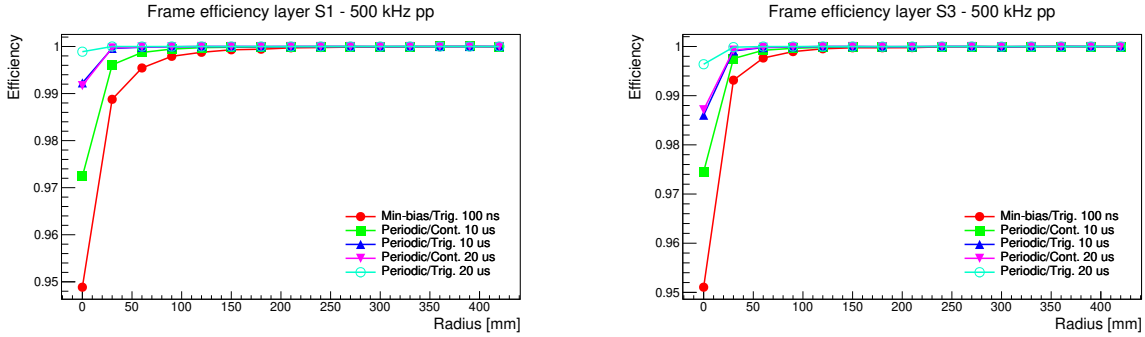
(B) Layer S3

FIGURE H.17: Average data rate versus radius for different trigger and strobe settings, simulated for 1 MHz pp in layer S1 (A) and layer S3 (B) of the FoCal detector.



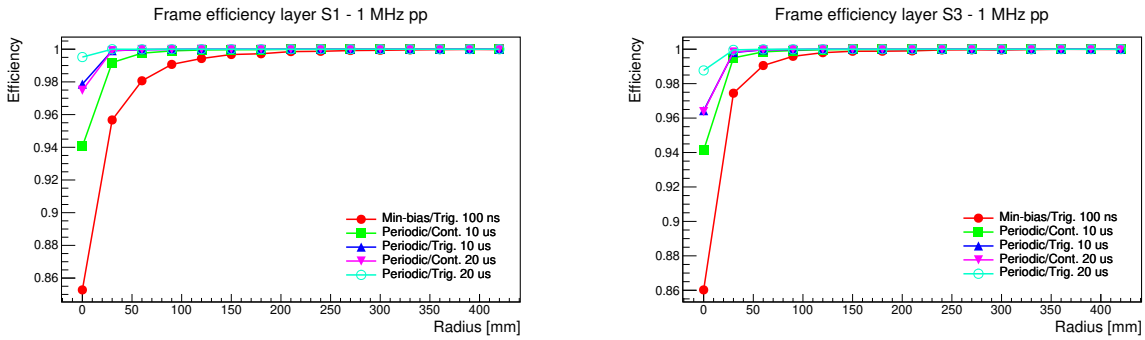
(A) Layer S1 (B) Layer S3

FIGURE H.18: Readout efficiency versus radius for different trigger and strobe settings, simulated for 200 kHz pp in layer S1 (A) and layer S3 (B) of the FoCal detector.



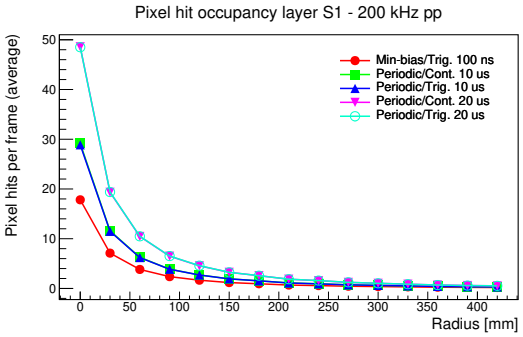
(A) Layer S1 (B) Layer S3

FIGURE H.19: Readout efficiency versus radius for different trigger and strobe settings, simulated for 500 kHz pp in layer S1 (A) and layer S3 (B) of the FoCal detector.

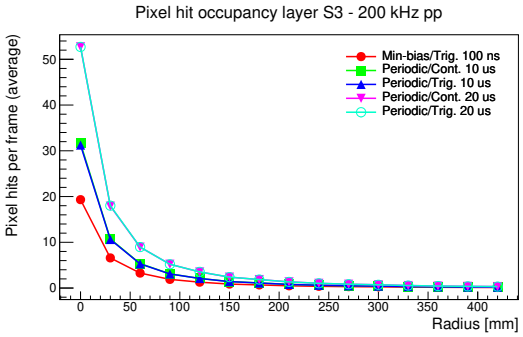


(A) Layer S1 (B) Layer S3

FIGURE H.20: Readout efficiency versus radius for different trigger and strobe settings, simulated for 1 MHz pp in layer S1 (A) and layer S3 (B) of the FoCal detector.

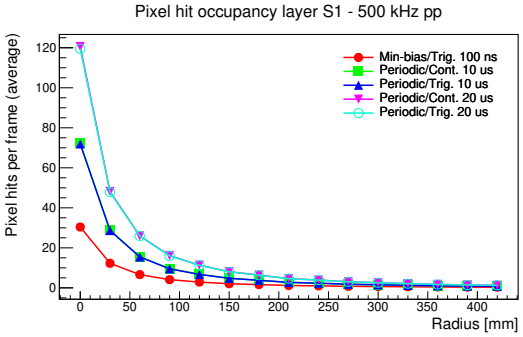


(A) Layer S1

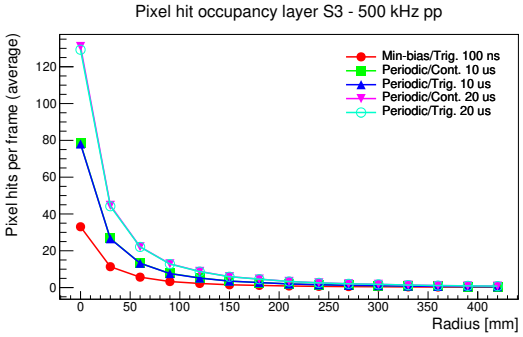


(B) Layer S3

FIGURE H.21: Pixel hit occupancy versus radius for different trigger and strobe settings, simulated for 200 kHz pp in layer S1 (A) and layer S3 (B) of the FoCal detector.

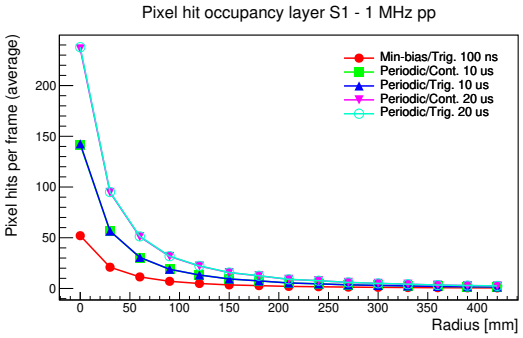


(A) Layer S1

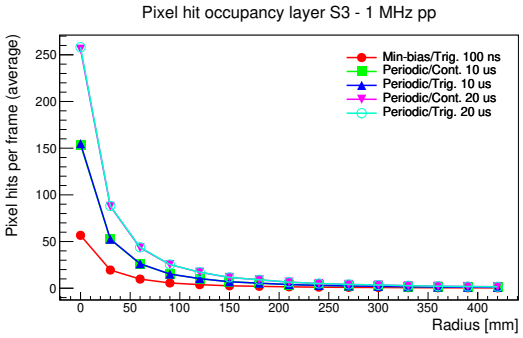


(B) Layer S3

FIGURE H.22: Pixel hit occupancy versus radius for different trigger and strobe settings, simulated for 500 kHz pp in layer S1 (A) and layer S3 (B) of the FoCal detector.



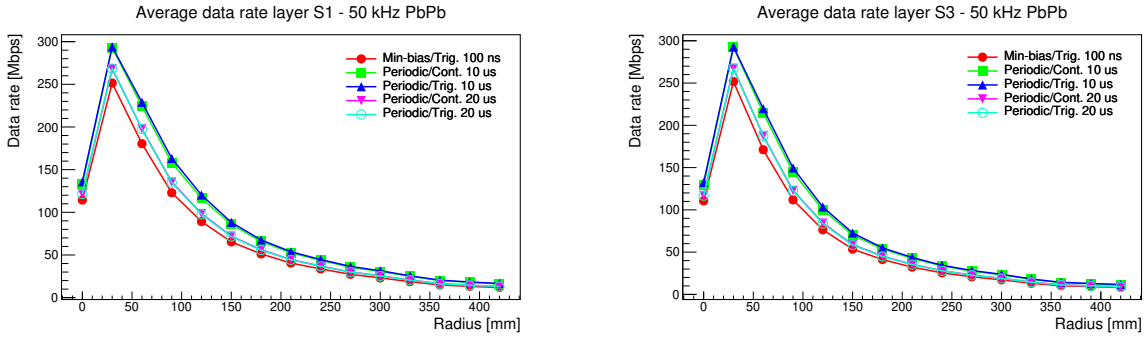
(A) Layer S1



(B) Layer S3

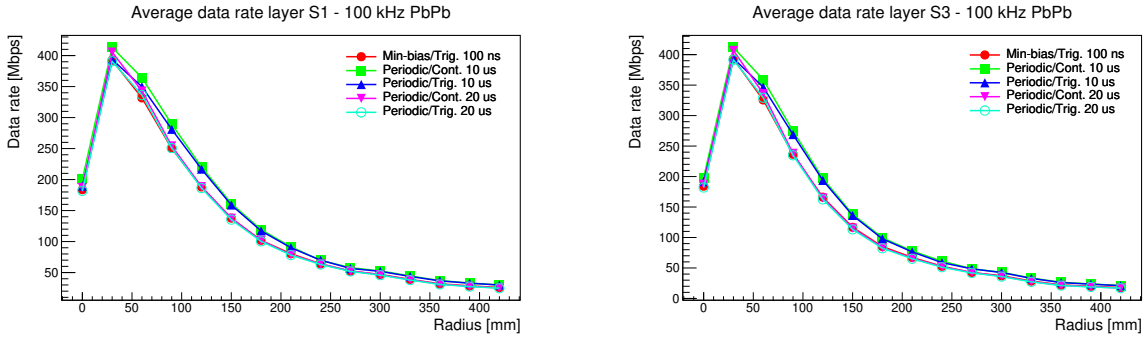
FIGURE H.23: Pixel hit occupancy versus radius for different trigger and strobe settings, simulated for 1 MHz pp in layer S1 (A) and layer S3 (B) of the FoCal detector.

H.4 FoCal - Pb–Pb



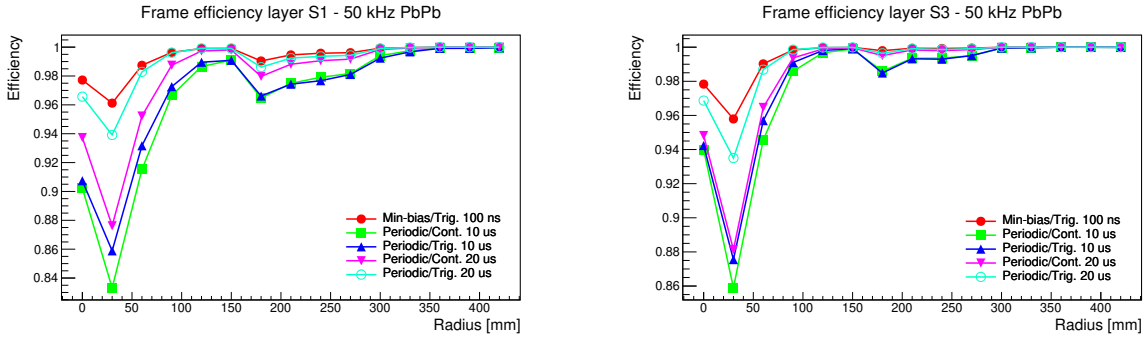
(A) Layer S1 (B) Layer S3

FIGURE H.24: Average data rate versus radius for different trigger and strobe settings, simulated for 50 kHz Pb–Pb in layer S1 (A) and layer S3 (B) of the FoCal detector.



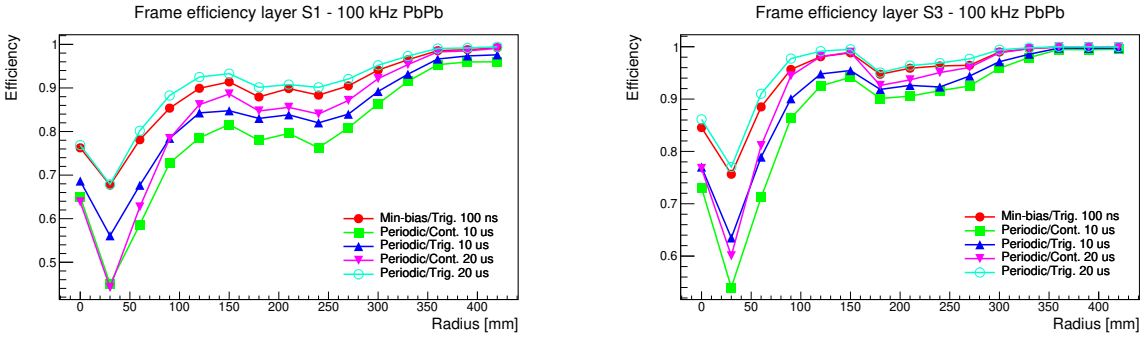
(A) Layer S1 (B) Layer S3

FIGURE H.25: Average data rate versus radius for different trigger and strobe settings, simulated for 100 kHz Pb–Pb in layer S1 (A) and layer S3 (B) of the FoCal detector.



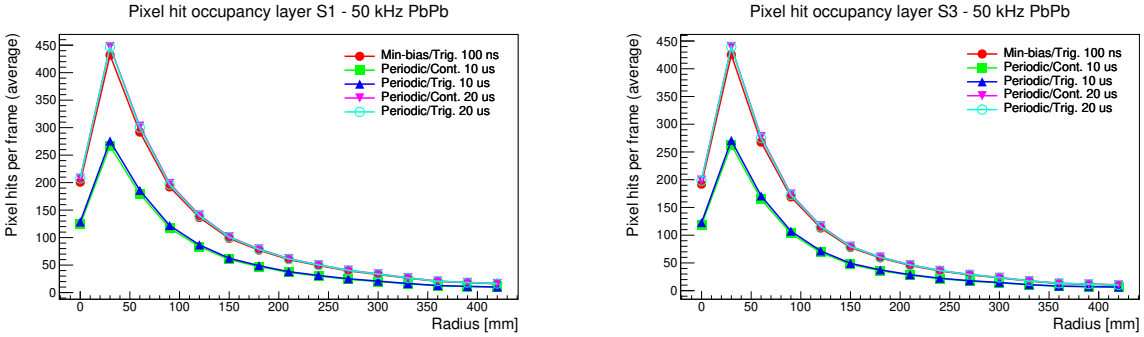
(A) Layer S1 (B) Layer S3

FIGURE H.26: Readout efficiency versus radius for different trigger and strobe settings, simulated for 50 kHz Pb–Pb in layer S1 (A) and layer S3 (B) of the FoCal detector.



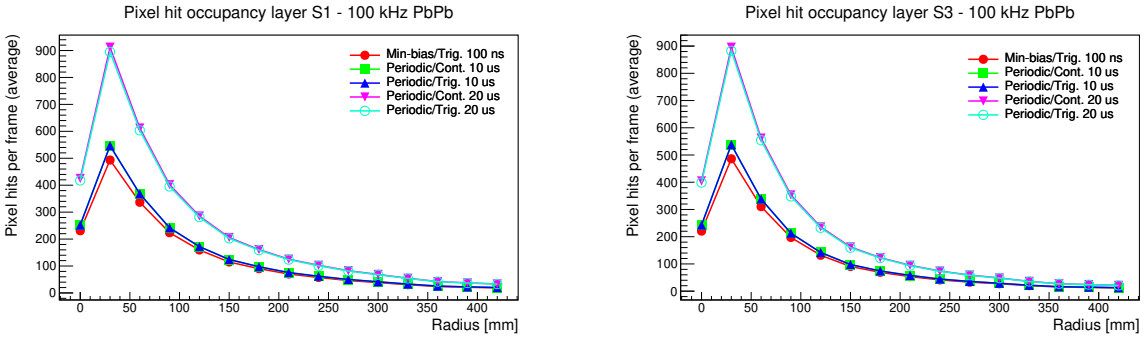
(A) Layer S1 (B) Layer S3

FIGURE H.27: Readout efficiency versus radius for different trigger and strobe settings, simulated for 100 kHz Pb–Pb in layer S1 (A) and layer S3 (B) of the FoCal detector.



(A) Layer S1 (B) Layer S3

FIGURE H.28: Pixel hit occupancy versus radius for different trigger and strobe settings, simulated for 50 kHz Pb–Pb in layer S1 (A) and layer S3 (B) of the FoCal detector.



(A) Layer S1 (B) Layer S3

FIGURE H.29: Pixel hit occupancy versus radius for different trigger and strobe settings, simulated for 100 kHz Pb–Pb in layer S1 (A) and layer S3 (B) of the FoCal detector.

Bibliography

[1] Wikipedia contributors, *Electron — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Electron&oldid=1068855589, [Online; accessed 20-February-2022], 2022.

[2] ——, *Parton (particle physics) — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Parton_(particle_physics)&oldid=1070425688, [Online; accessed 20-February-2022], 2022.

[3] ——, *Quark — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Quark&oldid=1071943911, [Online; accessed 20-February-2022], 2022.

[4] ——, *Standard Model — Wikipedia, The Free Encyclopedia*, [Online; accessed 4-December-2019], 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Standard_Model&oldid=928956838.

[5] W. Busza, K. Rajagopal, and W. van der Schee, "Heavy Ion Collisions: The Big Picture and the Big Questions," *Annual Review of Nuclear and Particle Science*, vol. 68, no. 1, pp. 339–376, 2018. DOI: 10.1146/annurev-nucl-101917-020852.

[6] OpenStax, *Chemistry: Atom First*. OpenStax CNX, 2013. [Online]. Available: http://cnx.org/contents/4539ae23-1ccc-421e-9b25-843acbb6c4b0@3.1.

[7] G. Charpak, R. Bouclier, T. Bressani, J. Favier, and C. Zupancic, "The use of multiwire proportional counters to select and localize charged particles," *Nucl. Instrum. Methods*, vol. 62, pp. 262–268, 1968. DOI: 10.1016/0029-554X(68)90371-6. [Online]. Available: http://cds.cern.ch/record/347202.

[8] G. Charpak and F. Sauli, "Multiwire proportional chambers and drift chambers," *Nucl. Instrum. Methods*, vol. 162, 405–428. 25 p, 1979. DOI: 10.1016/0029-554X(79)90726-2. [Online]. Available: http://cds.cern.ch/record/133177.

[9] V. Frigo, "LHC map in 3D. Schéma du LHC en 3D," AC Collection. Legacy of AC. Pictures from 1992 to 2002., 1997, [Online]. Available: https://cds.cern.ch/record/842700.

[10] "CERN", *"Powering CERN"*, [Online; accessed 26-September-2020]. [Online]. Available: https://home.cern/science/engineering/powering-cern.

[11] K Aamodt, A Abrahantes Quintana, R Achenbach, *et al.*, "The ALICE experiment at the CERN LHC. A Large Ion Collider Experiment," *JINST*, vol. 3, S08002. 259 p, 2008, Also published by CERN Geneva in 2010. DOI: 10.1088/1748-0221/3/08/S08002. [Online]. Available: https://cds.cern.ch/record/1129812.

[12] The ALICE Collaboration, "ALICE Inner Tracking System (ITS): Technical Design Report," Tech. Rep., 1999. [Online]. Available: https://cds.cern.ch/record/391175.

[13] ——, "Technical Design Report for the Upgrade of the ALICE Inner Tracking System," Tech. Rep., 2013. DOI: 10.1088/0954-3899/41/8/087002. [Online]. Available: https://cds.cern.ch/record/1625842.

[14] Arturo Tauro, *3D ALICE Schematic RUN3 - with Description*, [Online; accessed 19-June-2019], 2019. [Online]. Available: https://alice-figure.web.cern.ch/node/11220.

[15] "Performance of the ALICE experiment at the CERN LHC," *International Journal of Modern Physics A*, vol. 29, no. 24, p. 1 430 044, 2014, ISSN: 1793-656X. DOI: 10.1142/s0217751x14300440. [Online]. Available: http://dx.doi.org/10.1142/S0217751X14300440.

[16] M. Vito, G. Anelli, F Antinori, A Boccardi, G Bruno, M Burns, I. Cali, M. Campbell, M. Caselle, P Chochula, M. Cinausero, A Dalessandro, R. Dima, R. Dinapoli, D Elia, D Fabris, R. Fini, E Fioretto, F. Formenti, and T Virgili, "The silicon pixel detector (SPD) for the ALICE experiment," *Journal of Physics G: Nuclear and Particle Physics*, vol. 30, S1091, Jul. 2004. DOI: 10.1088/0954-3899/30/8/065.

[17] C. Gemme, "The ATLAS tracker pixel detector for HL-LHC," in *Proceedings of The 25th International workshop on vertex detectors — PoS(Vertex 2016)*, vol. 287, 2017, p. 019. DOI: 10.22323/1.287.0019.

[18] The Tracker Group of the CMS Collaboration, *The CMS Phase-1 Pixel Detector Upgrade*, 2020. arXiv: 2012.14304 (physics.ins-det).

[19] C. Dean, "The sPHENIX experiment at RHIC," *PoS*, vol. ICHEP2020, p. 731, 2021. DOI: 10.22323/1.390.0731.

[20] A. Alscher, K. Hencken, D. Trautmann, and G. Baur, "Multiple electromagnetic electron-positron pair production in relativistic heavy-ion collisions," *Physical Review A*, vol. 55, no. 1, 396–401, 1997, ISSN: 1094-1622. DOI: 10.1103/physreva.55.396. [Online]. Available: http://dx.doi.org/10.1103/PhysRevA.55.396.

[21] T. A. Collaboration, *ITS stand-alone tracking efficiency for current and upgraded ITS*, [This is a replotting of Fig 7.12 of the ITS TDR], 2015. [Online]. Available: https://alice-figure.web.cern.ch/node/8785.

[22] ——, *Impact parameter resolution (rphi and z) for current ITS (PbPb data) and upgraded ITS*, [This is a replotting of Fig 7.6 of the ITS TDR], 2015. [Online]. Available: https://alice-figure.web.cern.ch/node/8784.

[23] T. Kugathasan, C. Cavicchioli, P. L. Chalmet, P. Giubilato, H. Hillemanns, A. Junique, M. Mager, C. A. Marin Tobon, P. Martinengo, S. Mattiazzo, H. Mugnier, L. Musa, D. Pantano, J. Rousset, F. Reidt, P. Riedler, W. Snoeys, J. W. van Hoorne, and P. Yang, "Explorer-0: A Monolithic Pixel Sensor in a 180 nm CMOS process with an 18 µm thick high resistivity epitaxial layer," in *2013 IEEE Nuclear Science Symposium and Medical Imaging Conference (2013 NSS/MIC)*, 2013, pp. 1–5. DOI: 10.1109/NSSMIC.2013.6829476.

[24] P. Giubilato, *Readout Electronic - WP10*, Internal communication, 2016.

[25] G. Tambave, J. Alme, G. Barnaföldi, R. Barthel, A. van den Brink, S. Brons, M. Chaar, V. Eikeland, G. Genov, O. Grøttvik, H. Pettersen, Z. Pastuovic, S. Huiberts, H. Helstrup, K. Hetland, S. Mehendale, I. Meric, Q. Malik, O. Odland, G. Papp, T. Peitzmann, P. Piersimoni, A. Ur Rehman, F. Reidt, M. Richter, D. Röhrich, A. Sudar, A. Samnøy, J. Seco, H. Shafiee, E. Skjæveland, J. Sølie, K. Ullaland, M. Varga-Kofarago, L. Volz, B. Wagner, and S. Yang, "Characterization of monolithic CMOS pixel sensor chip with ion beams for application in particle computed tomography," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 958, p. 162 626, 2020, Proceedings of the Vienna Conference on Instrumentation 2019, ISSN: 0168-9002. DOI: 10.1016/j.nima.2019.162626. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168900219311258.

[26] ALICE ITS ALPIDE development team, *ALPIDE Operations Manual*, ALICE Internal Communication, 2016.

[27] Manzari, Vito, *OB HIC Construction and Prototypes*, Internal communication: Presentation at ITS Upgrade - Stave Production Readiness Review, 2017.

[28] F Costa, A Kluge, and P. V. Vyvre, "The detector read-out in ALICE during Run 3 and 4," *Journal of Physics: Conference Series*, vol. 898, 2017. DOI: 10.1088/1742-6596/898/3/032011. [Online]. Available: https://doi.org/10.1088/1742-6596/898/3/032011.

[29] P Antonioli, A Kluge, and W Riegler, "Upgrade of the ALICE Readout & Trigger System," Tech. Rep., 2013. [Online]. Available: https://cds.cern.ch/record/1603472.

[30] P. Giubilato, *System timing and trigger*, Internal communication: Presentation at ITS Upgrade - Readout Electronics Engineering Design Review, 2017.

[31] K. Sielewicz, G. A. Rinella, M. Bonora, J. Ferencei, P. Giubilato, M. J. Rossewij, J. Schambach, and T. Vanat, "*Prototype readout electronics for the upgraded ALICE Inner Tracking System*," *Journal of Instrumentation*, vol. 12, 01 2017. DOI: 10.1088/1748-0221/12/01/C01008.

[32] "Radiation Dose and Fluence in ALICE after LS2," 2018. [Online]. Available: http://cds.cern.ch/record/2642401.

[33] Microsemi, *Neutron-Induced Single Event Upset (SEU) FAQ*, [Online; accessed 6-March-2022], 2011. [Online]. Available: https://www.microsemi.com/document-portal/doc_view/130760-neutron-seu-faq.

[34] K Roed, J Alme, D Fehlker, C Lippmann, and A Rehman, "First measurement of single event upsets in the readout control FPGA of the ALICE TPC detector," *JINST*, vol. 6, p. C12022, 2011. DOI: 10.1088/1748-0221/6/12/C12022. [Online]. Available: http://cds.cern.ch/record/1452942.

[35] S. V. Nesbo, J. Alme, M. Bonora, M Rentsch Ersdal, P. Giubilato, H. Helstrup, M. Lupi, G Aglieri Rinella, D. Röhrich, J. Schambach, A. Velure, and S. Yuan, "Implementation of a CANbus interface for the Detector Control System in the ALICE ITS Upgrade," *PoS*, vol. TWEPP2019, p. 083, 2020. DOI: 10.22323/1.370.0083.

[36] O. S. Groettvik, J. Alme, R. Barthel, T. Bodova, V. Borshchov, A. van den Brink, V. Eikeland, A. Herland, N. Van Der Kolk, S. Voigt Nesbø, T. Peitzmann, D. Röhrich, G. Tambave, I. Tymchuk, K. Ullaland, and S. Yang, "Development of Readout Electronics for a Digital Tracking Calorimeter," *PoS*, vol. TWEPP2019, p. 090, 2020. DOI: 10.22323/1.370.0090.

[37] J. Alme, G. G. Barnafoldi, R. Barthel, V. Borshchov, T. Bodova, A. van den Brink, S. Brons, M. Chaar, V. Eikeland, G. Feofilov, G. Genov, S. Grimstad, O. Grottvik, H. Helstrup, A. Herland, A. E. Hilde, S. Igolkin, R. Keidel, C. Kobdaj, N. van der Kolk, O. Listratenko, Q. W. Malik, S. Mehendale, I. Meric, S. V. Nesbo, O. H. Odland, G. Papp, T. Peitzmann, H. E. Seime Pettersen, P. Piersimoni, M. Protsenko, A. U. Rehman, M. Richter, D. Rohrich, A. T. Samnoy, J. Seco, L. Setterdahl, H. Shafiee, O. J. Skjolddal, E. Solheim, A. Songmoolnak, A. Sudar, J. R. Solie, G. Tambave, I. Tymchuk, K. Ullaland, H. A. Underdal, M. Varga-Kofarago, L. Volz, B. Wagner, F. M. Widerøe, R. Xiao, S. Yang, and H. Yokoyama, "A High-Granularity Digital Tracking Calorimeter Optimized for Proton CT," *Frontiers in Physics*, vol. 8, p. 460, 2020, ISSN: 2296-424X. DOI: 10.3389/fphy.2020.568243. [Online]. Available: https://www.frontiersin.org/article/10.3389/fphy.2020.568243.

[38] Xilinx, *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics*, https://www.xilinx.com/support/documentation/data_sheets/ds922-kintex-ultrascale-plus.pdf, [Online; accessed 12-June-2020], 2019.

[39] "Technical Design Report for the Muon Forward Tracker," Tech. Rep. CERN-LHCC-2015-001. ALICE-TDR-018, 2015. [Online]. Available: http://cds.cern.ch/record/1981898.

[40] A. Szczepankiewicz, "*Readout of the upgraded ALICE-ITS*," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 824, pp. 465–469, 2016, ISSN: 01689002. DOI: 10.1016/j.nima.2015.10.056. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0168900215012681.

[41] M. J. Rossewij, P. Giubilato, and J. Schambach, *Readout Unit production (RUv2)*, Internal communication: Presentation at ITS Upgrade - Readout Electronics Production Readiness Review, 2018.

[42] B. G. Taylor, "Timing distribution at the LHC," 2002. DOI: 10.5170/CERN-2002-003.63. [Online]. Available: https://cds.cern.ch/record/592719.

[43] R. Bailey and P. Collier, *Standard Filling Schemes for Various LHC Operation Modes (Revised)*, https://cds.cern.ch/record/691782/files/project-note-323.pdf.

[44] P. Moreira and J. Christiansen and K. Wyllie, *GBTx Manual*, *V0.18 DRAFT*, 2021.

[45] S. Yuan, J. Alme, D. Röhrich, M. Richter, M. R. Ersdal, P. Giubilato, G. A. Rinella, A. Velure, M. Lupi, and J. J. Schambach, *Remote Configuration of the ProASIC3 on the ALICE Inner Tracking System Readout Unit*, 2020. arXiv: 2011.03815 [physics.ins-det].

[46] P Moreira, R Ballabriga, S Baron, S Bonacini, O Cobanoglu, F Faccio, T Fedorov, R Francisco, P Gui, P Hartin, K Kloukinas, X Llopart, A Marchioro, C Paillard, N Pinilla, K Wyllie, and B Yu, "The GBT Project," 2009. DOI: 10.5170/CERN-2009-006.342. [Online]. Available: https://cds.cern.ch/record/1235836.

[47] Collu, Alberto, *ALICE ITS Production Power System - Operation Manual V1.5*, ALICE Internal Communication, 2020.

[48] L. Greiner, *Power Distribution*, Internal communication: Presentation at ITS Upgrade - Stave Production Readiness Review, 2017.

[49] The ALICE Collaboration, "Technical Design Report for the Upgrade of the Online-Offline Computing System," Tech. Rep. CERN-LHCC-2015-006. ALICE-TDR-019, 2015. [Online]. Available: https://cds.cern.ch/record/2011297.

[50] Schambach, Joachim, *ITS Upgrade Data Format v6.07*, Internal communication, 2019.

[51] G. A. Rinella and J. Schambach and M. Lupi and A. Velure, *ITS Readout FW Specifications*, ALICE Internal Communication, 2021.

[52] Xilinx, *UG974 UltraScale Architecture Libraries Guide*, [Online; accessed 13-December-2019], 2016. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_3/ug974-vivado-ultrascale-libraries.pdf.

[53] P. Chochula, A. Augustinus, P. Bond, A. Kurepin, M. Lechman, J. Lang, and O. Pinazza, "Challenges of the ALICE Detector Control System for the LHC RUN3," TUMPL09. 5 p, 2018. DOI: 10.18429/JACoW-ICALEPCS2017-TUMPL09. [Online]. Available: https://cds.cern.ch/record/2306220.

[54] A. Kurepin, A. Augustinus, P. Bond, P. Chochula, J. Lang, M. Lechman, O. Pinazza, and K. Salas, "ALICE DCS preparation for run 3," in *Selected Papers of the 8th International Conference "Distributed Computing and Grid-technologies in Science and Education"*, ser. CEUR Workshop Proceedings,

Germany: Rheinisch-Westfaelische Technische Hochschule Aachen, 2018, pp. 65–69. [Online]. Available: http://www.jinr.ru/posts/8th-international-conference-distributed-computing-and-grid-technologies-in-science-and-education/.

[55] J Schambach, L Bridges, W Burton, G Eppley, K Kajimoto, and T Nussbaum, "CANbus protocol and applications for STAR TOF control," *Journal of Physics: Conference Series*, vol. 331, no. 2, p. 022 038, 2011. DOI: 10.1088/1742-6596/331/2/022038. [Online]. Available: https://doi.org/10.1088%2F1742-6596%2F331%2F2%2F022038.

[56] I. Mohor, *CAN Protocol Controller*, https://opencores.org/projects/can, [Online; accessed 9-September-2019], 2003.

[57] M. D. Berg, H. S. Kim, M. Friendlich, C. E. Perez, C. M. Seidlick, and K. A. Label, "Incorporating Probability Models of Complex Test Structures to Perform Technology Independent FPGA Single Event Upset Analysis," 2011.

[58] M. D. Berg, K. A. LaBel, and J. Pellish, "Single event effects in FPGA devices 2014-2015," in *NASA Electronic Parts and Packaging Program, Electronics Technology Workshop (NEPP-ETW)*, 2015.

[59] M. Lupi, P. Giubilato, M. Bonora, and K. Sielewicz, "Design of Finite State Machines for SRAM-based FPGAs operated in radiation field," *PoS*, vol. TWEPP-19, 2019. [Online]. Available: https://pos.sissa.it/370/129/.

[60] R. B. GmbH, *CAN specification 2.0*, 1991.

[61] S. V. Nesbo, *Canola - a CAN controller for FPGAs written in VHDL*, https://github.com/svnesbo/canola, 2019.

[62] Wikipedia contributors, *CAN bus — Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=CAN_bus&oldid=933159513, [Online; accessed 31-December-2019], 2019.

[63] Wikimedia Commons, *File:CAN-Bus-frame in base format without stuffbits.svg — Wikimedia Commons, the free media repository*, [Online; accessed 3-August-2020], 2017. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:CAN-Bus-frame_in_base_format_without_stuffbits.svg&oldid=232916576.

[64] Alme, Johan, *ITS RU Aux FPGA Manual*, Internal communication, 2019.

[65] M. Litochevski, *UART to Bus*, https://opencores.org/projects/uart2bus, [Online; accessed 9-March-2020], 2010.

[66] Xilinx, *UG570 UltraScale Architecture Configuration*, [Online; accessed 27-September-2020], 2020. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug570-ultrascale-configuration.pdf.

[67] Samsung, *2G x 8 Bit / 4G x 8 Bit / 8G x 8 Bit NAND Flash Memory*, (K9XXG08XXM Flash Memory Datasheet), 2007.

[68] M. R. Ersdal, "External scrubber implementation for the ALICE ITS Readout Unit," *PoS*, vol. TWEPP2019, p. 136, 2020. DOI: 10.22323/1.370.0136.

[69] Wikipedia contributors, *Hamming code — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Hamming_code&oldid=1026877298, [Online; accessed 19-July-2021], 2021.

[70] Micron Technology, Inc., *TN-29-08: Hamming Codes for NAND Flash Memory Devices*, [Online; accessed 18-October-2020], 2005. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/technical-note/nand-flash/tn2908_nand_hamming_ecc_code.pdf.

[71] ——, *TN-29-63: Error Correction Code (ECC) in SLC NAND*, [Online; accessed 18-October-2020], 2011. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/technical-note/nand-flash/tn2963_ecc_in_slc_nand.pdf.

[72] A. McGuffey, M. D. Berg, and J. A. Pellish, "Localized Triple Modular Redundancy vs. Distributed Triple Modular Redundancy on a ProASIC3E Reprogrammable FPGA," 2010.

[73] Microsemi, *Application Note AC139: Using Synplify to Design in Microsemi Radiation-Hardened FPGAs*, [Online; accessed 28-October-2020], 2012. [Online]. Available: https://www.microsemi.com/document-portal/doc_download/130053-ac139-using-synplify-to-design-in-microsemi-radiation-hardened-fpgas-app-note.

[74] A. Velure, "Design, Verification and Testing of a Digital Signal Processor for Particle Detectors," Presented 14 Sep 2019, 2019. [Online]. Available: http://cds.cern.ch/record/2688945.

[75] Digilent, *Zybo [Digilent Documentation]*, [Online; accessed 27-November-2020], 2020. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/zybo/start.

[76] ——, *Pmod CAN [Digilent Documentation]*, [Online; accessed 27-November-2020], 2020. [Online]. Available: https://reference.digilentinc.com/reference/pmod/pmodcan/start.

[77] PEAK System, *PCAN-USB: PEAK-System*, [Online; accessed 27-November-2020], 2020. [Online]. Available: https://www.peak-system.com/PCAN-USB.199.0.html?&L=1.

[78] O. Grottvik, *Bust bus tool*, https://github.com/olagrottvik/bust, 2020.

[79] AnaGate, *AnaGate CAN Quattro*, [Online; accessed 28-November-2020], 2020. [Online]. Available: http://www.anagate.de/en/products/AnaGateCANquattro.htm.

[80] K. M. Sielewicz, G. A. Rinella, M. Bonora, P. Giubilato, M. Lupi, M. J. Rossewij, J. Schambach, and T. Vanat, "Experimental Methods and Results for the Evaluation of Triple Modular Redundancy SEU Mitigation Techniques with the Xilinx Kintex-7 FPGA," in *2017 IEEE Radiation Effects Data Workshop (REDW)*, 2017, pp. 1–7. DOI: 10.1109/NSREC.2017.8115451.

[81] M. Ersland, "Radiation mitigation design and test for the ALICE ITS Readout Unit," 2018.

[82] CERN, *About CHARM*, [Online; accessed 10-December-2020], 2017. [Online]. Available: https://charm.web.cern.ch/about-charm.

[83] G. Mikkelsen, "Integration and design for the ALICE ITS readout chain," 2018. [Online]. Available: https://bora.uib.no/bora-xmlui/handle/1956/18459.

[84] M. Lupi, *FPGA Design and Scrubbing Testing*, Internal communication: Presentation at ITS Upgrade - Readout Electronics Production Readiness Review, 2018.

[85] ——, *XCKU060 Radiation Testing*, Internal communication: Presentation at ITS Upgrade - Readout Electronics Production Readiness Review, 2018.

[86] D. Colella, "ALICE ITS Upgrade for LHC Run 3: Commissioning in the Laboratory," Tech. Rep. arXiv:2012.01564, 2020, 8 pages, 5 figures, proceedings at The 29th International Workshop on Vertex Detectors (VERTEX) held virtually on October 5th-8th 2020. [Online]. Available: http://cds.cern.ch/record/2746554.

[87] ——, *ALICE Inner Tracking System Upgrade: construction and commissioning*, 2019. arXiv: 1912.12188 (physics.ins-det).

[88] A. Velure, "Integration, Commissioning and First Experience of ALICE ITS Control and Readout Electronics," *PoS*, vol. TWEPP2019, p. 113, 2020. DOI: 10.22323/1.370.0113.

[89] K. T. Mcdonald, "Deadtime When Using a FIFO Buffer," 1996, Princeton/BaBar/TNDC-96-44. [Online]. Available: https://www.hep.princeton.edu//~mcdonald/tndc/fifo.pdf.

[90] Snyder, Wilson, *Intro - Verilator - Veripool*, [Online; accessed 3-January-2021], 2021. [Online]. Available: https://www.veripool.org/wiki/verilator.

[91] M. Hostettler, K. Fuchsberger, G. Papotti, Y. Papaphilippou, and T. Pieloni, "Luminosity scans for beam diagnostics," *Physical Review Accelerators and Beams*, vol. 21, no. 10, 2018, ISSN: 2469-9888. DOI: 10.1103/physrevaccelbeams.21.102801. [Online]. Available: http://dx.doi.org/10.1103/PhysRevAccelBeams.21.102801.

[92] The ALICE Collaboration, "*Elliptic Flow of Charged Particles in Pb-Pb Collisions at* $\sqrt{s_{NN}} = 2.76$ *TeV*," *Physical Review Letters*, 2010. DOI: 10.1103/PhysRevLett.105.252302.

[93] J. Adam, D. Adamová, M. Aggarwal, G. Aglieri Rinella, M. Agnello, N. Agrawal, Z. Ahammed, S. Ahmad, S. Ahn, S. Aiola, and et al., "Centrality dependence of the pseudorapidity density distribution for charged particles in Pb–Pb collisions atsNN=5.02 TeV," *Physics Letters B*, vol. 772, 567–577, 2017, ISSN: 0370-2693. DOI: 10.1016/j.physletb.2017.07.017. [Online]. Available: http://dx.doi.org/10.1016/j.physletb.2017.07.017.

[94] S. Kushpil, F. Krizek, and A. Isakov, "Recent Results From Beam Tests of the ALPIDE Pixel Chip for the Upgrade of the ALICE Inner Tracker," *IEEE Transactions on Nuclear Science*, vol. 66, no. 11, pp. 2319–2323, 2019, ISSN: 1558-1578. DOI: 10.1109/TNS.2019.2945234.

[95] Shahoyan, Ruben, *AliRoot GitHub repository - AliGenQEDBg.cxx*, [Online; accessed August 31, 2021], 2018. [Online]. Available: https://github.com/alisw/AliRoot/blob/master/TEPEMGEN/AliGenQEDBg.cxx.

[96] The ALICE Collaboration, *AliRoot GitHub repository - itsuTestBench*, [Online; accessed October 28, 2019], 2019. [Online]. Available: https://github.com/alisw/AliRoot/tree/master/ITSMFT/ITS/itsuTestBench.

[97] S. V. Nesbo, J. Alme, M. Bonora, P. Giubilato, H. Helstrup, S. Hristozkov, G. Aglieri Rinella, D. Röhrich, J. Schambach, R. Shahoyan, and K. Ullaland, "Simulations of busy probabilities in the ALPIDE chip and the upgraded ALICE ITS detector," *PoS*, vol. TWEPP-17, 147. 5 p, 2017. DOI: 10.22323/1.313.0147. [Online]. Available: https://cds.cern.ch/record/2312057.

[98] D. Marras, G. R. Aglieri, and C. Flouzat, *ALPIDE Periphery & Readout*, Internal communication: Presentation at ITS Upgrade - ALPIDE Chip Engineering Design Review, 2015.

[99] The ALICE Collaboration, "Letter of Intent: A Forward Calorimeter (FoCal) in the ALICE experiment," CERN, Geneva, Tech. Rep. CERN-LHCC-2020-009. LHCC-I-036, 2020. [Online]. Available: https://cds.cern.ch/record/2719928.

[100] Wikimedia Commons contributors, *File:Comparison of dose profiles for proton v. x-ray radiotherapy.png*, [Online; accessed 5-February-2021], 2021. [Online]. Available: https://commons.wikimedia.org/wiki/File:Comparison_of_dose_profiles_for_proton_v._x-ray_radiotherapy.png.

[101] H. Pettersen, J. Alme, A. Biegun, A. van den Brink, M. Chaar, D. Fehlker, I. Meric, O. Odland, T. Peitzmann, E. Rocco, and et al., "Proton tracking in a high-granularity Digital Tracking Calorimeter for proton CT purposes," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 860, 51–61, 2017, ISSN: 0168-9002. DOI: 10.1016/j.nima.2017.02.007. [Online]. Available: http://dx.doi.org/10.1016/j.nima.2017.02.007.

[102] H. E. S. Pettersen, "A Digital Tracking Calorimeter for Proton Computed Tomography," 2018. [Online]. Available: https://bora.uib.no/bora-xmlui/handle/1956/17757.

[103] Voigt Nesbo, Simon, Alme, Johan, Bonora, Matthias, Giubilato, Piero, Helstrup, Håvard, Lupi, Matteo, Aglieri Rinella, Gianluca, Röhrich, Dieter, Schambach, Joachim, Shahoyan, Ruben, and Velure, Arild, "System simulations for the ALICE ITS detector upgrade," *EPJ Web Conf.*, vol. 245, p. 02 011, 2020. DOI: 10.1051/epjconf/202024502011. [Online]. Available: https://doi.org/10.1051/epjconf/202024502011.

[104] M. Suljic, "Study of Monolithic Active Pixel Sensors for the Upgrade of the ALICE Inner Tracking System," Presented 02 Feb 2018, 2017. [Online]. Available: http://cds.cern.ch/record/2303618.

[105] S. Afroz, "Noise and Cluster Size Studies of ALPIDE-CMOS Pixel Sensor for pCT," 2018. [Online]. Available: https://bora.uib.no/bora-xmlui/handle/1956/18057.

[106] CERN, *LS2 Report: An upgraded Inner Tracking System joins the ALICE detector*, [Online; accessed 16-June-2021], 2021. [Online]. Available: https://home.cern/news/news/experiments/ls2-report-upgraded-inner-tracking-system-joins-alice-detector.

[107] J. Schambach, J. Alme, M. Bonora, P. Giubilato, R. Hannigan, H. Hillemanns, M. Lupi, S. V. Nesbø, A. Rehman, G. A. Rinella, M. J. Rossewij, K. M. Sielewicz, and A. Velure, "A Radiation-Tolerant Readout System for the ALICE Inner Tracking System Upgrade," in *2018 IEEE Nuclear Science Symposium and Medical Imaging Conference Proceedings (NSS/MIC)*, 2018, pp. 1–6. DOI: 10.1109/NSSMIC.2018.8824419.

[108] G Mazza, G. A. Rinella, F Benotto, Y. C. Morales, T Kugathasan, A Lattuca, M Lupi, and I Ravasenga, "A 1.2 Gb/s Data Transmission Unit in CMOS 0.18 µm technology for the ALICE Inner Tracking System front-end ASIC," *JINST*, vol. 12, no. 02, C02009. 9 p, 2017. DOI: 10.1088/1748-0221/12/02/C02009. [Online]. Available: https://cds.cern.ch/record/2275297.

[109] J. Schambach, *ITS Upgrade Data Format*, ALICE Internal Communication, 2020.

[110] O. Bourrion, D. Evans, J. Imrek, A. Jusko, A. Kluge, M. Krivda, J. Kvapil, R. Lietava, L. A. P. Moreno, O. Villalobos-Baillie, and E. Willsher, *Interface between CTS-CRU and CTS-Detector Front Ends*, ALICE Internal Communication, 2019.

[111] F. Costa, *ALICE RUN 3 Raw Data Header*, ALICE Internal Communication, 2020.

Abbreviations and Index

ACF	Acceptance Filter. 67
ADC	Analog to Digital Converter. 19, 56, 59, 121
ALICE	A Large Ion Collider Experiment. iii–viii, 5, 8–11, 13, 15–17, 22, 23, 25, 32–34, 39, 48, 114, 116, 118, 119, 132, 133, 137, 143, 145, 149, 154, 156, 163, 164, 166, 170, 184, 187, 199
ALPIDE	ALice PIxel DEtector. iii, iv, 12, 13, 15, 18–28, 31, 34–36, 38, 44–49, 55–59, 61, 62, 96–98, 110–113, 116–122, 124–129, 132–135, 137–140, 143–145, 147, 149, 151–154, 156–158, 160–163, 166, 171, 173–177, 179, 181, 182, 184, 192, 195, 196, 204, 217–219, 221–224
ASIC	Application Specific Integrated Circuits. 7, 18, 35, 37, 99
ATLAS	A Toroidal LHC ApparatuS. 8, 16
AXI	Advanced eXtensible Interface. 102, 104, 165
BB	Bias Bus. 24, 26
BC	Bunch Crossing. 177, 180, 185
BCID	Bunch Crossing ID. 47
BFM	Bus Functional Model. 96–100
BG	Baud Generator. 68
BMU	Busy Management Unit. xiv, 178, 181
BNL	Brookhaven National Laboratory. 5
BRAM	Block RAM. 50, 58–61
BSP	Bit Stream Processor. 70–73, 75, 76, 98
BSP	Board Support Package. 93–96
BTL	Bit Timing Logic. 67–73, 98
BU	Busy Unit. 31, 38, 111, 166, 222–224
BUST	Bus Tool. 104
CAN	Controller Area Network. iii, iv, 12, 35, 38, 39, 41–43, 49–55, 66–78, 93–96, 98–100, 102–106, 164, 165, 226, 227
CDC	Clock Domain Crossing. 50, 57
CERN	European Organization for Nuclear Research. iii–vii, 5, 7, 9, 11, 12, 35, 37, 41, 104, 105, 109, 110, 137, 163, 164, 195
CHARM	CERN High energy AcceleRator Mixed field facility. 107, 108

CI Continuous Integration. 41, 97, 98, 101
CMOS Complementary Metal Oxide Semiconductor. 18, 163
CMS Compact Muon Solenoid. 8, 16
CMU Control Management Unit. 176
COTS Commercial off-the-shelf. 37
CPU Central Processing Unit. 42, 102, 213
CRAM Configuration RAM. 107
CRC Cyclic Redundancy Check. 50, 70, 76, 77, 82, 84, 99, 108, 109, 232
CRU Common Readout Unit. 26, 29, 30, 43, 46, 102, 150, 183, 184, 186–189, 224
CT Computed Tomography. 137, 138
CTP Central Trigger Processor. 22, 25–30, 46, 47, 49, 116, 144, 183–185, 224

DAC Digital to Analog Converter. 19, 20, 110, 121
DC Direct Current. 175
DC/DC DC-to-DC. 38
DCal Di-Jet Calorimeter. 10
DCS Detector Control System. 35, 39, 51–56, 61, 104, 105, 110, 164
DCTRL Differential Control bus. 21, 55, 175, 176
DDR Double Data Rate. 181
DIP Dual In-line Package. 43, 52, 91
DMU Data Management Unit. xiv, 181
DTC Digital Tracking Calorimeter. iv, 138–140
DTMR Distributed TMR. 91
DTU Data Transmission Unit. xiv, 121, 181, 182
DTUL Data Transmission Unit Logic. 181, 182

ECC Error Correction Code. 50, 56, 59–61, 63, 77, 79, 85, 87, 89–91, 101, 164, 232
EDR Engineering Design Review. 125
EMCal Electromagnetic Calorimeter. 10
EML Error Management Logic. 76, 77, 98, 99
EOP End Of Packet. 47, 186, 187
ESA European Space Agency. 54

FEC Focused Expression Coverage. 100, 101
FEC Focused Error Correction. 183
FEE Front End Electronics. 7, 31, 43, 44, 46, 52, 183, 187, 188
FF Flip-Flop. 50, 91, 96
FIFO First In First Out. 12, 20, 23, 29, 38, 41, 43–46, 49–51, 53, 56, 61–63, 67, 81, 89, 90, 96, 98, 101, 113, 121, 125, 127, 151, 164, 177–181, 210, 227, 230

FIT Fast Interaction Trigger. 10, 28, 116
FLP First Level Processor. 30, 35, 39, 40, 81, 93, 150, 183, 184, 188, 190
FoCal Forward Calorimeter. iv, 13, 133–136, 143, 153–160, 166, 167, 193, 195, 238–242
FPC Flexible PCB. 23, 24, 26
FPGA Field-Programmable Gate Array. iii, iv, 7, 11, 12, 27, 31, 33–39, 41–45, 47–55, 62–67, 77–85, 87, 89–93, 96–102, 104, 107–113, 129, 138, 163–165, 209, 210, 212–214, 223, 225–232
FROMU Frame and ReadOut Management Unit. xiv, 121, 125, 127, 177, 178, 180
FSM Finite State Machine. 53, 54, 57, 58, 61, 64–66, 69–76, 91, 99, 101, 120, 125, 127–129, 131, 164, 178–180, 227
FWFT First Word Fall Through. 61

GATE Geant4 Application for Tomographic Emission. 139
GBT GigaBit Transceiver. 26, 27, 31, 35–39, 41, 42, 44–51, 81, 82, 90, 93, 96, 113, 152, 154, 164, 183, 185, 186, 188
GBT-SCA GBT Slow Control Adapter. 35, 37, 79, 81, 82, 100, 183
GBTX GigaBit Transceiver ASIC. 31, 35–37, 47, 51
GEM Gas Electron Multiplier. 6, 10
GPIO General Purpose IO. 34, 36, 45
GUI Graphical User Interface. 82, 209, 212

HB HeartBeat. 27, 29, 47, 49, 184, 185
HBF HeartBeat Frame. 184, 185
HDL Hardware Description Language. 42, 54, 63, 91, 94, 96–98, 100, 104, 113, 117
HIC Hybrid Integrated Circuit. 24, 110
HLP High Level Protocol. 51–55, 67, 94–96, 98, 102, 104–106, 226, 227
HVL Høgskulen på Vestlandet. v–ix

I^2C Inter-Integrated Circuit. 35, 37, 39, 79, 81, 89, 92, 100, 101
IB Inner Barrel. 17, 20–22, 24, 25, 34, 44–46, 55, 102, 110, 113, 121, 124, 125, 133–135, 138, 151, 154, 155, 157, 160, 166, 173, 175, 176, 178, 179, 181, 219, 221
IC Integrated Circuit. 79, 86
IFS Inter-Frame Spacing. 72, 76
IL Inner Layer. 17, 38, 39, 109
IO Input/Output. 31, 50, 85, 89, 90, 101, 139, 176
IP Intellectual Property. 54, 55, 82, 209

IP Interaction Point. 8, 15–17, 31, 48, 114, 163
IP2 Interaction Point 2. 9
ITS Inner Tracking System. iii, iv, vi, vii, 9–13, 15–18, 21, 22, 25–27, 29–36, 38, 39, 48, 51, 55, 67, 80, 93, 107, 109–114, 116–120, 130, 132, 133, 138, 143–157, 160, 163, 165, 166, 175, 176, 184–186, 188, 193–195, 197, 199, 200, 203–207, 221, 222, 233–237

JCM JTAG Configuration Manager. 107
JTAG Joint Test Action Group. 33, 35, 37, 84

LED Light Emitting Diode. 35
LEP Large Electron-Positron Collider. 8
LHC Large Hadron Collider. iii, v, 7–10, 16, 35, 36, 48, 50, 51, 63, 107, 114, 119, 163–166, 177, 184, 199
LHCb Large Hadron Collider beauty. 8, 114
LHCC LHC Experiments Committee. iv, 166
LM Level Minus. 27–29, 46, 144, 176
LOI Letter Of Intent. 154
LS Long Shutdown. iii, 9–11, 27, 30, 163, 184
LSB Least Significant Bit. 27, 43, 52, 53, 173, 175, 176
LTMR Local TMR. 54, 91, 92
LTU Local Trigger Unit. 25–28, 30, 47, 183–185, 224
LUT Look-Up Table. 63, 78
LVDS Low-Voltage Differential Signaling. 34, 173

MAPS Monolithic Active Pixel Sensor. 15, 18, 19, 163
MC Monte Carlo. 117–119, 132, 133, 135, 136, 139, 140, 143, 154, 155, 159, 162, 199, 204, 207
MEB Multi Event Buffer. 20, 22, 23, 113, 121, 125, 144, 171, 177–179, 195, 217, 218
MFT Muon Forward Tracker. 10, 34
MGT Multi Gigabit Transceiver. 34
MIP Minimum Ionizing Particle. 154
ML Middle Layer. 17, 24, 26, 39, 47, 120, 147
MSB Most Significant Bit. 25, 27, 44, 52, 177, 180, 186
MTTF Mean Time To Failure. 65, 109
MVTX MAPS-based Vertex Detector. 16
MWPC Multi-Wire Proportional Chamber. 6, 7, 10

NEF Neutron Equivalent Fluence. 31

O^2 Online-Offline. 40, 93
OB Outer Barrel. 17, 20–22, 24–27, 34, 39, 44–47, 55, 110, 113, 121, 124, 125, 133, 134, 151, 154–157, 173, 175, 176, 178, 179, 181, 221
OL Outer Layer. 17, 24, 26, 39, 47, 109, 120, 147

PA3 ProASIC3. 12, 33, 37, 38, 41, 79, 91, 227
PB Power Board. 39
PB Power Bus. 24, 26
PCB Printed Circuit Board. 11, 33, 34, 38, 39, 103
pCT Proton CT. iv, 13, 34, 133, 138–140, 143, 159–162, 166, 193–195, 197, 199
PHOS Photon Spectrometer. 10
PID Particle Identification. 15
PLL Phase Locked Loop. 181
PON Passive Optical Network. 26
pRU pCT Readout Unit. 34, 138, 139, 160, 162
PS Proton Synchrotron. 108
PSA Pulse Shaping Amplifier. 20, 28
PU Power Unit. 30, 35, 39, 41, 107, 108

QCD Quantum chromodynamics. 4, 9, 163
QED Quantum electrodynamics. 4, 17, 118, 119, 143, 197, 199, 200, 202, 204, 206
QGP Quark Gluon Plasma. 4, 5, 9, 163

RAM Random Access Memory. 59, 63
RCU Readout Control Unit. 33, 84
RDH Raw Data Header. 46, 47, 186–190
REC Receive Error Count. 77, 99
RHIC Relativistic Heavy Ion Collider. 5
RPC Remote Procedure Call. 98
RRU Region Readout Unit. xiv, 121, 125, 127, 178–180
RSP Relative Stopping Power. 138
RTL Register Transfer Level. 42, 54, 96, 97, 112, 113
RU Readout Unit. iii, iv, 11–13, 27, 29–39, 42, 43, 46, 49, 51, 52, 55, 57, 67, 79, 80, 84, 86, 87, 91, 93, 96, 100, 101, 104, 105, 107–113, 117, 129, 130, 133, 151, 152, 154, 160, 161, 163, 164, 166, 175, 176, 179, 183, 184, 186, 188, 195, 196, 209, 214, 220–224, 233–237

SAMPA Serialized Analogue-digital Multi Purpose ASIC. 79
SCADA Supervisory Control And Data Acquisition. 39
SDC Synopsys Design Constraint. 97
SDD Silicon Drift Detector. 15
SEE Single Event Effect. 38, 54, 180
SEMIP Soft Error Mitigation IP. 79, 80, 107
SEU Single Event Upset. 31, 33, 52, 62, 79, 80, 87, 91, 107–109, 179
SJW Synchronization Jump Width. 69
SM Standard Model. 2–4

SOBP Spread Out Bragg Peak. 137
SOF Start Of Frame. 68
SOP Start Of Packet. 47, 186, 187
SPD Silicon Pixel Detector. 15
SPI Serial Peripheral Interface. 84, 102
SRAM Static Random Access Memory. 31, 54, 62, 79, 164
SSD Silicon Strip Detector. 15
STAR Solenoidal Tracker at RHIC. 51
SWT Single Word Transaction. 51, 93, 186, 187

TDR Technical Design Report. 16, 116, 119, 205
TEC Transmit Error Count. 77, 99
TID Total Ionizing Dose. 31, 32, 108
TMR Triple Modular Redundancy. iv, 42, 43, 50, 54, 55, 63–65, 70, 77, 78, 91, 92, 99, 100, 104, 107–109, 164, 165
TOF Time Of Flight. 51
ToT Time over Threshold. 28, 121
TPC Time Projection Chamber. 6, 9, 10, 31, 33, 79, 84, 114, 115, 149
TQG Time Quanta Generator. 68, 77
TRU Top Readout Unit. xiv, 121, 125, 127, 131, 177, 180, 181
TTC Timing, Trigger and Control. 26
TTS Trigger and Timing System. 25, 27, 47, 187

UART Universal Asynchronous Receiver Transceiver. 12, 79, 81–83, 89, 107, 209–211
UiB University of Bergen. iv–viii, 11, 13, 34, 79, 102, 133, 137, 138, 166, 167
UML Unified Modeling Language. 120, 123, 132
USB Universal Serial Bus. 42, 49, 50, 93, 103
UVM Universal Verification Methodology. 41, 97, 98
UVVM Universal VHDL Verification Methodology. 97, 98, 100

VHDL VHSIC Hardware Description Language. 65, 66, 68, 71, 77, 92, 97–99, 124
VME Versa Module Eurocard. 35
VTRx Versatile TransReceiver. 35, 37
VTTx Versatile Twin-Transmitter. 35, 37

WB Wishbone. 43, 49–57, 60–64, 67, 79–82, 89–91, 93, 94, 96, 100, 164, 209, 210, 212

XML Extensible Markup Language. 203, 204
XPM Xilinx Parameterized Macros. 50

Graphic design: Communication Division, UiB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230856819 (print)
9788230865521 (PDF)