Design and Implementation of a High-Speed Readout and Control System for a Digital Tracking Calorimeter for proton CT

Ola Slettevoll Grøttvik

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2021

UNIVERSITY OF BERGEN

Design and Implementation of a High-Speed Readout and Control System for a Digital Tracking Calorimeter for proton CT

Ola Slettevoll Grøttvik

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

Date of defense: 18.02.2021

Year: 2021

Title: Design and Implementation of a High-Speed Readout and Control System for a Digital Tracking Calorimeter for proton CT

Name: Ola Slettevoll Grøttvik

Print: Skipnes Kommunikasjon / University of Bergen

Acknowledgments

For the last four and a half years, I have been deeply involved with the Bergen pCT project, of which the last three years have revolved around the work of this dissertation. I will consider this as one of the most demanding but important periods of my life. I am indebted to everyone I have had the chance to meet and get to know these last couple of years.

Thank you, Dieter Röhrich and Kjetil Ullaland, for believing in me and giving me the opportunity to work with the pCT project, and allowing me to take the role I eventually took. Thank you for always sticking up for me when it was needed and for the long and fruitful discussions.

A large thank you goes to Johan Alme for putting me on this path as early as the autumn of 2014 when you indicated that I had what it took to proceed with both an M.Sc. and a Ph.D. Thank you for supervising me on both my B.Eng. and M.Sc. thesis. But mostly, thank you for being a genuinely great guy with a terrific sense of humor that also shares my interests in beer and good music.

I would also like to thank many of the engineers at IFT. First of all, Shiming Yang, thank you for enduring all my stupid questions and some of my most gloomy days in the lab. Thank you also to Attiq Rehman and Bilal Hasan Qureshi.

One of the most exciting but also intimidating aspects of the last couple of years involved the collaboration with international partners. However, I can not be more grateful for the patience and the guidance I've received. First of all, thank you, Ton van den Brink and Rene Barthel, at Utrecht University in the Netherlands. You are honestly both some of the most knowledgeable and friendly people I have met to date. Thank you both for your hospitality when Tea and I visited Utrecht. Thank you also, Marcel Rossewij, Naomi van der Kolk, Jody Wisman, and Thomas Peitzmann. A huge thank you for the fruitful and friendly collaboration, Ihor Tymchuk and Viatcheslav Borshchov, at LTU Ltd in Kharkiv, Ukraine.

Several people involved with the pCT group deserve considerable recognition. Thank you, Viljar Eikeland, for continually being available for working together on frustrating issues and socializing afterward. Thanks to Matthias Richter, Helge Pettersen, Jarle Sølie, Ganesh Tambave, Shruti Mehendale, Håvard Helstrup, Boris Wagner, and Pierluigi Piersimoni.

The last couple of years would have been quite dull without the company of my colleagues at the institute. I'm grateful for all the fun we had together, especially Magnus Ersdal, Simon Voigt Nesbø, Lucas Altenkämper, Shiming Yuan, and Are Haslum.

I had the pleasure of co-supervising several M.Sc.-students. Thank you all for your hard work — especially the star, Tea Bodova. Also, thanks to Karl Emil Sandvik Bohne, Alf Herland, and Håkon Underdal.

This work would've never been completed without the support from my family and all the friends I love. Thanks to my sisters, Ida and Kristin, you guys give the best pep-talks and support. My parents have been helpful with both proofreading and any other help when it was urgently needed. Thanks to Jenny for the nights out, Eva for hygge, and a huge thanks to Signe for getting me to the finish line. Finally, thank you, guys. You know who you are.

Ola,
Bergen, November 2020

Abstract

Particle therapy, a non-invasive technique for treating cancer using protons and light ions, has become more and more common. For example, a particle treatment facility is currently being built, in Bergen, Norway. Proton beams deposit a large fraction of their energy at the end of their paths, i.e., the delivered dose can be focused on the tumor, sparing nearby tissue with a low entry and almost no exit dose. A novel imaging modality using protons promises to overcome some limitations of particle therapy and allowing to fully exploit its potential. Being able to position the so-called Bragg peak accurately inside the tumor is a major advantage of charged particles, but incomplete knowledge about a crucial tissue property, the stopping power, limits its precision. A proton CT scanner provides direct information about the stopping power. It has the potential to reduce range uncertainties significantly, but no proton CT system has yet been shown to be suitable for clinical use. The aim of the Bergen proton CT project is to design and build a proton CT scanner that overcomes most of the critical limitations of the currently existing prototypes and which can be operated in clinical settings.

A proton CT prototype, the Digital Tracking Calorimeter, is being developed as a range telescope consisting of high-granularity pixel sensors. The prototype is a combined position-sensitive detector and residual energy-range detector which will allow a substantial rate of protons, speeding up the imaging process. The detector is single-sided, meaning that it employs information from the beam delivery system to omit tracker layers in front of the phantom. The detector operates by tracking the charged particles traversing through the detector material behind the phantom. The proton CT prototype will be used to determine the feasibility of using proton CT to increase the dose planning accuracy for particle treatment of cancer cells.

The detector is designed as a telescope of 43 layers of sensors, where the two front layers act as the position-sensitive detector providing an accurate vector of each incoming particle. The remaining layers are used to measure the residual energy of each particle by observing in which layer they stop and by using the cluster size in each layer.

The Digital Tracking Calorimeter employs the ALPIDE sensor, a monolithic active pixel sensor, each utilizing a 1.2 Gb/s data link. Each layer of 18×27 cm consists of 108 ALPIDE sensors, roughly corresponding to the width and height of the head of a grown person. The sensors are connected to intermediary transition boards that route the data and control links to dedicated readout electronics and supply the sensors with power. The readout unit is the main component of both the data acquisition and the detector control system. The power control unit controls the power supply and monitors the current usage of the sensors. Both of these devices are mainly implemented in FPGAs.

The main purpose of this work has been to explore and implement possible design solutions for the proton CT electronics, including the front-end, as well as the readout electronics architecture. The resulting architecture is modular, allowing the further scale-up of the system in the future. A major obstacle to the design is the high amount of sensors and the corresponding high-speed data links. Thus, a large emphasis has been on the signal integrity of the front-end electronics and a dynamic phase alignment sampling method of the readout electronics firmware. The readout FPGA employs regular I/O pins for the high-speed data interface, instead of high-speed transceiver pins, which significantly reduces the magnitude of the data acquisition system.

A consistent design approach with detailed and systematic verification of the FPGA firmware modules, along with a continuous integration build system, has resulted in a stable and highly adaptive system. Significant effort has been put into the testing of the various system components. This also includes the design and implementation of a set of production test tools for use during the manufacturing of the detector front-end.

Contents

List of Abbreviations

μC	Microcontroller
10GbE	10Gb Ethernet
ADC	Analog-to-Digital Converter
ALICE	A Large Ion Collider Experiment
ALPIDE	ALICE Pixel Detector
ALWM	ALPIDE Lightweight Model
API	Application Programming Interface
ARP	Address Resolution Protocol
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
AXIS	AXI Stream
BER	Bit Error Rate
BFM	Bus Functional Model
BTBI	Board-to-Board Interface
BTMR	Block triple modular redundancy (TMR)
CAD	Computer-Aided Design
CDC	Clock Domain Crossing
CERN	The European Organization for Nuclear Research
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
CT	Computed Tomography
DAQ	Data Acquisition
DCS	Detector Control System
DDR	Double Data Rate

DMAPS	Fully Depleted Monolithic Active Pixel Sensor
DPA	Dynamic Phase Alignment
DTC	Digital Tracking Calorimeter
DTMR	Distributed TMR
ELT	Enclosed Layout Transistor
FEC	Focused Expression Coverage
FEE	Front-End Electronics
FIFO	First In First Out
FMC	FPGA Mezzanine Card
FPC	Flexible Printed Circuit
FPGA	Field Programmable Gate Array
FSM	Finite-State Machine
GbE	Gigabit Ethernet
GBT	GigaBit Transceiver
HAL	Hardware Abstraction Layer
HDL	Hardware Description Language
HEH	High Energy Hadron
HEP	High Energy Physics
IP	Internet Protocol
IP	Intellectual Property
ITS	Inner Tracking System
LHC	Large Hadron Collider
LSB	Least Significant Bit
LTMR	Local TMR
LUT	Look-up Table
LVDS	Low-Voltage Differential Signaling
MAC	Media Access Control
MAPS	Monolithic Active Pixel Sensor
MGT	Multi-Gigabit Transceiver
MLVDS	Multi-point LVDS
MMCM	Mixed-Mode Clock Manager
PCB	Printed Circuit Board
pCT	Proton CT

PCU	Power Control Unit
pDTP	pCT Data Transfer Protocol
PHY	Physical Layer
PLL	Phase-Locked Loop
PPM	Parts Per Million
PRBS	Pseudo-Random Binary Sequence
pRG	Proton Radiography
pRU	pCT Readout Unit
PSD	Position-Sensitive Detector
PTB	Production Test Box
QSFP	Quad Small Form-factor Pluggable
RAM	Random Access Memory
RERD	Residual Energy-Range Detector
RIU	Register Interface Unit
RMW	Read-Modify-Write
RSP	Relative Stopping Power
RTL	Register Transfer Level
SBI	Simple Bus Interface
SEL	Single Event Latch-up
SerDes	Serializer/Deserializer
SEU	Single Event Upset
SEUPI	Single Event Upset Probability Impact
SFP	Small Form-factor Pluggable
SGMII	Serial Gigabit Media Independent Interface
SMD	Surface-Mount Device
SNR	Signal-to-Noise Ratio
SOBP	Spread-Out Bragg Peak
SoC	System-On-Chip
SpTAB	Single point Tape Automated Bonding
SRAM	Static Random Access Memory
TC	Transition Card
TCP	Tape Carrier Package
TCP	Transmission Control Protocol

TDD	Test-Driven Development
TID	Total Ionizing Dose
TMR	Triple Modular Redundancy
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UI	Unit Interval
ULTM	Ultra-Light Test Modules
UUT	Unit Under Test
UVVM	Universal VHDL Verification Methodology
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VVC	VHDL Verification Component
XGMII	10 Gigabit Media-Independent Interface
ZIF	Zero Insertion Force

CHAPTER 1

Introduction

This chapter begins with a brief introduction to particle therapy for cancer treatment. Imaging for particle treatment dose planning is then concisely discussed. Furthermore, proton CT is introduced as a concept. Subsequently, the Bergen proton CT prototype, the Digital Tracking Calorimeter, is outlined. Finally, the primary objective and the major contributions of this work are concisely presented.

Cancer treatment research is an important and active field. Millions of people worldwide get diagnosed with cancer every year, and the disease is the second leading cause of death, trailing only cardiovascular diseases. Based on the cancer type, the stage of the disease, and the risk for critical organs, various types of therapy are offered to the patient: (1) chemotherapy, (2) surgery, and (3) radiation treatment. The purpose of radiation treatment, also known as radiotherapy, is to control or kill cancerous cells using ionizing radiation. It is important, however, to also avoid damaging the healthy tissue surrounding the cancer cells. The most common type of radiation treatment utilizes X-ray photons, yet, in the last couple of decades, the use of charged particles for radiation treatment has increased.

By the end of 2016, more than 170 000 patients had been treated with particle therapy [1], most of whom with protons. Carbon ions are also used at some facilities like the HIT facility in Heidelberg. In 2018, it was decided that Norway should provide particle therapy as an option to cancer patients in the form of proton therapy and that two treatment centers would be built, one in Oslo and one in Bergen [2].

1.1 Particle Therapy

Particle therapy is distinguished from conventional X-ray therapy in that it utilizes massive particles rather than massless photons [3]. Although both neutral and charged particles can be employed, neutral particles are not often used [1]. Particle therapy promises several advantages compared to X-ray therapy. The advantages stem from the fact that a charged particle interacts with matter in a fundamentally different way than photons. Contrary to photons, protons and other charged particles stop. While traversing through matter, a charged particle will interact with orbital electrons, and in each of these collisions, it will lose a tiny amount of energy and slow down. Eventually, close to the stopping point, the particle will release most of its energy. This is called the Bragg peak.

The relative depth dose distributions of photons versus protons are given in Figure 1.1. A photon beam, as shown by the red line, delivers a large dose to the healthy tissue. A harmful dose bath is shown both in front of and behind the tumor. A proton will deposit only about 30 percent of the Bragg peak maximum dose to the area in front of the Bragg peak [3]. Beyond the Bragg peak, the dose deposited falls to virtually zero. It is clear, however, that a mono-energetic proton beam will deliver most of its dose to only a small portion of the tumor. This is illustrated by the multiple blue lines, each representing a beam with a certain energy. A therapeutic proton beam needs to cover the entire tumor area illustrated by the grey box. Thus the beam distribution is artificially expanded by using protons with varying energies. The dotted blue line shows the expanded beam's depth dose distribution. The combination of protons with different characteristics produces a Spread-Out Bragg Peak (SOBP); the area of the accumulated Bragg peaks of many protons. Bear in mind that some broadening of the Bragg peak also occurs when combining protons in a mono-energetic beam. This broadening is due to the statistical nature of the energy loss process and is called range straggling [4].

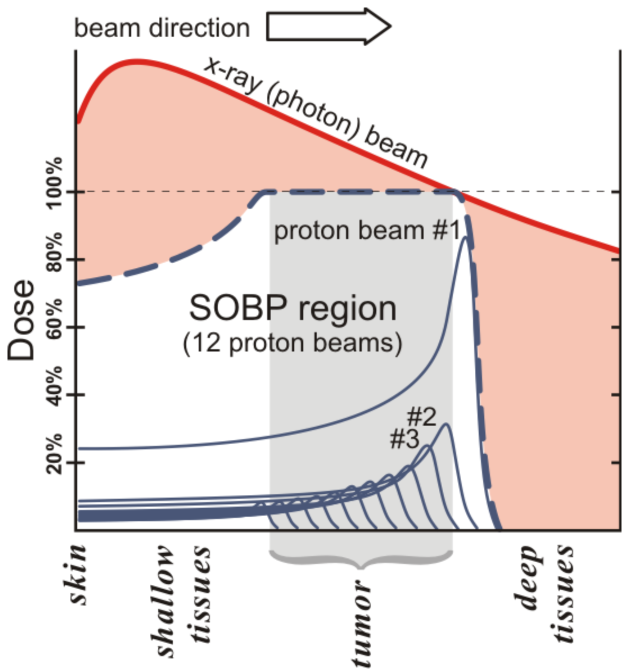


Figure 1.1: *Comparison of relative depth dose distributions of photons versus protons. From Filipak [5]. Licensed under CC BY-SA 3.0.*

Particle therapy treatment has several potential advantages compared to conventional X-ray therapy:

(1) Less total dose to healthy tissue

A photon beam delivers a much larger dose to healthy tissue than a proton beam. Any reduction of dose entails that secondary radiation effects may also be reduced [6]. There is an increasing number of studies indicating that this will translate into clinical advantages [7]. Because of this, particle radiotherapy can be considered for the treatment of pediatric tumors in the hope of reducing complications later in life for child patients.

(2) Finite range and sharp dose cut-off

Since charged particles stop and because healthy tissue behind a tumor will receive next to no dose, it is appealing to consider particle treatment for tumors next to critical structures. There is some clinical evidence that patients with tumors close to critical structures are benefiting the most from proton treatment [8].

(3) Increased tumor control

Because the dose delivered to healthy tissue is dramatically lower, it is possible to increase the dose to the tumor. The sharp dose cut-off following the Bragg peak also contributes. A higher dose to the tumor helps to control the tumor.

In Figure 1.2 we clearly see how some of these advantages are manifested in the dose plans for treatment. The photon treatment has a sizeable low-dose bath behind the tumor, shown in blue, while the proton treatment virtually leaves no dose trailing the tumor.

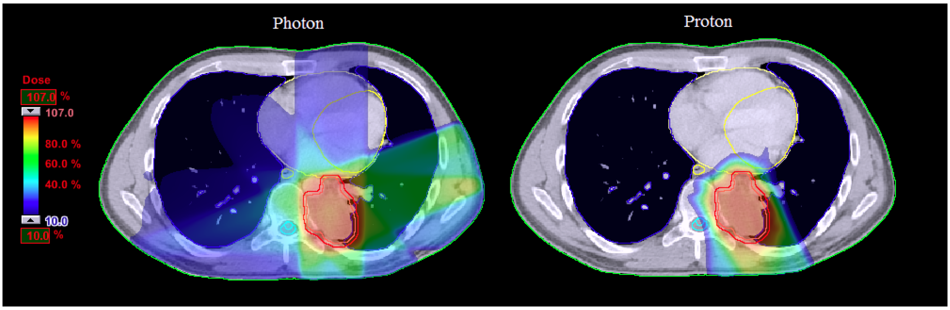


Figure 1.2: *A comparison between two dose plans for irradiation of a paravertebral sarcoma in the lung, overlaid on CT images [4]. Left: (conventional) Intensity Modulated Radiation Therapy with photons. Right: Intensity Modulated Proton Therapy.*

1.2 Proton CT

A single proton's range is uncertain because of the range straggling. Even in simple water phantoms, predicting where a proton will stop is a statistical process. Nevertheless, range straggling is an unavoidable uncertainty and is not a significant problem for particle treatment. However, the tissue in front of the tumor is a mixture of muscles, bones, and fat, and the incomplete knowledge of the chemical composition and density of this and the tumor itself can become a significant source of range uncertainty [7]. When creating dose plans for particle treatment, sharp gradients of the dose distribution are generated around the tumor. Even a small uncertainty about the range of the beam can cause drastic changes to the dose deposited to healthy tissue [7]. This can be detrimental to patients that have a tumor that lies close to critical structures, i.e., one of the patient groups which can potentially benefit the most from particle radiotherapy. Although it is impossible to remove the range uncertainty completely, one can strive to diminish it compared to the values used in the clinic today.

Every type of radiation treatment is preceded by the calculation of a treatment or dose plan that indicates the dosage delivered to various tissue of the treatment volume. Today, one obtains information from conventional X-ray computed tomography (CT) imaging to prepare dose plans for particle treatment. CT imaging provides the so-called Hounsfield units, or the CT number, of the volume in question. This number describes how the intensity of an X-ray beam is attenuated by the volume. Particle treatment, however, uses the relative stopping power (RSP) of the volume to calculate the beam's range. It is possible to convert Hounsfield units to RSP, but this conversion is not a one-to-one mapping [9]. The conversion introduces range uncertainties of up to a total of 2 to 3 percent. Most of this uncertainty comes from the table lookup of the ionization potential [4].

By directly measuring the proton RSP of the tissue, this uncertainty is minimized to the uncertainty of the measurement. One way to do this measurement is by using protons or other charged particles for imaging. This involves transmitting high-energy protons through the volume and measure how the volume in question affected the energy of the particles. Since the 1960s, several types of proton radiography (pRG) and proton computed tomography (pCT) have been proposed. Some of the systems are proton integrating systems that calibrate a signal in a detector with the length traversed, averaging over multiple protons. This approach has some significant negative implications in terms of image quality [10]. Thus, the field of pCT research has mostly started to focus on proton tracking devices.

The proton tracking method involves the measurement of individual protons and the calculation of the residual energy of each particle. The residual energy is the particle's remaining energy after passing through the volume to be measured. This way, one can reconstruct estimates of the particles' trajectories or tracks through the patient. The tracks are estimated based on advanced most-likely path algorithms, that also minimizes the effect of the multiple Coulomb scattering on the spatial resolution [11]. Based on these tracks the proton's average energy loss along the path is calculated. Finally, by transmitting a large number of protons from different angles, the average RSP of each voxel of the volume can be calculated [4]. Proton tracking pCT systems typically consist of multiple position-sensitive detectors (PSD), two in front of the patient, and two behind the patient [10]. These detectors obtain crucial information about the particles' position and direction at both entrance and exit of the measured volume. In addition, a residual energy-range detector (RERD) is used to obtain the particles' residual energy. In Figure 1.3 we see how the different detectors obtain the information about each particle.

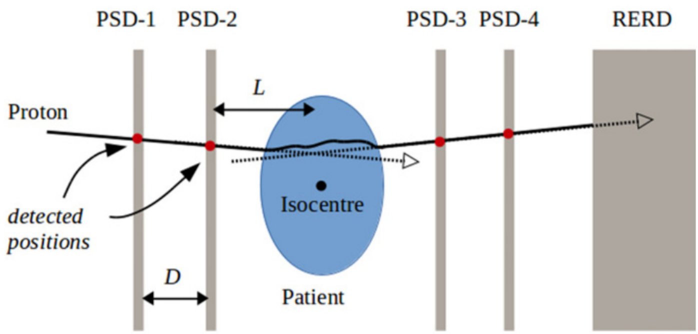


Figure 1.3: *A typical double-sided proton-tracking proton radiography/proton CT system. Consists of four PSDs layers and a RERD. Figure from Poludniowski [10]. Licensed under CC BY 4.0.*

1.3 Digital Tracking Calorimeter

Several of the suggested pCT prototypes make use of a crystal-based RERD. For instance, a collaboration of Loma Linda University (LLU) and others have presented a head-scanner prototype that employs CsI crystals for the RERD [12]. The use of a crystal-based calorimeter significantly reduces the proton rate capabilities of the device, and for the LLU-prototype, this leads to a CT scan time of several hours [10]. To use protons for clinical imaging the scan time must be reduced. Not only because it is unreasonable to keep a patient still for so long, but also because the anatomy of the patient is susceptible to change over time. This change might potentially dramatically increase the proton range uncertainty [13]. To increase the proton rate, and thus reduce the imaging time, the use of a range telescope-system to construct a RERD has been proposed by Esposito et al. (with silicon strip detectors) [14] and by Pettersen et al. (with pixel detectors) [15]. Specifically, Pettersen et al. introduces the concept of a Digital Tracking Calorimeter (DTC) that:

(1) is digital in that it only detects energy depositions over a certain threshold, i.e. either a 0 (no hit) or a 1 (hit).
(2) tracks the path of individual particles through the detector medium.
(3) calculates the energy of each particle based on the particle range.

In the proposed DTC by Pettersen et al., the RERD is constructed by several layers of high-granularity pixel detectors. In fact, it is constructed the same way as the PSD in the same system. Therefore, the entire prototype is simply several layers of pixel detectors. Between each layer of the RERD, metal sheets are installed to gradually absorb energy, and eventually, stop the particles within the detector. These sheets are also used to obtain structural stability for the sensor chips. For the RERD, aluminum is the metal of choice based on the proton stopping and scattering power, durability, secondary neutron production, and more [16]. To minimize the uncertainty of the measurement, physics simulations with different thicknesses of the energy-absorbing aluminum layers between each sensor layers were done. The study concluded that an absorber thickness of 3.5 mm both minimizes the uncertainty of the measurement and restricts the total number of sensor layers required to provide sufficient tracking and range resolution. A total of 41 sensor layers are needed for the RERD-part of the DTC to make sure protons with an energy range of 35 MeV to 245 MeV will stop in the detector, and thus, be appropriately measured.

The PSD layers must be very thin and should contain as little mass as possible to reduce multiple Coulomb scattering while measuring the position and angle of the incoming particle track. Thus, the absorber is replaced with a low-mass stabilizer that the sensors are mounted on. A 0.2 mm thick carbon fiber sheet is chosen based on its low mass and its structural properties.

1.3.1 Single-sided proton CT

Sølie and Voltz et al. investigated the feasibility of proton imaging without front trackers and instead rely on the information given by the beam delivery system [11]. Avoiding the use of front trackers has several benefits, especially during the design and assembly stages. First, it significantly reduces the cost of sensor layers, but one can also increase the particle rate as the pairing of hits on the front and the rear trackers are avoided [17]. Furthermore, in a clinical setting, one avoids the complexities involved in aligning the set of front and rear trackers. Critically, one will not be required to move the front trackers away before starting proton therapy operations, which allows the DTC to be used as an online treatment dose deposition monitor via online in-vivo imaging. Single-sided pCT simulations indicate a somewhat reduced spatial resolution, but Sølie and Voltz suggest the development of dedicated reconstruction algorithms for this type of detector setup [11]. See Figure 1.4 for an illustration of the single-sided pCT prototype.

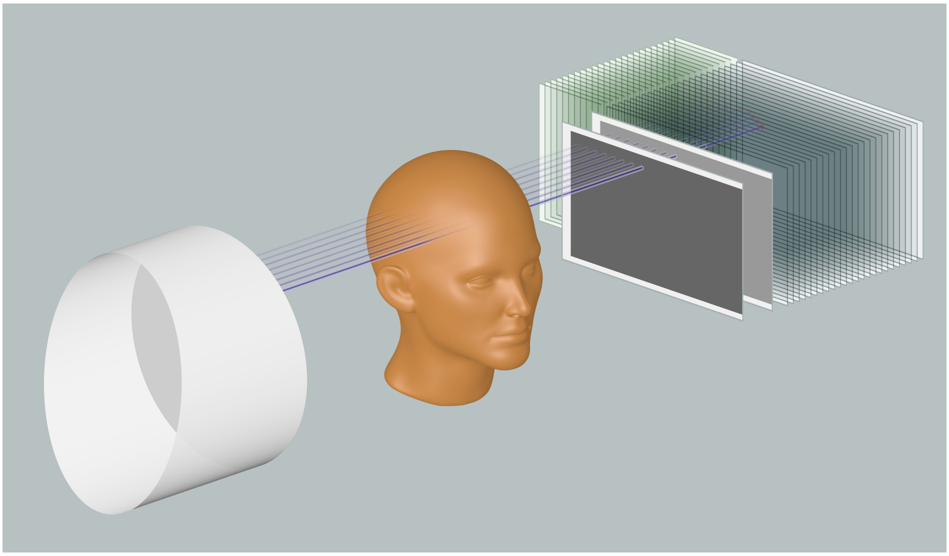


Figure 1.4: *Artistic illustration of the Bergen proton CT prototype. A single-sided scanner does not use tracking layers in front of the measured object.*

1.3.2 Particle Energy Calculations

The residual energy of a particle can be calculated by observing in which layer of the RERD the particle stops. This calculation is derived from the Bethe range-energy relationship. Several analytical models are investigated by the Bergen pCT group [18]. One can further improve the residual energy calculation by employing more information from the pCT detector. Even though the DTC only provides digital pixel hit data, each particle that moves through the detector will deposit energy relative to its velocity/energy. Tambave et al. showed how the mean cluster size changes relative to the energy loss of the particle using the same sensor as the DTC, the ALPIDE, as seen in Figure 1.5 [19]. The ALPIDE is further discussed in Section 2.3. Fitting a Bragg peak curve to the combined information of cluster size and particle range will provide an accurate measure of the residual energy [4, Figure 5.5].

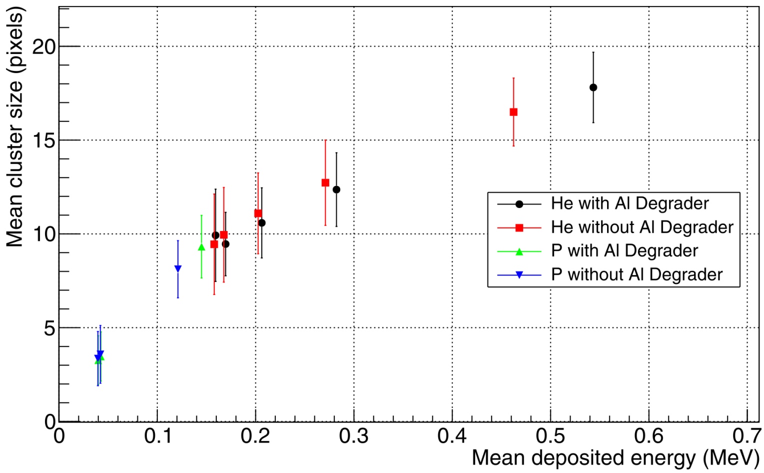


Figure 1.5: *The mean cluster size versus the energy loss of the ^{4}He ions and protons in the ALPIDE chip [19].*

1.4 Primary Objective and Main Contributions

In this introductory chapter, we have introduced some of the motivation for developing a proton CT detector. Note that the development of such a detector is a big collaborative effort, of which this thesis is an integral part. The Bergen pCT group started the initial work in 2012 and received funding for the design and the construction of the DTC prototype in late 2016. The proof-of-principle of using a digital tracking calorimeter for pCT was based on experiments with an electromagnetic sampling calorimeter containing MIMOSA23 sensor chips as sensitive layers [15].

The work to develop the pCT readout electronics, however, was built upon the development of the readout electronics of the ALICE Inner Tracking System (ITS). As is discussed later in this thesis, it became evident that the pCT electronics had different requirements, and therefore, the development efforts were separated. Nevertheless, several components from the ITS development remain in the pCT system.

At the time of writing, the Bergen pCT prototype, although delayed by COVID-19, is getting close to the start of production of the front-end electronics. I have been involved with many parts of the detector development, including design and testing of (1) system design (Chapter 2), (2) front-end electronics (Chapter 3), (3) readout and control electronics and field programmable gate array (FPGA) firmware (Chapter 4, 5, 6 and 7), and various other tasks. The prime objective of this thesis is to present a near-complete technical electronics solution to the Bergen pCT prototype. Although I have been involved with many parts of the detector development, most of my focus has been on the data acquisition and detector control electronics FPGA firmware, of which I have been the sole developer. The thesis reflects this since most of Chapter 4, 5, 6 and 7 are concerned with the FPGA firmware development. Parts of this work have been presented at an international conference and have been subject to peer review for publication in journals (see Appendix A).

CHAPTER 2

System Design and Electronics Components

In this chapter, the requirements of the proton CT DTC system are discussed in detail before an overview of the system electronics design is provided. The individual parts of the system are then presented in more detail. As the work of this thesis focuses on the electronics of the system, the chapter emphasizes the various parts of the front-end electronics and the readout system. Finally, some other prototypes related to the development of the project are mentioned.

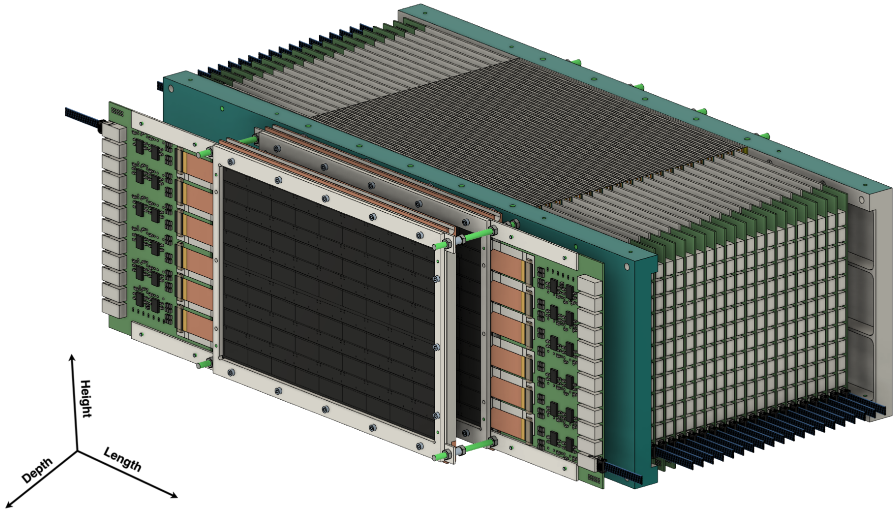


Figure 2.1: *3D design of the pCT Digital Tracking Calorimeter. The figure is provided by courtesy of Anthony van den Brink. The two tracking layers are shown in front, with the calorimeter layers behind. The detector is shown without the cooling plate and fans.*

2.1 Detector Requirements and Considerations

The goal of the pCT prototype is to prove that a DTC can be used for CT imaging with protons and overcome most of the critical limitations of the already existing prototypes. This will improve the dose planning accuracy for particle radiation therapy. Although both the conceptual and the technical requirements have been subject to revision during the research project period, a few major requirements are regarded as the most important. Note that the following requirements are focused mainly on the most basic physics requirements for particle tracking, and subsequently, the electronics requirements needed to fulfill the former. The following list of requirements is in addition to the specifications of the number of layers and absorber thickness already described in Section 1.3.

(1) pCT is most likely to be beneficial for tumors close to critical structures, because particle treatment of such tumors requires high accuracy. Consequently, the detector must be large enough to construct an image of a human head. The width and height of the detector are presented in Section 2.4. To verify that the DTC can be used to improve dose planning for radiation treatment, the detector will be tested using so-called phantoms. Phantoms are representations of an actual physical object and are often made to resemble a part of the human body. Phantoms are used both in the testing of new types of detectors and to calibrate existing detectors' real-world data to the data from advanced simulations.

(2) The DTC must be able to handle clinical beam settings so that it can be tested and used in a clinical facility like HIT in Heidelberg, Germany. This requirement entails that the DTC must withstand a particle flux within the nominal values of clinical proton beams. Also, the pCT will operate with a scanning pencil beam. The detector must be able to handle a rate of 10^7 particles/s. This number is in the upper range of protons required to construct a 2D-projection of a human head and is also in the lower range of the typical flux of the clinical proton beam machines like the one at HIT. To opt for a flux that is rarely used in these facilities might have consequences for the information that is possible to acquire from the beam optics of the facilities.

As mentioned in the introductory chapter, most pCT implementations are using a crystal-based RERD, limiting the rate of particles that can be concurrently tracked. By replacing this approach with silicon-pixel detectors, one of the main goals of the Bergen pCT prototype is to drastically increase the rate of particles. The increased particle rate will speed up the imaging process to the point where each 2D-projection is completed in a matter of seconds, and a complete 3D-tomography is completed in a matter of minutes [17]. A full 3D-tomography is necessary to generate a full RSP voxel map of the measured volume.

(3) The calculation of a particle's residual energy is mainly based on its *track* through the detector and where it stops. The track can be used together with knowledge of the detector design to accurately estimate how much material the particle has traversed. Monte Carlo simulations provide information about how particles are expected to behave in a precise model of the detector, and what the residual energy of particles should be if they stop in a particular layer. However, it is complicated to construct a reasonable model and simulation of the detector if the detector is built in an inhomogeneous way. This is the reason why one must strive to build the detector as homogeneous as possible and avoid intricate structures.

(4) A DTC needs to be able to track individual particles, but due to the high rate, it can be challenging to differentiate between particles. For this reason, the detector needs to have a high spatial resolution. This means that the size of each sensor-pixel needs to be small. Furthermore, there should be hardly any dead area between the pixels.

(5) It is an advantage for clinical use if the detector can be designed without the use of gas or high voltage. The complete detector should also be compact and have a simple air/water cooling system. Details about the mechanics and cooling can be found in [17].

(6) To be able to construct the track of each particle through the detector, the pixel address information from all the layers combined with the time of each hit must be collected. It is crucial that all sensors in all layers adhere to the same time-domain so that the particles hit in two different layers are marked with the same timestamp and/or frame ID. Hits in two layers with different

parameters will not be combined to form a track as this indicates that they do not emanate from the same particle. To ensure that all sensors in all layers adhere to the same time-domain, a deterministic and accurate synchronization system is needed.

(7) The pCT detector must continuously sample data with minimal dead-time or integration time. The detector data will be captured by a nominal trigger rate of 10 µs, and a sampling window of 9.75 µs, meaning a sampling window gap of 250 ns. However, these settings can be configured to optimize for certain conditions, for instance, to reduce the number of double hits (see Section 2.3.1). The detector electronics must handle the resulting data with minimal data loss. The data rate calculations are done with the worst-case trigger rate of 5 µs (see Section 5.4).

(8) In addition, the system should have a trigger-less readout architecture. This means that the data acquisition should be made possible without any external nor any high-level trigger system implemented. Note that this does not exclude the use of sensors that require low-level trigger signals as these can originate and be controlled from the data acquisition (DAQ) system itself. This simplifies both the design and the operation of the detector system.

(9) Although the primary objective of the pCT scanner is to investigate imaging with protons, the prototype can in principle be used to investigate CT with any charged ions, e.g., carbon or helium.

2.2 Detector Overview

The DTC prototype consists of the parts listed below.

Front-end High-granularity pixel sensors with high-speed data readout bonded to custom glue electronics providing an interface to the external detector electronics.

Transition Card A simple printed circuit board (PCB) that supplies power to the sensors, as well as providing the electrical connections between the sensors and the DAQ system.

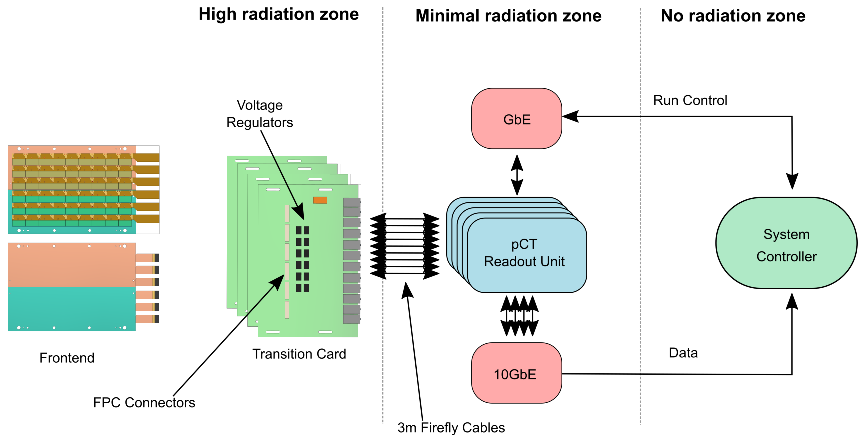


Figure 2.2: *The Proton CT readout and control architecture. Notice the different radiation zones of the different aspects of the system. The PCU is not depicted in the figure, but is placed within the same radiation zone as the readout unit.*

Readout Unit The pCT Readout Unit (pRU) is the interface to the sensor chips and acts as the communication hub between the sensors and the control room. The pRU is the main component of both the DAQ and the control system.

Power Control Unit The Power Control Unit (PCU) and the power network provides power to the front-end electronics and is also part of the control system.

Server Farm One or more computers are serving to handle the data stream (DAQ) from the sensors, as well as controlling the operation of the detector (DCS).

Network Infrastructure for connecting the readout units and the power control units to the control server farm.

The pCT readout and control architecture is shown in Figure 2.2. The front-end components are connected to the Transition Card (TC), which is further connected to the pRU. Note how these parts are located in zones with different radiation levels. In Section 4.6 the radiation environment of the pCT system is discussed. The pRUs are connected to the system controller via two separate Ethernet-based networks; one for run control and one for data readout.

The control server farm and the network components are not discussed further in this thesis. The PCU is only briefly mentioned in Section 2.8. The other parts of the system are discussed in detail below.

2.3 Monolithic Active Pixel Sensor

In general, monolithic active pixel sensors (MAPS) have a pixel matrix where each pixel has a dedicated signal amplifier and an ADC or a simple threshold discriminator. Often, the detector chip also incorporates a digital transmission unit combined with the pixel matrix integrated on the same piece of silicon. One can differentiate between high-voltage CMOS technology and MAPS [20]. MAPS generally have a very low power consumption and can be made radiation tolerant [21]. Low power consumption and radiation tolerance are often two of the most crucial requirements for sensors used in high energy physics (HEP) experiments and are also required for the pCT detector.

Several MAPS have been developed specifically to be used in HEP and similar applications. For instance, the Ultimate-2 sensor for the STAR experiment [22]. Generally, for MAPS, charge collection is dominated by diffusion, but by drift near the electrode [23]. However, for high-voltage monolithic CMOS sensors and fully depleted MAPS (DMAPS), for instance, the MuPix7 developed for the Mu3e Experiment [24], drift is the driving force of the charge collection. The drift component of the charge collection is an important variable of the signal-to-noise ratio of the sensor [20]. A DMAPS is considered used in the high-luminosity Large Hadron Collider (LHC) upgrade of the ATLAS detector in 2025 [25, 26]. Most of the DMAPS alternatives that would be usable for the pCT detector are still under prototyping or testing and were not available for consideration during the early stages of pCT development.

The selection of a MAPS for the pCT detector was made based on three main considerations: (1) data readout speed, (2) pixel size, and (3) availability. The University of Bergen is involved with the ALICE experiment at the LHC at The European Organization for Nuclear Research (CERN), and thus had experience with the ALPIDE, which was developed for the ALICE ITS detector. The ALPIDE was the first MAPS to implement a zero-suppression architecture to increase readout speed and reduce readout power usage [25]. Below, we provide the properties and characteristics that demonstrate that the ALPIDE is a valid sensor choice for the pCT detector.

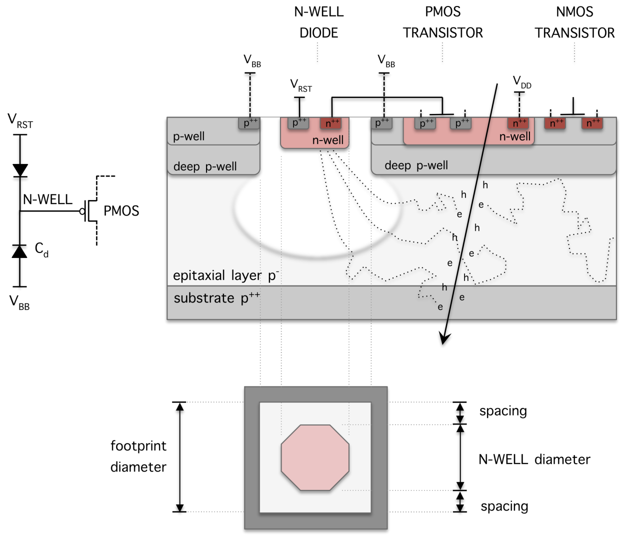


Figure 2.3: *Equivalent circuit schematic, cross-section and footprint of the ALPIDE pixel [27]. Notice how electron-hole pairs are produced in the epitaxial layer by the ionizing particle.*

2.3.1 The ALICE Pixel Detector

The ALPIDE consists of 512 × 1024 pixels, and the size of each pixel is only $29.24\,\mu m \times 26.88\,\mu m$ [28]. The chip measures $15\,mm(H) \times 30\,mm(W)$ of which 1.2×30 mm^2 is a digital periphery region devoted to control and readout functions. The pixel itself is constructed as a charge collection diode, indicated as C_d in equivalent circuit schematic in Figure 2.3. A particle hitting the chip will cause electron/hole-pairs as the particle ionizes the material along its path. These electrons diffuse in the epitaxial layer and can be collected by the diode when the force of the depletion region causes the electrons to drift towards the n-well. When electrons are collected the sensing diode experiences a change of potential of a few tens of mV relative to the number of electrons collected [29]. The electrons can also diffuse to nearby pixel diodes, causing several pixels to fire, causing so-called clusters. The size of the depletion region of the diode versus the drift volume can be controlled by supplying a negative bias voltage to the chip. An increased depletion region increases the granularity of the sensor by restricting the number of pixels affected by a particle but will also cause a reduction of the cluster sizes, which in turn yields a lower energy resolution. In any case, the ALPIDE is considered to be a high-granularity pixel sensor, which was mentioned as a requirement in Section 2.1.

The ALPIDEs can be produced with different substrate thicknesses. Specifically 100 µm and 50 µm. Employing a thin chip will reduce the multiple Coulomb scattering of the particles when they pass through the sensor material. Also, less energy is lost in the sensor itself. A very thin chip can be more difficult to handle during the bonding and mounting process, therefore the RERD layers, also called the calorimeter layers, of the detector will employ the 100 µm version. As discussed in Section 1.3, the position-sensitive detector (PSD) layers, also called the tracking layers, need to be very thin, thus the 50 µm version will be utilized in these layers.

MAPS can be constructed to be radiation tolerant. Previously, the ASICs used in the LHC often employed enclosed layout transistors (ELT) and guard rings to achieve the required radiation tolerance [30]. ELTs help keeping the leakage current at a low level after irradiation [31]. On the other hand, ELTs introduce several behavioral differences compared to standard linear MOS transistor models. Most critically limiting the W/L ratio to large values, thus severely constraining the designs [32]. With smaller CMOS technology nodes, one has observed a reduced total ionizing dose (TID) sensitivity, in turn making the ELT approach redundant and complicated [30]. The ALPIDE achieves sufficient TID tolerance by using the TowerJazz 0.18 µm CMOS technology with a gate oxide thickness of roughly 3 nm [33, 34]. To protect the sensitive digital control circuits of the sensor from radiation-induced failures (see Section 4.6), mitigation techniques like triple modular redundancy (TMR) are employed [29, 28, 35]. Also, a deep p-well, as seen in Figure 2.3, is shielding the digital logic and preventing the n-well from collecting signal charge [20, 36].

The charge collection diode is connected to an analog amplifier circuit. The amplifier uses a slow shaping time, between 1-2 µs, to save power [36]. In Figure 2.4 the pixel timing diagram is shown. A global threshold level is used to discriminate the signal, reducing the number of fake hits, i.e., signal changes caused by noise. The discriminator outputs a digital signal indicating a hit. Thus, the sensor only indicates whether there is a signal or not. The digital nature of the pixel electronics means that no information regarding the strength of the signal is stored, causing loss of information about the particle type and energy. However, as indicated in Section 1.3.2, some information about the deposited energy can be extracted from the cluster size. Also, because of the digital nature of the pixel, meaning that a signal below the global threshold level is ignored, we say that the data is naturally zero-suppressed.

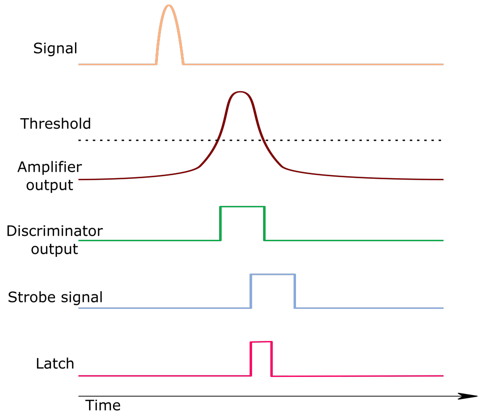


Figure 2.4: *ALPIDE Pixel Timing Diagram [37]. The latch signal is high when the discriminator output and the global strobe signal is asserted at the same time.*

Each pixel has a 3-bit memory called the multi-event buffer. For a hit to be stored the discriminator output must coincide with the global strobe window signal. The strobe signal usually originates from an external trigger signal, and the strobe window length can be configured globally. The use of a global strobe signal also implies that the ALPIDE has a global-shutter readout architecture. This contrasts with the rolling-shutter architecture of the Ultimate-2 sensor [22]. The slow shaping time of the analog pixel circuit means that the signal might be captured by several strobe windows if the gap between them is short.

2.3.2 Data Interface

The ALPIDE can be set in different modes depending on expected data rates and operation. The mode names are based on where in the ALICE ITS detector the specific mode is used. The inner barrel mode is used in the three innermost layers of the ITS, where the expected data rates are the highest. Outer barrel mode, however, is used in the middle and outer layers of the detector, where the data is low enough to combine the data outputs of multiple chips without buffer overflows and data loss. Because of the expected data rates of the pCT, see Section 5.4, all ALPIDE chips of the DTC are configured in inner barrel mode.

The use of the inner barrel mode entails that the chip utilizes a high-speed low-voltage differential signaling (LVDS) link with a maximum speed of 1.2 Gb/s to offload the pixel hit data. The data is 8B/10B-encoded by default to minimize data errors. This encoding scheme is standard in high-speed electronics and ensures link DC-balance, allows for checking for single-bit errors in an 8-bit word, and provides special control words for word alignment [38]. Thus, the maximum data throughput of the chip is 960 Mb/s. After configuration, the data link is always on, transmitting comma words when there is no data. The ALPIDE employs the K28.5 comma control word. As no clock is transmitted alongside the data, the comma word is used by the FPGA firmware for clock/data recovery and to align the first bit of the word.

Experience while handling the chip in different implementations has shown that the ALPIDE data link is susceptible to jitter, and careful considerations must be made at multiple levels of the electronics implementation. The on-chip phase-locked loop (PLL) must be very stable to ensure the optimal data link performance. This is further discussed in Section 2.3.5, 2.5, and 3.2.4.3.

2.3.3 Slow Control

A slow control link is used for non-data communication between the ALPIDE and the readout electronics. A multi-point LVDS (MLVDS) interface is used for this purpose so that multiple ALPIDEs may share the same slow control link. This interface enables the external reading and writing of the internal on-chip status and configuration registers. In addition, the interface acts as a command distribution channel for transmitting triggers to initiate the global strobe signal and synchronizing the clock counter, and more. The ALPIDE has multiple registers that control the operation of the chip, as well as report the current status. Communication via the slow control link is done with a set of opcodes. As multiple chips share the same link, one can communicate with multiple chips at the same time. By default, the output from the ALPIDE is Manchester encoded to ensure DC-balance.

2.3.4 Other interfaces

To separate the ALPIDEs sharing the same slow control link, a 7-bit chip ID field is used. Note that the ALPIDE has a bug that causes ID 7 to be interpreted as a broadcast ID. Therefore, this ID is skipped in the DTC implementation. Chips configured in inner barrel mode only uses CHIPID[3:0], allowing the remaining pins to be connected to ground.

The ALPIDE is designed to use the LHC system master clock of roughly 40 MHz[1]. The clock input expects an MLVDS signal so that multiple chips can share the same clock.

One of the chip interfaces are unused, the reset pin. The power-on-reset functionality of the chip does not function properly. Whether to supply a dedicated reset signal to the chip was discussed, but eventually dismissed. The space needed for a separate trace on the front-end (see Section 2.5), as well as the infrastructure required for the readout electronics, were ultimately the reasons why the use of a reset connection was abandoned. A proper reset procedure can instead be achieved by powering up the digital and analog power supplies in a certain sequence, and then provide reset commands on the slow control interface.

2.3.5 Power Supply

The ALPIDE power supply is separated into two domains: the digital and the analog supply. The analog domain is connected to the pixel-matrix analog front-end electronics, the analog biasing circuits, and the ADC block. The digital domain is connected to the pixel-matrix readout circuits, configuration

[1]This frequency is associated with the bunch crossing rate of the LHC.

registers, the peripheral readout circuits, as well as the input and output buffers and transceivers. In addition, a separate supply exists for the on-chip PLL for the data transmission unit. As noted above, the PLL supply integrity is vital to ensure the optimal performance of the high-speed data link. Furthermore, one must also supply a bias voltage reference for the substrate and the p-type wells of the pixel matrix region. These two nets are electrically connected via the conductance of the die substrate [29].

The nominal voltage supply of both the analog and digital domain is 1.8 V. However, by increasing this voltage slightly, one has observed better high-speed link performance. Therefore, the DTC system aims to supply roughly 1.9 V to all the sensors. The possible values for the p-well and substrate voltage are ranging from 0V to -6V. A larger negative voltage increases the depletion region of the pixel diode and effectively increases the drift component of the charge collection.

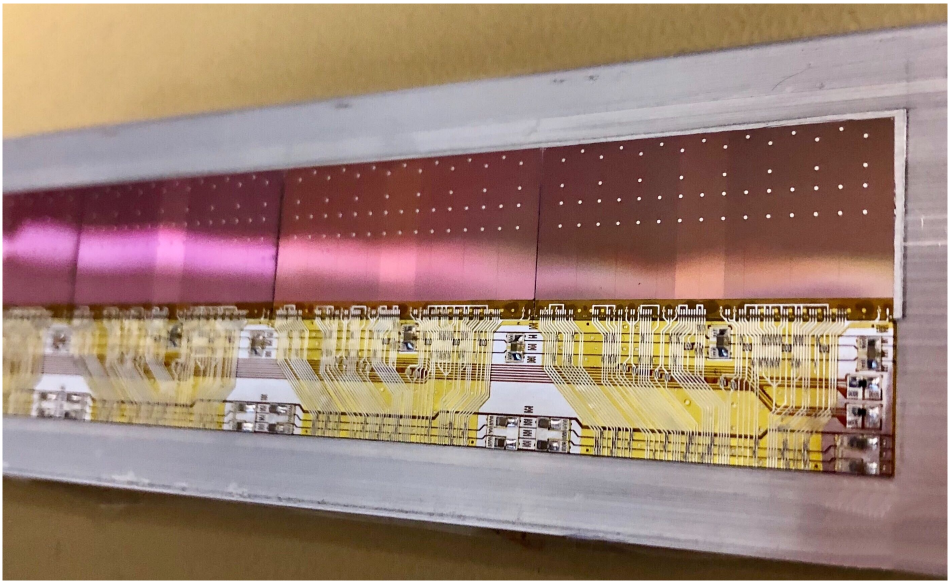


Figure 2.5: *Photograph of the manufactured string flex mounted on aluminum carrier with ALPIDEs and SMD-components bonded. The ALPIDEs are on the upper part of the image, while the chip cable and the string flex are visible in the bottom part of the image. The chip cable dielectric is shown with a slightly yellow color.*

2.4 Layer Design

The size of the layer area (*width* × *height*) is a major contributor to the cost of the detector, both in terms of the cost of each sensor and the complexity of the surrounding system. A size of 27 cm × 18 cm is chosen, as this size roughly corresponds to the width and the height of a grown man's head. A layer consists of several submodules:

String A string is the collection of 9 ALPIDE sensor chips mounted next to each other. The sensors are oriented with the short side facing the other. As the width of an ALPIDE is 3 cm, the total width of the string is 27 cm. On a string, all chips share the clock and slow control signal, and have unique IDs to separate them. Figure 2.5 shows parts of the finished string with ALPIDEs mounted.

Slab A slab is three strings combined to build height. The height of an ALPIDE is 1.5 cm, and the height of the flexible cable is slightly less than that. The total height of a slab is slightly less than 9 cm.

Half-layer A half-layer consists of a top and a bottom slab.

As illustrated by Figure 2.6, the sensitive area of a half-layer only covers about 50 % of the total area. This is caused by the need for the power supply and communication interface traces that must reach the sensors from the outside of the detector, and also because the ALPIDE itself has a digital periphery. However, it is crucial that all particles that penetrate the detector are registered by all layers. To ensure that a layer is completely covered with sensitive material, a type of double-module is constructed by flipping a slab to face another slab. From Figure 2.7 one can see the cross-section of multiple layers, and that the sensitive side of one slab covers the non-sensitive side of the other, and vice versa. This type of construction causes an air-gap of about 1.6 mm inside a layer, but also provides space for the required decoupling capacitors. Nevertheless, as the gap is relatively thin compared to the absorber, it is considered to have a minimal effect and constitutes a valid trade-off to ensure full layer coverage. A total of 108 ALPIDE sensors are needed to construct a complete layer. Thus, a layer consists of a total of ~56 Mpx.

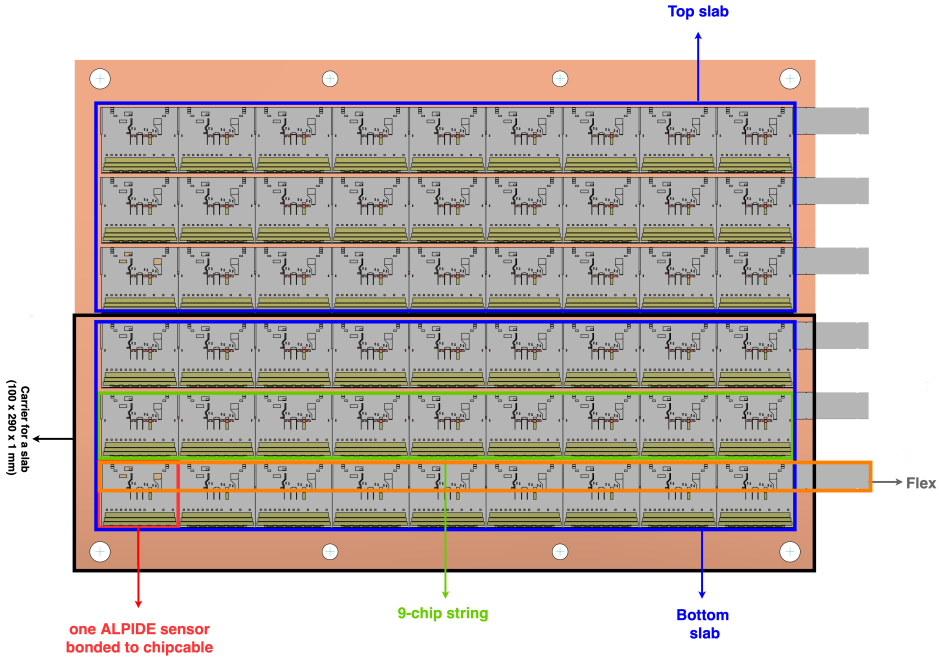


Figure 2.6: *Half sensor layer and the related glossary [39].*

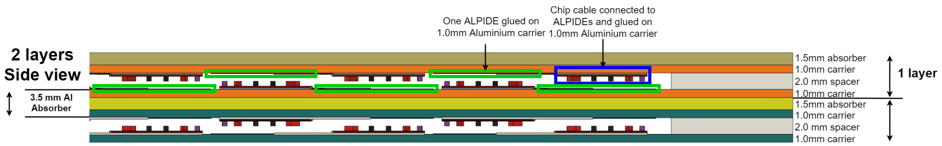


Figure 2.7: *Cross-section of two layers seen from the side of the DTC, rotated 90 degrees [39]. Note how the ALPIDEs of two corresponding half-layers are positioned at different locations, and that the sensitive area (marked in green) is slightly overlapping. The figure also illustrates how the 3.5 mm absorber layer is constructed of two 1 mm carrier layers, and a 1.5 mm absorber layer placed in between.*

2.5 Front-end Electronics Bonding and Mounting

Several types of bonding techniques exist to connect an ASIC's interfaces to back-end electronics. The ALPIDE has two types of bonding pads with different geometries. The type A pads are located on the south side of the chip, and the type B pads are scattered on top of the chip, as shown in Figure 2.8. The B-pads have rounded edges and are 290 µm wide. The A-pads are significantly smaller in size, 88 µm^2, and are electrically closer to the digital periphery of the chip. The two pad types were added to provide support for different bonding methods. The ALICE ITS adopted a traditional wire-bonding approach using the B-pads and carefully adding small wire bonds to a thin flexible printed circuit (FPC). This is a relatively simple method, but the wire bonds are susceptible to breaking if exposed to the wrong temperature and humidity. Quite extraordinary measures must be taken to make sure no damage is done to the wire bonds that will impede the operation of the sensor.

For this reason, the pCT group decided to opt for a different type of bonding

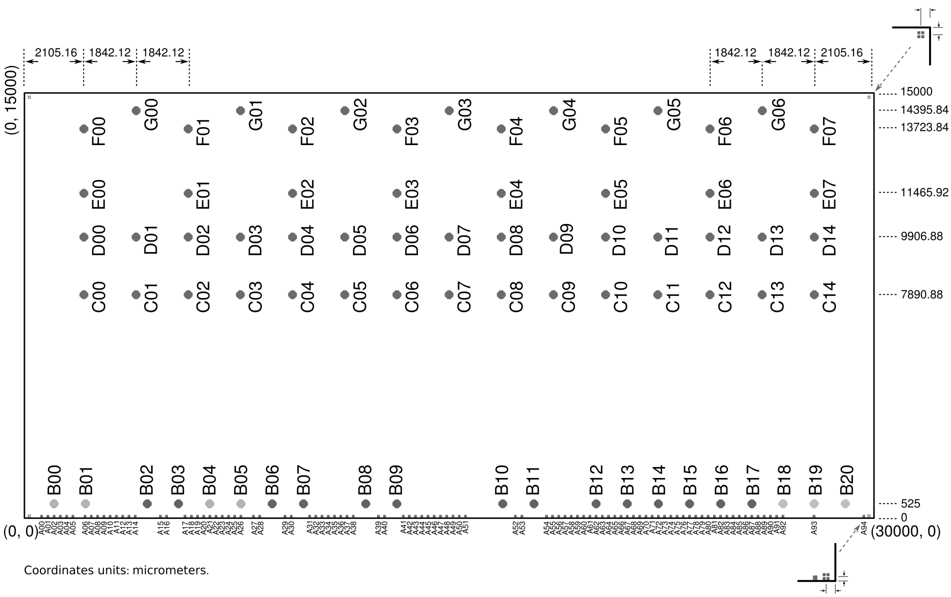


Figure 2.8: *The ALPIDE bonding pads [29]. Significantly smaller A-type bonds on the south side of the chip.*

technique. Single point tape automated bonding (SpTAB) is an alternative technique with several advantages compared to both wire and laser bonding [40]. The technique uses ultrasonic point welding, which involves an ultrasonically applied pressure and heat that connects a thin conductive material to the pad of the device. This type of bond is shown in Figure 2.9 and is considered to ensure a highly reliable and mechanically stable connection. Compared to wire-bonding, SpTAB also requires fewer interconnects to ensure reliability. Another reason for selecting SpTAB is to avoid the use of heavy metals since aluminum can be used as the conductive material. It is beneficial for the homogeneity of the detector that the metal used in the front-end is consistent with the absorber layer material.

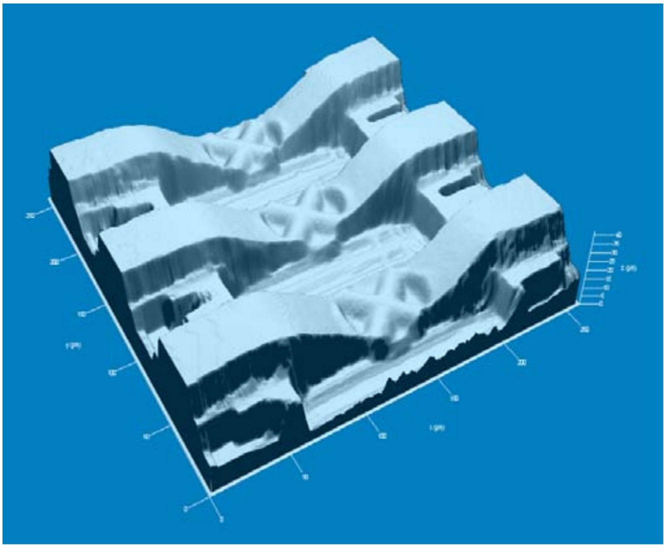


Figure 2.9: *Laser microscope image of SpTAB bonds from a related project [41].*

2.5.1 Chip Cable and String Flex

The chip cable and the string flex are both thin FPCs made of adhesive-less aluminum and polyimide [40]. Polyimide is a polymer that acts as a dielectric and a carrier for the conductive aluminum traces. The technology is specially developed for detector modules in HEP experiments by LTU Ltd. in Kharkiv, Ukraine. The manufacture process of the chip cable is based on precision photolithography and chemical wet etching. This approach has been utilized and tested and verified with the ALICE ITS silicon strip detectors [42]. The pCT front-end components must, however, be designed for significantly faster data readout and lower supply voltage compared to the ITS [41].

The chip cable serves as an interconnect between the ALPIDEs and the string flex. See Figure 3.9 for the transverse cross-section of the interconnection between the ALPIDE and the chip cable. A hole is etched in the polyimide dielectric and ultrasonic welding pushes the aluminum trace of the chip cable down onto the ALPIDE pad.

The string flex is an FPC based on the same principles as the chip cable. However, contrary to the chip cable, the string flex consists of two conductive layers. This is to deliver power to the sensors and to provide a proper ground plane for the high-speed links. The traces of the flex run perpendicular to the chip cable traces, and end a couple of centimeters outside of the detector area. A stiffener is added to the end of the FPC enabling connection in a Zero Insertion Force (ZIF)-connector. The height of the string flex is constrained. The height of the six-string flex in a half-layer cannot exceed the height of the detector. Several iterations of the chip cable and string flex FPCs have been produced and tested (see Section 3.2).

One could question the reasoning behind having two distinct parts in the front-end glue electronics: the chip cable and the string flex. It inevitably increases the material budget and complexity of the front-end. However, the choice is considered to improve the yield of the production as each production step can be tested. E.g., the bonding of a chip is deemed to be a critical process where many problems can occur. Thus, it is critical to test each chip after bonding. The bonding to the chip cable makes it possible to test the bonding of each chip before bonding to the string. If problems occur during the bonding to the string, this can be fixed without interfering with the chip bonds. The chip cable is designed to fit into a specific test-frame as discussed later in Section 3.2.1. A special tool was developed for testing the chips and the bonding, the Production Test Box (PTB) (see Section 7.5.1). This is also used to test the string as each of the chips are bonded, simplifying the debug process during production and again increasing the yield. The string flex, with the chip cables and the ALPIDEs connected, are glued to the absorber metal sheet.

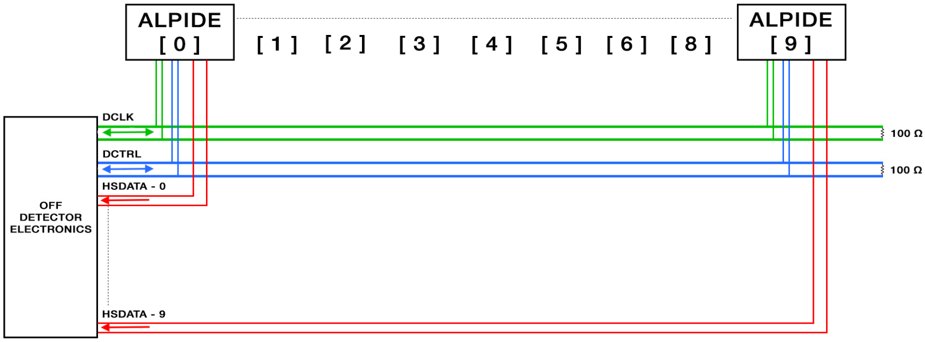


Figure 2.10: *Simplified schematic diagram of the interconnections between the string and the off-detector electronics [39]. The diagram also indicates that ID 7 is not used.*

A simplified schematic diagram of the signal interconnections between the string and the off-detector electronics is shown in Figure 2.10. All nine sensors share the clock and control signals, while each sensor embodies an exclusive high-speed data link. Note the 100 Ω resistors used for the clock and control signals. These resistors are placed on the far-end of the string to terminate the transmission lines. The termination of the high-speed links is done on the readout electronics.

2.6 Transition Card

The TC is a critical part of the front-end. As the dielectric loss of an aluminum FPC is considerable, one must strive to minimize the length of the traces on the string flex. One possibility would be to have the readout electronics placed directly on the outside of the detector, but unfortunately, the levels of radiation in this area makes this complicated and would require special care designing both the readout electronics and firmware. A transition card, however, solves this problem by simply acting as an interconnect and transforming all signals on to a different medium more suitable for transferring high-speed signals while minimizing the dielectric loss and without introducing other signal integrity issues. The string flex is thin and highly volatile and will only to a small degree withstand mechanical handling. The TC acts as a stabilizer for the string flex and it is carefully designed to not cause stress during connection. In addition, the TC acts as a distributor of power to the sensor chips, ensuring stable voltage supply and senses problems like single event latch-ups (SEL).

As seen in Figure 2.2 the TC is placed in close proximity to the front-end. Each layer incorporates a single TC. 12 ZIF-connectors are used to interface the 12 strings of a full layer. The TC is designed to be placed in a radiation zone, and each component is thoroughly tested to withstand the radiation levels of the LHC, far beyond the radiation levels expected in the operation of the pCT. As can be seen in Figure 2.1, every other layer is oriented differently, i.e. either left or right. This is because of the limited space available for the TC [39].

2.7 pCT Readout Unit

The pRU is designed to control all the sensors of the detector, as well as handling the data streams. As the ALPIDE data output is in a custom data format, only a custom solution is possible to handle the data streams. Also, as noted above, the ALPIDE slow control uses a custom protocol. These factors exclude the possibility of using any commercial-off-the-shelf DAQ systems. In modern-day physics experiments, like at the LHC, the use of FPGAs is commonplace. Not only because of the reduced cost and increased speed of development but also for the high performance that is now possible to achieve. Accordingly, the pRUs are employing an FPGA to perform the custom behavior required.

2.7.1 Board Overview

The pRU board's main component is an FPGA. The choice of FPGA is discussed in Chapter 4. The connection to the TC is done via twelve Samtec FireFly cables, one cable for each string. As shown previously in Figure 2.10, each string has a total of eleven signals that must be connected to the readout electronics. To obtain the best possible signal integrity for the clock signal, each string is supplied with its own separate clock signal. The use of the Samtec

FireFly ensures compatibility between the ITS and the pCT string design and the associated readout units.

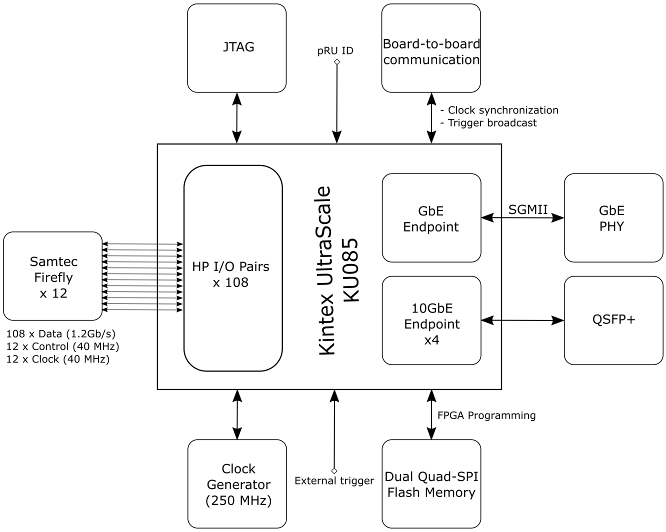


Figure 2.11: *Simplified block diagram of the pRU Board.*

Figure 2.11 shows a simplified block diagram of the pRU board. The board employs two different physical interfaces for control and data, QSFP+ for the high-speed data offload, and regular RJ45 interconnection for the run control link. In addition, the pRU also has a board-to-board communication interface. This is used for synchronization of the system and is discussed in Chapter 6.

2.7.2 FPGA Firmware Overview

As the design and development of the DTC has been a gradual process, the design of the pRU board was delayed until all design decisions had been made. Therefore, for any hardware implementation and testing of the firmware during the design process, a Xilinx VCU118 development kit has been employed. This platform was also used for parts of the front-end electronics (FEE) testing. The board has a Virtex UltraScale+ FPGA with similar architecture as the chosen FPGA (see Chapter 4). In addition, parts of the firmware have also been in use with the PTB as discussed in Section 7.5.1.

The FPGA firmware is designed to be modular and scalable. This is important

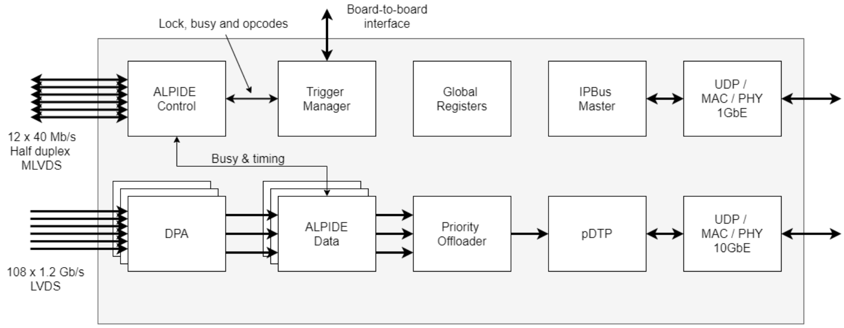


Figure 2.12: *Simplified pRU firmware block diagram.*

as the final design requirements have been fluctuating during the development process. All modules are centered around a global bus interconnect allowing full control and monitoring of the system from the control room, see Section 6.1. As seen in Figure 2.12, the ALPIDE communication is made possible by two separate blocks. The slow control communication is handled by the ALPIDE Control and the Trigger Manager. These two modules are fully controlled by the bus interconnect system and the pRU board-to-board interface. A detailed discussion of these modules is found in Chapter 6. The high-speed data processing, however, requires a complex block of various modules, as discussed in Chapter 5.

2.8 Power Supply and Control

The FEE are powered by several PCUs. This device is also responsible for monitoring the current and acts as a fail-safe in case of shorts, SELs, and other harmful events. The PCU controls the voltage regulators of the TC and monitors the temperature of the detector. An outline of the device and the interface to the TC is shown in Figure 2.13. The PCU is not discussed in detail in this thesis but is mentioned as part of the detector control system (see Chapter 6). More details about the PCU can be found in [39].

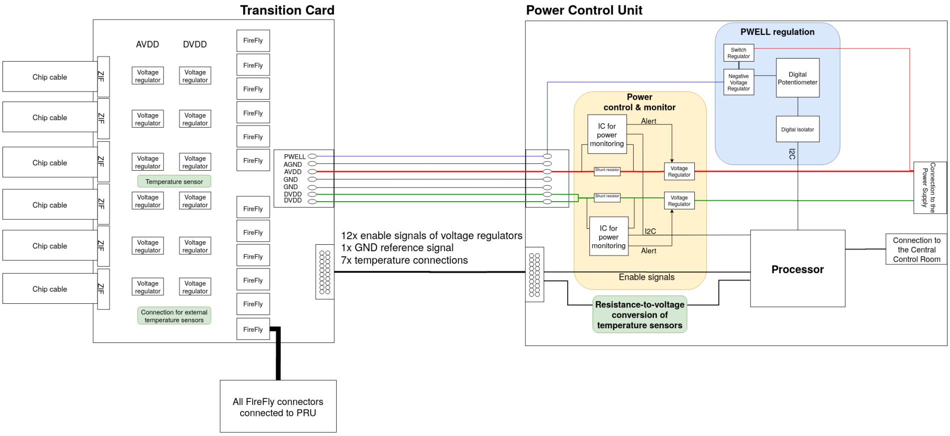


Figure 2.13: *Transition Card and Power Control Unit diagram. Figure provided by courtesy of Tea Bodova [39].*

2.9 Other Prototypes

The work reflected by this thesis has also involved efforts related to several other prototypes that are associated with the pCT detector. A short description is given here, but the prototypes are not described further in this thesis.

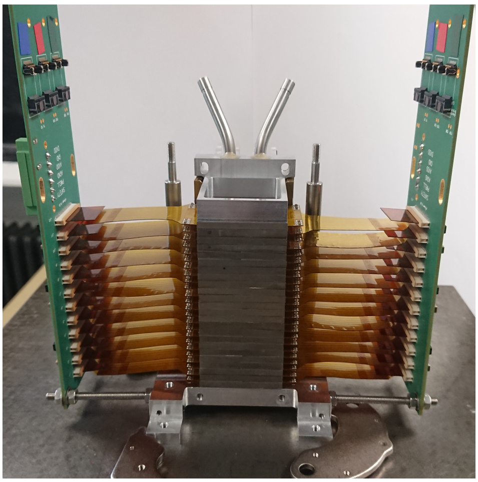


Figure 2.14: *Picture of the FoCal mTower.*

The FoCal mTower, as seen in Figure 2.14, is a prototype intended to be the first design step of both the ALICE FoCal detector and the pCT DTC. Rather than mounting 9 chips as a string, a two-chip FPC with the ALPIDE oriented with the wide side facing the other is chosen. With these 2-chip modules, a telescope of chips is constructed like a tower. A total of 24 modules, with a 3 mm tungsten absorber placed in between, are connected via two transition PCBs to the readout electronics. With this design, one could perform multiple tests such as qualification of the bonding method and the chip cable design. Also, several preliminary tests could be performed with the tracking of particles in the telescope.

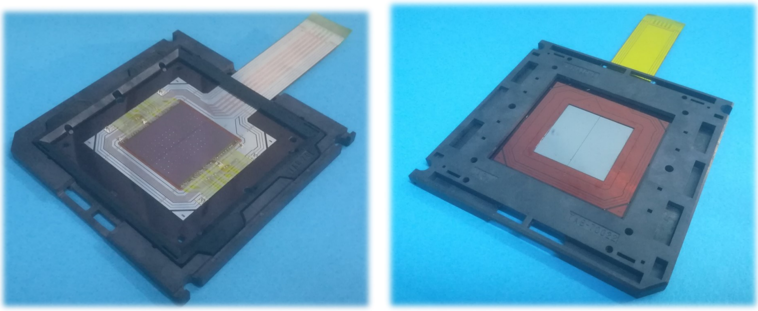


Figure 2.15: *Picture of the ULTM.*

The Ultra-Ligth Test Modules (ULTM) was designed to provide an open-chip solution for other physics experiments. The modules were the first chip cables designed utilizing the type A-pads.

Two 15-chip strings were designed for the ALICE FoCal experiment. The strings utilize both the inner and outer barrel mode of the ALPIDE, to multiplex the data streams of the chips with lower occupancy. Furthermore, the string design has also been proposed for the ALICE ITS3 upgrade, using bendable sensors.

CHAPTER 3

String Design and Performance Evaluation

In this chapter, the concept of signal integrity is introduced along with a general theory of transmission lines, characteristic impedance, and various types of signal loss. The front-end designs have been subject to several iterations. This chapter shows the first and final iterations of the chip cable and the string flex, and the changes are discussed in detail. Simulations that were done alongside the design are presented. Finally, we show specific lab test results that assess the quality of the front-end designs.

3.1 Introduction

Any custom high-speed electronics need to be designed with signal integrity in mind. Signal integrity refers to all problems that can arise when the frequency of a digital signal is increased, or when the rise time of a digital signal is decreased. These two factors are usually correlated. Signal integrity problems are generally related to noise and are commonly categorized by the following four elements [43]:

- Signal quality issues of a net, including reflections and attenuation.
- Power distribution network issues.
- Cross-talk between interconnects.
- Electromagnetic interference.

As a general rule of thumb, a reduction of the signal rise time worsens all types of signal integrity problems [43]. But if the rise time and the frequency of the signal are given, as with the ALPIDE high-speed link, one needs to design the rest of the electronics with these problems in mind. Both simplified equations and simulations can be helpful in that process. When designing custom electronics, with fairly rigid constraints in the material budget and space usage, near zero-loss traces are very difficult to attain. Transmission line loss is closely related to the conductor and dielectric material, the width and separation of the traces, as well as the dielectric thickness.

3.1.1 Transmission Line Model

Any conductive medium where the propagation time of the signal, i.e., the time the signal takes to travel from the source to the receiver is comparable to the rise time of the signal itself, benefits from transmission line analysis. A transmission line can be modeled using the lumped-element model in Figure 3.1. The lumped-element model represents an infinitely small part of the transmission line, and each of the elements is notated per length, e.g. Ω/m. G and C are primarily defined by the nature of the dielectric, R involves the effects of the conductor metal, and as the metals used for transmission have a relative permeability of roughly 1, L mainly implicates the geometry of the transmission line. L and C tend to dominate the effects, but when designing custom electronics, care must be taken regarding all factors.

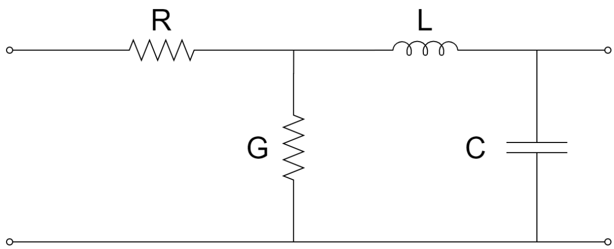


Figure 3.1: *Lumped-element model of an infinitely small part of a transmission line.*

3.1.2 Characteristic Impedance and Mismatch

The characteristic impedance of a transmission line can be calculated from the lumped-element model in Equation 3.1. Impedance mismatch is a prevalent signal integrity problem. As a signal propagates down a net, changes in the impedance will cause reflections of a signal back towards the transmitter. Thus, a mismatch will cause so-called rejection loss. Usually, this is resolved by carefully designing the electronics to adhere to the decided impedance, for example 100 Ω. However, with custom flexible printed circuit (FPC) with explicit constraints, this can be hard to achieve. For instance, the pCT detector design strives to minimize the material budget, restricting the thickness of dielectrics and conductive materials. Also, the width of traces might be limited because of space issues, and the desire to restrict other signal integrity issues like crosstalk. Thus, the optimal impedance might be unattainable. Note also that a matched termination on the end of a transmission line is necessary to avoid reflections and rejection loss.

$$Z_0 = \sqrt{\frac{R + j\omega L}{G + j\omega C}} \tag{3.1}$$

3.1.3 Conductive and Dielectric Loss

Although care must be taken regarding impedance matching, other mechanisms are the principal contributors to signal attenuation for signals with clocks higher than 1 GHz [43]. The propagation constant, γ in Equation 3.2, indicates how a wave is affected by the medium it traverses. A lossless medium will only have a complex part, $j\beta$, that details the phase shift of the signal. However, if the attenuation constant, α, is non-zero, the amplitude of the signal is also affected. For low-loss lines, one can approximate the attenuation per length unit to roughly equal Equation 3.3 [44, 43]. Conductive and dielectric loss dominate the signal attenuation.

$$\gamma = \alpha + j\beta = \sqrt{(R + j\omega L)(G + j\omega C)} \tag{3.2}$$

$$\alpha \approx \frac{1}{2}\left(\frac{R}{Z_0} + GZ_0 \right) \tag{3.3}$$

Conductive loss, α_c of Equation 3.4, can be simplified as the ratio of the resistance of the conductor and the characteristic impedance [43, 45]. The resistance of the conductor is the RF sheet resistance, Equation 3.5, divided by the width of the conductor [45]. The RF sheet resistance is a function of frequency and characteristics of the medium, like permeability and conductivity. Therefore, in a lossy transmission line, the resistance per length increases with the square root of the frequency due to skin depth [43]. In microstrip traces, one can often reduce the conductive loss by keeping the characteristic impedance constant and increasing the dielectric thickness. This increase must be compensated by an increased trace width to keep the characteristic impedance constant [46].

$$\alpha_c = \frac{(\frac{R_{RFSH}}{W})}{2Z_0} \tag{3.4}$$

$$R_{RFSH} = \sqrt{\frac{\pi f \mu_0 \mu_R}{\sigma}} \tag{3.5}$$

The dielectric loss of a transmission line, Equation 3.6, is proportional to frequency. Therefore, at a high frequency, dielectric loss will dominate over conductive loss.

$$\alpha_d = \tan(\delta)\pi f Z_0 \tag{3.6}$$

Of adjustable variables, the dielectric loss is mainly dependent on the dissipation factor of the dielectric, δ, and the characteristic impedance. Note that, for a fixed characteristic impedance, the dielectric loss will stay fairly constant for a change in dielectric thickness [46]. Thus, the dielectric loss is independent of the trace geometry when the characteristic impedance is given [43]. Thus, an increase in the dielectric thickness to reduce the conductive loss will not affect the dielectric loss of the conductor to any great extent.

3.2 Front-End Design

As mentioned in Section 2.5.1, the front-end components have gone through several iterations in the process of optimizing the signal and power integrity.

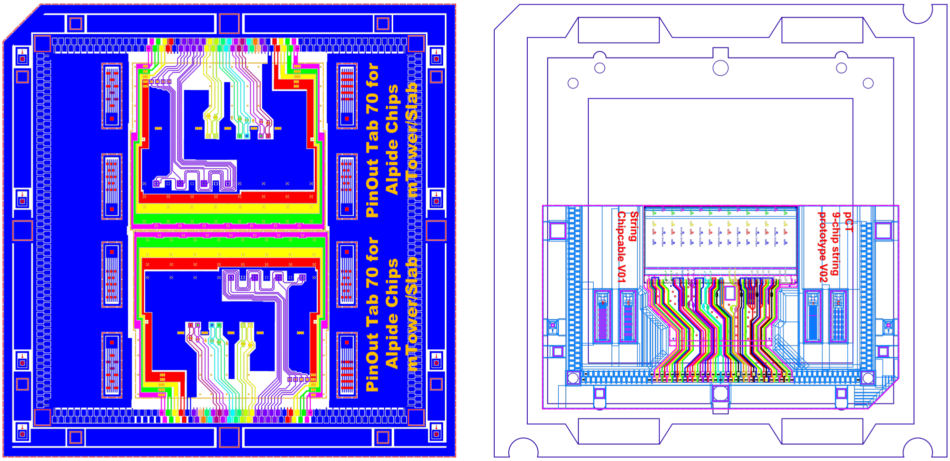
The designs were made by LTU Ltd. in Kharkiv, Ukraine, in collaboration with members of the Bergen pCT team. Using a mechanical CAD-tool, the different layers of the front-end components (the chip cable and the string flex) are drawn. As the tool is only structural and no netlist is used, all schematics must be checked manually. This unfortunately caused errors during the design that were not detected before testing in the lab. Figure 3.2 shows how a bond between two layers is made in the CAD-tool. The visual inspections of hundreds of such bonds turned out to be time-consuming and prone to error.

Figure 3.2: *Illustration of a bond between the chip cable and the string flex. The vertical purple trace is on the chip cable flex, while the horizontal purple trace is on the string flex. The grey square marks the etching hole of the polyimide dielectric. This particular bond is for one of the differential traces of the slow control MLVDS-link. The size of the etching hole is roughly* $500\,\mu\text{m}^2$*. The double bond increases reliability.*

3.2.1 Chip Cable

Figure 3.3 shows two different versions of the chip cable. The chip cables were designed to fit into a tape carrier package (TCP) frame to enable testing before further bonding steps[1]. Each trace is connected to a pad in the frame. This connection causes extra stubs as seen in Figure 3.4. After bonding and testing, the extra material of the chip cable is cut off and removed.

[1]The TCP frame is produced by Yamaichi.



(a) *First version.* **(b)** *Last version.*

Figure 3.3: *Chip cable designs with the outline of the TCP frame.*

Figure 3.3a was the first version that was used in the production of a stave for testing. Note that in this version, two identical chip cables fit into the frame. Also, each trace is connected twice to the TCP frame. This was done to enable testing of the chip cable design without bonding an ALPIDE chip first. However, we found that the extra stubs hurt the signal integrity by introducing significant reflection, especially for the high-speed links, and the stubs were removed in later versions. However, one of the extra stubs needed to remain, to allow connection to the TCP test contact. Simulation of the single remaining stub per interconnection showed only a minimal effect on the signal integrity.

The major difference between the two versions is the type of bonding pads employed. The first version of the chip cable applied the type B pads of the ALPIDE. This is illustrated by the blue, red, yellow, and green power planes placed on top of the chip, while the traces for the clock, chip ID, slow control, and the high-speed data create gaps in these planes. The type B pads are significantly larger than the type A-pads and were considered to be the easiest solution. Also, since the B-pads are positioned on top of the chip, it is substantially easier to execute the SpTAB bonding technique as the welding motion is simply up-down relative to the surface of the chip cable and the ALPIDE.

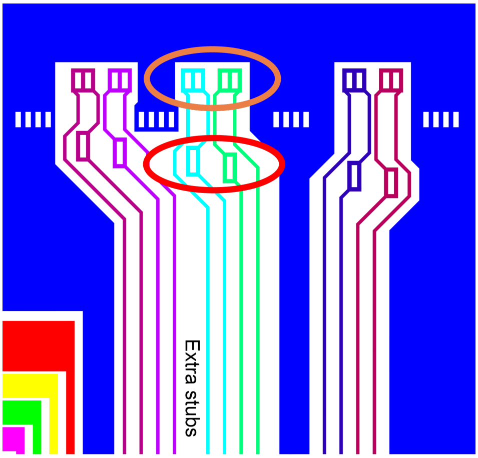


Figure 3.4: *Chip cable double-stubs, zoomed in from Figure 3.3a. The orange circle indicates the position of the bond to the ALPIDE pad. The red circle indicates the position of the string flex bond. The extra stubs are the interconnections to the TCP test pads.*

However, there are two significant benefits of using the A-pads. Firstly, the B-pads are spread out on the chip, and the power pads lie on the other side of the chips, relative to the digital periphery. This means that the length of the traces for DVDD and PVDD is increased by almost the height of the chip, 1.5 cm. Learning that the chip needs decoupling capacitors as close as possible to the PVDD pads in order to reduce jitter on the high-speed links, diminished the benefit of the B-pads.

Secondly, notice that the first version does not separate DVDD and PVDD. This means that any voltage drop on the DVDD supply rail will also be observed at the PVDD input. This is critical as the PVDD must be kept as stable as possible to reduce jitter on the internal clock of the chip, and to avoid deterioration of the high-speed link performance. It would be challenging to split DVDD and PVDD supplies and continue to use B-pads as it would require using the already minimal space on the edges of the chip cable, possibly causing a significant voltage drop and counteracting the benefit of splitting DVDD and PVDD.

Notice also, the tenuous connection of the blue ground plane to the TCP frame pad. Although this was probably not a source of error when bonded to the string flex, it might have been one of the contributions to the difficulties that emerged during the first lab tests.

In the last version, the power planes are entirely removed, and the bonds are using the type A-pads. This allows us to add material in other places of the front-end and keep the same material budget, for example, increasing the thickness of the ground plane of the string flex.

Another significant difference between the versions is the length of the traces. With the use of the A-pads and changes of the cross-section of the string flex discussed later in Section 3.2.2, the chip cable is required to bend more, which required a small increase in the trace length. This can be seen in the transverse cross-section in Figure 3.9.

3.2.1.1 Cross-Section and Microstrip

The chip cable consists of a single layer of aluminum traces built on top of a 20 µm polyimide dielectric. The thickness of the aluminum was 30 µm in the first version but was later reduced to only 14 µm without any performance loss. This was done to reduce the stiffness of the material and allow the material to bend. When the traces are interpreted as microstrips, conventional calculations of characteristic impedance can be done. Although the traces from the pads of the ALPIDE to the pads of the chip cable are relatively short ($\sim$2.6 mm), during testing, the traces are longer ($\sim$11 mm) because of the connection to the TCP test points. Therefore, care still needs to be taken when designing the traces to ensure proper performance during testing.

3.2.2 String Flex

The string flex is a multilayer FPC with long traces, and with a much longer signal propagation time than the signal rise time. Figure 3.5 shows the first version of the string flex. In this version two layers of identical thickness (30 µm/20 µm) are used facing each other, providing a total dielectric thickness of roughly 45 µm, including the epoxy glue. It was later learned that this dielectric thickness is not sufficient to provide a low enough resistive loss in the transmission line.

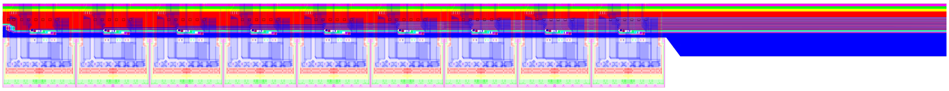


Figure 3.5: *String flex version 1 shown top down. The* 13 cm *cable extending the detector area is shown on the right. The nine ALPIDEs and the chip cables are located on the bottom of the figure.*

The first version of the string flex also employed the first version of the chip cable design. This entails that the bonds to the ALPIDE were done with a top-down bond to the type B-pads. In Figure 3.6 one can see that the bonds to the ALPIDE are to the left of the figure, and the bonds to the flex string are to the right. The chip cable traces, however, are relatively short compared to the string flex traces. In the longitudinal cross-section of Figure 3.7, we see that the signals are carried on the double layer flex, and then outside of the aluminum carrier for roughly 13 cm. The length of the trace on the double layer flex varies depending on which chip the traces are connected to. The high-speed signal traces for the near-end chip are rather short, only about 2 cm. However, for the far-end chip, the traces are about 27 cm long.

The drawn width and the separation of the differential microstrip traces are

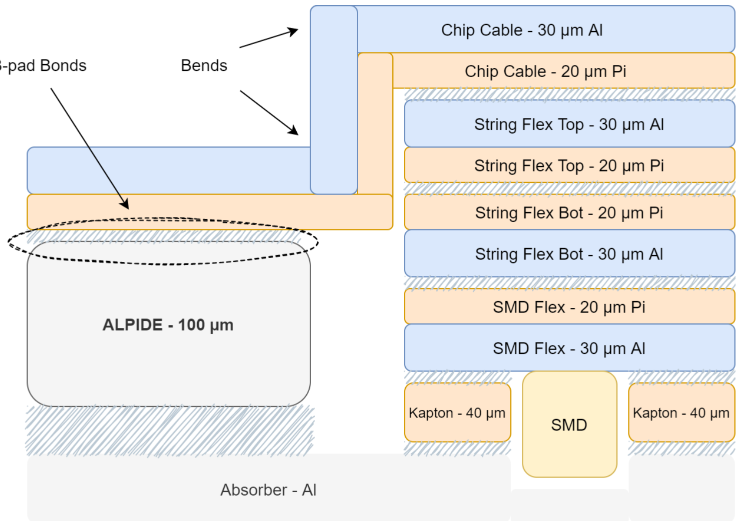


Figure 3.6: *String flex version 1 transverse cross-section. The surface layers are to scale, but the layer width and SMD component size are not to scale. Epoxy glue used between the layers are illustrated by the hatched grey areas. The figure also illustrates that a well must be created in the absorber metal to fit the SMD components.*

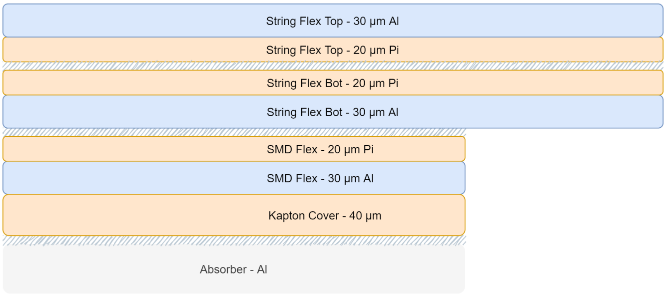


Figure 3.7: *String flex version 1 longitudinal cross-section. Layer length not to scale.*

110 µm and 95 µm, respectively. However, during the production process, the traces shrink slightly, and we assume that the manufactured width and separation are both roughly 100 µm.

Figure 3.6 and 3.7 shows the transverse and longitudinal cross-sections of the first version of the chip cable and string flex design. In this design, all the conductive aluminum layers are 30 µm thick, and the polyimide dielectric layers are 20 µm thick. The layout of this design makes it difficult to interpret the top layer of the chip cable. In terms of characteristic impedance and conductive loss of the high-speed traces, the effect is relatively large depending on whether the chip cable layer is interpreted as a plane or a signal layer. As we can see from Figure 3.3a, a large part of the layer consists of a ground plane, and this can influence the characteristic impedance. With a plane interpretation, the characteristic impedance is calculated to be 41.8 Ω, while a signal interpretation gives 68.1 Ω. Note that neither of these values are close to the goal impedance of 100 Ω. Figure 3.12a and 3.12b show the difference in terms of loss where the plane interpretation predicts a higher resistive loss than the signal interpretation.

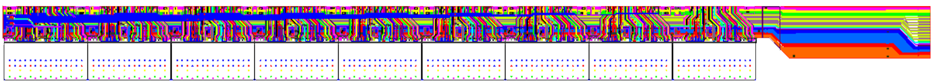


Figure 3.8: *String flex version 2 shown top down. The ALPIDEs are no longer covered by the chip cable plane.*

The most obvious change of the flex string one can observe from Figure 3.8, that shows the latest design, is the change of the chip cable type. The chip cable power planes do no longer cover the ALPIDE sensor and the string flex.

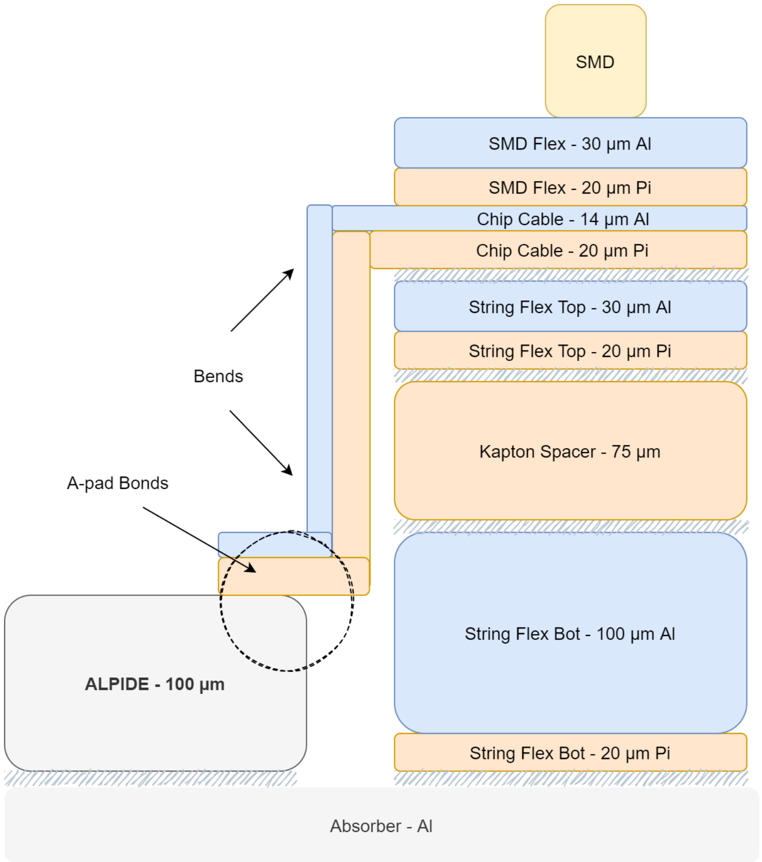


Figure 3.9: *String flex version 2 transverse cross-section. The layer width and SMD component size are not to scale.*

Because of this change, the interpretation of the chip cable layer is no longer disputed, and in all calculations of the string flex characteristic impedance and resistive loss, the layer is considered a signal layer. As we saw in the previous string flex version, this will improve both the characteristic impedance and the losses. Also, the split of the DVDD and PVDD power lines are continued on the string flex, allowing for a more stable PLL output on the ALPIDEs.

Moreover, another major change of design which is shown in both the transverse and longitudinal cross-sections in Figure 3.9 and Figure 3.10, is the addition of a 75 µm dielectric spacer layer in between the signal and the ground layer. This significantly improves the electrical characteristics for the high-speed traces, but also substantially increases the thickness of the string flex, reducing the flexibility of the string.

The reduction of aluminum resulted from chip cable change is allowing for an increase in the material use in the now *thicker* bottom ground layer. The thickness of the top layer stays the same at 30 µm/20 µm, but the bottom layer is increased substantially to 100 µm/20 µm. The bottom layer does not only act as the ground layer for the high-speed traces, but also provides most of the power to the chips. This change was done to reduce the voltage drop in the longitudinal direction of string flex, improving the performance of the ALPIDEs at the end of the string. As the DVDD and PVDD traces were split, a considerably narrower trace is allowed for the DVDD supply. The increase in the thickness of the bottom layer circumvents this change.

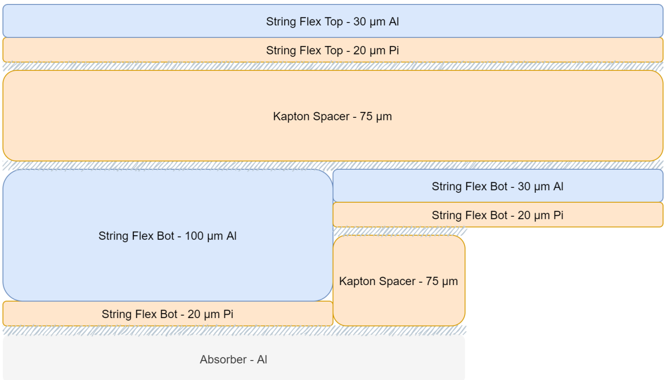
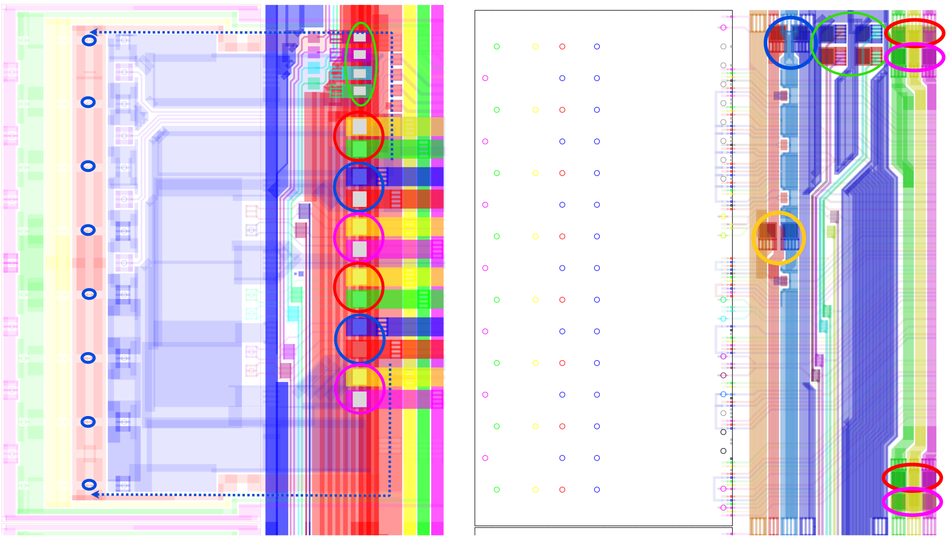


Figure 3.10: *String flex version 2 longitudinal cross-section. Layer length not to scale.*

To allow for the flexibility needed when connecting the string flex to the ZIF-connector, the bottom layer is separated into two parts, as seen in Figure 3.10, where a thinner layer is used where it extends from the absorber.

Because of the changes in the dielectric thickness, the width and the separation of the high-speed traces are somewhat modified, to 120 µm and 80 µm, respectively. With all these changes, the new characteristic impedance is calculated to be 86.3 Ω, much closer to the goal of 100 Ω. Also, the resistive loss is dramatically reduced, as we expected for the increase in dielectric thickness and trace width. This is observed in Figure 3.12c.

Lab experiments of the ALPIDE at CERN, showed that the decoupling capacitors need to be very close to the supply pads to ensure reliable behavior, especially of the high-speed link. The decoupling capacitors are intended to keep the power supply voltages stable. The locations of the decoupling capacitors are marked in Figure 3.11. On the first version of the string flex and chip cable, however, the distance from the capacitor to the power pads is almost 27 mm. This is because the power supply traces are routed along the edges of the chip cable. In addition, one also needs to consider the distance of the internal nets of the ALPIDE from the pad to the digital periphery where the PLL is located. With the latest version of the string flex, the distance from the decoupling capacitors, especially for PVDD and DVDD, is minimized.



(a) *First version.* **(b)** *Last version.*

Figure 3.11: *Placement of decoupling capacitors and termination resistors. The termination resistors of the clock and control signal are marked with the green circle. The analog and PWELL capacitors are marked with red and pink. The capacitors for the digital supply and PLL supply are marked with a blue and an yellow circle. The paths from digital supply capacitors to the pads are marked with blue arrows in the first version. The path lengths in the last version are negligible in comparison.*

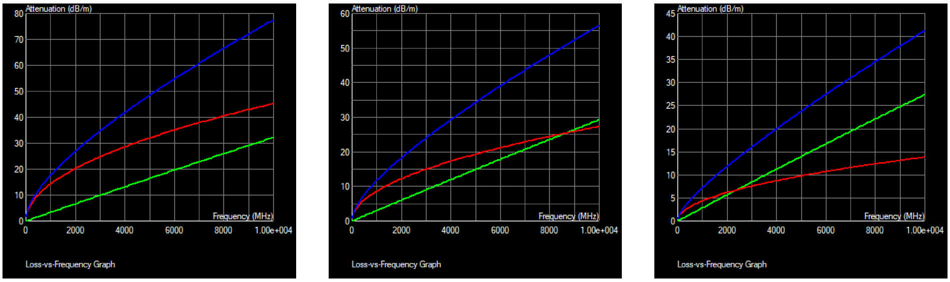
The slow control and clock interface are employing the MLVDS signal standard, which requires termination on both ends of the signal. In Figure 3.11, the placement of these termination resistors is also marked.

The following list summarizes the major improvements of the chip cable and string flex designs:

- Change from using type B-pads to type A-pads, thus removing the power planes on the chip cable.
- Remove double stubs from chip cable.
- Split the DVDD and PVDD power supplies for the whole front-end.
- Separate decoupling capacitors for DVDD and PVDD.
- Reduce the distance from the decoupling capacitor to power pads.
- Increase the dielectric thickness of the string flex.
- Modify the width and separation of the string flex traces to better characteristic impedance and resistive loss.
- Decrease the length of the post-carrier stub.
- Separate the flex strings into two parts; a thin external part allows for flexibility when connecting to ZIF-connector, while the thicker part reduces the voltage drop of the power supply in the longitudinal direction of the flex string.
- Move the termination resistor and decoupling capacitor SMDs to the top of the chip cable.

3.2.3 Simulation

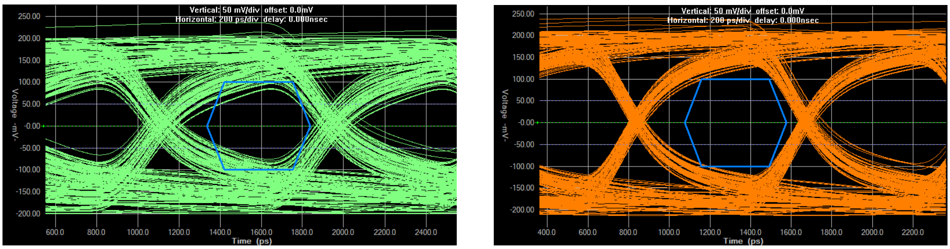
To support the design process of the front-end, i.e. the chip cable and the string flex, a Mentor Hyperlynx simulation environment was used. Hyperlynx employs a field solver to extract the behavior of electrical traces. Although it is made for conventional PCB simulation, it allows for enough customization to be applicable for custom-made FPCs as well. These types of simulations are useful to quickly see what kind of effect changing any parameter will have on major decision-points. The result, however, might differ significantly from the real-world results based on the level of detail in the model. In these simulations, we opted to find the characteristic impedance and the loss of particular traces based on the cross-section of the structure. As shown in Figure 3.12, the newest design has a significantly lower resistive loss, while the dielectric loss remained nearly the same. Additionally, a full system simulation is run to compare eye diagrams of various designs. With the eye diagram simulations shown in Figure 3.13 one can see improvement in the newest design. These simulations are done with the characteristics of the longest string trace, to the far-end sensor.



(a) *Flex v1. Chip cable interpreted as a plane layer.* **(b)** *Flex v1. Chip cable interpreted as a signal layer.* **(c)** *Flex v2.*

Figure 3.12: *String flex v1 and v2 trace dielectric and resistive loss. Red: resistive loss. Green: dielectric loss. Blue: total loss. Generated by Mentor Hyperlynx using the field solver.*

.



(a) *First version.* **(b)** *Last version.*

Figure 3.13: *Eye diagram simulations of the front-end components. The final version results in a wider and higher eye opening.*

.

3.2.4 Experimental Verification

Significant efforts are done to ensure proper operation of the pCT string design. An excerpt of the tests performed is shown below. Overall, the tests give reasons to have confidence in the latest string design and also in the sampling method which is discussed in Chapter 5.

3.2.4.1 Eye Diagram Measurements

Figure 3.14 shows the eye diagram of the high-speed signal of one of the ALPIDEs on the pCT string during pseudo-random binary sequence (PRBS) testing. We observe a proper open eye reaching the requirements of the LVDS-requirements of the Xilinx I/O-pins.

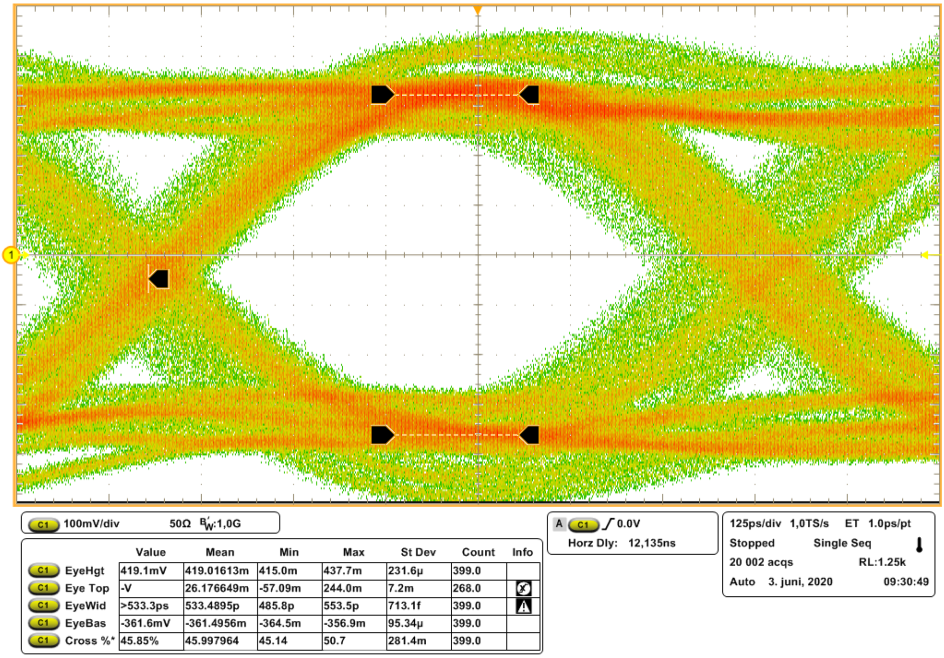


Figure 3.14: *Eye diagram of the pCT string during PRBS testing. Eye height is measured to be* 419 mV *and eye width* 533 ps*. Tests were done with nominal driver link settings and with a* 2 m *FireFly cable.*

3.2.4.2 Setting Sweep Tests

The ALPIDE high-speed driver settings can be configured to circumvent the detrimental effects of the electrical traces. Furthermore, we can also configure the PLL of the ALPIDE can be configured to minimize the effect of supply voltage drops. Figure 3.15 shows the decode error results from sweeping two of these settings for all chips on the pCT string: the driver strength and the pre-emphasis. Pre-emphasis is used to emphasize the signal strength of the first bit in a sequence of equal bits, opting to counteract the effects of reflections on the transmission line and thus improve the signal-to-noise ratio (SNR). Any error indicates that the 8B/10B decoder warns of inconsistencies when decoding a sequence of bits. The particular setting sweep shown in the figure shows that nearly all settings yield a perfect performance for most sensors, except one: chip ID 1. This indicates an issue with the bonding or the internal operation of this particular sensor.

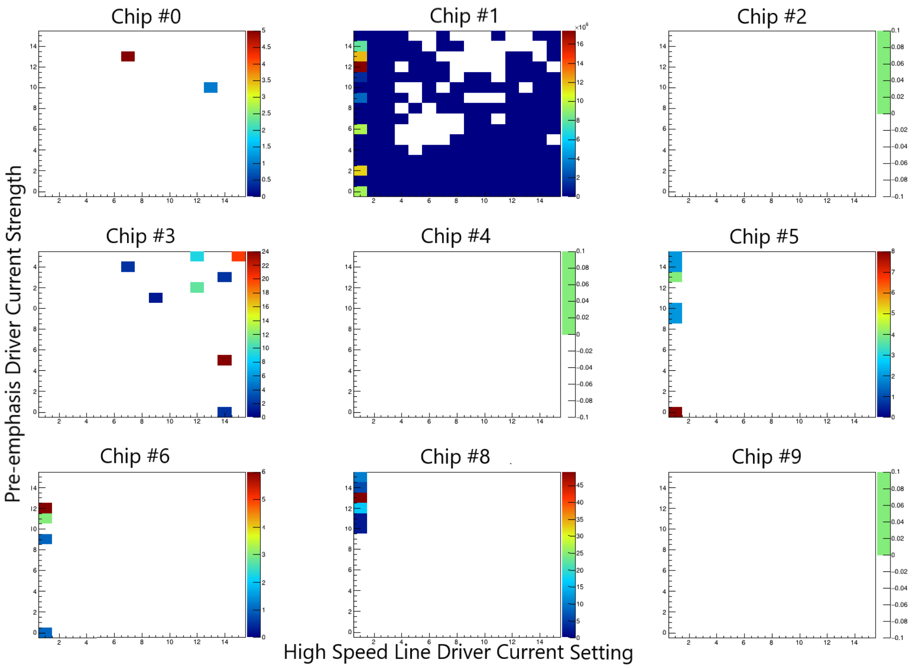
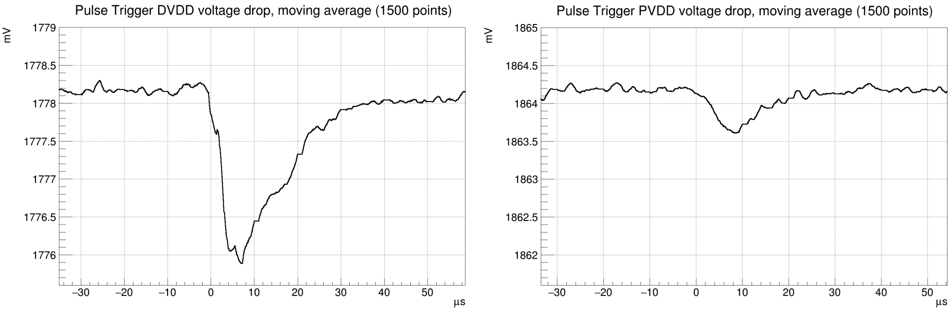


Figure 3.15: *Results of an ALPIDE link settings sweep test. The plots show the number of decode errors observed for each setting combination of the link driver current and pre-emphasis current strength. No errors were observed for chips 2, 4, 9.*

3.2.4.3 Voltage Drop

Figure 3.16 compares the voltage drop on the DVDD and PVDD power supplies on the pCT string following a trigger command. We observe that both supplies respond to the trigger, but the DVDD drop is significantly larger than PVDD. The PVDD voltage also returns faster to the baseline. This result is reassuring in that a trigger will not cause large disruptions to the PLL on the ALPIDE and that the separation of DVDD and PVDD is important to improve the signal integrity of the high-speed links.



(a) *DVDD.* **(b)** *PVDD.*

Figure 3.16: *Measurement of the voltage drop on DVDD and PVDD following a trigger. The results show a significantly larger drop on DVDD than PVDD. Measured using a differential probe over the decoupling capacitor.*

.

3.2.5 Conclusion

The verification efforts show that the front-end design is acceptable. The electrical behavior is well understood and is optimized for better performance.

CHAPTER 4

FPGA Design Considerations

The choice of an FPGA is an important part of the pRU board design. In this chapter, we briefly discuss the FPGA selection considerations. The chapter is a mix of both design descriptions and design choices. The chapter is introduced by a brief discussion about FPGA vendors and the differences between the ALICE ITS and the pCT DTC readout requirements. Furthermore, FPGA I/O technology is discussed in reference to the pRU data sampling approach. The chapter also discusses the clock network of the pRU and clock domain crossings. Also, the FPGA resource utilization is presented. Finally, the radiation environment of the pCT and consequences for the design are discussed.

4.1 Introduction

There are three major FPGA manufacturers whose products are used in HEP experiments and similar applications. The two market-leading companies are Intel (previously Altera) and Xilinx. The FPGA-devices of these two manufacturers are mostly SRAM-based and have a large range of various resources. Microchip (previously Microsemi) FPGAs are also used in some instances but as the Microchip FPGA-devices are mainly flash-based or anti-fuse, they are usually only used in experiments with very high radiation.

At the early stages of development the pCT detector was closely linked to the upgrade of the ALICE ITS. The prototype of the ITS readout unit was based on a Xilinx Kintex 7-Series FPGA. This FPGA was used for the initial pCT firmware experiments (see Section 5.2). However, the later units employ Xilinx UltraScale Kintex FPGAs. Porting firmware between FPGAs, and especially between different manufacturers, is a time-consuming process. This is in part due to the use of technology-specific primitives and IPs, but also because of the device-specific design of the I/O and FPGA time and physical constraints. As a consequence of the close relationship with the ITS development, it was decided to design for Xilinx FPGAs.

The ITS and the pCT readout electronics are quite similar. Both readout devices are interfacing multiple instances of the ALPIDE sensor chips. However, there are also major differences between the readout electronics. The main contributors to the different design choices are the following:

- The ITS readout unit only interfaces a single high-speed sensor stave (9 sensors), while the pCT readout unit interfaces twelve (12×9 sensors) to significantly reduce the magnitude of the data acquisition system.
- The ITS readout unit is placed in a high-radiation zone, requiring a high emphasis on mitigation of single event upset (SEU) induced soft-errors. Note that with a smaller number of sensors connected to the readout unit, the ITS readout unit can (1) employ a relatively small FPGA and (2) use more of the FPGA's resources on radiation mitigation like TMR. The pCT readout unit is intentionally placed in a radiation zone with minimal risk of soft-errors (see Section 4.6). This is done to avoid using TMR and other types of mitigation and thus saving FPGA size.

The choice of the pRU FPGA is based on three main considerations: (1) legacy code, (2) resource utilization and (3) cost. The pRU will use the Xilinx UltraScale KU085 FPGA. This FPGA is slightly larger than the FPGA chosen for the ITS readout unit. See the remaining parts of this chapter for the considerations that were taken for this selection.

4.2 High-Speed I/O

When the fractional phase detector was introduced, the technology for serial interfaces was boosted to the multi-gigabit range [47]. However, for FPGAs, the rise of bandwidth was mostly attributed to the multi-gigabit transceiver (MGT) pins. Most modern FPGAs have I/O-pins that are specifically designed to handle asynchronous high-speed data with electrical standards like LVDS. A MGT I/O-pin has extended features specifically added to handle outgoing and incoming high-speed data. However, FPGAs with many of these I/Os are usually very expensive. Besides, with many MGTs, other features on the FPGA might be reduced. This is for instance seen when comparing Xilinx UltraScale and UltraScale+ devices, like the KU11P and KU060, where the KU11P have more MGT I/O-pins, but less system logic cells than the KU060 [48, 49].

Historically, regular FPGA I/O-pins have been quite simple, with a certain set of hard primitives that allowed for the support of multiple standards, optional termination, and some input/output delay. For instance, the Xilinx Spartan 6 and Altera Cyclone V, released around 2010, supported LVDS signals up to a maximum of 1 Gb/s and 840 Mb/s, respectively [50]. With the emergence of faster memory devices like DDR4 and the need for more memory on FPGA applications, among other things, it was clear that the FPGA vendors needed to provide I/Os with improved performance.

The pCT detector readout electronics must be able to handle many high-speed links, one for each sensor, 108 high-speed links in total per layer. With 2 tracking layers and 41 calorimeter layers, the total amount of high-speed links is 4644. The cost of an FPGA with up to 108 MGTs would be roughly five times the acceptable cost of the pRU. Fortunately, with new and modern FPGAs, the regular I/O-pins have become more powerful, with more features, and most importantly for the pCT-application, an increase in the maximum bandwidth.

From Xilinx 7-series and later, the maximum LVDS bandwidth of the non-MGT I/O-pins is 1.25 Gb/s[1], thus the ALPIDE data rate of 1.2 Gb/s can be handled by regular I/O-pins. However, using regular I/Os for high-speed data sampling requires logic to deal with phase and word alignment, which are usually performed by the MGT primitives. Also, we cannot expect the same performance as the MGT in terms of bit error rate (BER). This is because MGTs include several utilities that can enhance the quality of the incoming

[1] With some variations depending on speed-grade and voltage supply.

signal, which regular I/Os are lacking. In Chapter 5 we discuss in detail how it is possible to reliably sample high-speed data using regular I/O-pins.

The chosen Kintex FPGA has a total of 264 high-performance differential I/O-pairs [49]. Thus, less than 50 % of the available pins of the device are used for the FEE data interconnections. The remaining I/O-pins are therefore free to be used for a variety of different purposes. Also, by not employing the MGT-pins for FEE interconnection, all 56 MGTs are available for run control and data offload.

4.3 Resource Planning

As the pRU is a fully custom PCB it is possible to add more than one FPGA to the board. By employing more than one FPGA we can opt for the use of a smaller and cheaper device. Furthermore, more than one FPGA would allow the use of MGT for high-speed data sampling. However, the use of more than one FPGA also complicates the design.

(1) Run control communication is required between the FPGAs and the control room. By using more than one FPGA device, it becomes necessary to either add a separate communication link for each FPGA or to develop a communication interface between the FPGAs. Either way, this complicates the design.

(2) The data offload of the FPGAs involves the same aspect, each FPGA must have at least one communication link with the control room. Using QSFP as the physical layer of this connection, however, makes it possible to have at least one independent 10Gb Ethernet (10GbE) link for each FPGA. The data rate simulations shown in Section 5.4 indicate that more than one 10GbE interface might be exaggerating the offloading requirements.

(3) The readout electronics must be fully synchronous. Adding more devices adds additional uncertainty and complexity to the synchronization process.

Because of the successful use of regular I/Os for data sampling (see Section 5.3), in addition to the above-mentioned points, it was decided to opt for the use of a single FPGA for the pRU.

4.4 Clocking Strategy

The pRUs are designed to function as either semi-autonomous with free-running clocks or as fully synchronous components. In the semi-autonomous mode, the pRUs only uses dedicated clock oscillators placed on the board to generate all clock frequencies and allow the clocks to drift between the components. In the fully synchronous mode, a clock is shared from either a master pRU to all other slave pRUs or from a dedicated clock distribution card. This concept is further discussed in Section 6.5.

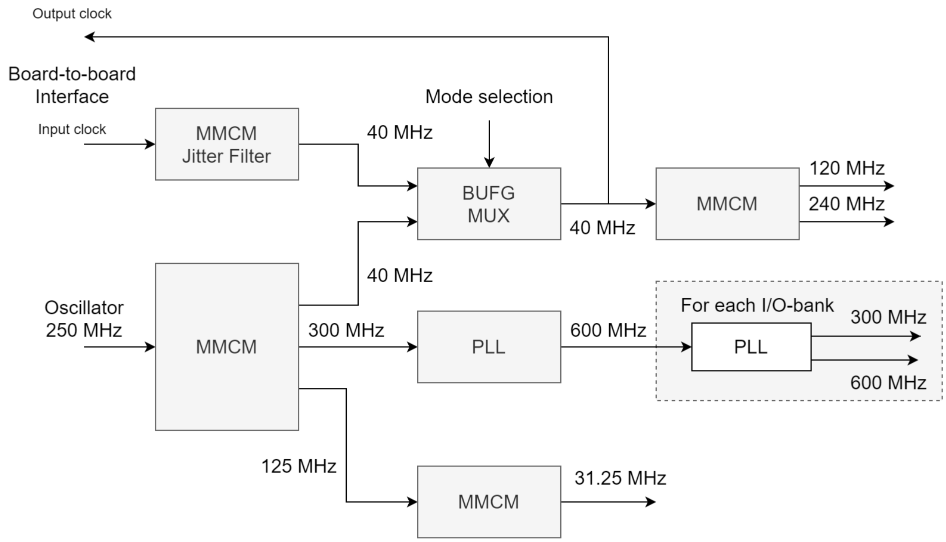


Figure 4.1: *Overview of the pRU FPGA clock network.*

To aid these two modes, the clock network shown in Figure 4.1 is used. All pRUs have an oscillator supplying a 250 MHz clock. This clock is used to generate the clocks used by the various blocks of the firmware. In addition to the 250 MHz clock, the pRUs must have two MGT reference clocks to support the Gigabit Ethernet (GbE) and 10GbE functionality of 625 MHz and 156.25 MHz, respectively. Listed below are the oscillator frequencies that supply the pRU FPGA.

250 MHz The main clock that is used as the source for most of the firmware modules.

625 MHz Provided by an external PHY used by the IPBus stack. Is connected to the MAC layer and is used as the SGMII communication clock (see Section 5.7.1).

156.25 MHz Reference clock for the MGT of the 10GbE link. Is also used by the UDP stack and the data transfer protocol block.

Regardless of whether the pRU is configured to use the local or external clock, the primary mixed-mode clock manager (MMCM) will generate a minimal set of clock frequencies that are needed for remote access to the pRU. This includes the clock used to access the global registers of the pRU, allowing to configure at all times which input clock to use. It is also possible to view the status of the external clock's MMCM, indicating whether the distributed clock can be used. Note that it is impossible to select the external clock as a source if the MMCM is unlocked. The second MMCM acts as a jitter cleaner of the external clock and ensures that the clock is phase-locked to the input clock. The MMCM instance requires that the input clock has a maximum input period jitter of 20 % or 1 ns [51].

Note that the use of the distributed clock signal is optional. By default, the pRU will employ the locally generated clock. This ensures that the pRU will be fully operational at startup and that all registers are accessible. The use of the BUFGMUX primitive ensures a glitch-free switch between the clocks. The 40 MHz clock, either the locally generated or the distributed, is used to generate the 120 MHz clock utilized for data processing, and the 240 MHz clock used for oversampling the ALPIDE slow control signal. The MMCM that generates these clocks is configured to keep these clocks phase-aligned.

The 125 MHz clock is used for MDIO auto-negotiation with the GbE PHY and is also used to generate the master bus interface clock of 31.25 MHz. Notice that the 600 MHz clock, which is used for the high-speed data sampling, is generated via the 300 MHz clock made by the primary MMCM. This is done for several reasons. First, the primary MMCM cannot create a 600 MHz directly in the combination of the other clocks. Also, as stated in [52], PLLs are the recommended clock source for the I/O-bank native primitives (see Section 5.3.2.1). Furthermore, it is important to minimize the sampling clock jitter. It is therefore imperative that it is generated from the local oscillator, and not the distributed clock, as the distributed clock might introduce more jitter, and thus degrading the data sampling. For each I/O-bank another PLL is used to generate the primary sampling clocks. This is done to minimize the fanout and further optimize the sampling clock quality.

4.4.1 Clock Domain Crossings

A clock domain crossing (CDC) is the transfer of data from a flop driven by one clock to a flop driven by another clock. This transfer might result in a metastable state, a signal that do not assume a stable value of 0 or 1 [53]. Digital designers are often advised to avoid or minimize the number of CDCs in a design. However, specific techniques can be used to minimize the risks of detrimental consequences of CDCs.

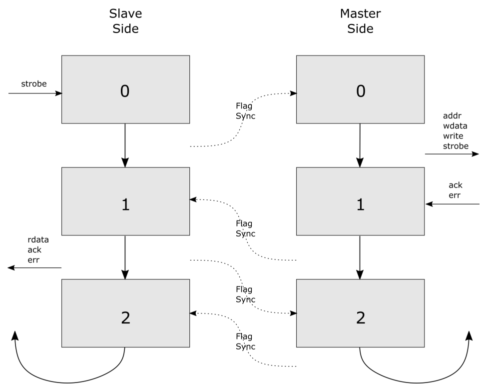


Figure 4.2: *The IPBus CDC synchronizer operation.*

The modules of the firmware operate in different clock domains. E.g., the modules responsible for the slow control communication with the ALPIDEs operate at 40 MHz. The data processing module, however, operates at 120 MHz. All modules are connected to the common bus interface master, which is in the clock domain 31.25 MHz. One way to solve this could be to have CDCs inside each module, i.e., for each register to synchronize between the domains with free-standing CDC synchronizers. However, since some modules have many registers, and because the ALPIDE Data module (see Section 5.6) is duplicated many times, it means that a large amount of logic would be needed to implement all the safe crossings. As even a simple two-stage synchronizer will double the number of registers used, it is important to reduce the number of CDCs to save resources. The method chosen to solve the problem is to implement CDC synchronizers within the bus datapath, as described below.

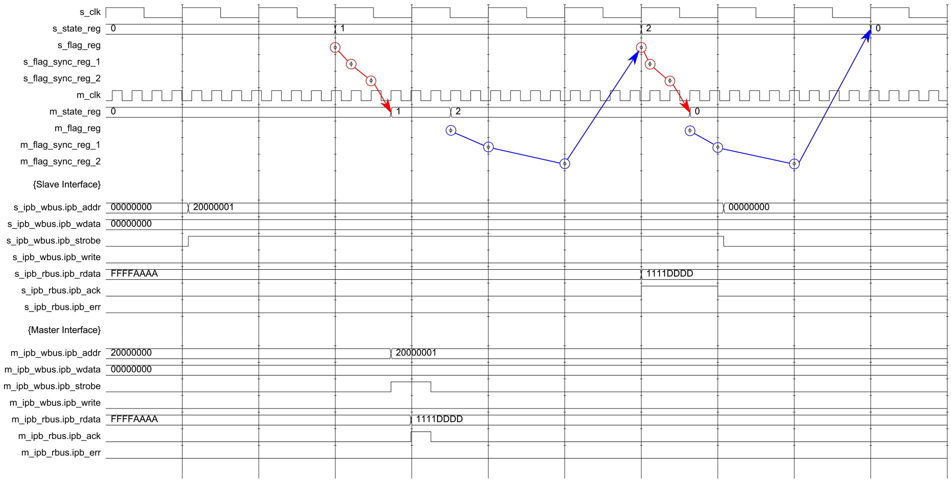


Figure 4.3: *IPBus synchronizer waveform of CDC between* 31.25 MHz *and* 120 MHz. *The red circles are indicating the synchronization flag from the slave to the master side, and the blue circles vice versa. A total of nine clock cycles are required for each operation.*

As data is transferred both to and from the slave, the synchronizer has both a master and a slave block. Figure 4.2 shows the synchronizer operation. Two finite-state machines (FSM) interact via two state-flags that are transferred via the two clock domains with a conventional two-stage multi-flop synchronizer. This method is called a handshake synchronization or a multi-cycle path formulation with feedback and solves the potential problem of data incoherence since the multi-bit data is guaranteed to be stable at the time of transfer [53, 54]. Figure 4.3 presents the synchronizer waveform. Each operation requires a total of nine clock cycles of the slowest clock.

Along the main data path of the firmware, there are several CDCs. Using a handshake synchronizer on the datapath would be inefficient and would cause data loss as the process requires several clock cycles to complete the transfer of one word. For this reason, asynchronous FIFO synchronization is used instead. This is a common method to transfer continuously changing data from one clock domain to another clock domain without causing any metastability problems [55]. This technique is usually safe by design if the write rate is lower than the read rate, guaranteeing no buffer overflows [53].

4.5 Resource Utilization

Obviously, the size of the FPGA, i.e., the amount of resources available on the device, is crucial for any firmware design. This includes the number of flip-flops, LUTs, memory blocks, MGTs and clock resources like MMCMs and PLLs.

Table 4.1: *Resource utilization overview of the pRU firmware on the Xilinx Kintex KU085. The bottom line shows the available resources on the FPGA.*

	Slice LUTs		**Slice Registers**		**Block RAM**	
Block: Data Sampling	243	0.05 %	204	0.1 %	0	
Accumulated - 108 Ch.	26446	5.3 %	22666	2.3 %	0	
Block: Data processing	1401	0.3 %	1096	0.1 %	5	0.3 %
Accumulated - 108 Ch.	149591	30 %	118368	11.9 %	540	33 %
Priority Offloader	7299	1.5 %	3223	0.3 %	59	0.3 %
pDTP & 10GbE	5774	1.2 %	5492	0.6 %	11.5	0.7 %
ALPIDE Control	2687	0.5 %	2922	0.3 %	0	
Trigger manager	764	1.5 %	657	0.1 %	0	
Global registers	68	0.1 %	115	0.1 %	0	
IPbus infrastructure	5489	1.1 %	5548	0.6 %	16.5	1 %
Top-level	198221	40 %	159272	16 %	627	38.7 %
KU085	**497520**		**995040**		**1620**	

The modular design of the pRU firmware aided the determination of the resource quantity needed for the complete design. By investigating each block of the design, we can establish an estimate of the resources needed when up-scaling the design by simply multiplying the resource in question. As most of the resources are used by the data sampling and processing blocks of the firmware, these were the most important blocks to investigate. In Table 4.1 the accumulated resource utilization is shown, as well as the contribution of the various blocks. Notice that the data processing block is by far the most resource-demanding block of the firmware. This is because each channel, i.e., each ALPIDE, requires independent data processing and that the resource usage scales with the number of channels. Furthermore, increasing the number of data channels also complicates the

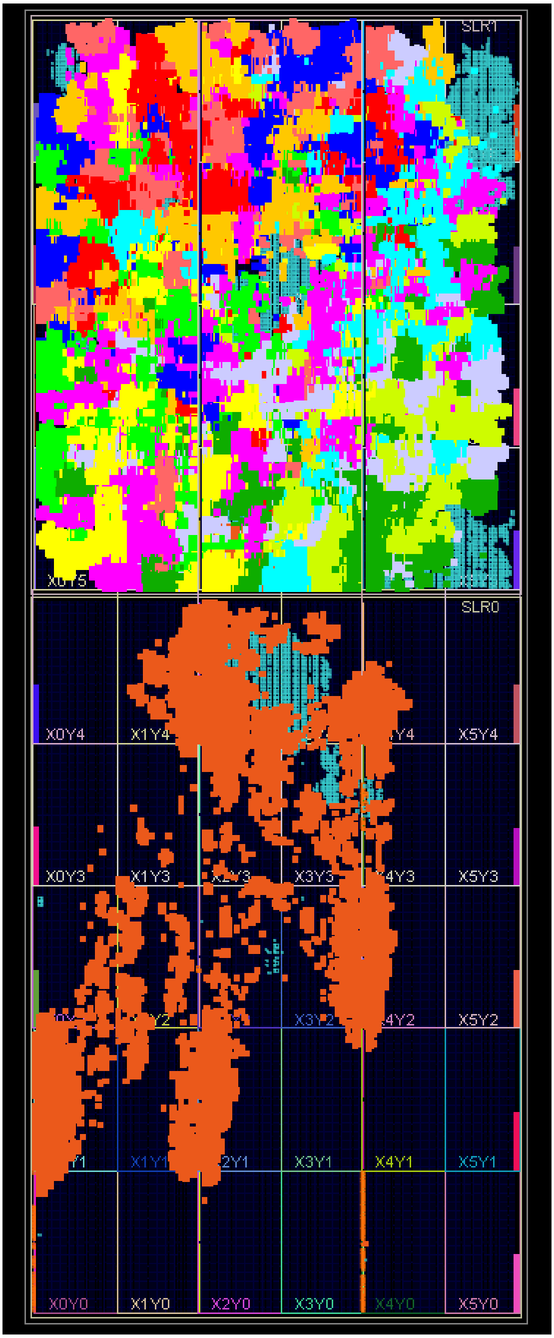


Figure 4.4: *FPGA resource utilization. The data processing blocks occupy most of the resources. The input stage, with dynamic phase alignment, is found in the bottom part of the FPGA, near the input pins, shown in maroon color. The data blocks for each data channel are shown in alternating colors at the top of the FPGA.*

multiplexing and priority offloader architecture. The full implementation of the current state of the pRU FPGA firmware is shown in Figure 4.4. Note also, however, that there is still a significant amount of free resources available, allowing further upgrades of the firmware.

A major weakness in the resource estimation is the uncertainty concerning the buffer sizes needed for the data path of the design (see Section 5.6.5). However, with the current block RAM utilization of 38.7 %, there is an option to increase the buffer sizes further if required. Furthermore, another option is to utilize unused LUTs as distributed RAM.

4.6 Radiation Environment of the pCT

One of the most challenging aspects of designing electronics for particle physics experiments is ionizing radiation. The radiation is obviously a necessary part of the system since it is what the detector is supposed to measure. Nevertheless, this radiation is also potentially harmful to the electronics of the system.

There are two specific kinds of radiation effects that one must be aware of while designing the electronics of such a system, (1) single-event effects caused by a single particle, and (2) TID, the cumulative effects over the lifetime of the system. Several types of single-event effects exist, one of the most damaging is the single event latch-up (SEL), which can cause physical damage to the transistors of the device. However, often, SEUs are the major problem for the electronics of particle physics experiments. An SEU is caused by a charged particle losing energy in the material of the microelectronic circuit, and sometimes flipping a bit of a memory cell. This bit flip can cause either corrupted data or, in the case of FPGAs, cause the device to change behavior. Such bit flips are usually considered soft errors because the error can be removed by rewriting the memory location that was affected by the particle. There are multiple techniques to mitigate the effects of SEUs. However, by avoiding the use of these techniques, we can reduce the utilization of costly resources on the FPGAs of the system or the cost of extra components for mitigation.

With the help of simulations of the radiation environment, several observations and decisions can be made. Figure 4.5 shows a simulation of the radiation of the area of the detector during a pCT projection and illustrates the high energy hadron (HEH) fluence top-down. The plot is normalized, and the hadron fluence must therefore be multiplied by the number of protons used in a projection.

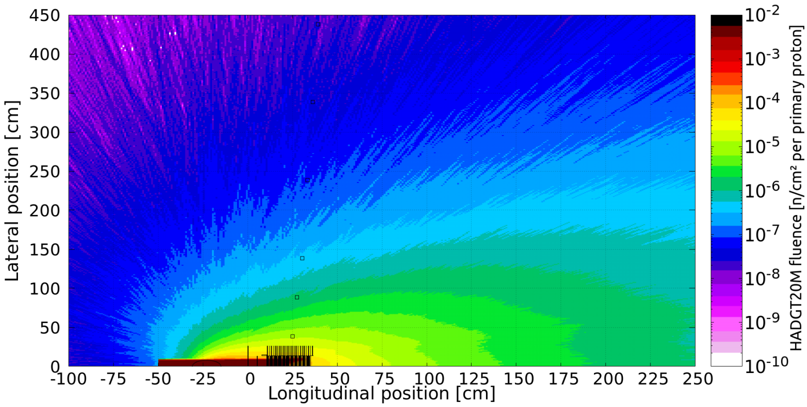


Figure 4.5: *Simulation of the HEH fluence during proton imaging. The black lines at the bottom of the plot illustrate the detector layers. The simulation was provided by courtesy of Jarle Sølie.*

The SEU rate on an FPGA can be calculated based on the HEH fluence together with the information of the FPGA device itself. The cross-section of an SEU, i.e., the probability that a HEH will cause an SEU, is 1.89×10^{-15} cm^2/bit on the chosen pRU FPGA [56]. Equation 4.1 is used to approximate the rate of SEUs, where σ_{SEU} the SEU cross-section of the FPGA, Φ_{HEH} the fluence of >20MeV hadrons, and N_{bit} is the amount of configuration memory bits [57].

$$N_{SEU} = \sigma_{SEU} \times \Phi_{HEH} \times N_{bit} \tag{4.1}$$

The expected SEU rates for both pCT and proton therapy are presented in Table 4.2. The calculations are considering continuous projections with a flux of 10^7 particles/s. The HEH rate was simulated at several points outside of the DTC[2], to find the sufficient distance where one could safely place the readout electronics rack. With these calculations, it is evident that a distance of 2 meters

[2]The distance is relative to the edge of the transition card.

is a safe distance, causing a single SEU roughly only every 45 min. Generally, the average number of SEUs needed to cause a functional failure, the single event upset probability impact (SEUPI), is dependent on the configuration memory usage. However, it is generally assumed that a SEUPI value of 10 is a conservative number [57, 58]. This means that only every tenth SEU will cause a functional error. Therefore, it was concluded that advanced mitigation techniques like TMR would exceed the reliability requirement and be too costly for the pCT readout electronics. However, to avoid the accumulation of errors in the configuration memory of the FPGA over time, some form of scrubbing must be implemented. The use of the built-in Xilinx SEM IP would be preferable to the use of the custom scrubbing methods. Custom scrubbing requires the utilization of separate electronics components, exemplified by the ITS readout electronics [59].

Table 4.2: *Expected FPGA SEU rate and life-time based on HEH rate. A safety factor of 10 is included and all 43 FPGAs are considered in the FPGA SEU rate. A conservative TID limit of 1000 Gy for the FPGA is used. The table is provided by courtesy of Jarle Sølie and previously published in [17].*

		Proton CT	**Proton therapy**
	Distance from DTC [cm]	Time [s]	Time [s]
1 SEU	10	20.4	8.0
	50	197.2	66.2
	100	689.7	224.7
	200	2932.6	1022.5
	300	6410.3	2415.5
	400	11286.7	4329.0
	Distance from DTC [cm]	Life-time [s]	Life-time [s]
TID	10	$1.40 \cdot 10^{10}$	$1.30 \cdot 10^{10}$
	50	$1.23 \cdot 10^{11}$	$7.66 \cdot 10^{10}$
	100	$6.02 \cdot 10^{11}$	$2.89 \cdot 10^{11}$
	200	$4.12 \cdot 10^{12}$	$1.14 \cdot 10^{12}$
	300	$6.45 \cdot 10^{12}$	$2.35 \cdot 10^{12}$
	400	$1.43 \cdot 10^{13}$	$3.91 \cdot 10^{12}$

4.6.1 Radiation Mitigation Resources

Although SRAM-based FPGAs are susceptible to ionizing radiation, we saw in Section 4.6 that by placing the pRU at a certain distance from the beam-center, we can expect a relatively low rate of SEUs. Thus, the current firmware design does not employ any type of radiation mitigation. By avoiding the use of mitigation techniques like TMR, a lot of resources are free to be used elsewhere. It is, however, possible to implement different levels of TMR as long as there are available resources. TMR is, in principle, just a technique to improve our confidence in a certain logic state by having multiple copies of critical nodes [60].

There are multiple types of TMR schemes and topologies [61]. Local TMR (LTMR) is the simplest form, and entails a triplication of all registers. Each register triplication is followed by a voter deciding the correct value based on a simple majority. Block TMR (BTMR) is when an advanced function is triplicated in its entirety, with voters operating on the output of the function, which may be a multibit array. Distributed TMR (DTMR), however, entails the triplication of an entire design, excluding global routes like clocks. According to Berg [61], LTMR should not be employed for SRAM FPGAs, as the results are similar or even worse than non-mitigated systems. BTMR can be used for blocks that are often flushed/reset, as errors might accumulate over time. This makes this type of mitigation applicable for parts of the detector control system (DCS) block, where the block can be reset for each transaction. DTMR is naturally very resource-demanding but is usually the most effective mitigation technique. Berg is critical of TMR that is not implemented with care, as it may actually worsen the system reliability [61].

Not only the type of TMR must be evaluated, but we must also assess which parts of the logic that are critical for the reliability of the system. For instance, it is not possible to triplicate the data path registers and buffers of each data channel of the system. This is evident by referring to Table 4.1 and observing the resources required of each data channel. However, it is more feasible to triplicate the FSM logic controlling the protocol checker (Section 5.6.3) or the data formatter (Section 5.6.5). Note, that since LTMR is unwanted, this type of replication must also include the combinational logic. In any case, we do not consider the actual detector data critical to ensure system reliability and therefore accept the risk of some data corruption. Rather the DCS system is. Naturally, it would make more sense to triplicate the blocks that control sensor communication and synchronization; the ALPIDE Control and the Trigger Manager. As these blocks combined take up a mere 2 % of the LUTs and 0.4 % of the registers, triplicating these blocks seems reasonable.

It is important to note that the pRU firmware is not managing anything that can cause damage to the sensor devices as the power control and the main temperature monitoring of the detector is performed by a separate unit. To conclude, provided that the pCT is not employed in a clinical setting, we can disregard the need for TMR completely.

4.6.2 Radiation and the Data Link

With low radiation, we also can avoid the use of radiation hard offload-links, e.g., the GigaBit Transceiver (GBT) architecture and transmission protocol employed by the ITS readout unit [62]. Note, however, that the GBT can handle single event transients in the photo-diodes of an optical system [63]. As the distance between the pRU and the control room is in the order of meters, optical communication is not required and a copper solution can be employed. Thus, the GBT can be replaced with an interface with higher throughput, which reduces the need for buffering resources on the pRU FPGA.

4.7 Conclusion

Avoiding the use of a large scale TMR implementation and increasing the throughput of the offload data link, leads to a significant resource reduction per channel for the pRU firmware. These are the two main reasons why it is possible to use an FPGA that is only slightly larger than the ITS FPGA and still interface more than ten times as many ALPIDEs.

CHAPTER 5

Detector Data Readout

The pCT detector compromises many high-speed data links and will produce a large amount of data during operation. This chapter describes the challenges that arise from these two facts. The chapter is introduced by discussing methods to reliably sample high-speed data with FPGAs. Specifically, two different methods are discussed, a semi-dynamic and a dynamic phase alignment approach. Both methods are implemented and tested. Furthermore, the detector data rate during pCT operation is discussed. Next, the full data flow of the readout firmware is discussed in detail. This entails a discussion on how the data of each channel is handled and how the data from all channels are combined. Finally, the pRU data offloading method is outlined.

5.1 Data Sampling Methods

As serial data is propagating in a non-perfect medium, several effects cause the signal shape to deteriorate (see Chapter 3). The sampling window is the period where we consider the signal to be stable and adhering to a value within an I/O-standard. For instance, for Xilinx I/Os, an LVDS signal is logical 1 if the differential value is above 100 mV, and logical 0 if below −100 mV. A value between these two definitions will produce an unpredictable logical value if sampled. Figure 5.1 shows an example of how the sampling window size can be reduced by various factors. This is especially critical for static phase alignment (see Section 5.1.1). Recall from Section 2.3.2 how the ALPIDEs on-chip PLL jitter was a source for the reduced sampling window of the ALPIDE high-speed data. Naturally, the receiver's task is to reliably sample data inside the sampling window by aligning to the correct phase and thus distinguishing one bit from another.

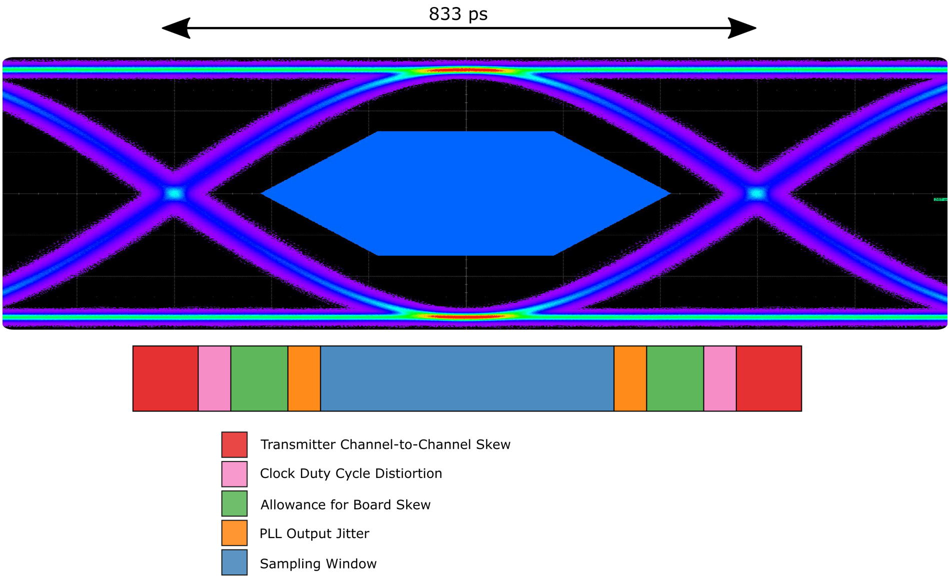


Figure 5.1: *Illustration of an example eye diagram, sampling window and the timing budget of a* 1.2 Gb/s *system. Several components are included as potential sources of noise and reduction of the sampling window as discussed by [64].*

5.1.1 Static Phase Alignment

In low-speed applications, phase alignment can be solved statically. Static phase alignment methods might include calculation of the signal propagation delay and matching the sampling clock to this calculation. Not only is this work labor-intensive in terms of the number of data traces used, but it is also highly unpredictable if one does not perform proper analog simulations of the board. Furthermore, any changes to the board layout, cables, connectors, etc., require reiteration of the delays involved.

5.1.2 Clock Data Recovery

A common method to ensure the correct frequency of the signal is to transmit a clock along-side the signal, especially if there are multiple data links from the same source. The receiver thus gets the correct *frequency* of the signal, but still needs to lock on to the correct *phase* to find the middle of the sampling window. In some cases, the phases of the clock and data signal are assumed to be equal, *source-synchronous*, and thus one simply needs to adjust the clock signal by 90 degrees to place the sampling edges within the sampling window. This adjustment is easily achieved using PLLs which are abundantly available on modern FPGAs. At higher frequencies, it becomes more difficult to obtain a source-synchronous relationship between the transmitter and the receiver, and one starts to consider the system as either *mesochronous* or *asynchronous*. This means that it is necessary to adjust the phase of the sampling clock based on the incoming data, or vice versa. Phase detectors like Hogge and Alexander bang-bang, for instance, can be used to deal with random data streams [65, Chapter 12]. These detectors detect the timing difference between the clock and the data, and the output can be used to adjust the phase of the sampling relative to the sampling window. See Section 5.2 and 5.3 for more details about phase alignment.

5.1.3 Asynchronous Data Recovery

Asynchronous data recovery is becoming the norm in high-speed applications as it gets harder to keep two devices synchronized at the clock rate that they operate. Also, as phase adjustment of the incoming data becomes crucial, the benefit of transmitting a clock alongside the signal is becoming less significant. In asynchronous systems, both the transmitter and the receiver agree on a certain transmission frequency, and two clocks of nearly the same frequency operate in separate domains. If done correctly, the variation of frequency between these two clocks is countered by a phase adjustment of the sampling clock to the phase of the incoming data.

5.2 Semi-Dynamic Phase Alignment

An early prototype of the pRU, employing a Xilinx 7-series FPGA, was intended to test the approach of using regular I/Os for data recovery [66]. In this approach we opted to test the potential of using the input delay primitives of the regular I/Os and use an initial stage to search for the 8B/10B comma-words to obtain a phase lock. Thus, this approach is semi-dynamic, because a re-lock can be done at given intervals. The approach uses a clock of 600 MHz and double data rate (DDR) sampling. Figure 5.2 illustrates the concept of input delay taps, and how by adjusting the delay tap, one can move the data to align the middle of the sampling window to the sampling clock edges. Small increments and decrements of the delay tap can be performed without glitches in the data stream [67, p. 123].

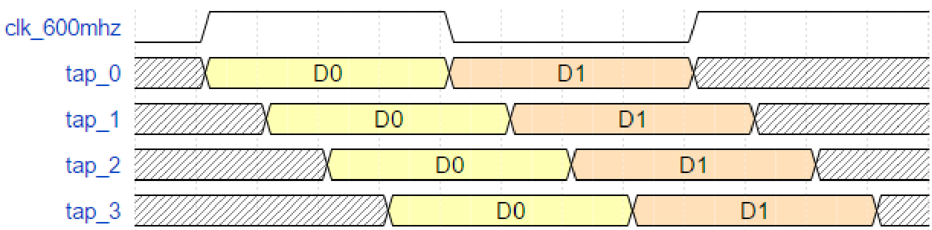


Figure 5.2: *Illustration of the concept of data delay chain [66]. The data in delay tap_2 will be sampled more reliably than the data in the other taps, as the clock edges are placed in the middle of the sampling window.*

Figure 5.3 shows the block diagram of the approach. The input stage, i.e., IDELAYE2, ISERDESE2 and IDELAYCTRL, are primitives provided by the Xilinx FPGA architecture. All other building blocks are developed by the author, except the 8B/10B decoder[1]. When the ALPIDEs high-speed link is enabled, it continuously outputs 8B/10B comma-words. IDELAYE2 provides a 32-tap programmable input delay with calibrated resolution. The Phase Aligner block checks if a comma is observed (without errors) for each delay tap setting for a given number of samples. The result of this brute force method is a 32-bit word, where each bit indicates if the setting resulted in a successfully sampled comma word. As the settings which provide the correct result would

[1]The 8B/10B decoder utilized is a combinational design by Chuck Benz [68].

normally be next to each other, the delay tap chosen is the middle of the longest consecutive row of *valid* taps. When the Phase Aligner finishes the initial search phase, it indicates that the 8b10b Decoder can start the decoding of each 10-bit word. The Phase Aligner, 8b10b Decoder and the Bitslip block will co-operate to align the comma-word. This method proved to be quite promising when testing with the ALPIDE's PRBS testing pattern [66].

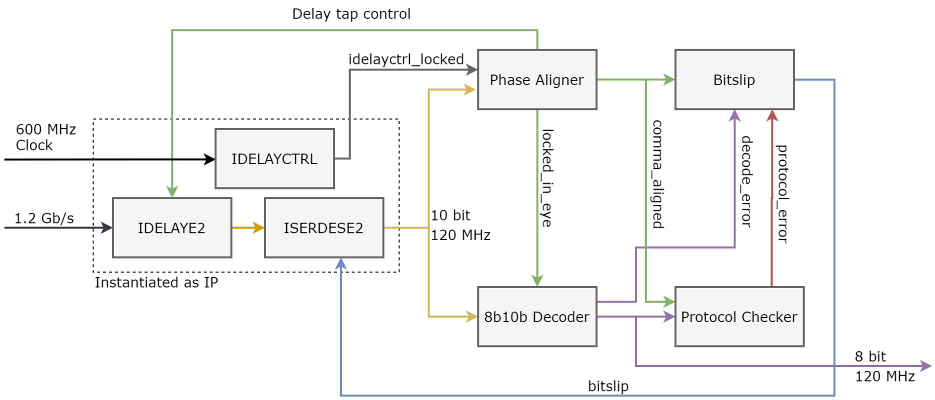


Figure 5.3: *Block diagram of the semi-dynamic phase and word alignment method [66].*

Unfortunately, although the approach was successful in sampling PRBS-data, severe issues with the method was discovered during actual data-taking, i.e., sampling data following triggering of the ALPIDE with a radiation source. As discussed in Section 3.2.4.3, triggering of the ALPIDE causes voltage fluctuations of the digital supply, and in turn, jitter on the high-speed link. Also, another contributing factor to the poor results is that the 7-series FPGA only provides 32 delay taps, giving a coarse resolution of roughly 26 ps delay per tap [66]. This is more than double the worst-case and twelve times more than the best-case input delay tap coarseness of the Ultrascale devices (see Section 5.3.2.1). Despite further efforts trying to re-lock during data-taking, the jitter was too severe to be handled by the phase lock approach.

Although the method ultimately proved unsuccessful, the results were promising regarding the use of regular I/Os for the data capture. Therefore, upgrading to a Xilinx Ultrascale device to increase the granularity of the input delay (increase to 512 taps), was the logical next step. Also, because of the change of the internal primitives of the I/O-blocks from the 7-series to Ultrascale, the semi-dynamic phase alignment method was abandoned for dynamic phase alignment (see Section 5.3).

5.3 Dynamic Phase Alignment

Dynamic phase alignment (DPA) is a technique that was developed to address inadequacies of static phase alignment methods, see for example [64]. The technique involves continuously updating the phase of the sampling clock relative to the phase of the incoming signal, solving the problem of drift and, if fast enough, corrects for jitter. One important advantage of DPA, aside from solving the problem of asynchronous data recovery, is that each channel is independent of each other, and thus makes the problem of trace length mismatches between channels obsolete. Instead of using a clock recovery circuit, with DPA one assumes that the frequency of the on-chip clock closely resembles that of the data frequency. As the difference between these two frequencies can be interpreted as drift, the DPA-process will correct for it.

Because DPA is a form of asynchronous data recovery, i.e. the incoming signal is independent of any clock signal, it requires that the data transition density[2] is sufficient to reliably detect the phase difference between the signal and the sampling clock. Therefore, the DPA-method needs coded data. Since the ALPIDE employs 8B/10B encoding, this is not an issue in the pCT application. Note that 8B/10B has a transition density that varies between 0.3 and 1.0, while the $K28.5$ comma-word, which is used in the initial critical stage of the alignment, has a transition density of 0.5. Also, a PRBS sequence can be used with DPA if sequence entails frequent transitions. The ALPIDE employs a PRBS-7 test pattern with a transition density comparable to that of 8B/10B encoded data.

[2]The ratio of transitions/edges to the number of unit intervals in a serial data stream.

5.3.1 DPA Implementation

The following section describes the implementation of a DPA approach for data recovery of the ALPIDE high-speed links using a Xilinx Ultrascale(+) FPGA[3]. To detect the difference between the phase of the signal and the sampling clock, the approach is exploiting the possibility to delay the p- and n-side of the differential signal differently and by comparing the sampled values. By delaying them by a half unit interval (UI)[4] relative to the each other, we gain the following two effects:
(1) a DDR oversampling without using a faster clock, meaning two samples per unit interval.
(2) by comparing the values of the two signals, an indication of whether the primary signal is sampled in the middle of the sampling window. One can use this information to purposely move the sampling point of the active signal to the middle of the sampling window by incrementing or decrementing the delay of both channels equally. This is the same operation as to move the sample clock edges relative to the incoming signal.

Notice that the signal comparison is only valuable if the consecutive bits of the transmission differs, so that it is possible to determine whether the two samples are done within the same sampling window. This is aided by 8B/10B-encoding the signal. Furthermore, the implementation is performing the comparison of the signal after a SerDes. Because the SerDes outputs 4-bits at the time, the comparison process can be performed on the 4 middle bits of a series of 8 bits. That means that for each bit pair that is compared, one also has access to the next bit in the transmission. If there is no change between two consecutive bits, the delay remains unchanged. However, by basing a decision on 4 bits at the time, chances are that at least one of the bit pairs involve a transition.

To illustrate this, two signal samples in a sequence are compared. The *active* is the signal used as data, while *monitor* is the extra signal used to determine the sample quality. Both p and n can be used as the active signal. Imagine that active[n] and monitor[n] are equal, and active[n+1] also matches these

[3]The principle and method is derived from [52].

[4]In this case, the unit interval coincides with the bit period of the data stream.

values. In this case, no change of the delay will be done as we cannot determine whether the monitor[n] was sampled at the sampling window of either active[n] or active[n+1]. In the case where active[n+1] does not match, however, the delay for both channels will be decreased. This can be observed in the upper case of Figure 5.4, where the sampling edges *a* and *b* occur within the sampling window of the same bit. Decreasing the delay will move the signal to the left relative to the clock phase, moving the sampling edge *a* closer to the middle of the data eye, thus improving the sample of the active signal for the next sample.

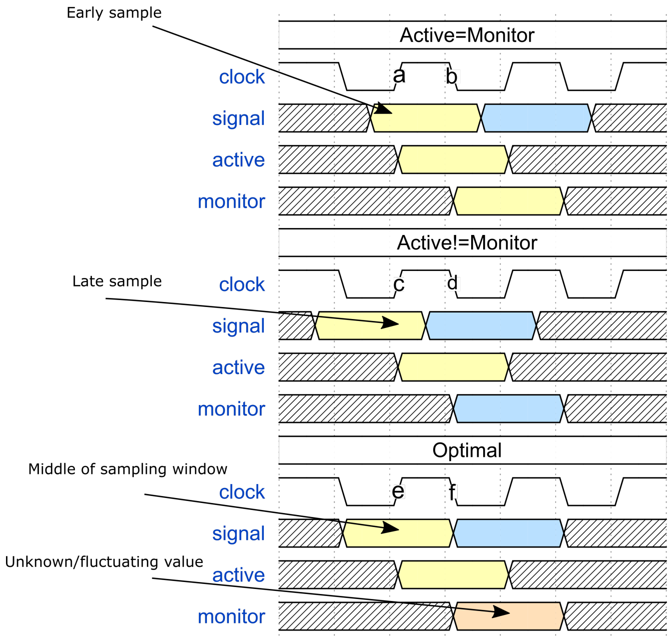


Figure 5.4: *Wave diagram illustrating how the DPA application is using input delay and comparing the two samples of a differential signal to move the sample point. For illustration purposes the two signals are sampled at different clock transitions; the active signal is sampled at the rising edge and the monitor signal is sampled at the falling transition. When the active signal matches the monitor signal, the input delay is decreased. When the active signal does not match the monitor signal, the input delay is increased. This continuous update of the input delay keeps the sampling edge in the middle of the window.*

In the case where active[n] and monitor[n] do not match, however, and also, active[n] and active[n+1] do not match, one will increase the delay of both the signals. This will move the signal to the left. As seen in the middle of Figure 5.4, this will move the sampling edge c closer to the middle of the sampling window. At the bottom of the figure, the optimal situation where the sampling edge e is exactly in the middle of the window is illustrated. The case is not stable because of jitter, noise and that the value of the monitor signal is unknown, making the active signal sample point fluctuate around the middle of the eye. See Table 5.1 for an overview of all delay actions based on the signal values.

Table 5.1: *An overview of the executed delay action based on the values of the active and monitor signals. Notice that the values of two consecutive active bits must differ to provoke a delay change.*

Active[n]	Monitor[n]	Active[n+1]	Delay
0	0	0	No change
0	0	1	Decrease
0	1	0	No change
0	1	1	Increase
1	0	0	Increase
1	0	1	No change
1	1	0	Decrease
1	1	1	No change

5.3.2 High-Performance I/O

The comparison of two values of a single differential signal at two different sampling times are made possible with the newest I/O-features of the Xilinx Ultrascale FPGA. According to [69] the Xilinx Ultrascale I/O-banks are differentiated in three groups; (1) high-performance (HP), (2) high-range (HR) and (3) high-density (HD) where the HP-banks are designed to meet performance requirements of several high-speed interfaces. Thus, in the following discussion, we are referring to HP-bank I/O-pins, and the built-in features the I/O-bank architecture. Each bank consists of 52 I/O-pins, whereas 48 can be used as a differential pair with another, i.e. 24 differential pairs in a bank. The I/Os are further divided into 4 subgroups called bytes, and further divided into an upper and a bottom nibble. See the layout of a byte in Figure 5.5. Each bank has an MMCM and two PLLs at disposal to create clocks that are used for sampling of data and other household tasks of the input stage.

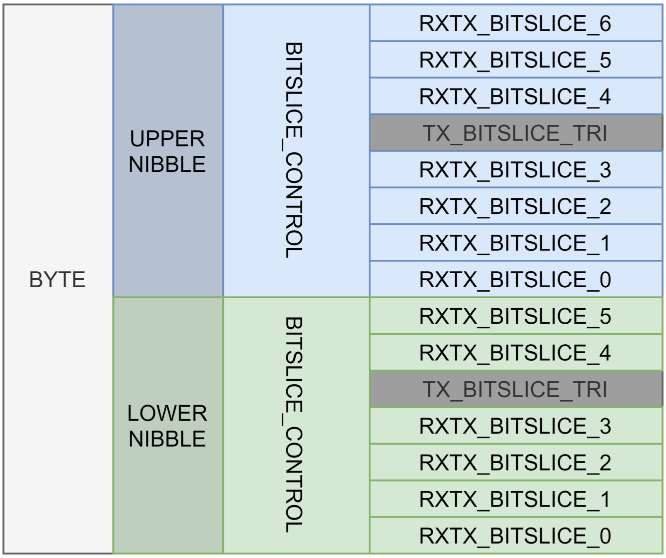


Figure 5.5: *Layout of a byte in an I/O-bank [52]. Each bank contains four byte blocks. Each byte contains two nibbles, where each nibble comprises BITSLICE primitives to handle up to three differential I/Os.*

Like with most FPGA I/Os, the HP I/Os can be used with a range for I/O-standards, most importantly LVDS, with or without internal 100 Ω termination. Furthermore, LVDS can be used with DC- or AC-coupling. DC-coupling requires that the driver keeps the common-mode voltage stable within in the LVDS-standard. AC-coupling allows for changing the common-mode voltage at the input stage, either via an external biasing circuit or internally on the FPGA. According to [70] AC-coupling is recommended only if the signal is DC-balanced as AC-coupling elicits baseline-wandering in non-balanced signals in high-speed applications. Equally, DC-coupling is only recommended if the signal is non-balanced or if a wide bandwidth is required [71]. As the ALPIDE transmits an 8B/10B-encoded signal, we will employ AC-coupling for the pRU application. Also, AC-coupling is required to employ the equalization features of the I/O-pin to counteract the attenuation of the transmission line [72]. Equalization can be set to five different values based on the strength required to rebuild the signal. Unfortunately, only limited information about how the equalization is implemented is provided by Xilinx [69], and therefore the level of equalization must be selected based on performance tests.

5.3.2.1 I/O Primitives

For the Ultrascale FPGA family, the I/O-primitives are divided into component and native mode primitives. Component mode resembles the I/O-primitives of the older 7-series FPGA family, while native mode is new for the UltraScale architecture. The component mode primitives are built from the native mode components, and thus, have less functionality. Furthermore, these primitives also lack some of the features that exist in the 7-series FPGAs. The native mode primitives, however, are more complex to use and require that the design abides by a comprehensive set of rules.

As seen in [72, Table 23,24], native mode exceeds component mode when it comes to maximum bandwidth, whereas native mode can achieve a bandwidth up to 1600 Mb/s for synchronous data capture, and up to 1300 Mb/s for asynchronous data capture. The native mode primitives are employed by the DPA reference project, as the primitives provide features required for this purpose [52].

Each I/O incorporates various native mode primitives. The RX_BITSLICE is a special primitive that handles receiving data. This primitive is used together with BITSLICE_CONTROL. Both primitives' functionalities are managed via attributes during instantiation. A RX_BITSLICE primitive contains four blocks, as seen in the diagram in Figure 5.6. The first block controls how much the signal is delayed before sampling. The signal can be sampled and registered at any of the 512 delay taps, controlled by the BITSLICE_CONTROL. The time resolution of each delay tap varies with temperature, voltage, and process variation, but is between 2.1 ps and 12 ps [72, Table 35]. This is significantly less coarse than the 7-series delay resolution. Note that the receiver/sampling clocks are also distributed from BITSLICE_CONTROL. Further, a SerDes block deserializes the incoming data. A shallow FIFO is also included. All these features are included in the hard fabric of the I/O, meaning that a negligible amount of soft fabric resources are needed to implement these primitives. In the Ultrascale devices the output of both RX_BITSLICE-primitives of a differential pair can be accessed. Consequently, the same signal can be sampled at two different times by delaying the primitives differently.

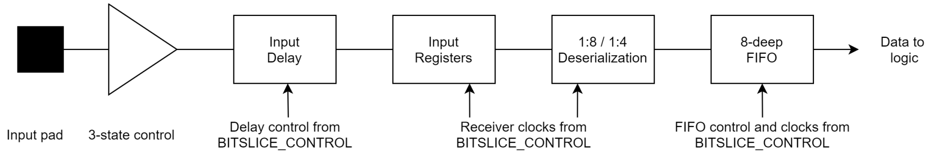


Figure 5.6: *Block diagram of the RX_BITSLICE based on Figure 2-31 in [69].*

5.3.3 DPA Sequence

The DPA sequence implemented in the pRU firmware is based on reference project in [52] and is divided into the following steps:

- The transmitter starts transmitting 8B/10B-encoded data. At this stage, this must only be comma words and not actual data.
- The master and slave channels (p and n) of the data are captured by separate RX_BITSLICEs via an IBUFDS_DIFF_OUT-primitive, seen in Figure 5.7.
- The PLL-generated 600 MHz clock supplied to the BITSLICE_CONTROL which is used to sample the data has an unknown phase relationship with the incoming data.
- The finite-state machine (FSM) shown in Figure 5.7 starts a calibration sequence of the I/O-primitives. During this process, the BITSLICE_CONTROL registers are accessed via an RIU-interface to obtain the number of delay taps that constitute a delay equivalent to the UI of the data transmission rate. E.g., the UI of the 1.2 Gb/s data link is 833 ps. Thus, the calibration sequence determines how many delay taps this is equivalent to.
- Based on the master channel it is determined whether there is data activity and if the following sequence can be initiated.
- When data activity is detected, both the master and slave data are delayed incrementally while the captured data are compared. If the data remain equal while gradually incrementing the delay a .5 UI, the p-side is selected as the active signal and is considered *in front* of the n-side. However, if

the data is unequal during the incrementing, we expect that the sample point has been moved through a transition. In this case, the n-side is selected as the active signal and the p-side is *behind* the n-side.

- The delay is modified so that the active signal is sampled a half UI before the monitoring signal.
- From this point onward, the FSM continuously compares the active and the monitor signal and keeps the relative input delay between them constant. Any changes to the delay are based on the values of Table 5.1.

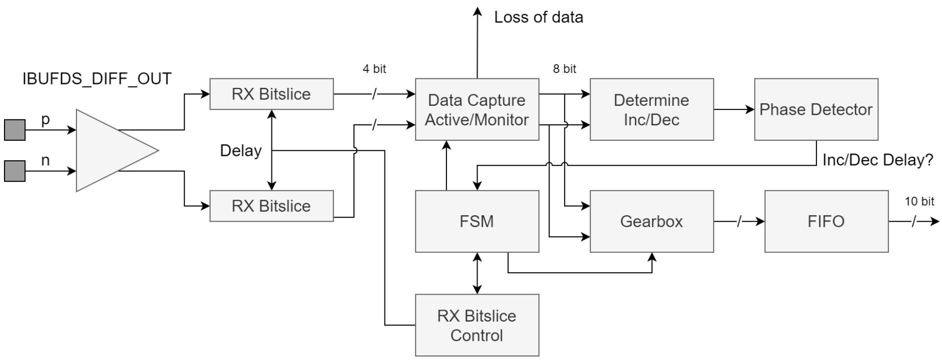


Figure 5.7: *Block diagram of the alignment and tracking circuit.*

The DPA logic is built with four components, (1) *data capture*, (2) the *comparator*, (3) the *phase detector* and (4) the *FSM*. The *data capture* block registers the 4-bit words from both RX_BITSLICEs into either the active or the monitor 8-bit registers. The block also determines whether there is a loss of data. The values of these registers are used by the *comparator* block, which is strictly combinational and produces a result according to Table 5.1. Note that the block compares 4 bits each clock cycle and considers how many of the bit-comparisons that are pointing in the same direction. E.g., if two of the bit comparisons point to increase the delay, and the other two point to decrease it, the next stage, the *phase detector*, will ignore the result, and no delay change will be performed. Note that the bit-comparison only measures the polarity, and not the size, of the phase error, so the type of phase-detection resembles an Alexander bang-bang implementation. However, as the comparator measures multiple bits at the same time, the size of the timing error can be extrapolated by the number of bits pointing in the same direction.

Consequently, the *phase detector's* task is to make sure that a given number of bits point in the same direction, and prevent any oscillation of the sampling point if the phase error is minimal. Although this is a pseudo-linear phase detector because we can determine whether the phase error is large by observing how many of the bits within the same 4-bit sequence points in the same direction, any movement of the delay is done with a fixed interval. The reason for this is simply that a shift of 8 delay taps is the maximum that can be done without risking a glitch in the sampled data [69, Chapter 2]. Because the average delay is in the order of ps/tap, eight taps constitute a very small portion of the 1.2 Gb/s UI. Any smaller steps are not needed. However, this maximum step-size every clock cycle of the 300 MHz DPA clock, might limit the response to sizable jitter.

The FSM of Figure 5.7 controls the whole operation from the initial sequence searching for any transition, to keeping the sampling point stable in the middle of the window. Also, the FSM needs to be aware of if the clock drift, the small frequency differences between the transmitter and the receiver, makes it necessary to wrap around the 512 available delay taps. This will cause a glitch-less swap between the active and the monitor signal. See Section 7.4.2 for the results of the DPA implementation.

5.3.4 Block Architecture

Several boards have been used during the design process of the pRU and for production testing (see Section 7.5). To streamline the firmware design process for the various FPGAs, an attribute-driven architecture for the input stage was designed, shown in Figure 5.8. As the usage of the native-mode components of the I/O-bank increased the threshold of knowledge about the placement rules and the required instantiations[5], the code block was required to abstract away the complexity of these rules.

The architecture is based upon the layout of an I/O-bank, so the designer is still required to know the location of each of the I/O-pins used. The top-level

[5] For instance, the usage of a nibble without employing the lowest I/O-pair still requires the instantiation of the RX_BITSLICE, or the place and route process will fail.

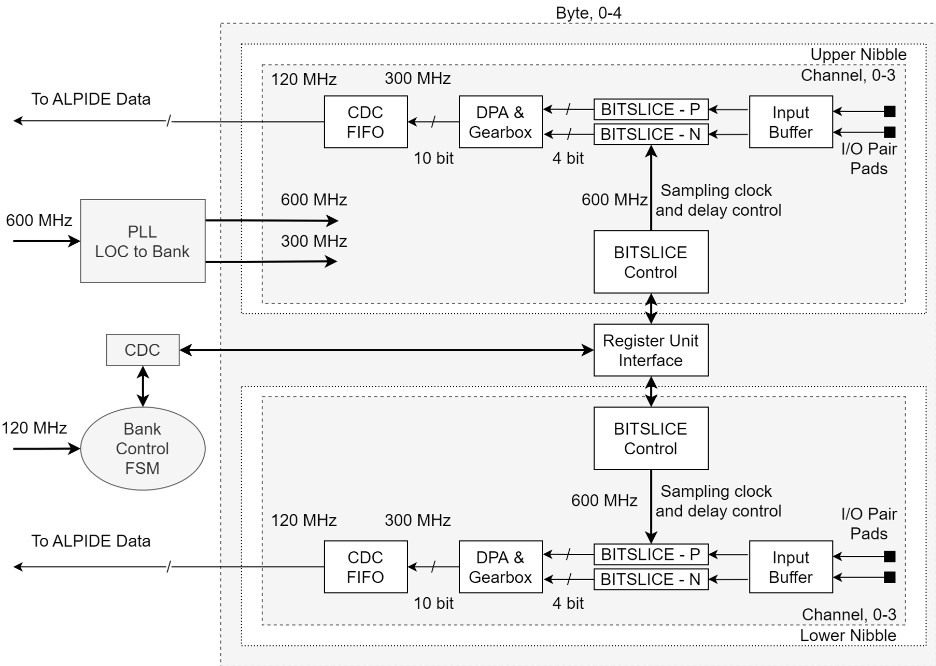


Figure 5.8: *Simplified block diagram of the input stage architecture.*

of the architecture may use several I/O banks, and all the required routing to and from the next level of the data flow is done here. Figure 5.8, however, shows the instantiation of an I/O-bank. In the code instantiation of the bank, we can indicate which channels to be used, and this enables the generic code to adhere to the placement rules.

Each bank requires the supply of two different clocks, the system 120 MHz clock[6], which is used for the bank-FSM and for clock domain crossing (CDC), and the DDR sampling clock of 600 MHz. However, the rules of the *native* primitives prohibit the sharing of the sampling clock between I/O-banks. Therefore, each bank instantiates a PLL that distributes the clock to each BITSLICE_CONTROL primitive in all nibbles used. Furthermore, a 300 MHz clock is created by the same PLL. This is the clock the DPA block operates at, as the SerDes outputs 4 bits every clock cycle. The PLL is locked to a certain physical position in the bank. This is done in the placement constraint file

[6]Based on the ALPIDE data transmission rate.

of the design. Note also that the entire bank must be associated with a fixed physical block, *pblock*, to adhere to the Xilinx rules.

The bank block includes a FSM that resets and calibrates the I/O-primitives. Each I/O-channel is associated with a *bit value*, i.e. the number of delay taps required to delay the signal with a full UI based on the sampling clock. The value might be different depending on temperature, voltage, and process variations. The FSM extracts this value from the BITSLICE_CONTROL via the RIU interface during the calibration stage and conveys the information to the DPA block of the corresponding channel. The value is important for the DPA to determine how to delay the two samples of the differential signal a half UI relative to each other and to know whether one of the signal delays has wrapped around a full UI.

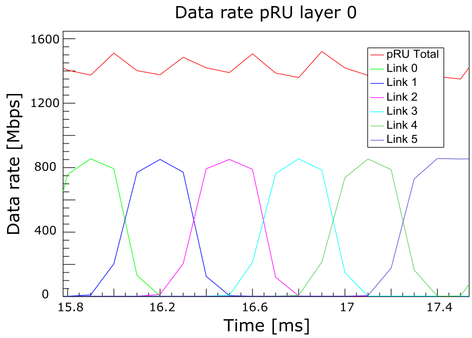
An unused byte or nibble will not be instantiated, but an unused bitslice might be forced to be instantiated depending on whether any other bitslice is used in the same nibble. For byte blocks with both nibbles instantiated, a RIU_OR primitive is instantiated. This is a native-specific primitive that demultiplexes a common RIU-signal to two BITSLICE_CONTROL primitives.

For every used channel an IBUFDS_DIFF_OUT-primitive is instantiated along with a BITSLICE_RX-pair, a DPA-block, the gearbox[7] and a FIFO. The data from the DPA and gearbox are stored in a FIFO. This FIFO is always read out if data exist, and its only purpose is to perform CDC from the 300 MHz to the 120 MHz domain that is used later in the data flow. Note that the data stored in the FIFO is neither comma-aligned nor 8B/10B decoded.

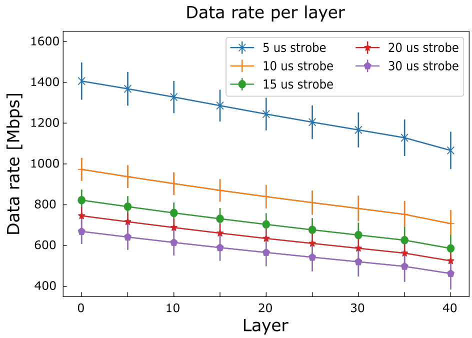
5.4 DTC Data Rates

Although a beam with a flux of 10^7 particles/s is considered a low-intensity beam, this rate of particles will produce a large amount of data in each layer of the DTC. Figure 5.9 shows a Monte Carlo simulation of a fast scanning proton beam using a System C-model to anticipate how much data is generated and

[7]The gearbox transforms the 4-bit word output of the DPA into a 10-bit word.



(a) *Data rate for the first layer with a trigger rate of 5 µs with data rates of a few selected data links.*



(b) *Total data rate per layer. The strobe length is the data taking window. A minimal gap of 25 ns is used between each strobe.*

Figure 5.9: *Monte Carlo simulation of data rates of a 230 MeV proton scanning beam with an intensity of* $10^7\ s^{-1}$*. The beam scans over the detector plane in 65 ms. The simulation result and figures are provided by courtesy of Simon Voigt Nesbø and were first published in [73].*

.

offloaded of the sensors. With a trigger rate of 5 µs, considered to be the highest rate to be used with the detector, each sensor will produce an actual data rate of up to 900 Mb/s for a brief period [73]. In Figure 5.9b, the accumulated data rate of each layer is given. Notice that the data rate varies with the strobe window length and that the average data rate will peak at roughly 1.4 Gb/s. Furthermore, the aggregated data for each layer is relatively stable. At the same time, the individual sensors will see a data burst as the beam is gradually scanning over the detector area. This fact can be exploited in the design of the back-end electronics, as discussed in Section 5.6.

The data rate and the distribution of that data rate of each sensor, and in turn, each layer, is dependent on many variables. First, the rate varies significantly with the intensity of the beam used, i.e., the number of particles per second. The beam particle also affects the data rate as different types of particles affect the degree of secondaries and the mean cluster size. Furthermore, background radiation and noise are probably contributing somewhat to the data rate. Moreover, to accurately predict the data rates, one must also consider the analog characteristics of the sensor pixels, the threshold of the digital front-end, and the bias voltage. It is difficult to find an exact value of the data rate before the complete detector is built and characterized using different settings. For this reason, and because the simulations inevitably have shortcomings[8], the electronics must be designed to handle a more substantial data rate than shown by the Monte Carlo-simulation.

5.5 ALPIDE Data Protocol

The data transferred from the ALPIDE are generated based on each strobe window. All pixel hits that occur during a window are combined in a data *frame*, of which the address of each pixel hit is the data. See Figure 5.10 for an example data frame. The frame itself is encapsulated by a chip header and a trailer, which contains the chip ID, time information, and various flags. Pixels hit that belong to the same chip region are grouped together with a region header preceding them. A single-pixel hit is represented by a 16-bit data word, data short, consisting of the two-dimensional address of the pixel within the region. A special case is reserved for pixel hits that are geometrically close to each other. In this case, the address of the pixel with the lowest address is combined with a 7-bit hit-map field that indicates a cluster shape relative to the aforementioned pixel. This case is indicated with the data long word. This clustering feature can be turned off, but it is generally a good idea to leave it enabled to compress the data payload. This is especially true when the substrate bias is set to 0 V, minimizing the depletion region of the sensing diode, which in turn will increase the mean cluster size.

Figure 5.10: *Wavediagram of an example ALPIDE data frame. In this particular frame only pixels within one region has been hit.*

[8]For instance, not including noisy pixels and background radiation.

5.6 Data Flow

The front-end architecture of the pRU firmware only takes care of the data sampling. Figure 5.11 illustrates the next steps of the data flow chain. First, the data must be comma-aligned, meaning that the first bit of the word must be determined. Furthermore, to make sense of the content of the data, it must be decoded. These two steps are outlined in Section 5.6.1. The reason why the data must be decoded on the FPGA before transmitting it to the server farm is that the data must be processed further in order to (1) reduce the payload by removing redundant data words, (2) filter out data, e.g., empty frames, (3) detect errors, and (4) efficiently pack data to reduce the throughput requirements of the network.

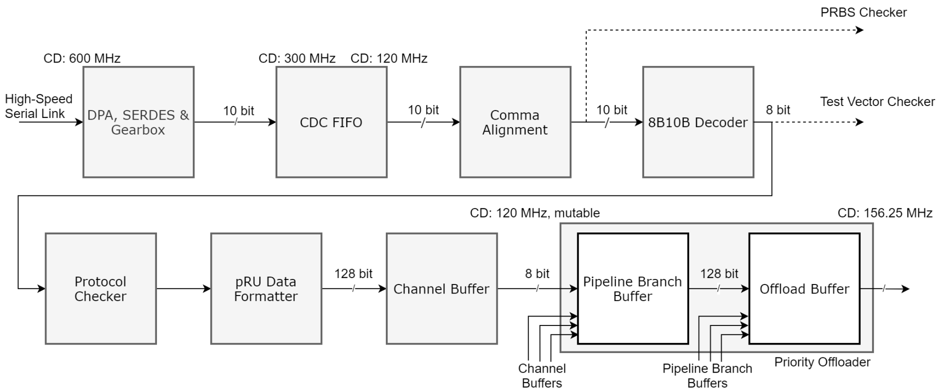


Figure 5.11: *Block diagram of the data flow of the pRU firmware.*

The ALPIDE, as explained in Section 5.5, transmits data in a specific format with a given protocol. The first step after the decoding is, therefore, to check whether the data adhere to this protocol and to check for other types of errors, as seen in Section 5.6.3. A special data format is constructed to ensure that all required information is relayed to the data server. The format itself, and the packing mechanism, are explained in Section 5.6.4.

All the previously mentioned blocks are part of the ALPIDE Data firmware module. This is, in addition to the bank blocks of the front-end architecture, the only major FPGA block that is duplicated. Each I/O-channel is associated with a separate ALPIDE Data module that is connected to the DPA logic and the CDC FIFO.

As seen in Figure 5.11, however, the data flow does not end with the buffering. The data must also be offloaded from the FPGA. The next step is therefore a priority offloader for multiplexing the data from each channel, see Section 5.6.6. When the data are combined, they can be transmitted to the data farm, see Section 5.7

5.6.1 Word Alignment and Decoding

The data-taking process on the FPGA is initiated by the ALPIDE Data module. By enabling the module via the bus interface, the front-end channel reset is de-asserted and the DPA process is initiated. The front-end is indicating that data are sampled successfully by deasserting the loss of signal and FIFO empty flags, signifying that data words are available to the rest of the chain.

Word alignment of 8B/10B-encoded data is in principle a trivial task. However, adding custom untested code to any project might introduce errors. Therefore, the pRU employs a block of System Verilog code originating from the ALICE ITS firmware for this task. A 20-bit shift register shifts the 10-bit word from the front-end FIFO every clock cycle. A combinational block searches over the 10 possible positions for either the positive or negative polarity of the K28.5 comma-word. If found, the position of the comma is stored. A clocked process checks if the position of the comma remains stable, and a counter keeps track of this. After 255 clock cycles of a stable position of the comma, the block raises a flag that indicates that word alignment has succeeded, and continuously outputs 10 bits from the shift register starting from the comma position.

Immediately after completion of the word alignment, the next block, the 8B/10B decoder, is enabled. The decoding block adopted is an IP core developed by Chuck Benz [68], only slightly modified to support enable/disable. The operation of this block is purely combinational. Based on the 10-bit input word and the current running disparity[9], the block outputs the 8-bit decoded word, the resulting disparity, as well as an indication of code error or disparity error.

[9]The running disparity is the difference between the number of 0s and 1s transmitted. As an 8B/10B code is DC-balanced, the running disparity must be 0 for 20 consecutive bits.

Additionally, the block indicates whether the symbol decoded was a control character or not. For example, the K28.5 comma-word is decoded as $0xBC$. Without the indication that this is a control character, this word could be interpreted as an ALPIDE chip trailer word with two of the error flags set. To avoid this confusion, the control character indication flag is forwarded to the protocol checker along with the 8-bit decoded data and the error flag signals.

5.6.2 PRBS Checker

One block of ALPIDE Data is using data that is neither word-aligned nor decoded; the PRBS check block. This block is essentially a Xilinx developed block, PRBS_ANY, which can be instantiated with a specific polynomial length and tap. Note that using the PRBS checker disables and resets the word alignment and decoder blocks. The output of PRBS checker block is monitored, and the errors are counted in a readable register.

This functionality is critical for testing (1) the quality of the high-speed link electronics, (2) whether the ALPIDE is bonded correctly and is alive. It is used both in the testing of the front-end electronics and the testing of the DPA-functionality of the FPGA.

5.6.3 Protocol and Error Checking

Although the 8B/10B decoder reliably detects 1-bit errors, there are no guarantees that multi-bit errors can't occur in the ALPIDE data stream. However, with the ALPIDE adhering to a strict data format protocol, it is possible to detect some of these errors by checking for protocol violations. The data are routed directly from the 8B/10B decoder to the protocol checker block. A principle sketch of the FSM for the checking sequence is given in Figure 5.12. A valid ALPIDE data frame will always require a chip header word, but there are a couple of exceptions. E.g., an empty frame, i.e. a frame where no pixels are fired during the strobe window, causes the transmission of a chip empty frame data word. Also, either busy on or off are allowed outside of a frame. However, all other data words are strictly forbidden. The protocol checker does not only check for and count errors and violations of the ALPIDE data format, it also indicates to the data-formatter block how to pack its event (see Section 5.6.5).

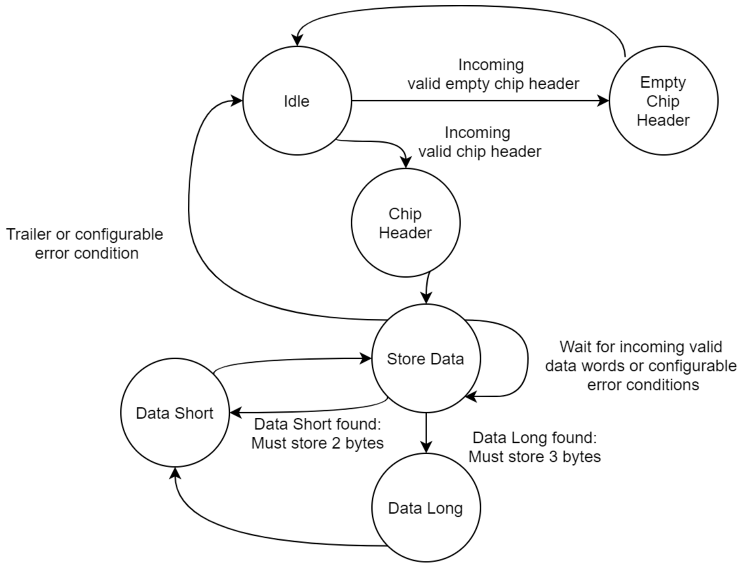


Figure 5.12: *Simplified state machine diagram of the protocol checker.*

The validity of a chip header depends on several configurable parameters. Most important is that the 8B10B decoder does not report any error from decoding the first byte of the word. However, the protocol checker can be configured to also confirm that the chip ID number is correct. This check can prevent so-called fake frames to be processed. These are frames that occur because a word with multi-bit flips erroneously resembles a chip header word. See Section 7.4.2 for an example of a fake frame captured on the FPGA. Note from Figure 5.12 that when the protocol checker detects a valid chip header the state machine will not return to the idle state until it determines that the frame is completely processed. However, to ensure that the FSM won't be locked in a frame either by a fake chip header or by failing to detect a chip trailer, several error checks can be configured. E.g., the protocol checker can check for a maximum number of idle bytes between each data word or a maximum number of bytes within a frame. Also, words that cannot be interpreted by the protocol checker can cause the FSM to abort the frame processing. Making these parameters configurable makes the firmware flexible and ensures that it can handle various situations.

Keep in mind that, if there are multi-bit errors coincidentally located within the pixel address bits of a data word, they can be impossible to detect[10]. The result will be corrupt data that will be a source of noise in the data analysis. Therefore, it is imperative to reduce the errors already at the electrical level of the interaction, as discussed in detail in Chapter 3.

5.6.4 The pRU Data Format

The primary task of the DAQ system is to provide sufficient information from the detector so that track reconstruction can be performed on the data server. However, to minimize the throughput requirements, one will also strive to send the minimum amount of data necessary. For track reconstruction, one needs a series of 4-dimensional data points, i.e. the pixel hits of a single particle as it traverses through the detector; x, y, z, and T[11]. As discussed previously, a single particle hit can cause several pixels to fire on the ALPIDE, and it is necessary to transmit this complete information further in order to analyze the cluster size for higher energy resolution, i.e. all pixel hits must be redirected further off the pRU. Each pixel hit that occurs during the same strobe window, are combined in a data *frame*. This frame is tagged with timing information from the ALPIDE, the bunch counter, as well as an ID that indicates which chip on the string is transmitting the data. Note that it is possible for a single particle to cause pixel hits within multiple strobe windows. This is due to the intrinsic separation of the analog front-end domain of the pixel and the digital domain, and the relatively long shaping time of the analog amplifier. Also, a short gap between each strobe window will contribute to the number of double hits, but a longer gap may cause loss of hit information. However, these double pixel hits will occur in consecutive strobe windows and are usually easily identified by a simple software algorithm. They are therefore not a problem as long as the FPGA buffers and the network data throughput are not saturated.

[10] Depending on whether the word passes the 8B10B checker.

[11] T represents the time of the pixel hit.

Evidently, the ALPIDE data format is missing information needed for a successful track reconstruction in the pCT detector. First, the format is lacking information about which string the ALPIDE in question is located. Thus, where on the detector layer the pixel is located, is a vital part of the full 2D position of the pixel in question. Furthermore, it is also missing which *layer* the chip is located on, providing the full 3D position of the pixel. That is why it is necessary to define a data format for transmitting data from the pRU.

At the firmware level, it is often complicated and resource-costly to extract information from several sources at the same time. Especially, if there is no guarantee that the data arrive at the same time since this makes it necessary to temporarily store the data. The nature of the ALPIDE on-chip data packing causes this exact behavior. Usually, the number of pixels hit on the various ALPIDEs during the same strobe window will differ, causing varying processing and transmission times of the data packets. Also, the actual arrival time of the start of the data packet will often differ slightly from channel to channel, as the ALPIDE data packing process might vary based on the number of pixels and amount of clustering, and because the previous data packet might not have been completely offloaded when a trigger occurs on the chip. To avoid having the different channels wait for each other and in turn cause buffer overflow as new data is arriving, it is easier to encapsulate data rather than to unpack and combine the data of multiple channels.

The pRU data format is constructed by using certain pre-defined words, each word consisting of 128 bits. 5 types of words are defined: (1) HEADER, (2) DATA, (3) TRAILER, (4) EMPTY and (5) DELIMITER. All words, except the DELIMITER word, have a 16-bit preamble that contains the following information: (1) the word type, (2) the pRU ID, (3) the string ID, and (4) the chip ID. This will naturally cause an overhead in the packaging which increases the throughput requirements but improves efficiency in the latter parts of the data flow where the data channels are multiplexed (see Section 5.6.6 for a detailed discussion). However, this means that each word has a 112-bit content field, in which the word-specific information is contained. The general format of these words is illustrated in Table 5.2. The DELIMITER word is defined solely to be used during the development of the pRU and will be removed in the final firmware, and will thus not be discussed further.

Table 5.2: *The general pRU data format.*

Name	WORD TYPE	RU ID	STAVE ID	CHIP ID	CONTENT
Length	2	6	4	4	112
Bits	127:126	125:120	119:116	115:112	111:0

WORD TYPE Determines the type of the pRU word.

- **0x0** `DATA`
- **0x1** `HEADER`
- **0x2** `TRAILER`
- **0x3** `EMPTY` or `DELIMITER`

RU ID Identification of which specific readout unit the data originated from.

STAVE ID Identification of which specific stave the data originated from.

CHIP ID Identification of which specific ALPIDE chip the data originated from.

CONTENT Either collection of ALPIDE data or pRU tag data.

5.6.4.1 pRU Packet/Frame

Each ALPIDE frame that contains pixel hit information will produce a pRU data packet, also called a pRU frame. The packet contains all the pixel hit data from a single ALPIDE produced by a single trigger. The pRU packet is strictly defined as follows:

- A single HEADER word.
- One or more DATA words. The number of data words allowed in a packet can be limited by a configurable amount. The reason for limiting the number of data words is to avoid forwarding data from a chip that only transmits noise. Data that exceed the limit are discarded.
- A single TRAILER word.

Note also, that a single pixel hit on an ALPIDE will cause a minimum transmission of 3×128 *bits*, 8 times the minimal number of bits transferred from the ALPIDE; 8×6 *bits*. However, we assume that this is rarely the case since particle hits will normally trigger multiple pixels and, therefore, this will not cause a significant increase in the actual data throughput rates. Furthermore, the word size of 128 bits was chosen to reduce the number of times the 16-bit preamble needs to be transmitted.

5.6.4.2 The HEADER Word

The 14-byte content field of the HEADER word contains information that is available to the formatter immediately after receiving the 2-byte chip header word, thus it can be created while processing the remaining part of the ALPIDE data frame. The structure of the HEADER is shown in Table 5.3. The specific fields of the HEADER are listed in Appendix B.

5.6.4.3 The DATA Word

The DATA word contains the ALPIDE data words in the order that they arrive, i.e., the most significant byte first. All words of an ALPIDE frame are included except the comma and idle words that might appear during frame processing. Based on register settings of the pRU, the user can specify other words that should be filtered out as well. This might be useful during development. Note that words that appear outside of an ALPIDE frame, for instance, chip empty frame and possibly, busy on and off, will never appear in a pRU packet, and thus never in a DATA word. If the ALPIDE frame does not completely fill up the last DATA word in the packet, the remaining bytes are padded with $0xFF$, which is identical to the idle word, and thus cannot be mistaken for real data.

Note, however, that because of the irregular sizes of the ALPIDE data words, some words might be split between two pRU DATA words. The parsing software must take care of these cases. Note also that the DATA interpretation order is critical to avoid data corruption. Therefore, the remaining data chain must take proper care not to scramble the words originating from a channel, meaning that the order of the words must not change.

Table 5.3: *pRU* `HEADER` *Word.*

Name	WORD_TYPE	RU	STAVE	CHIPID	DATA_FORMAT	UNUSED	BUSY_ON	BUSY_OFF	SPILL_ID	TRIG_SOURCE	MODE	FRAME_ID	ABS_TIME
Length	2	6	4	4	8	19	1	1	16	2	1	32	32
Bits	127:126	125:120	119:116	115:112	111:104	103:85	84	83	82:67	66:65	64	63:32	31:0
Value	0x1				0x1	0x0							120 MHz Clock

Table 5.4: *pRU* `DATA` *Word.*

Name	WORD_TYPE	RU	STAVE	CHIPID	DATA
Length	2	6	4	4	112
Bits	127:126	125:120	119:116	115:112	111:0
Value	0x0				

Table 5.5: *pRU* `TRAILER` *Word.*

Name	WORD_TYPE	RU	STAVE	CHIPID	UNUSED	ERROR_FLAGS	FRAME_ID	FRAME_SIZE
Length	2	6	4	4	40	8	32	32
Bits	127:126	125:120	119:116	115:112	111:72	71:64	63:32	31:0
Value	0x2					0x0		

Table 5.6: *pRU* `EMPTY` *Word.*

Name	WORD_TYPE	RU	STAVE	CHIPID	RESERVED	UNUSED	NUM_EMPTY	BUNCH_CNT	SPILL_ID	TRIG_SOURCE	MODE	FRAME_ID	ABS_TIME
Length	2	6	4	4	1	4	16	8	16	2	1	32	32
Bits	127:126	125:120	119:116	115:112	111	110:107	106:91	90:83	82:67	66:65	64	63:32	31:0
Value	0x3				0x0	0x0							120 MHz Clock

5.6.4.4 The TRAILER Word

The TRAILER word marks the end of the frame and guarantees that no more pixel data from the current channel and the current strobe window exist. The word contains several fields that are meant to ensure that the preceding data are not corrupted. See Appendix B for the data fields.

5.6.4.5 The EMPTY Word

Intuitively, the EMPTY word indicates that an ALPIDE has no pixel hits in a strobe window. The word will always appear outside of a pRU frame sequence. However, the ALPIDE chip empty frame word indicates only two things: (1) that a TRIGGER and a subsequent strobe window have been invoked on the chip in question, and (2) at what time the trigger occurred in terms of the bunch counter. This information might be useful to verify that all chips are in sync and that the triggering scheme works as expected. Thus, one might want to transmit all this data from the pRU during an initialization phase or during test runs. However, during data taking, there is no need to transmit the empty data frames from the pRU for further analysis. Therefore, there is an option to compress this data and reduce the total throughput requirements, either by completely filtering out all chip empty frame words or requiring a configurable amount of empty frames to arrive on a channel before transmitting the EMPTY word. These settings are set via the pRU registers.

5.6.5 Data Formatter

The data formatter design is sensitive to the trade-off between a high throughput and the success rate of the FPGA place and route process. The design of the data formatter is deeply connected to the design of the priority offloader and multiplexer of the next stage of the data flow. At a given clock frequency, higher throughput requires more parallel data lines. When we scaled up the system, from 9 to 108 data links, it became evident that some efficiency had to be sacrificed to reduce the design size.

The primary task of the block is to continuously encapsulate the ALPIDE data within the pRU data format and to temporarily store the data in a buffer. The first version of the data formatter pre-formatted all pRU words and stored the data in a 128-bit wide FIFO. This scheme results in the most efficient data flow design, as the throughput could be as high as 15 Gb/s when using the system clock to offload the FIFOs, but became problematic when aiming to scale-up the design to the required number of data channels. The number of routes required to support this scheme caused congestion during place and route, and the process eventually failed to meet the timing requirements.

To reduce the number of routes needed, to avoid congestion, one can reduce the width of the data FIFO. However, a reduction of the FIFO width has two major consequences: (1) the reduction of throughput as less data is transmitted each clock cycle, (2) multiple clock cycles are required to write the words. Using multiple clock cycles to store the HEADER and TRAILER words requires that the block temporarily store the incoming ALPIDE data while the FIFO input is busy. This will increase the resources needed and is inefficient. Fortunately, we can opt for a FIFO with different read and write widths, meaning that we can write the full pRU words in one clock cycle. However, the reduction in throughput cannot be redeemed and must be compensated for in other blocks (see Section 5.6.6).

The data formatter FSM is controlled solely by the flag outputs from the protocol checker. When an ALPIDE header is observed, the process is started to generate a packet. The pRU HEADER is written immediately to the buffer, while the raw ALPIDE data is shifted into a 14-byte wide registers. The value of the register is stored in the buffer if (1) the register is full or (2) an ALPIDE trailer is observed. When the trailer occurs, the pRU TRAILER is stored in the following clock cycle. This is safe only because it is guaranteed that back-to-back data frames, meaning the transmission of an ALPIDE header directly following a trailer, does not occur [74]. A minimum of two idle words are always observed after a trailed word. This fact makes it possible to use an extra clock cycle to store both the DATA and the TRAILER word. In fact, if this was not guaranteed, it might have been necessary to use two separate buffers, one for HEADER and TRAILER words and one for DATA words.

The data buffer is located in separate instance blocks simplifying any change to buffer size and type. This instance also has the option to connect a different clock to the read-side of the FIFOs potentially increasing the throughput. This option is one of the ways one can compensate for the reduced output width.

The data buffer is instantiated as an asynchronous AXI Stream (AXIS) FIFO. The main reason for this is the additional feature of the TLAST signal. This signal can be used to indicate to the subsequent data flow sections that an end of a frame has been reached. Thus, there is no need for the following logic to interpret the data that are transmitted. Also, this buffer type supports the programmable empty and full flags to determine the buffer usage in addition to the traditional empty and full flags. Note that the almost full flag is used instead of the full flag in order to provide one extra clock cycle for the priority offloader to react.

5.6.6 Priority Offloader

As the data from all the channels are temporarily stored in relatively small buffers, an efficient mechanism to combine the data for further transmission must be adopted. A priority offloader block is used for this purpose. It has access to the read-side of all the channel buffers in the firmware and uses the flags of these buffers to determine which ones need to be offloaded first to avoid buffer overflows. Here, the purpose of the 16-bit preamble of the pRU DATA words becomes apparent.

Because the actual data throughput of an ALPIDE is lower than 960 Mb/s[12], the priority offloader can transfer data from a buffer faster than it is filled. This makes it possible, and efficient, to read data from multiple sources in a frame-agnostic way. The priority offloader will switch between the buffers at will, regardless of the pRU frame. Therefore, a scrambling of the data between the different channels will occur. As every word is tagged with its origin in the 16-bit preamble, it is a trivial software task to sort the data for each channel when it arrives on the data server. Note that this scrambling will not change

[12] An ALPIDE frame will contain several idle words in between the actual data words. These redundant words are filtered out of the data stream by the pRU firmware, reducing the actual throughput.

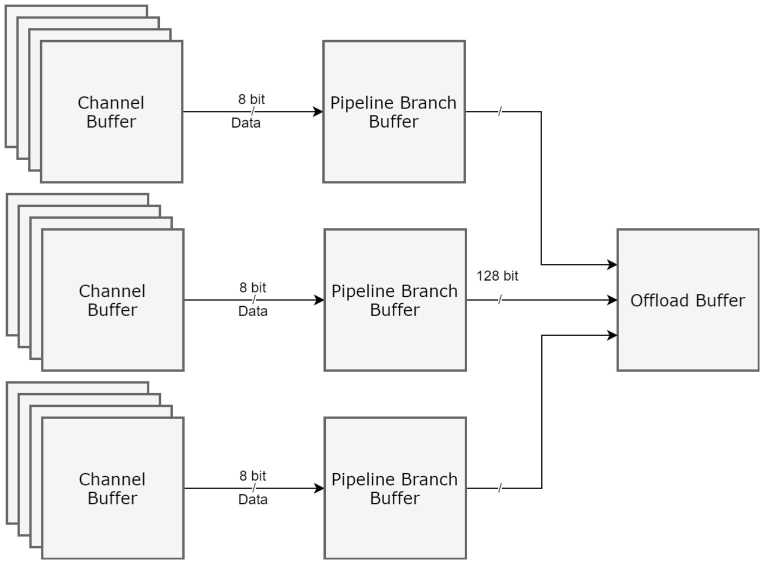


Figure 5.13: *The data pipeline branch flow.*

the order of the words within the pRU frame, and thus will not corrupt the data as warned in Section 5.6.4.3. If desired, this scrambling can be avoided with the sacrifice of efficiency, forcing the priority offloader to read entire frames before switching channel buffers. This is configured via the module registers.

The main method to compensate for the FIFO output width reduction discussed in Section 5.6.5 is to branch out the reading of the buffers in question. This means that we parallelize the extraction of the data into pipeline branches, and for each branch multiplying the throughput capabilities. Figure 5.13 illustrates the concept. Each readout branch is responsible for a fixed set of data channels to read out and to further transmit to a global buffer. Whether there will be only one or several global buffers is still to be determined. As the throughput of a single FIFO matches the theoretical data throughput of a single channel (960 Mb/s), the question remains how many branches that are necessary to avoid buffer overflow. We saw earlier, by the Monte Carlo simulations of the data rates in Section 5.4, that the accumulated data rates of an entire layer are well below 2 Gb/s. Thus, the lower limit required is roughly four branches. Nine branches, operating twelve channels each, provide a maximum throughput of 8.64 Gb/s. The transmission from the branches to the global buffer is completely agnostic to any data format and is completely based on the buffer status in the branches. Furthermore, the transmission from the branches to the global buffers is done with 128-bit words per clock cycle, 15.3 Gb/s.

The branching method can also exploit the fact that data are produced in a known scanning motion. Sensors that lie next to each other can be connected to separate offloading branches, maximizing the throughput to the theoretical limit of the widths and clock rates.

The data buffers of several channels are connected to the pipeline branch logic and then stored in AXIS FIFOs. This simplifies the transmission from the branch buffers to the common global buffer used to combine the data from all channels before offloading the data. Note how the interfaces operate at various clock rates. The interfaces between the channels, branches, and global buffer can be further manipulated to increase the throughput rate by using a higher clock rate.

5.7 Data Offloading

With the number of independent channels, and with the possibility of a random data burst, it is evident that a very fast data offloading scheme is needed to avoid buffer overflow and data loss. With fast offloading, the on-chip buffers for each channel can be reduced to a bare minimum. It is also important that the data offload link is completely separated from the control link to avoid disruptions to the data flow.

With the use of an Ethernet stack, commercial-off-the-shelf components could build up the back-end system of the detector, both simplifying development and reducing costs. 10GbE has become common and many low-cost solutions now exist in terms of network switches and network cards. Furthermore, several 10GbE links can be combined into a 40GbE link if an even higher throughput is required. Thus, several independent 10GbE interfaces can be instantiated, each responsible for offloading its own buffer. UDP was chosen as the transport layer of choice for three main reasons. First, the UDP packet contains a checksum-field, so it is possible to determine if any bits were flipped in the process. This comes in addition to the Ethernet frame check sequence code, so a higher level of reliability is achieved. Second, for software development it is easier to use UDP than raw Ethernet packets, as many software libraries are developed for UDP. Lastly, although UDP is considered unsafe, the use of TCP is ruled out because of the increased complexity needed on the pRU.

5.7.1 10 Gb/s UDP Stack

Although the UDP, IP and the Ethernet protocols are considered simple in terms of software, a somewhat complicated block of logic on the FPGA is required for proper operation. Fortunately, several IP cores exist free of use. An IP core developed by Alex Forenchic was chosen for its simplicity and the provided testbenches that assured correct operation [75]. This core contains a full-stack UDP/IP/Ethernet with example implementations for several development boards, e.g. the VCU118 board used for the development of the pRU. The core also includes full ARP support, allowing the pRU to be agnostic to the addresses of the extended network. The core is complete with blocks adhering to the protocols, as well as implementations of several types of MACs and PHYs, i.e., the link and physical layers of the network interface controller. In the case of 10GbE[13] a PHY can be connected directly to the Xilinx MGT block, and the interface between the PHY and the MAC is XGMII. All other blocks are connected with a 64-bit wide AXIS interface operating on a 156.25 MHz clock, giving a throughput of 10 Gb/s. Furthermore, the in and out interfaces of the block use the AXIS protocol.

Figure 5.14 shows the block diagram of the 10GbE scheme. The complete protocol stack is located within the UDP Complete block. This is connected to the MAC via two helper blocks (RX and TX). These blocks separate the Ethernet payload from the Ethernet header fields. The payload is transmitted as an AXIS, marking the end of the payload with the TLAST indicator. This is also the method employed when transmitting data to and from the UDP Complete block. However, for this block, the UDP header fields are separated from the

[13]The core also included both a 1GbE and a 25GbE implementation.

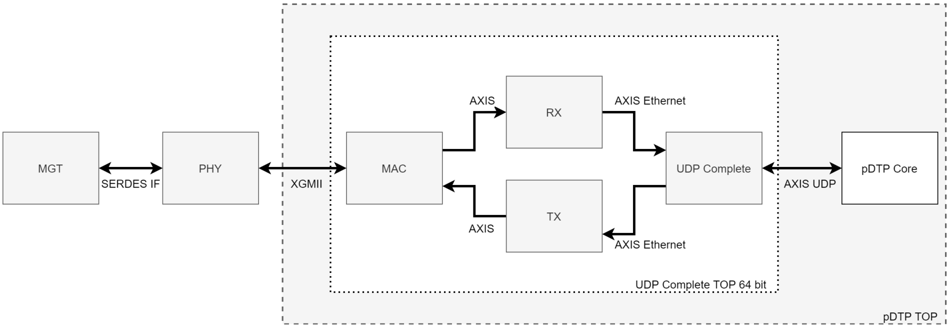


Figure 5.14: *Block Diagram of the 10GbE Scheme.*

UDP payload. The only UDP header field not required when transmitting data is the checksum field, because the block calculates the checksum automatically.

The checksum calculation requires that the block holds the entire payload in a buffer, increasing the resource usage of the block. A large packet increases the efficiency of transmission, thus there is a trade-off between resource usage and efficiency. An Ethernet jumbo frame is defined as a packet with a payload larger than 1500 bytes and usually no larger than 9000 bytes. Because of the diminishing returns of larger packets, the greater risks of data loss with them, and lastly the increased FPGA resource usage associated with them, it was decided that the maximum packet size of the block would be $(2^8 - 1)$ pRU words, which amounts to 4080 bytes. Given the UDP, IP, and Ethernet header fields, this yields a maximum link efficiency of 98 %. Table 5.7 shows the FPGA resource usage of both the full UDP stack and the custom protocol discussed below. Note that the resource utilization is low, leaving the possibility open to increase the number of links per pRU. Such a decision will only require the use of several global data buffers to interface the parallelized pDTP blocks.

Table 5.7: *pDTP and UDP Stack Resource Utilization on Xilinx Kintex KU085.*

	Slice LUTs	**Slice Registers**	**Block RAM Tiles**
pDTP Core	989 (0.2 %)	570 (0.06 %)	3.5 (0.06 %)
UDP Stack	4924 (1 %)	4914 (0.5 %)	11 (0.5 %)

5.7.2 pCT Data Transfer Protocol

UDP packets can be lost without the receiving end being aware of that a packet was transmitted. Also, the transmitter gets no acknowledgment from the receiver that the packet was received correctly. To increase the reliability of UDP one can add a protocol layer on top of it, with features like acknowledgments and resend options. This is the purpose of the pCT Data Transfer Protocol (pDTP). Note that the pDTP does not strive to be a perfectly safe protocol, only to somewhat increase reliability. Therefore, the user has the option of using three modes with different levels of safety. In a lossy network, meaning a network where packets are lost regularly, the safest mode can be utilized with the cost of reducing the throughput. However, if the network is proven to be reliable, higher throughput is achieved with the other modes. The mode selection is done at will, and the various modes can thus be used interchangeably with varying needs.

In the pDTP architecture, the pRU is considered the server. A client sends requests and the pRU responds, as illustrated in Figure 5.15. Two header formats are defined, one for the client and one for the server. The detailed header formats, the opcodes, and fields are shown in Appendix C. The server will mainly transmit data, but can also transmit its status, for instance, the buffer usage, firmware build details, and the current system clock time.

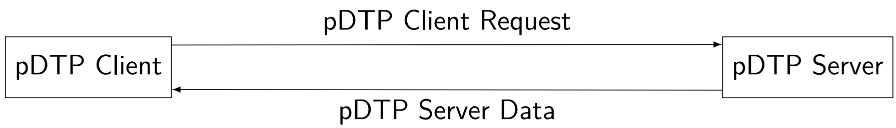


Figure 5.15: *The pDTP architecture.*

Figure 5.16 shows the pDTP block diagram. Data from the UDP stack is fed to the port decision logic. UDP packets with a specific destination port number (1234) will be fed to a buffer and automatically looped back to its origin. The loopback feature was added for the user to ensure that the system is alive and for testing performance. Packets with the pDTP destination port will be routed to the pDTP RX block. This block will interpret the client request and forward a client record to the pDTP TX block. The client record contains the opcode and the other fields of the request. Note that the pDTP TX block has access to the pRU data via the AXIS interface of the global buffer.

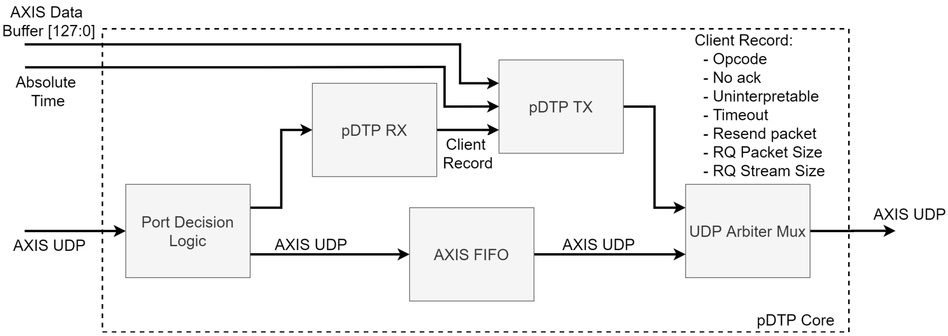


Figure 5.16: *Block Diagram of the pDTP Core.*

The pDTP TX block is a quite complex block controlled by a comprehensive FSM. A simplified FSM diagram is shown in Figure 5.17. Three main branches are shown in the diagram: (1) the RQR branch, (2) the RQS/RQFS branch, and (3) the STATUS branch. These branches represent the modes of pDTP along with the status request.

RQR: Request Read A request for a single packet. This request will provoke the server to transmit a single packet of data of the specified size. The client can also determine whether the server only transmits data if the buffer contains the specified amount of data and whether the server should maximize the packet size if data is available. These features allow a certain kind of flexibility in the transmission, allowing the client to ensure an efficient scheme. With this mode, the client has the opportunity to validate the transmitted data by the optional use of an acknowledgment packet. However, this will naturally reduce throughput. The client can also initiate the resending of the last packet. Therefore, during transmission of a packet in RQR mode, a copy of the data will be stored in a temporary buffer. However, this buffer is always overwritten when a new request is transmitted to the client, so a resend can only be performed directly after a RQR.

RQS: Request Stream A request for a stream of packets. This request will cause the server to transmit a certain number of packets. The client can specify whether the server should wait for new data to arrive in the buffer, or end

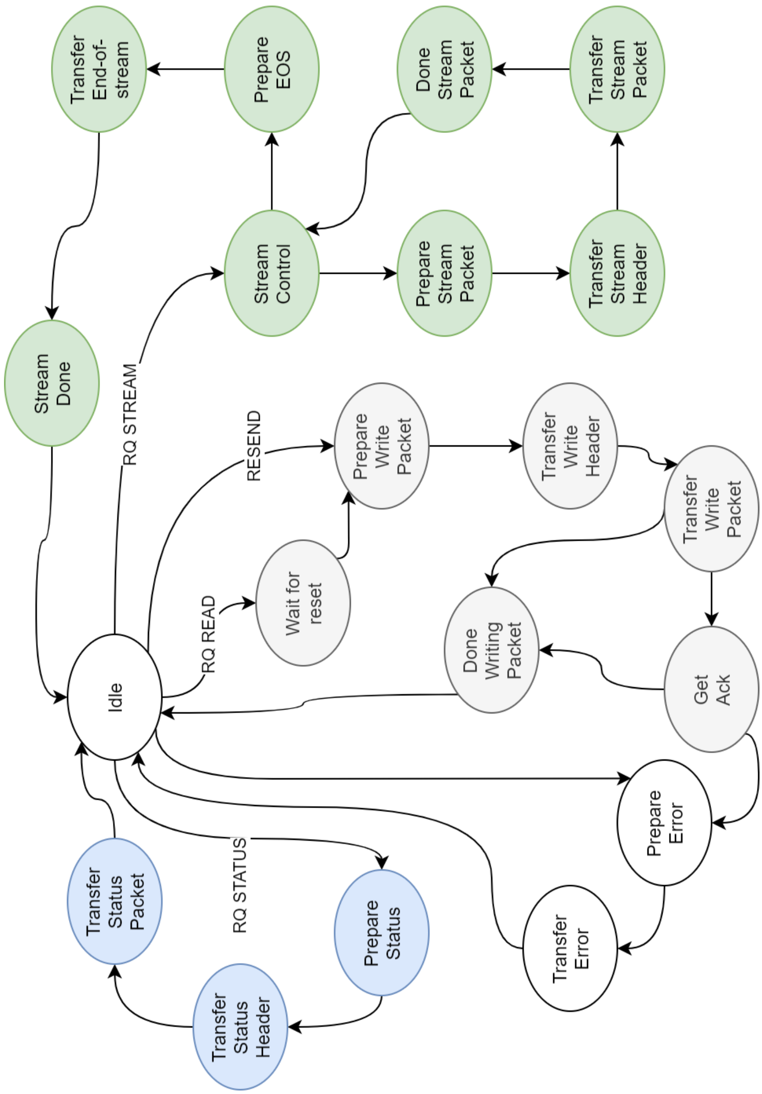


Figure 5.17: *Simplified FSM diagram for the pDTP TX block.*

transmission as the buffer is emptied. The completion of the request will be marked with a specific end-of-stream packet notifying the client that the server has completed the request. In this mode, the client has no option to resend packets as new packets are transmitted back-to-back, and there is no possibility of temporarily storing the packet in a separate buffer.

RQFS: Request Full Stream A request for a continuous stream of packets. By using this request, the client is asking the server to always transmit packets as data become available. The client can, as with the other request, determine whether the server should maximize the packet size and require a minimum payload size. With this mode, the client ensures the highest throughput possible. However, one loses the possibility to control the stream in case the client is busy, resulting in a significant loss of reliability. Like RQS, there is no possibility of recovery in case of packet failures.

It is the client's task to optimize the use of modes as they have specific strengths and weaknesses in different situations. In the case of a very stable network where no packets are dropped, the client should always be using RQFS, as this will provide the highest throughput. However, to ensure correct operation the client can choose a more varied use of the modes based on the network's packet loss rate and the amount of data currently in the buffer. As packet loss can occur because of packets being transmitted without a sufficient gap between them, the client can specify whether the server should throttle the streams with a configurable spacing between packets.

5.7.3 Addressing

To ensure that no conflict exists between the pRUs and the network, a specific IP and MAC addressing scheme is employed. The addresses are based on the pRU ID and the link number. Each pRU will contain all its links within its unique gateway address. This scheme was adopted to allow for multiple links per pRU, and for the possibility that the number of pRUs could increase in the future.

5.7.4 pDTP Client Software

The pDTP protocol has been fully implemented and tested, and significant efforts have been made to develop the client software. The DAQ software is designed to be highly efficient and to avoid bottlenecks when processing data. For this reason, the software is split into three separate parts: the pDTP client, the parser, and the output. These parts operate as a pipeline, as shown in Figure 5.18. The pDTP client is the input module and is responsible for the communication with the pRU server-side. The client will optimize the requests depending on the pDTP server buffer usage and the network performance. The data is transmitted to the next step of the pipeline via a thread-safe queue.

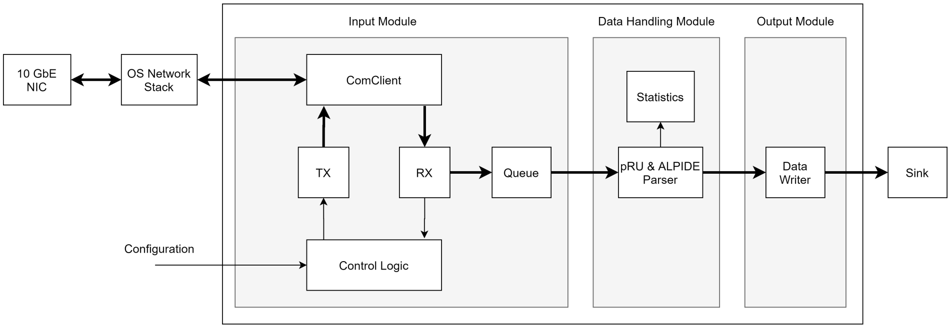


Figure 5.18: *Block diagram of the data readout software.*

The parser must sort data and combine pRU words that belong together. Furthermore, the parser extracts 4D data points and builds the full detector events. This means that all pixel hits of the entire detector that are within the same time period are combined. These full events can be used by the most-likely path algorithms to establish the particle tracks. Importantly, the parser does full sanity and coherence testing of the data. Corrupted data are stored by itself. This allows an attempt to manually recover the data later.

The final stage of the pipeline is responsible for transmitting the data to other parts of the software. For instance, during the configuration of the detector, test-data is needed to determine the addresses of the noisy pixel. In general, data can be stored to file to be analyzed later.

5.7.5 pDTP Server Emulator

A pDTP server emulator has been made to aid the development and verification of the pDTP client software. The software is organized in a similar fashion as the actual firmware and two state machines run as threads and control the RX and TX parts of the server. The RX state machine communicates with the TX state machine via a queue. Another thread is used to generate random data or load data from file. The emulator can, thus, be used to test the complete data acquisition software chain by pre-generating actual known data and load it to the emulator. Also, as the pDTP Server is agnostic to the data format, the data can be intentionally filled with errors to test the parsing capabilities of the software.

5.8 Conclusion

The design of the data flow is fully functional. The major blocks have been extensively tested with both simulation and in hardware (see Section 7.2.2 and 7.4). Verification of the firmware design in the lab and during beam tests have not shown significant weaknesses of the current design. The design can simply be modified to further increase the data throughput of the priority offloader by either changing the number of pipeline branches or by increasing the readout frequency of the channels. Furthermore, if desired in the future, the offload scheme can be upgraded to use more than one 10GbE link.

CHAPTER 6

Detector Control and Monitoring System

In addition to retrieving data, the pCT system electronics must also ensure proper operation of the sensors and the DAQ system itself. Thus, a detector control and monitoring system must be included in the design. The combined detector control and monitoring system, henceforth just called the Detector Control System (DCS), should be understood as the collection of software and hardware components that are used for controlling and monitoring the pCT detector. This chapter is mostly concerned with the hardware components of the DCS, specifically the FPGA firmware. The chapter is introduced by a discussion about the DCS architecture design choices. After that, several parts of the DCS layer are given special attention. An essential part of the DCS is the trigger and clock synchronization between all the sensors and the readout electronics. This is given particular emphasis. The chapter is concluded with a discussion about the DCS time budget.

6.1 DCS Architecture

The two main components of the DCS are the pRU and the PCU, introduced in Chapter 2. The PCU supplies and monitors the ALPIDE power usage. This is used to detect shorts, SELs, and other problems. The PCU will shut off the power to prevent any damage to the detector in case of critical failure. The pRU, however, controls the operation of the ALPIDEs. Naturally, we must establish communication links between the system control room and both the

PCU and the pRU. Figure 6.1 outlines the general components of the DCS and their interconnections. Note that there is no direct communication between the pRU and the PCU.

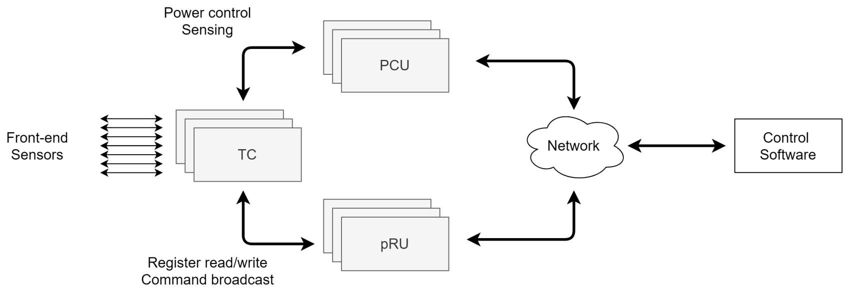


Figure 6.1: *General overview of the DCS components.*

There are multiple ways to establish communication between the control room and the DCS hardware components. As was concluded in the discussion about the radiation environment in Chapter 4, we know that radiation will not be a particular problem if the pRU is placed at a sufficient distance from the beam center. Therefore, one can opt for a commercial-off-the-shelf Ethernet solution and use a standard communication protocol like TCP or UDP.

6.1.1 Microcontroller-based DCS

We know from Chapter 5 that the data link of the pRU must be separated from all other communication links to avoid disruption of the data throughput. Therefore, the control link must be implemented with a separate physical interface. This separation opens the possibility of using a microcontroller (μC) to simplify the use of Ethernet and higher-level protocols on the pRU. The μC can either be implemented on the pRU as a separate chip, or on the FPGA itself as a soft-CPU system. The use of a μC-based system would allow running embedded software directly on the pRU for control and monitoring purposes. However, it would also require the development of that software, noting that embedded software is often more complicated, and performs less efficiently, than high-level software running on a server system.

During the development of the pRU firmware, a μC-based system has been implemented and tested [76]. Using the VCU118-development kit, this was

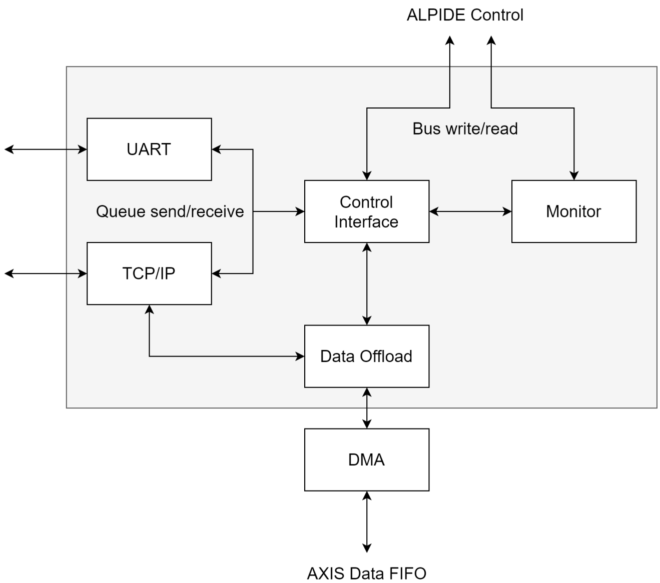


Figure 6.2: *Block diagram of the embedded software stack [76].*

implemented with a Microblaze soft-CPU and Xilinx MAC and PHY IP cores. The PTB, which is described later in Section 7.5.1, also uses the μC-based system but employs a hard-CPU that includes MAC and PHY blocks.

A custom protocol to be used with TCP communication has been developed. This protocol is based on the specific tasks of the software. This includes tasks like reading and writing registers of the ALPIDEs and the modules of the FPGA firmware, and the transmission of commands to the ALPIDEs for synchronization and triggering. Using the Xilinx software library for handling the TCP protocol, the embedded software is instructed to decode the protocol and perform the tasks one by one. Almost all tasks involve reading and writing registers via the FPGA bus interface (see Section 6.2). The software was designed to run on the FreeRTOS platform and included the possibility of sending and receiving packets with UART instead of TCP. Figure 6.2 shows a simplified block diagram of the software stack. Note also that the software has support for offloading the sensor data since no other data offload scheme was done at the time of development.

The pRU would require the use of a soft-CPU or an external μC because of the pure FPGA chosen. However, the soft-CPU was determined to be a severe bottleneck in the system. Many of the performance limitations could be addressed to the selection of FreeRTOS. However, the work needed to port the software to another OS was considered excessive. For this reason, one opted to research the possibility of developing a pure FPGA-based DCS.

6.1.2 Pure FPGA-based DCS with IPBus

Pure FPGA-based DCS components will face some of the same challenges as the data offload link of the pRU (see Section 5.7). Specifically, the implementation of TCP on an FPGA is ruled out because of the complexity. Similarly to how pDTP was developed for data offload, it would be possible to develop a custom protocol on top of a GbE UDP implementation. Fortunately, however, the use of Ethernet has already become semi-standard for detector control within the HEP community, driving further the development of protocols and standards. This is the result of standards like VME becoming antiquated and experiments like CMS upgrading to xTCA standards [77].

IPBus is an Ethernet-based protocol standard developed for particle and HEP experiments using FPGA-based hardware devices [78]. Recent developments are explicitly done to be used in the CMS upgrade for the LHC 2015 run. Both FPGA firmware and an integrated software suite are developed to simplify instantiating a reliable communication channel using UDP with GbE.

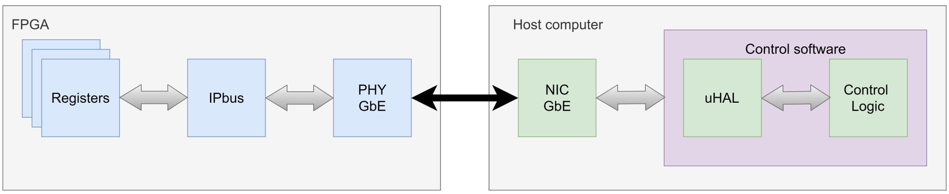


Figure 6.3: *Simplified block diagram of the pCT DCS as implemented with IPBus during testing.*

Figure 6.3 shows a simplified block diagram of how the pCT DCS is implemented with IPBus during testing. IPBus provides a hardware abstraction layer (HAL)

API for both C++ and Python, called μHAL. This API provides functions for register read, write, and read-modify-write (RMW) operations for accessing the bus devices. Additionally, the IPBus suite contains the Control Hub software, creating a single point of access for multiple devices when scaling up to multiple DCS components.

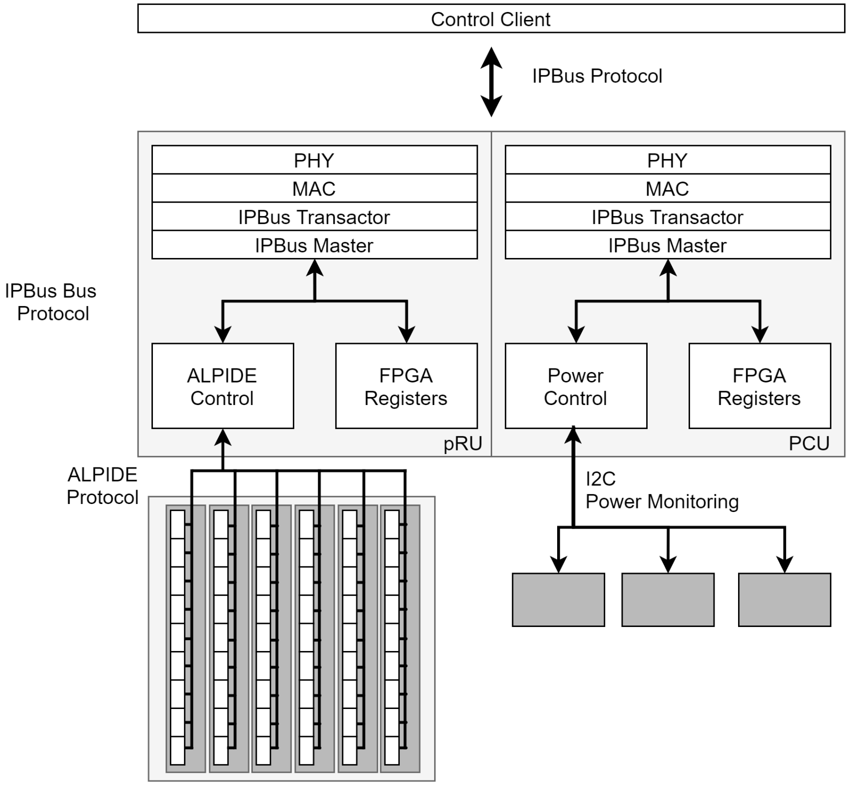


Figure 6.4: *Sketch of the different layers and devices of the DCS.*

Using IPBus, the development of an interface between the DCS components and the control room becomes significantly simpler than when using any custom protocol. Figure 6.4 shows the sketch of all the layers of the system. On the left side, the pRU is connected to all the ALPIDEs, whereas on the right, the PCU FPGA is connected to the power monitoring sensors via I2C.

6.2 Bus Interface

The pRU bus interface provides read and write access to the on-chip registers and, indirectly, the ALPIDE registers via the ALPIDE Control module (see Section 6.4). Several bus protocol standards are extensively used in FPGA applications, e.g., AXI-lite, Wishbone, Avalon, and Simple Bus Interface (SBI) [79, 80, 81, 82]. AXI-lite was the preferred bus choice in the early stages of the pRU development, as this is the built-in protocol of the Xilinx CPU memory-mapped interface. However, as the μC-based system was discarded and IPBus was selected for external communication, the on-chip bus protocol choice became rather self-evident. With the use of *bust* (see Section 6.3), the bus interface change is fast, safe, and straightforward.

The IPBus on-chip bus protocol is unsophisticated and is based upon the Wishbone standard[1]. The bus protocol uses only seven signals. A write and a read transaction are shown in Figure 6.5. The master requires the slave to acknowledge the transaction. IPBus has an intrinsic timeout after 256 clock cycles of the master IPBus clock[2,3]. Note that by waiting to transmit the acknowledgment signal, a slave can *hold* the bus (see Section 6.3.1).

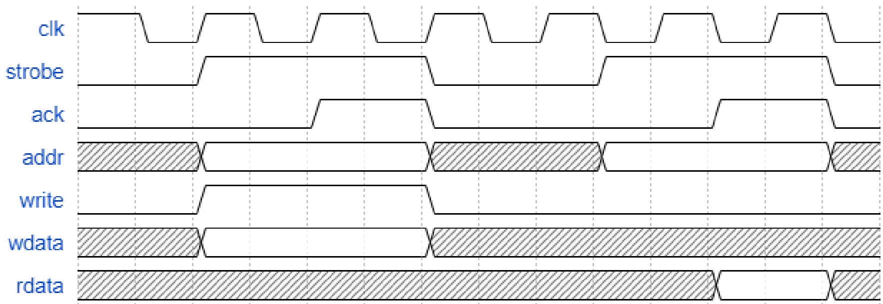


Figure 6.5: *Waveform showing both a write and read transaction on the IPBus protocol. The protocol does only require three control signals in addition to the three 32-bit data and address signals.*

[1] The protocol does also resemble the SBI used extensively by Bitvis [82].

[2] As seen in Figure 6.6, the master IPBus clock is 31.25 MHz.

[3] Note that, contrary to IPBus, AXI-lite has no intrinsic time-out, risking a locked situation by requesting a read or write from a non-existent address.

The IPBus bus topology is point-to-point. This means that address decoders are used to separate the slave interface signals and multiplex the returning data. Figure 6.6 shows the pRU firmware bus tree in which the path from the IPBus master to the various modules are outlined. Notice how simple register transfer level (RTL) interconnect modules are instantiated between the master and the slaves and used to decode the addresses based on given base addresses. These blocks are purely combinational and do not add latency.

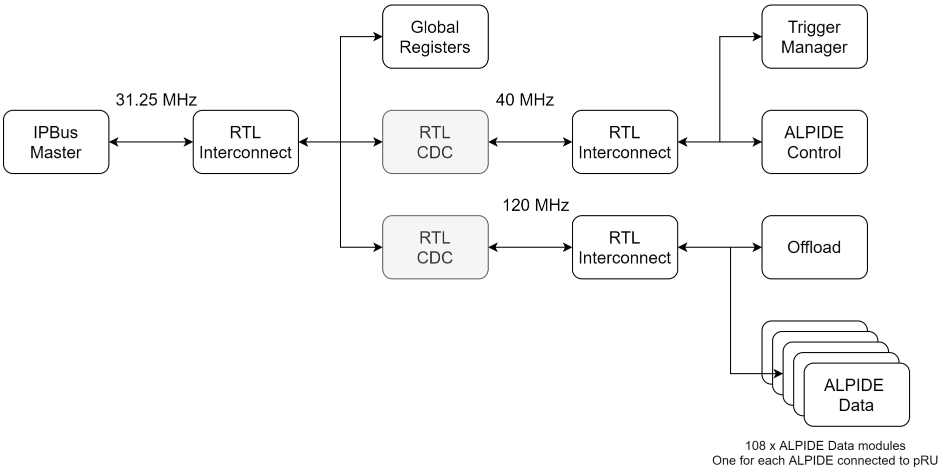


Figure 6.6: *The pRU firmware bus interface tree.*

6.2.1 Clock Domain Crossings

Recall from Section 4.4.1 that CDC synchronization modules are placed into the bus interface datapath. These modules are shown as gray boxes in Figure 6.6. There are two CDC synchronizers on the bus path: between 31.25 MHz and 120 MHz, and between 31.25 MHz and 40 MHz. This means that an operation targeting the 120 MHz and 40 MHz domain modules will be delayed by nine clock cycles of 31.25 MHz. Thus, there is a total transaction time of just below 0.3 µs.

6.3 Bus Tool

A common, and often neglected, source of error in FPGA designs is the bus interface and discrepancies between the actual and the documented register map. During the design process, the register map is frequently updated, especially if the specifications are changing. Because of this, the implementation, documentation, and more crucially, verification, might be mismatched. To counteract this, designers can use tools that automatically generate hardware description language (HDL), documentation, and testbenches. Several open-source tools exist for this purpose, but what they have in common is that they usually only support a single bus protocol. This is true for both the Bitvis Register Wizard (SBI), AirHDL (AXI-light), and several Wishbone-tools. As discussed in Section 6.2, both AXI-lite and IPBus have been employed by the pRU firmware during development. Therefore, a custom bus software tool (bust)[4], was developed to provide HDL, header-files, documentation, and testbenches from a JSON-file containing the specification of a module and its registers. Figure 6.7 outlines the bust concept.

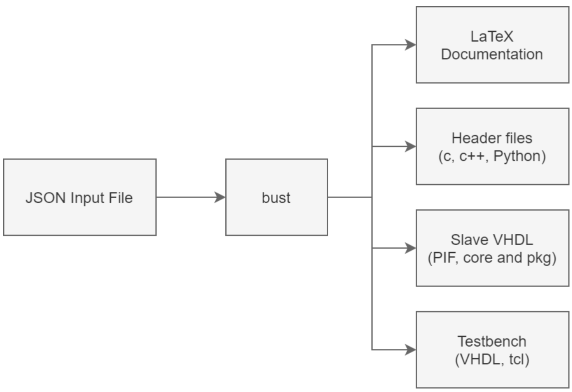


Figure 6.7: *The general concept of bust.*

A bust-generated module adheres to typical encapsulation principles often used to prevent implementation errors. This means that the bus interface mechanics are hidden from the general user logic. This is seen in Figure 6.8. The user logic

[4]bust is a freely available open-source project found at `https://github.com/olagrottvik/bust`

accesses the registers via the VHDL record construct. All register addresses, sizes, pulse lengths, and more, are hidden within the module peripheral interface VHDL package.

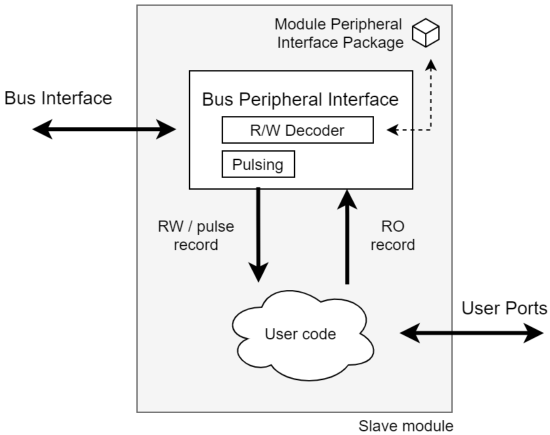


Figure 6.8: *The resulting firmware module generated by bust.*

6.3.1 Register Types

In addition to regular read/write and read-only register types, the pRU firmware employs a special type of register; the pulse register. Writing to this kind of register will set the register value for a certain amount of clock cycles before self-resetting back to its original value. This kind of register is useful for the initiation of procedures and more.

Furthermore, some registers can also stall the bus for a certain amount of clock cycles. Holding the bus avoids cases where the user tries to initiate commands before other procedures are completed. This minimizes the risk of user error. The stall feature is for instance used when the pRU communicates with the ALPIDEs (see Section 6.4). Both register mechanics are automatically generated by the bust software.

6.4 ALPIDE Control

The ALPIDE Control module is a firmware module of the pRU firmware designed for the communication with the sensors utilizing the slow control interface of the ALPIDE[5]. The block contains a separate control instance for each sensor string connected to the pRU, and it is possible to communicate with individual sensors, a full string, or even a complete layer at the same time. A simplified schematic drawing is shown in Figure 6.9. Notice that a common finite-state machine (FSM) is used to control all links. Thus, the block is designed so that no link can operate independently from the others. A significant limitation is that read operations must target only a single ALPIDE. This is due to the nature of the shared slow control interface.

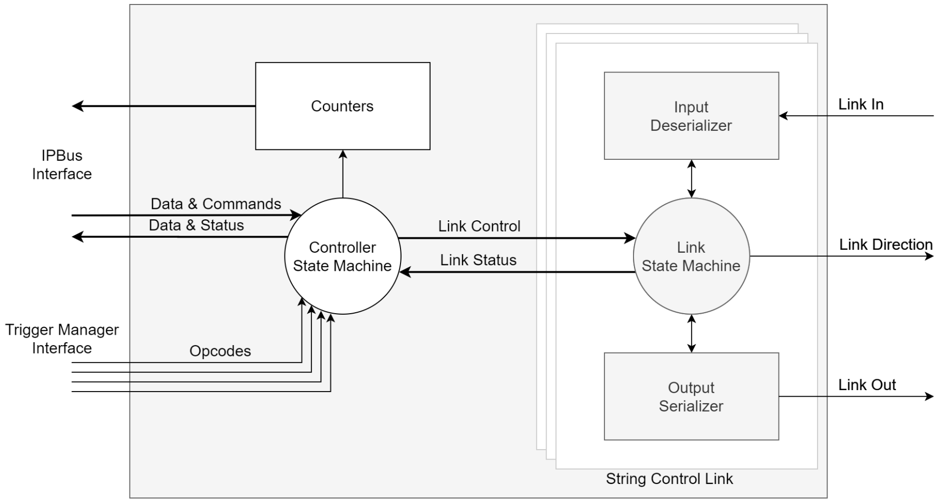


Figure 6.9: *Simplified schematic drawing of the ALPIDE Control block in charge of the slow control interface.*

6.4.1 Half-Duplex MLVDS Bus Interface

The ALPIDE slow control link uses a bi-directional MLVDS signal. The current direction of the control link is managed by a simple signal for each link. The pRU and its predecessors employ a half-duplex MLVDS-driver chip[6] between the FPGA and the ALPIDEs. This way, the data signal for each direction is separated on the FPGA-side, as seen in Figure 6.9. The chip also allows for

[5]The module is based on an early version from the ITS RU firmware but has been considerably changed to be agnostic to cable lengths and to be integrated with the Trigger Manager module.

[6]Texas Instruments SN65MLVD080 [83].

listen-while-talking, meaning that we can observe the actual MLVDS signal state at any time by measuring the return signal, regardless of the current link direction.

6.4.2 Transactions

The ALPIDE Control module can perform three types of transactions with the sensors of the layer. These types are based on the number of ALPIDEs that are affected by the transaction. See Figure 6.10 for the format of valid ALPIDE transactions.

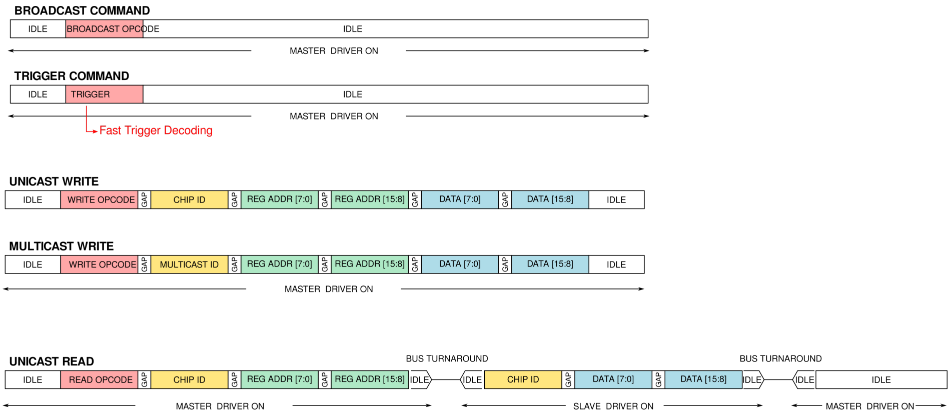


Figure 6.10: *The format of valid ALPIDE slow control transactions [29]. Note that the read transaction requires a bus turnaround period.*

Unicast A unicast transaction is either a register write or a register read operation for a single ALPIDE chip.

Multicast/multistavecast Multicasting refers to the write operation of a register on several ALPIDEs at the same time. Note that the register *read* transaction cannot be performed for multiple chips at the same time. The ALPIDE slow control format allows the writing of all sensors sharing the same interface. However, the ALPIDE Control module also enables writing to a specific chip ID on all connected strings (multistavecast) or even to *all* ALPIDEs connected to the pRU.

Broadcast A broadcast is a *command* transaction to every single sensor of either a specific string or the complete layer. Note that command transactions also can be performed by writing to the command register on the ALPIDE, and thus can also be carried out by unicast and multicast transactions. The trigger and various resets are examples of ALPIDE commands that are usually transmitted as broadcasts.

All the types of transactions can be started via the IPBus interface. Additionally, a subset of the broadcast commands is available to another block of the firmware design; the Trigger Manager (see Section 6.5.6). The type of transaction to be executed is selected by setting specific registers. A write or a read operation is initiated by writing to certain pulse/stall registers. By writing to these registers, the bus interface is stalled until the operation is completed. Each link is controlled by its own FSM, which communicates with the main FSM.

A simplified flowchart showing the combined main and link FSMs is given in Figure 6.11. Depending on which command is requested, three main paths are followed. The main FSM is giving instructions to the link while waiting for either a fixed amount of clock cycles or for the link to notify about completion. As no acknowledgment is given from the ALPIDE to confirm that a broadcast or a write transaction has been completed, the main FSM is simply waiting a fixed amount of clock cycles. The only way to confirm that a write operation has been completed successfully is to read back the register to verify the written value[7]. This can be done by the DCS software. During read transactions, however, one can confirm that the returned transmission from the ALPIDE is within the valid format and thus can confirm success or not. Note how the read transactions require an additional block, the input deserializer, which is outlined below in Section 6.4.3.

[7]Note, that some ALPIDE registers are write-only and cannot be read.

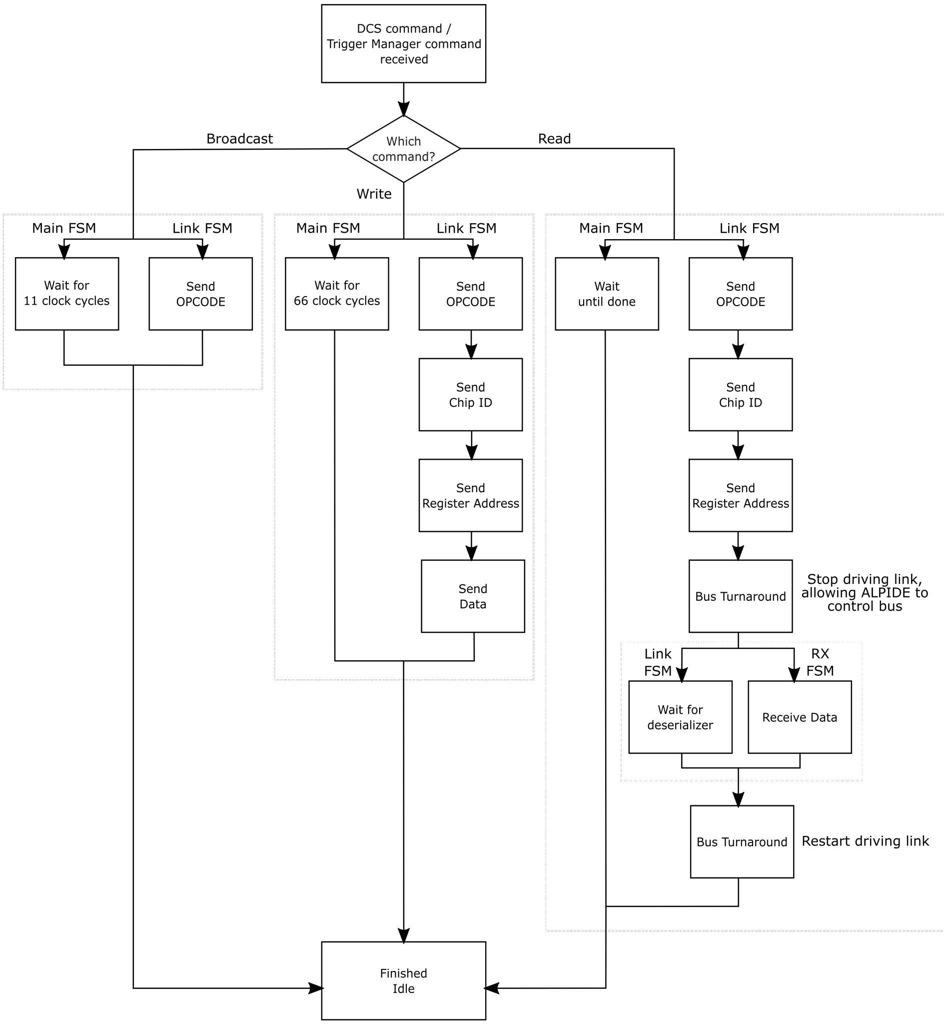


Figure 6.11: *Simplified combined flowchart of the main and link state machines.*

6.4.3 Input Deserializer

The input deserializer is initiated by the link FSM. The deserializer is oversampling the input signal and will, based on a given set of parameters, automatically determine which sample is the most reliable. The sample selection is performed for every 8-bit word transmitted from the ALPIDE, which adds significant reliability compared to a fixed-phase approach. Furthermore, the block also extracts the start-bit of the transmission and can extrapolate which word is currently sampled. For each word, the block checks that the end-bit is correct.

The block is designed to work with different oversampling rates. However, based on the results from early lab testing, it was decided that the incoming data will be sampled at six times the transmission rate, 240 MHz, to ensure a near-zero risk of errors. The block continuously shifts the sampled data into a vector whose size reflects the oversampling rate. The deserializing sequence for a single read operation is laid out below and in Figure 6.12:

- Idle until the link FSM notifies
- Wait for a configurable number of clock cycles. This can be used to tune to various cable lengths. A fixed number of clock cycles is found that supports cable lengths from 0.3 m to 5 m, as well as the zero meter setup for the PTB (see Section 7.5.1).
- Wait for a stable input signal. This step makes sure that the ALPIDE has started driving the link and that the MLVDS-driver chip has completed the turnaround of the bus. Note that the default setting of the ALPIDE is to use Manchester encoding, but this can be turned off in special circumstances. Thus, the block must be able to detect a stable signal for both cases, which is accomplished by observing that a configurable number of samples are identical. This check works for Manchester encoded data only because the protocol uses a start-bit to mark the beginning of a data word. A configurable timeout of this check causes an error flag to be raised, and that $0xDEAD$ is reported as the data received from chip ID $0xFF$ (which does not exist).
- Wait for the start bit. Based on the position of the discovered start bit in the sequence of samples, the sample point is selected for the following word.

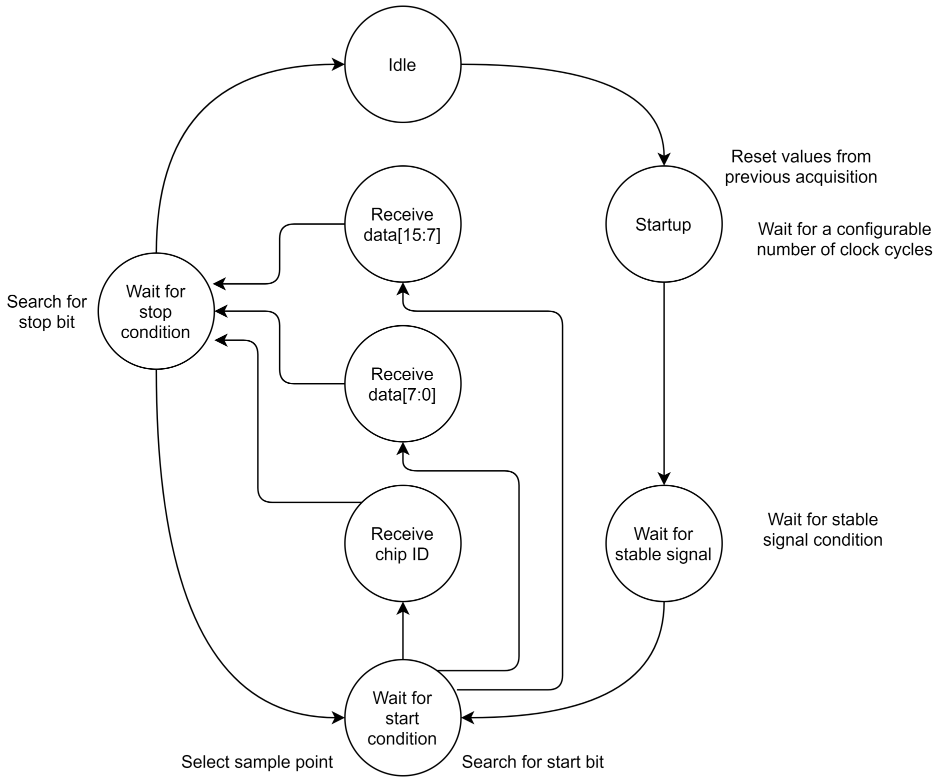


Figure 6.12: *The state machine diagram of the input deserializer block of the ALPIDE Control module.*

The middle sample of a sequence of identical sample values is selected as the sample point, lowering the probability of noise when sampling the bits of the word. If the start-bit fails to occur before a configurable timeout condition, the process is aborted, and the link FSM is notified.

- Sample the data word by shifting the selected sample position data bit into an 8-bit vector.
- Check that the stop bit occurs at the expected position. If no stop-bit is observed, the process is aborted, and the link FSM is notified.
- The last three steps are repeated for both the chip ID, data-low, and data-high words, as seen in Figure 6.10. When finished and all checks are met, the block notifies the link FSM that the process has completed successfully and that the resulting data and status can be latched to bus interface registers.

6.5 Trigger and Clock Synchronization

As discussed in Section 5.6, each detector frame needs to be associated with a certain trigger. This is because the frame must be combined with all other frames originating from the same trigger to generate a 3D snapshot of the detector that can be used for particle tracking. The trigger signals must be transmitted to all sensors of the system at the same time. The pRUs use a board-to-board interface (BTBI) to achieve this and other synchronization tasks. The BTBI is controlled by a single master pRU and is discussed in detail in Section 6.5.5.

6.5.1 Synchronization Parameters

To combine the data frames from different sensors and layers, the DAQ software relies on four separate parameters of the pRU data format. The values of all these parameters can be altered with the IPBus communication channel. However, as Ethernet is non-deterministic, reliable synchronization can only be achieved by using the BTBI. The origins of the synchronization parameters are shown in Figure 6.13.

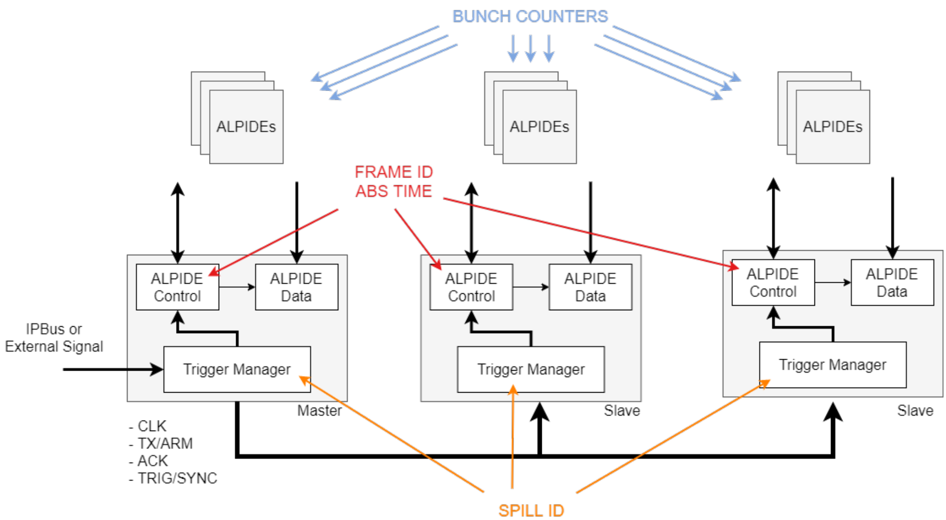


Figure 6.13: *The pRU synchronization architecture. The arrows show the origin of the various synchronization parameters. The board-to-board interface (BTBI) is shown connected in a multi-drop scheme with a single master pRU.*

6.5.1.1 Spill ID

The spill ID is intended to separate frames from different 2D projections. The value is stored by the Trigger Manager module of each pRU; therefore, so the value can be out-of-sync between each layer.

6.5.1.2 Frame ID

The frame ID is intended to separate frames originating from different triggers. The frame ID counter is incremented each time the ALPIDE Control module transmits a trigger command[8]. Similarly to the spill ID, the frame ID can be out-of-sync between two layers. However, in some rare cases, the frame ID can also be out-of-sync between channels in a layer. This situation is discussed in Section 6.5.2.

6.5.1.3 ALPIDE Bunch Counter

Each ALPIDE is supplied with a 40 MHz clock distributed by the pRU, and this is also the update rate of the ALPIDE bunch counter, a 16-bit internal counter. The counter value from the time the ALPIDE is triggered is used to tag an outgoing data frame. Note, however, that the three LSBs are not included in the data and that the resulting resolution, therefore, is 200 ns. As each ALPIDE incorporates a separate counter, the values might differ between them depending. The counters can be synchronized by transmitting a specific reset command to the ALPIDE.

6.5.1.4 pRU Absolute Time

Naturally, the 16-bit bunch counter of the ALPIDE overflows after only 1.6 ms. As each pCT projection is in the order of seconds, we must also tag the data with the pRU absolute time to ensure that we can separate all frames within a projection. The 32-bit pRU absolute time counter ticks at the 120 MHz system clock and thus, the time resolution is higher than with the ALPIDE bunch

[8]The trigger command can be sent to one or more ALPIDEs depending on the mode. However, the frame ID is incremented for ALL ALPIDEs regardless of how many ALPIDEs that are triggered.

counter. The counters can be out-of-sync between different layers. Note that the absolute time can be used to track the potential drift between the layers. This is valuable depending on the clock configuration which is discussed in Section 6.5.4.

6.5.2 Frame ID and Absolute Time Issues

The frame ID and the absolute time are both updated and latched to registers when the ALPIDE Control transmits a trigger command. The values are then fixed until another trigger command is transmitted. The registers are connected to the ALPIDE Data modules which use the current values of the registers to tag the incoming data frames. As illustrated in Figure 6.14, a channel handling a very large data frame might become out of sync when the data processing time exceeds two trigger periods. The second frame arriving on that specific channel, marked in red, is tagged with the incorrect frame ID and absolute time. Furthermore, the third frame, if arriving within the same window as the second, can be tagged with the same frame ID and absolute time counter as the second frame. Note, however, that these packets will still obtain the correct bunch counter values. Therefore, the DAQ software should be able to detect and correct for such situations.

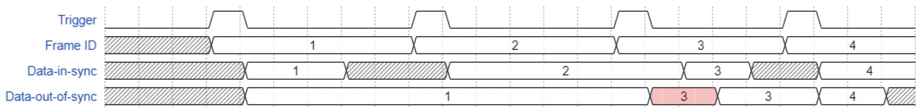


Figure 6.14: *Waveform illustrating the Frame ID and Absolute Time sync issue. The incorrectly tagged frame is marked in red.*

Because the nominal trigger rate of the pCT is 10 µs, such situations will occur only if a frame is large enough to exceed 20 µs[9]. Based on the throughput of 960 Mb/s, the frame must be over 2.4 kB for this to happen. If no clustering is enabled, or the data is solely single-pixel hits, this can occurs if an ALPIDE has over 800 single-pixel hits. However, if we conservatively assume that a

[9]Or if two frames combined are large enough to exceed 30 µs.

particle on average produces four data long words[10], 200 particles are required to generate such a frame. Based on the requirement of 10^7 particles/s scanning beam and a frame rate of 10 µs, giving an average rate of 100 particles per frame, this situation will rarely occur. However, this phenomenon must be taken into account before the detector is used with a faster trigger rate[11].

6.5.2.1 Alternative Approach

The above timing ambiguity situation is both rare and recoverable. However, an alternative approach for managing the frame ID and the absolute time counter has been explored. The above issue can be solved by having separated registers for the frame ID and the absolute time counter values within each ALPIDE Data module. For instance, the frame ID counter is incremented whenever a frame is observed by the protocol checker. Moreover, for each trigger transmitted by the ALPIDE Control, the ALPIDE Data module stores the absolute time counter value within a FIFO, ready to be extracted when a frame arrives.

However, this approach is perilous for other reasons. With an increasing number of instances of both the frame ID and the absolute time counter, we intensify the risk of asynchrony. Also, by using the incoming data from the ALPIDEs to increment the parameters, we expose the system synchronization to be vulnerable to data link noise. Specifically, a potential error can arise when the ALPIDE Data module interprets noise and jitter as actual data. The result of this event will be (1) an erroneous frame containing only garbage data that is tagged with the frame ID and the absolute time from the triggering, and (2) a proper frame tagged with an erroneous frame ID and absolute time information.

Consequently, the second frame's pRU time will not match the other sensors' frames initiated by the same trigger. Although it might be possible for the DAQ software to correctly identify the situation based on the bunch counter

[10] This entails a pixel cluster of up to 32 pixels firing. As shown earlier in Figure 1.5, this is rather unlikely.

[11] Note that if the detector was utilized with a scatter beam, rather than a scanning beam, the rate of particles per frame would significantly drop because the particles would be spread to all sensors of the layer.

value, the frame ID value will never recover from this situation without a reset. Furthermore, if more erroneous frames are observed, the frame ID error will accumulate, making the task of correctly identifying proper frames even more difficult for the DAQ software. Based on this observation, the alternate approach is rejected.

6.5.3 Synchronization Levels

By using these four parameters together, the DAQ software can validate and combine the data frames. But, in certain situations, like the ones described above, some of these parameters might not match. However, as we shall see, this does not mean that data frames cannot be appropriately combined.

Figure 6.15 shows four different levels of synchronization between the sensors and layers. Four sensors, two from different layers, are used to illustrate the state of synchrony based on their parameter values. In the top left corner, all ALPIDEs and pRUs are fully synchronized, i.e., all four parameters match each other. This is the optimal situation where we are confident that all frames originate from the same trigger signal.

In the top right corner, however, the frame ID and absolute time counter of one sensor do not match the values of the other sensors. Nevertheless, based on the bunch counter and the spill ID, the frame can easily be combined with the frames of the other chips. The DAQ software can search for this situation by combining all frames from within a layer with a matching bunch counter. Any frame where the frame ID and the absolute time counter are slightly off will be a candidate for this temporary out-of-sync situation. Further examination might reveal that the next frame for this channel has the same frame ID and bunch counter value as the previous.

In the bottom left corner of Figure 6.15, a drifting clock situation has arisen. This has caused all the bunch counter and the absolute time counter values of a layer to be different from those of another layer. However, in this situation, the values will still match within a layer. Therefore, the DAQ software can first combine all frames within a layer, and then next combine these frames with the other layers based on the frame ID and spill ID values.

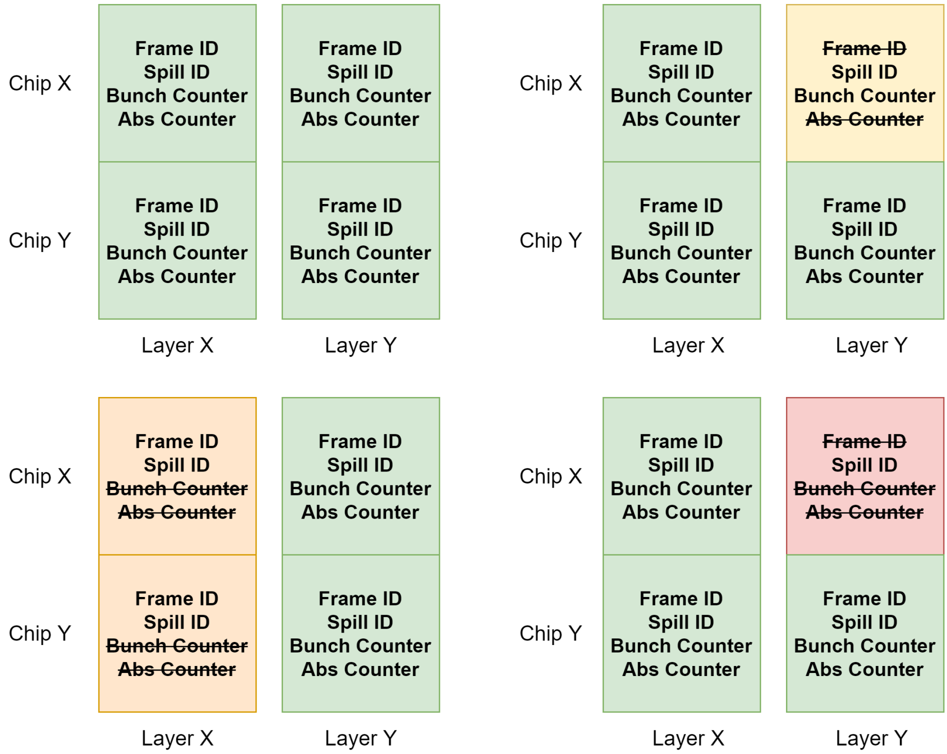


Figure 6.15: *Illustrations of the various levels of data frame synchronization. Parameters that are crossed over, illustrate values that do not match the others.*

In the bottom right corner, a situation has occurred where neither the frame ID, the bunch counter value, nor the absolute time counter value of one sensor's data frame match any other sensors' data. There is no way to combine this frame with the others. If the frame bunch counter value does not match the bunch counter value of any of the other sensors, this sensor is most likely out-of-sync, or the data is corrupted.

What the different levels of synchrony illustrate is that the bunch counter value is the most critical synchronization parameter and that consistency between sensors is imperative. Furthermore, the frame ID and, ideally, the absolute time counter must be synchronized between the layers. Within a layer, the spill ID and the bunch counter value are the two most reliable parameters. This is because these values cannot become out-of-sync based on activity on the data links. However, between two layers, the bunch counter is susceptible to clock drift depending on the clock mode (see Section 6.5.4). Note that the frame ID must never be used without the bunch counter to combine data frames within a layer. But rather, the value can be used to combine all frames from a layer with all the frames of another layer.

In a sense, the absolute time counter is redundant as the spill ID can be incremented when the bunch counter overflows. Also, in principle, the frame ID and the absolute time parameters have the same purpose and behave similarly. Nonetheless, the absolute time counter can also be used to keep track of clock drift between the layers and is thus a valuable extra parameter improving reliability.

6.5.4 Synchronization Architecture

All clock oscillators are associated with some uncertainty. The accuracy of a clock is usually specified in parts-per-million (PPM). The oscillators supplying the pRU FPGA firmware will naturally differ and drift apart. Based on the clock accuracy, we can calculate the worst-case difference between two ALPIDEs on two separate pRUs. If we assume that the 40 MHz clocks on two different pRUs supplying the ALPIDEs have an accuracy of 20 PPM, the actual frequencies supplying the ALPIDE can differ with up to 1600 Hz. That means that the bunch counter of one of the ALPIDEs might have ticked up to 1600 times more than the other ALPIDE after just a single second. The same observation can be made regarding the pRU system clock. In the worst case, if two pRUs are configured to autonomously generate a sequence of triggers with a 100 kHz trigger rate, one of the pRUs will have finished transmitting all triggers 40 µs faster than the other. The resulting data would be completely useless.

Note that, since the pRU distributes the same 40 MHz clock to all sensors over a physical interface with only slightly different trace lengths[12], we expect that all the ALPIDE bunch counters of a layer will be synchronized after a simple bunch counter reset of the ALPIDEs. Thus, for the remaining discussion, we

[12]By assuming a propagation time of roughly 0.6c, the 27 cm difference between the far-end and the close-end ALPIDE on a string entails that a clock edge will occur at the close-end ALPIDE roughly 1.5 ns earlier.

assume that synchronization is required only between the various pRUs of the system.

The pRU synchronization architecture is designed to alleviate the clock uncertainty and provide sufficient synchronization by supporting the two clock modes discussed below. Both modes ensure that all triggers in the system are executed within a very short time window and that it is possible to extract an accurate time from the pixel hit data. For both modes, there is a single master pRU that controls the execution of triggers and other synchronization commands via the BTBI discussed in Section 6.5.5.

6.5.4.1 Fully Synchronous

A dedicated board distributes a common 40 MHz clock via coax cables of identical lengths. Also, a single master pRU can be used to distribute the clock over a multi-drop interface[13]. The pRUs use this clock to generate the ALPIDE and the system clocks (see Section 4.4). Assuming that the clock signal integrity is sufficient to generate low-jitter clocks, this scheme is the simplest way to ensure full synchronization between the pRUs and the sensors. However, the hardware implementation of the clock distribution can affect the clock integrity. Thus, we must ensure that the system can function without clock distribution before we can confirm and test the scheme with actual hardware. This scheme is also more costly than the alternative, which is free-running clocks.

6.5.4.2 Semi-Autonomous

All pRUs operate on the local oscillator clock, and the clocks of different pRUs are guaranteed to drift apart within milliseconds. This means that neither the pRU absolute time counter nor the ALPIDE bunch counter of a layer will match the counters of another layer. This situation can be acceptable if, and only if, we can guarantee that the triggers of all layers occur at the same time (see Section 6.5.5). Also, the frame and spill counters must be consistent between layers, meaning that all frames originating from the same trigger will have the

[13]Note that this might be reduce the synchronization accuracy because of different clock skew.

same frame and spill ID. The frame and spill counters can be used to combine the frames from different layers without the aid of time counters. For further examination of the data and to check for data corruption, the DAQ software can be used to control for consistency between the differences between the clock counters of two layers. One can expect that the absolute time counter of the pRU and the ALPIDE bunch counters will drift with the same factor since the system and the ALPIDE clocks are related.

6.5.5 Board-to-Board Interface

To synchronize the parameters of the system, but also to initiate trigger commands and more, we need a form of communication between the components. It would be natural to assume that one could use the Ethernet-based IPBus interface to issue these commands. However, one must consider that Ethernet, when controlled by software and routed through several switches, will not be deterministic in terms of packet arrival time. Thus, to synchronize the sensors and the pRUs, a separate deterministic system is needed.

A command distribution architecture is presented in Figure 6.13. In this scheme, a single pRU acts as the master, and the remaining pRUs are connected in a multi-drop scheme. The required signals are the following:

TX/ARM The serial command interface signal. Used by the master to arm the slaves before issuing the command pulse.

ACK The acknowledgment signal is connected in an open-drain scheme with a weak pull-up resistor. Thus, the slaves will pull the signal low until they all have acknowledged the arm code. Releasing the pull allows the signal to be asserted on the master-end. When returning to the unarmed state, the slaves re-pulls the signal. As seen in Figure 6.18, the master will time-out after a configurable number of clock cycles if one or more of the slaves continue to pull the signal low. This indicates a critical communication failure.

TRIG/SYNC When all the slaves are armed, the master can initiate the command by transmitting the TRIG/SYNC pulse. The slaves react on the rising edge of the pulse. If the specific command is part of a sequence, as specified in Table 6.1, the slaves remain armed until the master transmits the unarm command.

With the presented clocking architecture and synchronization interface, we can assure that commands are initiated within the same time window on all pRUs, regardless of the clocking mode. However, the time window is slightly larger when using the free-running clocks, based on the phase uncertainty. Equation 6.1 combines the three contributing factors of the maximum trigger time divergence. The first factor is the propagation time from the master to a slave pRU. We assume that the electrical distance is between 0 and 2 m and a signal propagation velocity of about $0.6c$. The second factor is the maximum clock skew between the master and the slave. The skew cannot be more than the actual clock period. Note that the fully synchronous clocking mode will have a negligible clock skew. Thus, this is the factor that separates the two modes. For the final factor, we include the propagation time difference between the far-end and close-end ALPIDE.

$$\begin{aligned} T_{MaxDivergenceFree} &= T_{pRUProp} + T_{MaxSkew} + T_{ALPProp} \\ &= 11.1\ \mathrm{ns} + 25\ \mathrm{ns} + 1.5\ \mathrm{ns} \\ &= 37.6\ \mathrm{ns} \\ T_{MaxDivergenceSync} &= T_{pRUProp} + T_{ALPProp} \\ &= 11.1\ \mathrm{ns} + 1.5\ \mathrm{ns} \\ &= 12.6\ \mathrm{ns} \end{aligned} \tag{6.1}$$

Note that all commands that affect the ALPIDEs are affected by the same divergence. This means that the bunch counter value across the layers will have an uncertainty of either 37.6 ns or 12.6 ns depending on the clock mode. Within a layer, the uncertainty of these values is only 1.5 ns. The pRU absolute time counter has an uncertainty of 36.1 ns or 11.1 ns based on the clocking mode.

The proposed command distribution architecture is only deterministic component-to-component. Transmitting the first command via the IPBus interface removes the accurate knowledge of when the command will be initiated. Its time will vary based on the IPBus packet size and the general network activity. In the case where we require the system to be fully deterministic, meaning that also the initiating first command must be executed at a known time, one cannot use the IPBus interface to start the procedure. For instance, one could imagine that a trigger sequence must be initiated at an exact time relative to the beam spill. Therefore, the pRU can also be controlled via external ports. This allows for the control and synchronization of the DCS by external parts of the detector system, e.g., the beam machinery.

Figure 6.16 shows the waveform of the dead-simple BTBI protocol using start and stop bits. Each transmission consists of an 8-bit data word, where the four preceding bits determine the arm command. This leaves the option of 16 unique commands, with optional information about the commands. The master can, e.g., transmit an arm code and also notify the slave if it should expect a sequence or not. The various arm codes are presented in Table 6.1. Transmitting the 8-bit command, including start and stop-bits, to the pRU slaves takes roughly 250 ns using the 40 MHz clock, assuming a negligible propagation time.

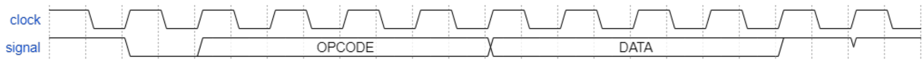


Figure 6.16: *Wavediagram of the simple board-to-board interface protocol.*

Table 6.1: *The arming codes of the board-to-board interface protocol. Note the special Sync All command which arms the pRUs for the reset of all the four synchronization parameters.*

Command	**Sequence**	**Arm Code**	**Lock ALPIDE Control**
Trigger	True	0x0	Yes
Pulse	True	0x1	Yes
Pulse/Trigger	True	0x2	Yes
Sync All	True	0x3	Yes
Unarm	False	0x4	Unlock
Absolute Time Reset	False	0x5	No
Spill ID Increment	False	0x6	No
Spill ID Reset	False	0x7	No
Frame ID Reset	False	0x8	No
ALPIDE Bunch Counter Reset	False	0x9	Yes
ALPIDE Global Reset	False	0xA	Yes
ALPIDE Pixel Reset	False	0xB	Yes
ALPIDE Debug	False	0xC	Yes
ALPIDE Readout Reset	False	0xD	Yes

6.5.6 Trigger Manager

The Trigger Manager is the firmware module in charge of the DCS synchronization procedures presented in Table 6.1. The Trigger Manager, based on configuration, operate as either the BTBI master or the slave. The block can initiate various commands via the ALPIDE Control module.

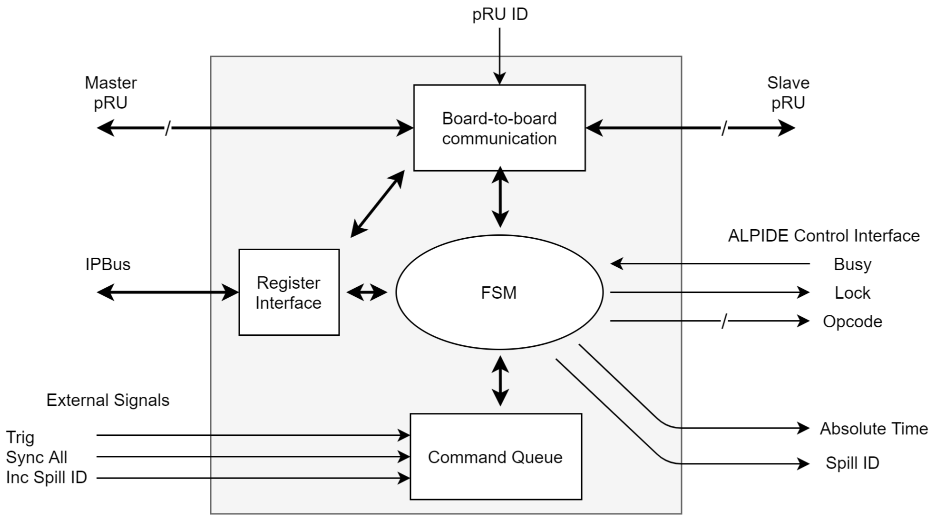


Figure 6.17: *Simplified block diagram of the Trigger Manager module.*

Figure 6.17 shows a simplified schematic of the Trigger Manager module. An FSM controls the command execution and has access to both the absolute time counter and the spill ID value. The Trigger Manager also has direct control of the ALPIDE Control module. However, to ensure that no conflicts arise by accessing the ALPIDE Control module, e.g., that the ALPIDE Control is already in use by the IPBus interface, the Trigger Manager monitors the ALPIDE Control status. Because the Trigger Manager tasks are more critical, i.e., synchronization and triggering are always more important than monitoring, it can lock the ALPIDE Control module by asserting the lock signal. When this signal is asserted, the ALPIDE Control will not initiate any requests from the IPBus interface. The user of the IPBus interface must be aware of this to ensure no critical configuration of the ALPIDE is being done at the same time as the Trigger Manager holds a lock. This feature is also important for certain sequences, e.g., when transmitting a trigger train, keeping all the ALPIDE

Control modules of all pRUs locked to the Trigger Manager interface.

Figure 6.17 also illustrates that the Trigger Manager incorporates the BTBI block. The multi-drop architecture requires that the pRUs are configured at boot-up to act as either master or slave. This is important so that the slaves do not drive the output of the TX/ARM but do drive the ACK signal. The pRUs will use the pRU ID number, set by a physical property on the board, to determine the master/slave status.

Figure 6.18 shows the combined flowchart for the master and the slave pRU. Note how both the master and the slave must consider the state of the ALPIDE Control before initiating commands. To prevent locking the ALPIDE Control unnecessary and to reduce the wait time, not all commands will require this lock.

6.6 DCS Time Budget

Each transaction of the DCS, either a register read/write or a synchronization command, is time-consuming. It is important to have a certain degree of control of how much time is spent by certain tasks, especially the initial configuration of the system, to ensure that the DCS is fast enough to be useful during the operation of the detector. During the configuration of the detector, the vast majority of the transactions will be targeting the ALPIDEs.

Each transaction via the ALPIDE slow control interface requires a fixed amount of clock cycles. However, for each transaction initiated via the memory-mapped interface, we must also consider the latency of the FPGA bus. As noted in Section 6.2.1, several clock cycles of delay is required for any write and read transaction of the ALPIDE control module because of the CDC modules.

Listed below are the exact times needed to perform the various ALPIDE transactions. An ALPIDE broadcast, as seen in Equation 6.2, only needs two register write operations, including the required 70 clock cycle bus stall. This transaction is completed within 2.3 µs. Register write and read, seen in Equation 6.3 and 6.4 respectively, requires more bus transactions. A register write requires that the user specifies both the register address and the data to

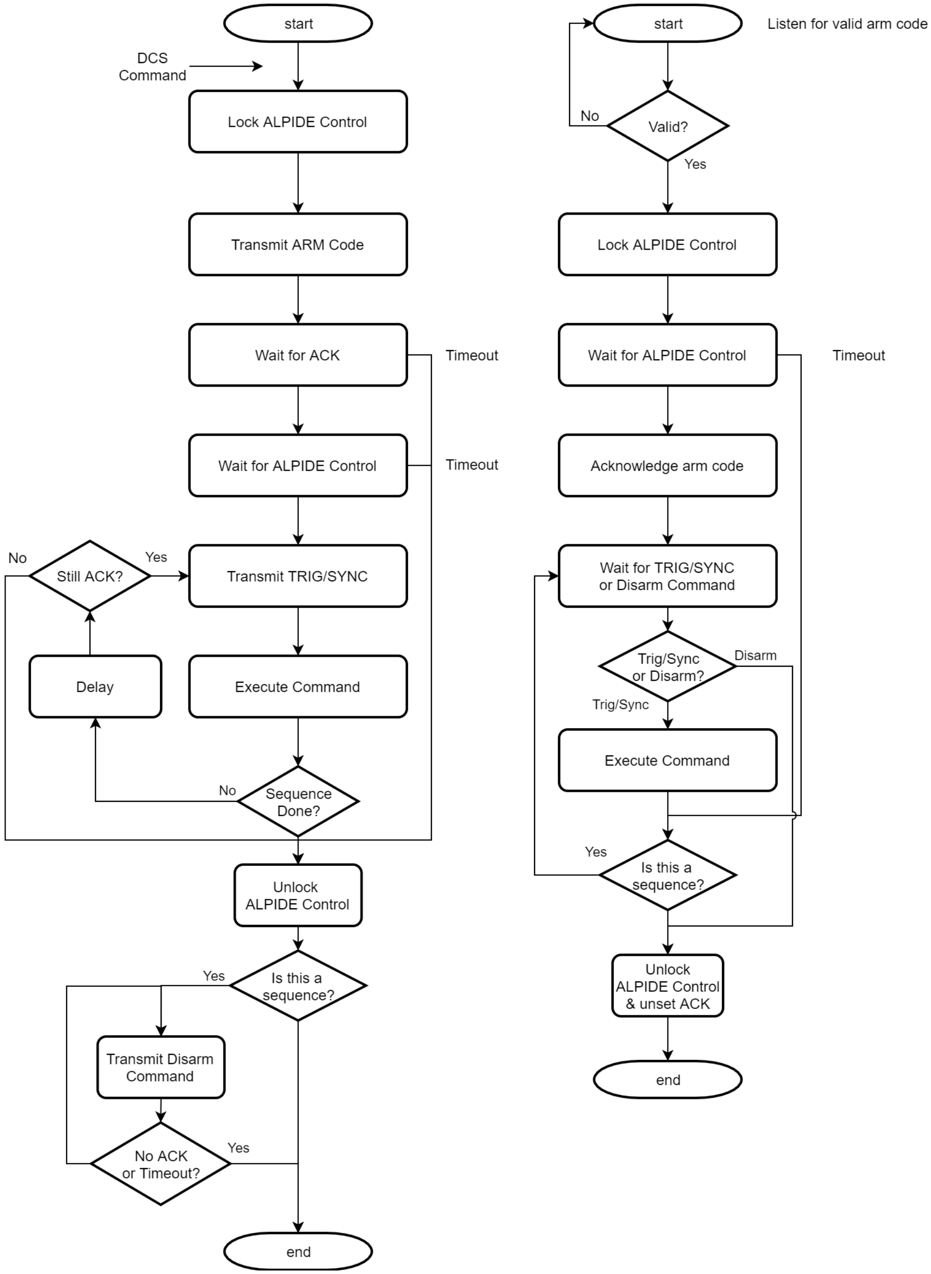


Figure 6.18: *A flowchart of board-to-board interface master and slave. The master-side is to the left, and the slave-side to the right.*

be written. A register read transaction also includes the transfer of the data read from the ALPIDE, as well as the read status flags, via the bus interface to the end-user. This is in addition to the extra clock cycles used for the bus turnaround process and recovering the data. The required time for register write and read, are 2.9 µs and 4.3 µs, respectively.

$$\begin{array}{rcl} & 1 \times & RegWrite \\ + & 1 \times & (RegWrite + 70\ \mathit{cycle\ stall}) \\ \hline = & & 18@31.25\,\mathrm{MHz} + 70@40\,\mathrm{MHz} \\ \hline = & & 2.3\,\mu\mathrm{s} \end{array} \tag{6.2}$$

$$\begin{array}{rcl} & 3 \times & RegWrite \\ + & 1 \times & (RegWrite + 70\ \mathit{cycle\ stall}) \\ \hline = & & 36@31.25\,\mathrm{MHz} + 70@40\,\mathrm{MHz} \\ \hline = & & 2.9\,\mu\mathrm{s} \end{array} \tag{6.3}$$

$$\begin{array}{rcl} & 2 \times & RegWrite \\ & 1 \times & (RegWrite + 116\ \mathit{cycle\ stall}) \\ + & 2 \times & RegRead \\ \hline = & & 45@31.25\,\mathrm{MHz} + 116@40\,\mathrm{MHz} \\ \hline = & & 4.3\,\mu\mathrm{s} \end{array} \tag{6.4}$$

Note that the time to transfer the IPBus read and write requests via Ethernet is not included in these calculations. According to [78], the single-word read/write latency of the IPBus is approximately 250 µs. This latency dwarfs what was outlined above. However, the IPBus μHAL can concatenate multiple transactions into a single packet transmitted to the device where each transaction is executed in order. This combining of up to an order of hundreds of transactions per IPBus Ethernet packet considerably compensates for the single-word latency.

As shown in Figure 6.19, when increasing the number of requests in an IPBus control packet, the time used per DCS transaction is considerably reduced. However, what is also clear, is that the latency introduced by the bus CDC modules are minimal compared to the IPBus and ALPIDE slow control overhead. The best-case time of an ALPIDE register write and read are measured to be roughly 15 µs and 21 µs, respectively.

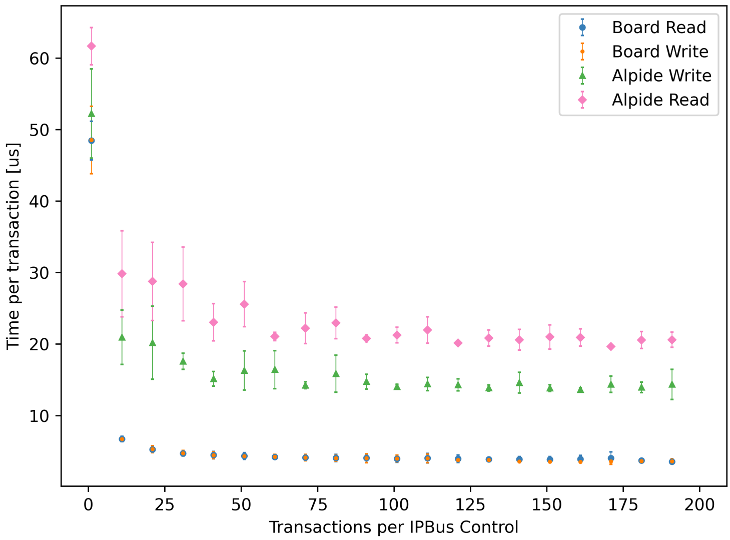


Figure 6.19: *Lab measurement of the time per DCS transaction based on the number of requests in the IPBus control packet. Note the stark diminishing returns when combining more than roughly 50 transactions within a packet. This yields the DCS software some slack when combining transactions in fixed packets.*

6.6.1 ALPIDE Configuration

$$
\begin{array}{ll}
 & Configuration\ Select \\
 & Row\ Select \\
 & Column\ Select \\
+ & Clear\ Pixel\ Select \\
\hline
= & 60\,\mu\text{s} \\
\times & 5000\ pixels \\
\hline
= & 300\,\text{ms} \\
\times & 108\ sensors \\
\hline
= & 32.4\,\text{s}
\end{array}
\tag{6.5}
$$

The most time-consuming task of the DCS is the configuration of the ALPIDEs, particularly the masking of noisy pixels. As the number, and the addresses, of noisy pixels will be unique for each ALPIDE, the masking procedure must be done one sensor at a time. Each pixel masking requires four ALPIDE register write operations. The worst-case scenario is if all ALPIDEs has up to 5000 noisy pixels[14]. Exploiting the IPBus concatenation procedure, one can calculate the time required to mask all the pixels of a (very noisy) layer to be 32.4 s, as seen calculated in Equation 6.5. The masking of pixels can be parallelized between each layer. The result is within what we consider to be a reasonable configuration period.

6.7 Conclusion

The DCS firmware is using register-based communication with both the IPBus Ethernet protocol and the IPBus bus protocol. This has simplified both firmware and software development of the DCS. Using the bust software for keeping documentation and verification up-to-date, facilitated a constructive basis for development and usage of the firmware. The DCS components of the pRU firmware are completed and tested extensively in simulation. Furthermore, the ALPIDE slow control communication module is both tested with multiple front-end designs and has been in widespread use during experiments. However, the trigger and clock synchronization components must be tested in a physical hardware setup to ensure correct behavior. Furthermore, the PCU and its firmware are still under development.

[14] A sensor with between 2100 and 5243 noisy pixels is considered to be of low quality or BRONZE (see Section 7.5.3).

CHAPTER 7

Verification and Testing

The firmware of the DAQ and DCS system has been extensively tested. This chapter describes a subset of these tests. First, we introduce verification as a concept, with techniques and methodology for digital design verification in general and specifically for FPGA design verification. The chapter continues to describe the verification and test setups for the various firmware modules. Furthermore, the integration tests are explained in detail. The chapter then changes focus to the hardware verification procedures and the use of the design.

7.1 Functional Verification

Functional verification ensures that the design operates as intended. This is one of the most important stages of the development cycle and is usually one of the most time-consuming phases.

Verification during ASIC development is often considered to be more critical than during FPGA firmware development. This is because of the considerable costs involved in the turnaround and production of ASIC devices. Most FPGAs are reconfigurable, and thus the firmware can be changed within the field if errors are found. However, FPGA firmware verification remains vital because of the complexity and effort involved with hardware testing. Once an FPGA is configured, it is difficult to observe what is going on inside it. Some tools exist for this purpose, like the Xilinx integrated logic analyzer. However, these do not allow for full control and observation of all the internal signals. They are often slow to operate and awkward to set up. They usually also require a significant amount of FPGA resources. These tools might, therefore, be impossible to use when the actual design occupies most of the FPGA resources. These are some of the reasons why FPGA firmware developers extensively use software simulation and other types of software-based verification before testing the resulting hardware.

7.1.1 Verification using Testbenches

Simulations of digital designs are generally done within a testbench architecture. A testbench contains, at minimum, a test sequencer and one or more instances of the unit under test (UUT). The test sequencer is a piece of code that provides stimuli to the UUT and checks and validates the UUT outputs following the stimuli.

The test sequencer can benefit from extracting the UUT interfaces to separate instances. This will significantly simplify the test sequencer code and allows the possibility of reusing the interface code in other testbenches. An extracted interface is called a Bus Functional Model (BFM). A BFM contains functions and procedures for interacting with a specific interface of the UUT. The test sequencer, thus, calls on these to interact with the UUT.

7.1.2 Test-Driven Development

Test-driven development (TDD) is a design paradigm used in software development. The paradigm states that one should design and write the tests before each feature is developed. This procedure can encourage simple design and increases confidence with the design. The development lead time is often reduced with the use of TDD [84]. Although TDD is usually associated with software development, the paradigm is now also used by some digital designers [85]. Specifically, for HDL developers, TDD discourages the use of the waveform to do manual verification.

During the development and verification of the pRU firmware, we have strived to follow the principles of TDD. This has resulted in multiple module-based verification testbenches, which have significantly contributed to relatively swift development and verification, in addition to high confidence in the resulting design. In some cases, however, TDD was rejected because of the indefinite module requirements during development. Specifically, this involved the development of the high-speed data input stage since this was designed during a research stage of the development (see Section 7.4.2). Also, for some of the data flow modules, because of changing requirements, the verification code was added after the design stage.

7.1.3 Bitvis Verification Library

The verification testbenches of the pRU firmware extensively use the Bitvis verification library Universal VHDL Verification Methodology (UVVM) [86]. The UVVM library provides a vast range of useful features for software simulation of digital design. For instance, UVVM includes a range of BFMs ready to be used with any custom test sequence. The library also provides features for logging, error checking, and randomization. The randomization functions are used to implement constrained random checking of some of the complex interfaces (see Section 7.2.6).

UVVM includes a particularly critical feature allowing concurrent control of multiple interfaces: the VHDL Verification Component (VVC). A VVC is, in principle, a BFM. However, the VVC includes a queue, making it possible to send a list of operations to the VVC, and while the VVC performs these tasks, the test sequencer can execute other transactions. This is particularly useful for testing how a module reacts when two of its interfaces are active simultaneously.

7.1.4 Design Correctness

It is not trivial to assert whether a design is *correct*. Naturally, the correctness definition is related to the module specifications. However, sometimes the specifications may be incomplete. This is especially true regarding multi-interface modules. Multiple interfaces add another layer of complexity to the module, and it might not always be clear how the module should tolerate situations where both interfaces are active simultaneously. Furthermore, in practice, it might be challenging to translate the simulation result metrics to match the linguistically expressed specification document objectives.

Therefore, the code coverage metric is often used to aid in the determination of correctness during digital design verification. Code coverage is a quantitative measure of the design and test code provided by the simulation software. High code coverage tells the verification engineer that the tests have activated a large part of the actual design code. Code coverage differentiates between several criteria. In particular, code coverage offers metrics for statements, branches, and FSM states and transitions. It is also possible to obtain a percentage of possible input combinations that have been activated for expressions and conditions. This metric is called Focused Expression Coverage (FEC). However, it can be difficult to obtain a high FEC score without sacrificing the verbosity and readability of the code. Thus, the pRU firmware verification focuses on the three former criteria. However, code coverage is an incomplete measure of correctness as it does not consider unimplemented features and does not ensure that the output of a given stimulus is verified.

Therefore, it is vital to write specification as a set of testable statements, and the pRU firmware is designed with this in mind. The pRU firmware verification is done with both code coverage and an emphasis on TDD to ensure correctness.

The pRU firmware testbenches use a combination of direct and random testing. Direct testing involves a set of pre-defined test cases and test data. Random testing is included in case we are incapable of predicting how the design might fail. The cases might include random data sets, but also random timing of events.

7.2 pRU Firmware Module Verification

As required by TDD, each main module of the pRU firmware is tested in isolation. A consequence of having a single person both designing and testing the firmware modules is the loss of the black box perspective. This might have caused hidden problems within the firmware modules, as the designer might have an obstructed view of the design. Evidently, two sets of eyes are beneficial to the verification process.

7.2.1 Bus Interface

Most of the pRU firmware modules incorporate an IPBus interface. As discussed in Section 6.3, the testbench creation for the bus interface is automated. These testbenches provide a code coverage of 100 %.

7.2.2 Data Flow

To verify the firmware data flow modules, significant efforts were made to develop a set of Python-scripts to generate a complete and realistic data set. These scripts are used to generate stimuli to both the data flow chain and the expected output data. The scripts are organized as follows:

(1) An ALPIDE pixel-data generator. This software block can generate both randomized and pre-defined data.

(2) An ALPIDE frame generator. The frame generator is simply generating an ALPIDE data frame based on pixel input data provided by the pixel-data generator. A large subset of inputs can be provided to ensure that the output data covers all possible situations. For instance, it is possible to choose whether the ALPIDE has the clustering feature enabled. The software then generates the data in the ALPIDE data format.

(3) An 8B/10B encoder. This is simply a block that encodes the generated ALPIDE data format to 8B10B encoding. This is required to test the input stage of the firmware data flow.

(4) A pRU data frame generator. This block packs ALPIDE data into the pRU data format. This data is needed to validate the output of the data flow firmware. The block can modify the data based on a substantial set of inputs. For instance, it is possible to set various error flags and manipulate the time tagging.

A sketch of the data flow test setup is shown in Figure 7.1. The test sequencer reads the pre-produced ALPIDE data set from a file and transmits it to the UUT via a data injector module. The test sequencer uses the bus interface via

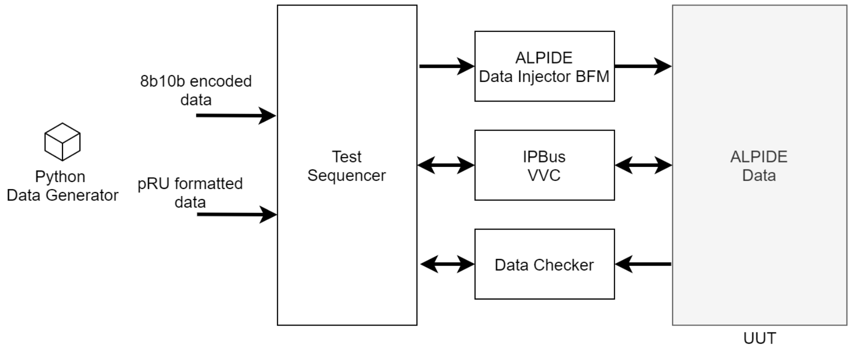


Figure 7.1: *The ALPIDE Data testbench setup.*

the IPBus VVC[1]. This allows the test sequencer to enable the various features of the module.

For each test case transmitted to the UTT, the test sequencer will check that the pRU formatted output matches the data pre-produced by the Python scripts. Naturally, some data content will differ from the pre-generated data, e.g., time-tagging. The data checking, therefore, ignores these fields. All fields where the value can be predicted by the Python software ahead of time are tested, including all error flags.

However, the test strategy does not only rely on checking the output data. The ALPIDE Data module and its submodules incorporate a significant number of various counters, and these can be used to verify that the module interpreted the input data correctly. For instance, the module will count the number of bytes in each ALPIDE frame. Furthermore, the different error conditions of a frame are associated with separate counters. For each test case, the test sequencer checks whether the counters have the correct values.

A quite extensive set of test data is generated by the Python software to ensure that the design is correct. The code coverage of the data flow simulation is close to 100 % for all criteria. The remaining untested code has been manually verified to be redundant statements that are required by either the VHDL language or by the Xilinx synthesis tool.

[1]The IPBus VVC was developed based on the community-provided Wishbone VVC.

7.2.3 Priority Offloader

Figure 7.2 shows a sketch of the priority offloader test setup. In addition to the module itself, the test uses several instances of the data formatter buffer entity. These instances are used to ensure that the priority offloader can handle multiple data sources concurrently, one of the module's main requirements.

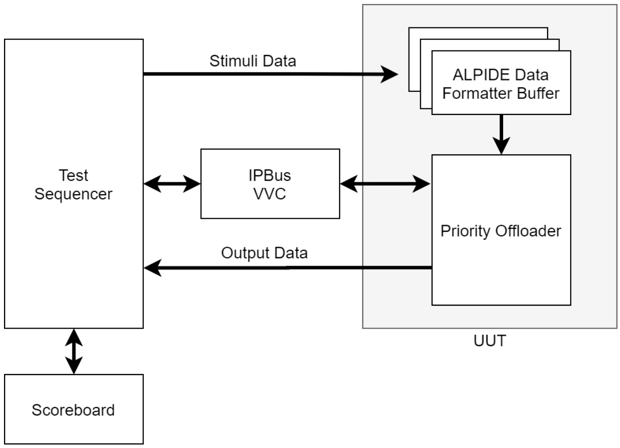


Figure 7.2: *The Priority Offloader testbench setup.*

With the Bitvis Scoreboard, one can test the data coherency based on the expected output. Since the priority offloader will scramble data words from different channels, one cannot expect that the output is precisely in the order as initially input to the scoreboard. However, the scoreboard can be configured to allow out-of-order words and, thus, only test whether all words have been recovered from the priority offloader.

The priority offloader test uses only random data, as the data content itself is irrelevant to the UUT. However, significant efforts have been made to provoke errors by manipulation of the buffers. The use of the out-of-order feature depends on the test case. E.g., to test that all the words of a pRU frame remain unscrambled, we require that the entered data order is fixed. By filling several modules with a known amount of data, we can predict which buffers will be offloaded first and transfer this data into the scoreboard first. When this test repeatedly succeeds, done with differing initial variables, we are comfortable that frame content will not be scrambled. Furthermore, the tests achieve an acceptable code coverage score (above 95 %) for all the components of the priority offloader block.

7.2.4 ALPIDE Control

The ALPIDE Control module is responsible for the ALPIDE slow control communication and is critical to ensure proper control of the detector. The testing of the module must be extensive to rule out potential errors. It is imperative to ensure that all possible errors are recoverable, meaning that it is impossible to lose control of the detector. It is also essential to check the performance of the phase alignment circuitry. However, we cannot expect the module to fix problems that arise with malfunctioning ALPIDEs, and it is sufficient that the module reports that the sensor is dead when it receives no reply.

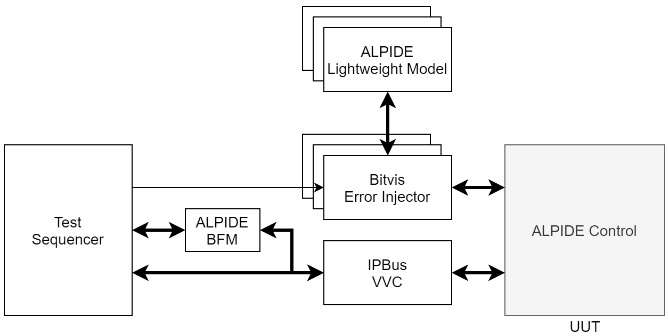


Figure 7.3: *The ALPIDE Control testbench setup.*

The ALPIDE Lightweight Model (ALWM) is a minimal SystemVerilog implementation of the ALPIDE digital circuitry and was provided by the developers of the ALPIDE. The model is particularly useful for testing the slow control interface as it incorporates all registers of the ALPIDEs digital periphery. The block is found within the test setup sketch, shown in Figure 7.3. The model also includes a data generator block, replacing the pixel matrix logic. However, the data generator is entirely random and does not produce errors in the data stream, and thus, it could not be used to ensure full coverage of the ALPIDE Data module.

Another characteristic of the ALPIDE Control test setup is the use of the error injector provided by the UVVM library. The module allows the test sequencer to inject various errors into the path between the ALPIDE Control and the ALWM. In the case of the ALPIDE Control verification, it is especially useful because it enables the test sequencer to add jitter and latency to the link, testing the phase alignment capabilities of the ALPIDE Control module.

A full BFM is developed for the communication with the ALPIDE via IPBus and the ALPIDE Control module. This module includes all the valid transactions that the ALPIDE Control can perform.

Several ALWMs and error injectors are instantiated in the test setup. This is to test the multi- and broadcasting features of the ALPIDE Control module. This is also used to check that ALPIDE Control will handle multiple links with different amounts of jitter and delay.

The implemented tests achieve a code coverage close to 100 %.

7.2.5 Power Control

The Power Control module is a block implemented in the PTB (see Section 7.5), but will also be used in the PCU. As the module controls power monitoring sensors via I2C, the test setup includes the instantiation of several I2C VVCs.

7.2.6 UDP Stack and Data Transfer Protocol

The test setup for UDP and pDTP is the most comprehensive of all the modules. This is because of the complexity involved with Ethernet protocols. The test setup, seen in Figure 7.4, seems relatively simple with only a few instantiated blocks. However, a complete VVC is developed to support XGMII communication with the 10GbE MAC. Also, significant efforts were made to develop support packages for the different protocol levels (see Section 7.2.6.1).

The MAC is connected to the pDTP block, which includes the full UDP stack. The pDTP block receives data from an AXIS buffer. The buffer is filled with random data, which is also entered into the scoreboard. This way, we can use the pDTP protocol to read data from the block and control the content using the scoreboard.

The tests emphasize verifying the behavior of the pDTP blocks and not the UDP stack. This is because the UDP is provided as an IP, and the designer of the IP supplies tests for these blocks. However, for completeness, a coverage-driven constrained random data set is used to verify the UDP stack behavior. The pDTP block tests, however, are focused on direct testing, as the block is agnostic to the data content. Similar to the other modules, these tests achieve a high code coverage score.

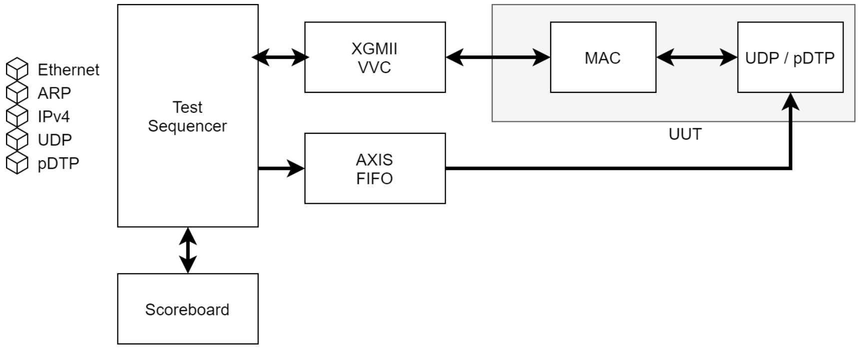


Figure 7.4: *The 10GbE and pDTP testbench setup.*

7.2.6.1 Ethernet Protocol Packages

The Ethernet protocol packages (Ethernet, ARP, IPv4, UDP, and pDTP) have several parallels. Most of the functions are for generating or parsing packets or parts of packets. Furthermore, the packages depend on each other. For instance, to generate a UDP packet of a data array, the UDP package will call on the IPv4 package to add proper encapsulation. Likewise, the IPv4 package function will call on the Ethernet package. The Ethernet package can then encapsulate the packet in the correct format, calculate the frame check sequence code, and more. At all stages, sanity checks are performed. The development of these packages was demanding and time-consuming. However, we expect that the lead time of the pDTP block in total was lower by using these for verification.

7.3 Integration Tests and Top-level Verification

For FPGA firmware, top-level verification is usually the most difficult to do correctly. As the complexity of the design grows, the simulation time increases. In some cases, the design might be so large that several days are needed to test even simple features. Also, with top-level simulation, we lose the possibility of adding stimulus to many parts of the system. Top-level simulation is, in any case, important as it can uncover errors within the inter-module interfaces that cannot be observed with simple module testing. Naturally, full code coverage might be difficult to attain with top-level simulation and is therefore not a specified objective.

The pRU firmware top-level test setup is sketched in Figure 7.5. The test sequencer re-uses BFMs and VCCs used in the tests discussed above to interface the top-level. The test sequencer is interfacing the MAC of the 10GbE interface and not the MGT or the PHY instance. This is done because it was considered too complex and time-consuming to develop a VVC with a continuously changing output, which is a requirement for the 10GBASE-R protocol. This approach was thought to be sufficient as long as the design simply worked in hardware. This is because the PHY, MAC, and the example MGT instance are provided as parts of the UDP IP, of which each part had been tested previously by the developer. The interactions between them should, therefore, be consistent.

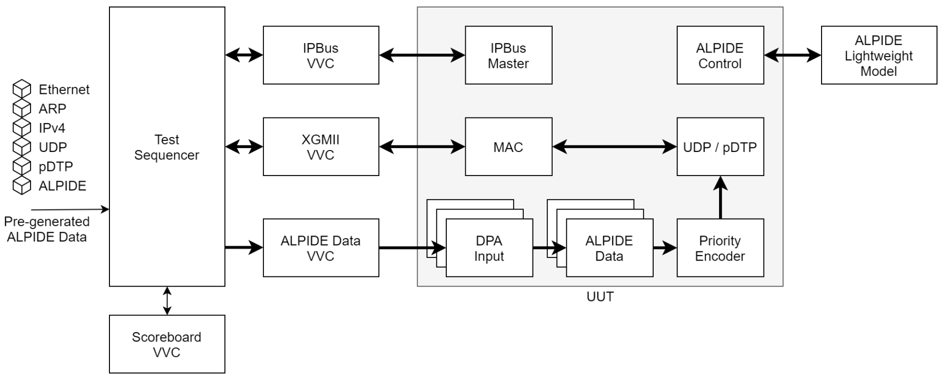


Figure 7.5: *The top-level testbench setup.*

The top-level test cases do not aim to fully test all features of the hardware. The tests verify that:

(1) The full data flow works, meaning that no data is lost or corrupted when transmitted through the full data chain, including the input and output stage.

(2) The ALPIDE Control and ALPIDE Data time tagging feature work as expected.

(3) The ALPIDE Control and Trigger Manager interface can be used to trigger and synchronize the ALPIDEs.

(4) The full bus interface works, meaning that no issues arise with the address-decoding or the CDC block.

(5) No unexpected errors occur with the module interaction.

7.4 Hardware Verification

The pRU firmware has, as explained in Section 2.7.2, been tested using the VCU118 evaluation kit. Although the FPGAs of the VCU118 and the pRU differ somewhat, we are confident that no severe issues will arise during the port, as the FPGAs have the same architecture. We are furthermore assured by the use of most of the firmware by the PTB (see Section 7.5).

7.4.1 FireFly FMC

An FPGA mezzanine card (FMC) was developed to allow the connection of FireFly cables to the VCU118 evaluation kit for testing. The FMC is shown in Figure 7.6. The card uses the Samtec VITA 57.4 FMC+ standard to connect to the VCU118 board. With the FMC, one could test different coupling and biasing circuits to optimize the high-speed link performance. Furthermore, two FireFly connections were added; one interfacing MGT-pins and one for regular I/O-pins. The FMC card also includes several other types of cable connectors. These were intended to be used to test an eventual direct connection of the string FPC to the readout unit.

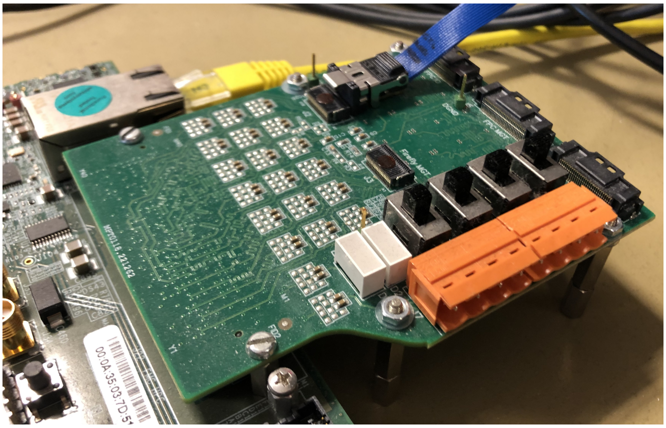


Figure 7.6: *The FPGA Mezzanine Card allowing connecting FireFly cables to the VCU118 board. The figure shows the FMC connected to the VCU118 board with a FireFly connected to the I/O-port.*

7.4.2 High-Speed Links

Because of the complexity involved, it was considered too time-consuming and unrealistic to test the DPA logic properly using software simulation. Thus, most of the verification and testing were done in hardware with the aid of integrated logic analyzers and oscilloscopes. The DPA feature was first tested using signal loopback. By using a known PRBS pattern, one could verify that the DPA logic was sampling the data correctly. With a data rate of 1.25 Gb/s, these tests were promising. During a 24-hours test, no errors were observed, yielding a BER of $< 9.64 \times 10^{-15}$.

However, as noted several times in this thesis, the ALPIDE high-speed data link suffers from potential jitter. It was, therefore, imperative to test the DPA using the real device. Furthermore, as triggers and occupancy affect the ALPIDE performance, as well as a broad range of other variables[2], significant efforts were made to cover the full test space.

The DPA testing occurred simultaneously with testing of new front-end designs. This complicated the testing of both the DPA and front-end as it was difficult to

[2]The high-speed driver strength, pre-emphasis strength, clock gating and the PLL bandwidth are some of the parameters that have been observed to have an effect on the high-speed link performance in the lab.

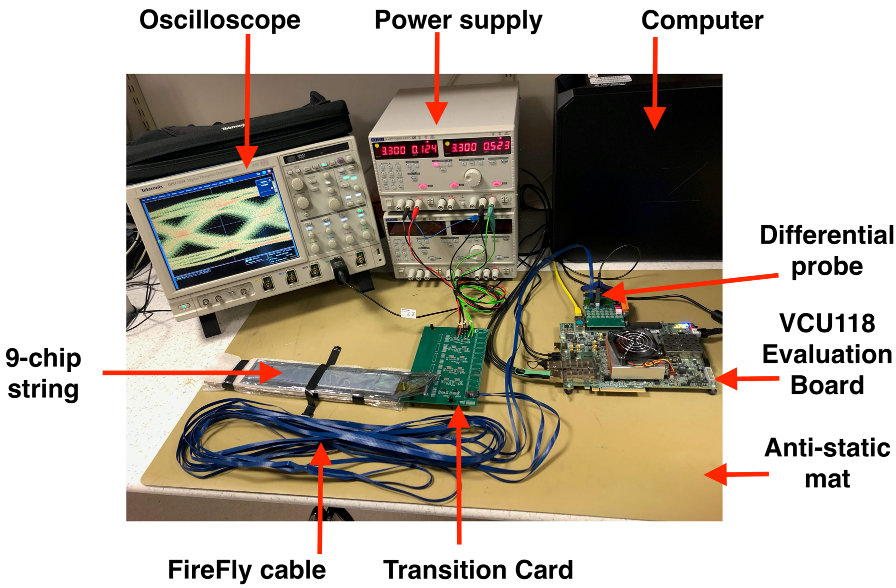


Figure 7.7: *The lab test setup. Figure provided by courtesy of Tea Bodova [39].*

interpret errors and determine which part of the design was the culprit. However, as new front-end designs were completed, the DPA testing results improved, which raised confidence that the cause of the errors had been found. Also, because many variables and parameters affect the high-speed link performance, our goal was to find settings that would validate the DPA performance, not to ensure that DPA worked at all times.

Figure 7.7 shows the lab test setup. It shows the VCU118 evaluation kit, with the FMC card (see Section 7.4.1), connected to a 9-chip string via a FireFly cable and an early version of the Transition Card. The FireFly cable length was shown by testing to be one of the most significant contributors to the signal deterioration. Early tests indicated an eye diagram opening height reduction of up to 50 % per cable meter. To ensure that the system would operate with the 3 m cable, tests were run with 5 m long cables.

Figure 7.8 shows the results of one of the many tests performed. In this test, all nine ALPIDEs on a string were simultaneously triggered with an artificial 1 % occupancy. Each step of the test was done with changing ALPIDE high-speed

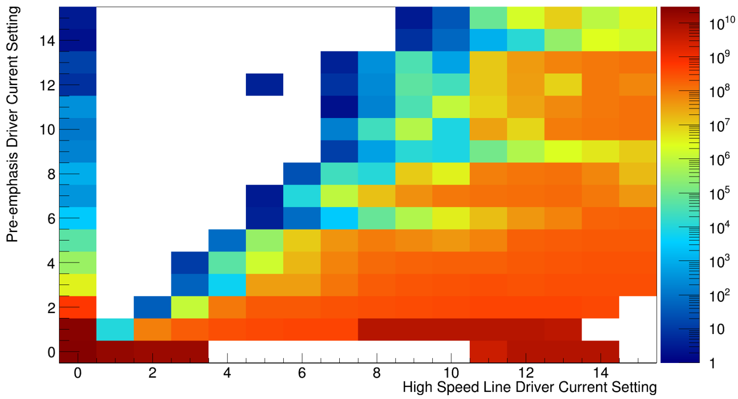


Figure 7.8: *Test results showing decode errors when testing the high-speed data sampling with 5m FireFly cable and varying the link driver settings of the ALPIDE. Setting combinations with no errors observed are shown in white. The bottom area indicates error-free settings, but on the contrary, with these settings, the signal is too weak or noisy for the DPA logic to lock to or detect data. The test results show that many driver settings produce zero sampling errors.*

driver and pre-emphasis strength. For each setting, 85 million triggers are transmitted to the ALPIDEs with 10 µs intervals[3]. The plot shows the number of errors observed for a single chip. The results show a typical case, where there is a *band* of settings that provoke errors. However, there is also a band of settings that results in zero sampling errors. Each setting without errors constitutes a BER of $< 9.8 \times 10^{-13}$. The same test was performed with fixed settings over a 66 hour period, i.e.,with 2×10^{10} triggers, yielding a BER of $< 3.5 \times 10^{-15}$.

The results of the range of tests performed confirm that:

(1) The DPA technique is sufficient for our purpose.

(2) The front-end design is acceptable.

(3) The combined activity of the chips on a string does not cause a detrimental voltage drop.

(4) Cross-talk between the high-speed links does not constitute a problem.

[3] 85 million triggers are roughly 20 % of the total triggers to all sensors of the DTC required for a 2D projection.

With the help of the internal logic analyzers of the FPGA, it was possible to observe the effect of the DPA logic failing to adjust to jitter. For instance, in Figure 7.9 we see a so-called fake frame. In this case, we know that the ALPIDE is only transmitting comma-words. However, what we observe is that the incoming data is interpreted as an ALPIDE frame. In addition, we observed that this error occurred directly following a trigger. Therefore, the error can be attributed to a voltage drop on the ALIPDE, and the resulting clock jitter (recall Section 3.2.4.3).

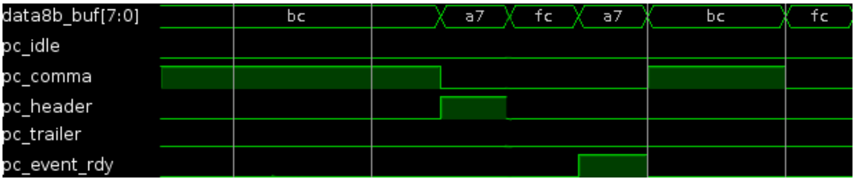


Figure 7.9: *Example of a situation where a fake frame is produced caused by jitter on the ALPIDE high-speed data link. Noise is causing the data sampled to resemble the beginning of an ALPIDE Data frame. However, the correct data should just be comma words. When the data words following the fake header does not comply with the protocol, the frame is aborted and tagged with an error. The event is captured using the Xilinx integrated logic analyzer.*

7.4.3 Slow Control

The ALPIDE slow control interface is tested with the same setup as the high-speed data link. The firmware is stress-tested by transmitting write commands to random registers and then reading the value back. Writing and then reading a register was done 200 million times without any errors observed. Adding the bits transmitted in both directions, this constitutes a BER of $< 5 \times 10^{-11}$. The read-direction-only BER is $< 2 \times 10^{-10}$. Furthermore, the ALPIDE Control module has been extensively provoked to fail, for instance, by disconnecting the FireFly cable during transactions. These efforts have confirmed that the module operates as desired.

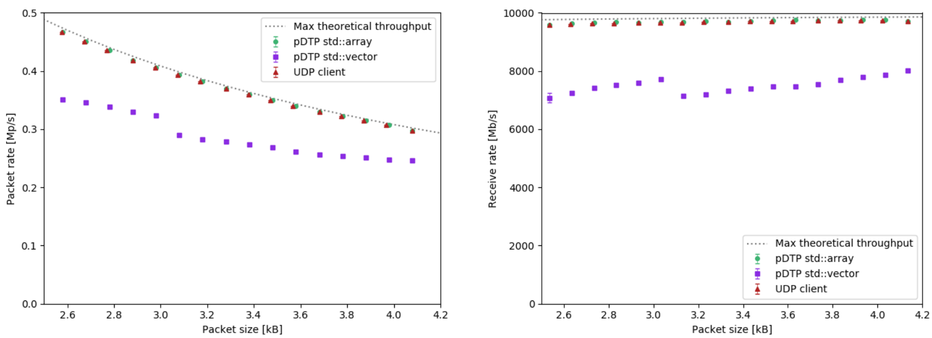
7.4.4 GbE and IPBus

The testing of the IPBus interface was done by utilizing the SGMII PHY chip of the VCU118 kit. A desktop computer was connected directly to the board via a standard network interface controller. No optimization was done on the computer-side of the system. Actual test results have been shown earlier in Section 6.6.

Investigation of both GbE and 10GbE protocols was needed to analyze errors that were not detected during the simulations. Most of these errors, however, involved the configuration of the physical interfaces and the MGTs. Aiding this analysis was the use of tools like Wireshark, allowing the observation of packet content.

7.4.5 10GbE and pDTP

Figure 7.10 shows the test results of the data transfer between the pDTP server and the pDTP client. The tests were performed in a simple environment with a typical desktop computer running CentOS with a standard Intel 10GbE X710-DA2 network interface controller. Copper SFP+ cables were used to connect to the VCU118 evaluation kit configured with the pRU firmware. Tests were run with the pDTP server buffer artificially filled with testing data. The tests show that we achieve a throughput near the theoretical maximum. The tests also show the importance of optimal memory management by the pDTP client software.



(a) *Packet rate.* **(b)** *Receive rate.*

Figure 7.10: *The measured pDTP packet and receive rate based on requested packet size.*

7.5 Production Testing

One of the most challenging stages of the pCT development is the production of the front-end electronics, i.e., the bonding of the ALPIDEs, chip cables, and strings. For each step of the production, one must verify all connections and that nothing is damaged in the process. The following section is dedicated to the efforts done to streamline the testing process.

7.5.1 Production Test Box

The PTB is a device developed to aid with the different stages of testing. The PCB is a custom device utilizing a Trenz FPGA module, a separate PCB that includes an FPGA and all necessary voltage regulators and clock generators required to operate the FPGA. This module is connected to the PTB board via high-density Samtec connectors. The PTB employs a Xilinx Zynq UltraScale System-On-Chip (SoC) FPGA. This FPGA module was chosen because of the low cost of the module and because the I/O-bank architecture is identical to the other pCT hardware boards. The latter allows the use of the same FPGA firmware for sampling and data processing. The PTB board uses Ethernet for both control and data offload. Both of these tasks are done by the hard processing system of the SoC and uses the architecture as outlined in Section 6.1.1.

One important feature of the PTB is the ability to test the integrity of the chip cable bonds to the ALPIDE before mounting to the 9-chip string FPC. This is done by attaching the chip cables to TCP frames and inserting them into the connector of the PTB. The interface allows electrical connection to all the ALPIDE bonds used, e.g., power, slow control, clock, and high-speed data. Figure 7.11 shows how a chip cable is inserted into the TCP frame and socket.

The PTB supplies power and monitors the bus voltage and the current usage of the ALPIDEs connected to the chip cable. The complete process is controlled by the FPGA. The power monitoring is done by INA226 chips managed via I2C by the FPGA. Schematics of the power supply and monitoring scheme is shown in Figure 7.12.

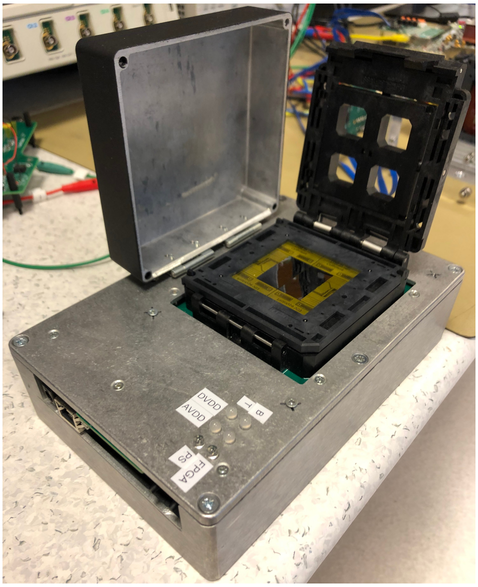


Figure 7.11: *The Production Test Box is used to test chip cables and strings. The Figure shows the Yamaichi TCP socket with the lid open with a chip cable with two bonded ALPIDEs inserted.*

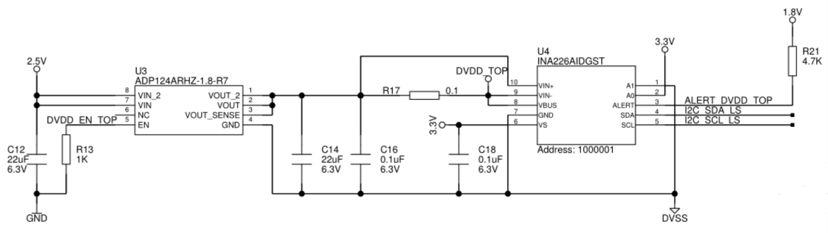


Figure 7.12: *Schematics of the PTB power supply and monitoring.*

The remaining ALPIDE pins are connected to the FPGA with the same procedure as the pRU design. The FPGA, thus, has full control of the sensor capabilities. Using this, the PTB can not only test that the bonding has succeeded but also the state of the ALPIDE chip itself (see Section 7.5.3).

The next iteration of the PTB also incorporates a FireFly connector. This allows the testing of the bonding of the chip cables to the string via a TC.

7.5.2 Mini-TC

The Mini Transition Card (Mini-TC) is developed to be used for testing of strings and slabs. Because the TC ZIF-connectors support relatively few insertions[4], these connectors will be damaged after testing just a few strings. We can also imagine that a single string needs to be tested several times because of errors during mounting. The Mini-TC is designed to be used solely during testing and only connects the three strings of a slab. The simplicity of the board makes it possible to produce and test many units so that it can be replaced after the ZIF-connectors are damaged. In Figure 7.13, we see a 3-string slab that is connected to the Mini-TC to be tested. The power consumption of the slab is monitored externally during testing by interfacing a typical lab power supply via USB [39].

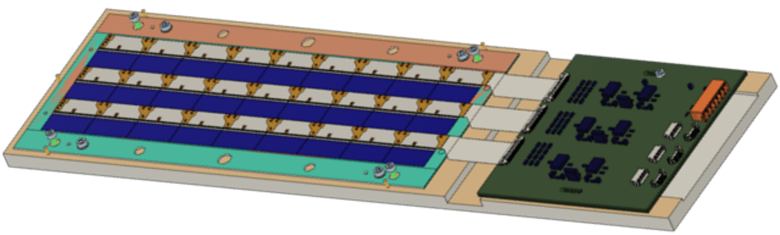


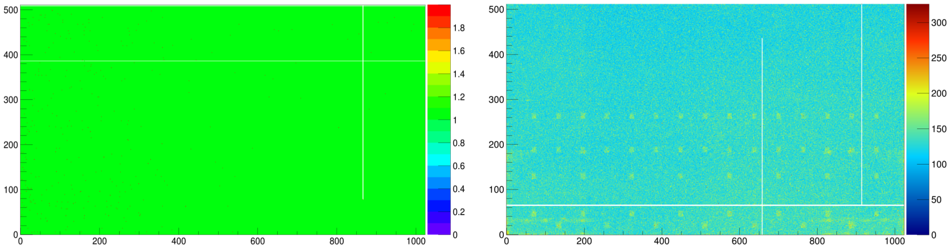
Figure 7.13: *3D model of the Mini-TC used to test a slab. Figure is provided by courtesy of Tea Bodova.*

7.5.3 ALPIDE Classification

The ALPIDEs are classified according to their performance and graded by several criteria. The criterion with the worst score sets the overall chip score. On delivery to the production stage, the ALPIDEs are already tested and classified by the scale of GOLD, SILVER, BRONZE, or NOK/NOTOK. In addition, the pCT group adds another grade called WOOD, between BRONZE and NOK, allowing the use of some sensors previously classified as NOK. The WOOD grade allows sensors with more noisy pixels and these sensors can be

[4]The PTB employs a Molex ZIF-connector that can withstand roughly 20 mating cycles [39].

used in the outer areas of the detector. The classification is used to determine where in the detector the sensors are used. For instance, the front-trackers of the detector will only employ GOLD standard sensors.



(a) *Analog scan.*

(b) *Threshold scan. The B-pads of the ALPIDE affect the sensitivity of the pixels in the surrounding area and can thus be observed in the lower middle of the chip in this scan.*

Figure 7.14: *Test results of two test scans. The results are from two separate ALPIDE sensors. Both sensors have dead areas. In these two cases these areas are manifested as dead rows and columns.*

7.5.3.1 Test Sequences

To test the various criteria of the ALPIDEs, several given sequences are defined. For instance, it is possible to test the quality of the analog front-end of the sensor pixel matrix. Figure 7.14 shows the qualitative results of two of these sequences as a plot. The analog scan tests, shown in Figure 7.14a, checks that each pixel's analog circuitry is functioning properly. The test is done by using the ALPIDEs internal pulsing logic[5], and directly after transfer a trigger to generate a strobe window. Pixels that do not report a pixel hit are perceived as malfunctioning. The threshold test in Figure 7.14b determines how sensitive the pixels are. The pixel discriminator threshold is set globally. However, because of process variations and voltage and temperature fluctuations, each pixel's threshold will vary somewhat. By pulsing the analog circuitry of the pixels with varying strength, we can detect the differences from pixel to pixel

[5]With the internal pulsing logic one can artificially create a pulse within both the analog and the digital part of the pixel.

by counting the number of hits. Pixels with the highest number of hits are the most sensitive.

Another important test is the noisy pixel test. This test is performed by triggering the ALPIDE a given number of times without performing any pulsing or masking. Single-pixel hits of the same pixel in these tests indicate that the pixel is noisy. The number of noisy pixels is one of the biggest contributors to a reduced classification grade. A noisy pixel can be masked to reduce the noise in the actual data, but many masked pixels might cause the loss of wanted data. The resulting data from the scans are analyzed using the CERN ROOT data analysis framework [87].

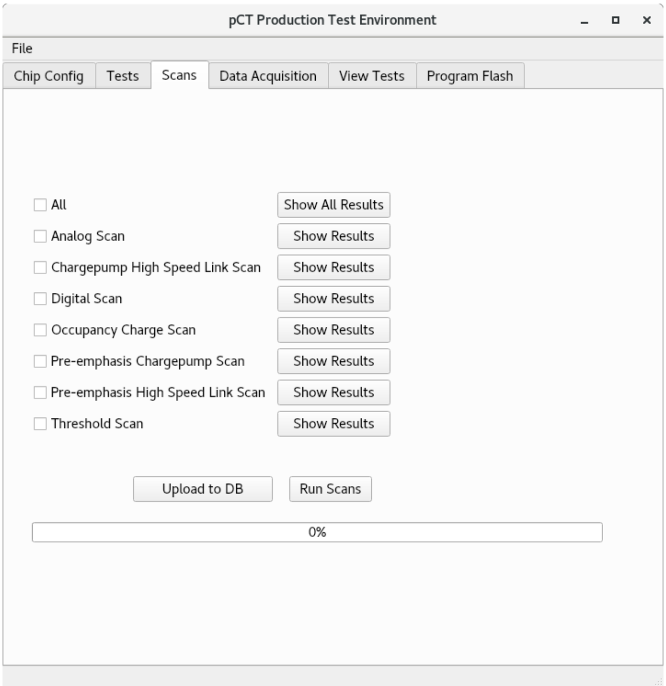
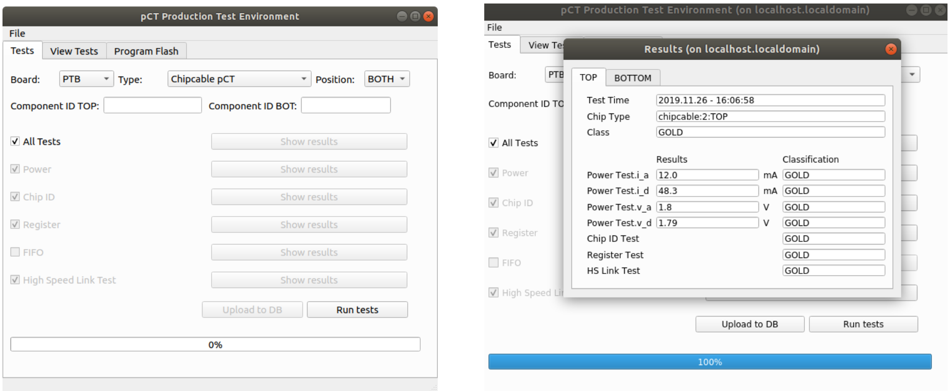


Figure 7.15: *GUI view of the selection of the various test sequences of the ALPIDEs. Figure provided by courtesy of Viljar Eikeland.*

7.5.4 Production Test Software

The production test environment software is developed to be used during the testing of chip cables, strings, and slabs. The environment is intentionally designed to be user-friendly for non-technical users. Figures 7.15 and 7.16 show screenshots from the software. The software can run a large range of different tests, change parameters of the testing hardware, and more. In addition to the communication with the PTB and the VCU118 hardware, the program is also able to control and monitor external power supplies used in the testing of strings and slabs.



(a) *Test selection.* (b) *Test results.*

Figure 7.16: *Screenshot of the pCT production test environment software suite.*

The production test environment is also connected to a MongoDB database. MongoDB differs from SQL-type databases in that it stores unstructured data, which makes it highly adaptable [88]. As the testing will be performed at several locations in different countries, it is important that the metrics from the testing is stored and shared. Each test result is time-stamped and tagged with the tester's name. The results from any location can also be viewed within the software.

7.6 Version Control and Continuous Integration

An integral part of the pRU firmware development has been the use of the version control tool git. Each production version of the firmware is tagged by both a version number and with the top-level name, e.g., `VCU118-v2.0-beta2`.

This has been imperative to keep track of changes and also made it manageable to develop firmware for multiple boards within one project without too many difficulties.

Furthermore, git is used in conjunction with GitLab, a web-based developer tool. GitLab incorporates a continuous integration tool, allowing standardization of the simulation and build process. By constructing a GitLab build server with all the required FPGA toolchains installed, the pRU firmware is automatically verified and built for each new version pushed to the repository main fork. The user of the firmware can download the built FPGA image directly from the pipeline webpage or via a scripting environment.

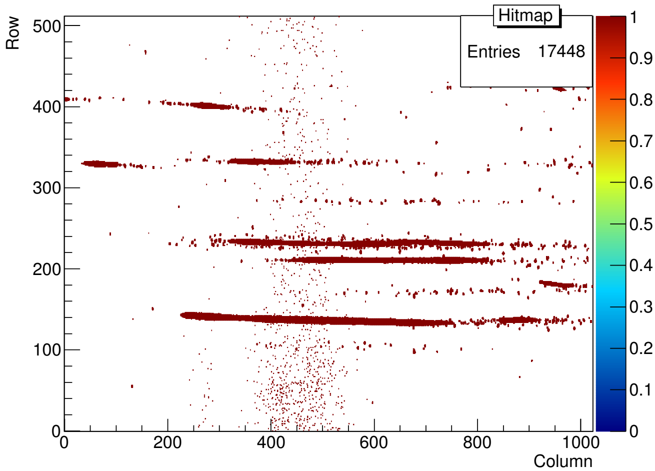


Figure 7.17: *Example hit map data captured at the Heidelberg HIT Facility. Data shows carbon ion-particles traversing in the sensor's sensitive layer coming in from the right. The figure was first published in [73].*

7.7 Beam Tests

The pRU firmware has been in use for experiments since the beginning of its development. This way, the firmware users have been involved in discovering errors and non-optimal solutions. One particular experiment was performed at the Heidelberg HIT facility during the summer of 2019, employing the pRU firmware with the VCU118 hardware. Figure 7.17 shows example data taken from this experiment. The experiment was set up to use a carbon beam entering the side of the ALPIDE. The aim was to deposit a large dose within the chip and to test the performance of the readout electronics and software. The results showed nice long tracks caused by the ionizing carbon ions and that the readout system was functioning. This experiment was done prior to the optimization of the front-end design, resulting in some corrupted data [73]. This did not, however, constitute a substantial percentage of the data frames and did not affect the results of the experiment.

CHAPTER 8

Conclusion and Outlook

8.1 Conclusion

When finished, the pCT detector will demonstrate whether it is feasible to employ pCT imaging clinically and if it will improve the accuracy of particle treatment dose-planning in the clinic. The presented work has largely focused on the study of how to achieve a reliable and efficient DAQ and DCS for a general particle detector. The results of the efforts have significantly contributed to the data acquisition and control scheme of the pCT.

The most significant achievements of this work, in no particular order, are:

- The conceptual design of the readout and control system.

    The conceptual DAQ and control system presented in the thesis has been adopted and will be applied in the completed pCT DTC prototype. The concept has impacted all significant parts of the detector, except the mechanical, and must be considered as a crucial element. As a result of this work, the design and development of the DAQ and DCS software and the FPGA firmware could be initiated.

- The design and testing of the detector front-end.

    In participation with the Kharkiv-team and their state-of-the-art technology, this work has provided valuable insight regarding the front-end design. With every iteration, systematic measurements and simulations were performed to detect weaknesses and potential improvements in the design. As it stands, the resulting front-end design adheres to both (1) the physical requirements of the detector in terms of homogeneity, material budget, and both convective and conductive heat transfer, and (2) the electronic characteristics requirements.

- Implementation and verification of the DAQ and DCS FPGA firmware.

  While DAQ systems and data flow FPGA designs are rarely scientifically novel, this thesis demonstrates some of the challenges that arise with custom electronics. Full implementation of the data chain has been completed, including sampling with DPA, simple compression, error checking, and packing. For the transmission of data from the readout electronics to the data server, a new data format has been developed, together with a completely new data transfer protocol to ensure reliable UDP communication. With a reasonable degree of code-reuse, the control architecture is implemented as a fully low-level system with no utilization of a soft-processing system. The latter design decision transferred the responsibility of the ALPIDE control and monitoring to the high-level software development but increased both the flexibility and the speed of the system. Considerable effort has been made in designing the DAQ and the DCS firmware to be adaptable to various scenarios and configurable from the control room.

- The design of test equipment and procedures for the manufacturing process of the front-end.

  The ALPIDE sensors and the front-end components are sensitive. This work also included considerable efforts with the design and development of comprehensive test equipment and procedures to ensure a sensible yield during the manufacturing process.

Due to the nature of a research project, in general, it is often hard, sometimes impossible, to provide precise requirements for the parts that are developed at the start of the project. During development, therefore, we have made both good and not so good, educated guesses about the requirements needed to push the development further while the accurate answers remained unclear. Naturally, some of the development choices can be criticized.

The responsibilities of the complete electronics system have been left to a relatively small group of people, some of whom also worked on other time-consuming projects. Therefore, parts of the development process were significantly slowed down; this included the work with the pRU. Furthermore, other non-design related efforts have been required, like the development and maintenance of the PTB, the PTB firmware, and the test software suite.

Because of the high ambition to interface a large number of sensors with a single FPGA, regular I/O-pins are used to sample the ALPIDE data. With this design decision, significant time and effort were needed to design and test the approach. Although successful, with a BER as low as $< 3.5 \times 10^{-15}$, one can assume that the development time was considerably increased compared to an MGT approach design.

8.2 Outlook

As the FPGA firmware is not yet tested with the final pRU, significant time has been invested in the system documentation. The firmware is also built with modularity in mind, making it adaptable to any potential changes to the system requirements. This should make it possible to finalize the pRU firmware without too much effort.

The Power Control Unit design is nearing completion. However, its firmware must be developed. This firmware is trivial in comparison to the pRU firmware and will primarily be based on the power control module used with the PTB.

A few open questions regarding the pRU firmware remain unanswered. First, the size of the data buffers and the distribution of buffer size between the channels and the pipeline branches can be further optimized. The optimization might use the buffer usage statistics from a scanning mode beam test of a full DTC layer. Second, is the decision of using local clocks or a master clock for synchronization between the pRUs. This decision can be made after experiments testing the performance of the pRU and by measuring the data processing power required to synchronize data frames occurring in semi-autonomous mode. No major changes to the pRU firmware are foreseen.

The preparation for the mass-production of the pCT front-end components is now in progress. The produced chip cables, strings, and slabs will be tested with the PTB and the mini transition card, as described in Section 7.5. The complete layers will be tested when the pRU development is finalized. The schematic and layout design of the pRU is now in progress, and the board is expected to be tested and in use in 2021. This leaves at least a full year for testing and integration of hardware, firmware, and software. Within the end of 2022, it is projected that the detector is operational.

APPENDIX A

List of Publications

A.1 As Primary Author

O. S. Groettvik et al., “Development of Readout Electronics for a Digital Tracking Calorimeter,” in Proceedings of Topical Workshop on Electronics for Particle Physics — PoS(TWEPP2019), 2020, vol. 370, p. 090.
DOI: 10.22323/1.370.0090

A.2 Publications Significantly Contributed To

J. Alme et al., “A High-Granularity Digital Tracking Calorimeter Optimized for Proton CT,” in Frontiers of Physics, 2020.
DOI: 10.3389/fphy.2020.568243

A.3 Master’s Theses as Co-Supervisor

K. E. S. Bohne, “Ethernet-Based Control System and Data Readout for a Proton Computed Tomography Prototype,” Master’s thesis, The University of Bergen, jun 2018. [Online]. Available: http://hdl.handle.net/1956/18466

H. A. Underdal, “Data Acquisition and Testing Software for a Proton Computed Tomography System,” Master’s thesis, jun 2019. [Online]. Available: http://hdl.handle.net/1956/20846

Ø. Jelmert, “Scalable Readout for Proton CT,” Master’s thesis, jul 2020.

T. Bodova, “High-Speed Signal and Power Distribution of a Digital Tracking Calorimeter for Proton Computed Tomography,” Master’s thesis, University of Bergen, aug 2020. [Online]. Available: http://hdl.handle.net/1956/23535

A. Herland, “Development and Implementation of Data Acquisition Software for proton Computed Tomography,” Master’s thesis, 2021.

A.4 All Publications

G. Tambave, et al., “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, p.162626, aug 2019. DOI: 10.1016/j.nima.2019.162626

J. Rambo Sølie, L. Voltz, et al., “Image quality of list-mode proton imaging without front trackers,” Phys. Med. Biol., vol. 65, no. 13, p. 135012, jul 2020. DOI: 10.1088/1361-6560/ab8ddb

H. E. S. Pettersen, et al., “Design optimization of a pixel-based range telescope for proton computed tomography,” Physica Medica, vol. 63, pp.87–97, jul 2019. DOI: 10.1016/j.ejmp.2019.05.026

H. Pettersen, et al., “Helium Radiography with a Digital Tracking Calorimeter—a Monte Carlo Study for Secondary Track Rejection,” in review.

APPENDIX B

pRU Data Format

This chapter lists and explains the data fields of the pRU data format. To illustrate how the words are combined a short example frame sequence is shown.

B.1 The `HEADER` Word

ABS_TIME The counter of the 120 MHz system clock sampled at the moment the pRU transmits a trigger command to the ALPIDEs.

FRAME_ID An unique ID of the current frame. The number is associated with a specific trigger command transmitted from the ALPIDE Control module. Any other frame with matching ID and matching time information from another channel belongs to the same strobe window.

MODE The readout mode the ALPIDEs are configured in. This information is not required for the data reconstruction but might be useful to determine the nature behind any busy-conditions of the ALPIDE. Note that this information is not obtained automatically by the data formatter, but must be manually set in the pRU settings register ahead of starting the data taking sequence. If the bit is low it indicates that the ALPIDEs are set in `TRIGGERED` mode, while if it is high they are set in `CONTINUOUS` mode.

TRIG_SOURCE The source of the ALPIDE trigger signal. This is also a nonessential data field, but is useful for debugging any synchronization procedure of the sensors.

0x0 ALPIDE Internal Strobe Sequencer
0x1 External pRU Hardware Signal
0x2 DCS Trigger

BUSY_OFF Indicates that an ALPIDE busy off word was received in the time between the previous and the current frame.

BUSY_ON Indicates that an ALPIDE busy on word was received in the time between the previous and the current frame.

SPILL_ID The current value of the SPILL_ID register. The SPILL_ID is an arbitrary number for the DAQ system, but it is intended to be used to separate frames from different 2D projections.

DATA_FORMAT The data format version. Any changes to the data format specification will cause this number to be incremented. Any software written to interpret pRU data can use this field to determine which fields are available.

B.2 The `TRAILER` Word

FRAME_SIZE A critical metric to check that no errors have occurred during the processing and transmission of the data is the amount of ALPIDE data bytes transmitted in the preceding pRU data words. If this field matches the above ALPIDE data bytes *and* no protocol irregularities were found, one can be confident of the data coherency.

FRAME_ID An unique ID of the current frame. The number is associated with a specific trigger command transmitted from the ALPIDE Control module. Any other frame with matching ID and matching time information from another channel belongs to the same strobe window.

ERROR_FLAGS This field contains certain flags that indicate whether any errors have occurred during transmission of the frame on the pRU. The following list describes each bit within the field starting from the LSB:

0 Decode/Protocol Error This flag is asserted whenever the 8B/10B decoder has been unable to decode a byte during the processing of the frame. However, the processing of the frame does continue and the remaining bytes of the frame will still be included. The flag is also asserted whenever other protocol errors are observed. Any frame with this flag set might be salvageable by identifying which byte, and thus which pixels, are corrupted. By removing these, the remaining pixel data can still be used for track reconstruction. This check, however, requires significant efforts by the DAQ software and whether this is applicable depends on the rate of frames with these errors.

1 Frame Error Asserted whenever a fatal error occurred during the processing of the frame. This error causes the frame processing to be aborted and instantly produces the trailer.

2 Empty Region Error Asserted when a REGION identifier is detected but no short or long words come directly after.

3 Double Busy On Error Asserted when two BUSY ON is detected, without a BUSY OFF in between.

4 Double Busy Off Error Asserted when two BUSY OFF is detected, without a BUSY ON in between.

5 Buffer Overflow Error Asserted whenever a pRU buffer overflow has occurred.

6 Max Size Error Asserted whenever the size of the frame is over the pre-defined maximum bytes. This type of error is usually caused by an error in detecting the ALPIDE trailer. The error will cause a cancellation of the frame processing and forces a pRU trailer word. The maximum number of bytes is configured in the pRU settings registers.

7 Max Wait Time Error Asserted whenever the data tagger has been waiting for more than pre-defined clock cycles (120 MHz) for valid data content during the frame, meaning other words than comma and idle. This error will cause a cancellation of the frame processing and forces a trailer word. The maximum number of clock cycles to wait is configured in the pRU settings registers.

B.3 The `EMPTY` Word

NUM_EMPTY The number of consecutive empty frames that have arrived at the pRU. The field must be added as the data parsing software might not have information about the compression settings on the pRU during readout.

FRAME_ID The frame ID associated with the trigger command for the first empty frame in the sequence.

BUNCH_CNT The bunch counter value of the ALPIDE in the first empty frame in the sequence.

RESERVED Must be zero to separate from potential conflict with a `DELIMITER` word only used in the PTB.

B.4 Example of a pRU frame

The following is an example of a frame where $pRUID = 2$, $StaveID = 10$, $ChipID = 3$, $SpillID = 300$, $TriggerSource = Software$, $Mode = TRIGGERED$, $FrameID = 25000$, $ABS\_TIME = 500.000.000$:

Name	WORD_TYPE	RU	STAVE	CHIPID	UNUSED	SPILL_ID	TRIG_SOURCE	MODE	FRAME_ID	ABS_TIME
Value	0x1	0x2	0xA	0x3	0x0	0x12C	0x2	0x0	0x61A8	0x1DCD_6500

Name	WORD_TYPE	RU	STAVE	CHIPID	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13
Value	0x0	0x2	0xA	0x3	HDR	HDR	RGN	DS	DS	DL	DL	DL	RGN	DL	DL	DL	RGN	DS

Name	WORD_TYPE	RU	STAVE	CHIPID	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13
Value	0x0	0x2	0xA	0x3	DS	RGN	DS	DS	RGN	DS	DS	RGN	DL	DL	DL	TRLR	0xFF	0xFF

Name	WORD_TYPE	RU	STAVE	CHIPID	UNUSED	ERROR_FLAGS	FRAME_SIZE
Value	0x2	0x2	0xA	0x3	0x0	0x0	0x1A

Two of the data fields are padded in the second DATA_WORD, and this is reflected in the FRAME_SIZE field in the trailer. $HDR = Header$, $RGN = RegionHeader$, $DS = DataShort$, $DL = DataLong$, $TRLR = Trailer$.

APPENDIX C

pCT Data Transfer Protocol

This chapter details the specific requests and responses defined by the pDTP. In addition, the flowcharts of the specific blocks of the pDTP are given.

C.1 pDTP Client

Table C.1: *pDTP Client Header.*

<table>
<tr><td>Offsets</td><td>Octet</td><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td>Octet</td><td>Bit</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>0</td><td>0</td><td colspan="32">IPv4 (20 Octets)</td></tr>
<tr><td>20</td><td>160</td><td colspan="32">UDP (8 Octets)</td></tr>
<tr><td>28</td><td>224</td><td colspan="4">DTP Client Opcode</td><td colspan="4">Flags</td><td colspan="16">DTP Special Commands</td><td colspan="8">Requested DTP Packet Size - in pRU words</td></tr>
</table>

Table C.2: *pDTP Client Opcodes.*

Opcode Name	Opcode	Description
CLIENT_RQR	0x0	Client request the server to transmit a packet of a certain amount of pRU words
CLIENT_RQT	0x1	Client request the server to transmit a test packet of a certain amount of pRU words
CLIENT_RQS	0x2	Client request the server to transmit a certain number of packets of a certain amount of pRU words
CLIENT_RQFS	0x3	Client requests the server to continuously transmit packets as data gets available
CLIENT_ERROR	0x4	Timeout while waiting for packet, or some other error
CLIENT_ACK	0x5	Acknowledge to the server that the last packet were obtained
CLIENT_ABRT	0x6	Instructs the server to abort the current operation, e.g. CLIENT_RQFS
CLIENT_GS	0x7	Ask the server to transmits its buffer status, firmware information, and absolute clock
CLIENT_THROTTLE	0x8	Ask the server to throttle the output stream, by waiting for a certain number of clk cycles between each stream packet

Table C.3: *pDTP Client Special Commands.*

<table>
<tr><td>Opcode / Bit</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>CLIENT_RQR</td><td>NO_ACK</td><td>MIN_RQ</td><td>MAXIMIZE</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="8">RQ_PACKET_SIZE</td></tr>
<tr><td>CLIENT_RQT</td><td>NO_ACK</td><td colspan="19"></td><td colspan="8">RQ_PACKET_SIZE</td></tr>
<tr><td>CLIENT_RQS</td><td></td><td>MIN_RQ</td><td>MAXIMIZE</td><td>NO_WAIT</td><td colspan="16">RQ_STREAM_SIZE</td><td colspan="8">RQ_PACKET_SIZE</td></tr>
<tr><td>CLIENT_RQSFS</td><td></td><td>MIN_RQ</td><td>MAXIMIZE</td><td>NO_WAIT</td><td colspan="16"></td><td colspan="8">RQ_PACKET_SIZE</td></tr>
<tr><td>CLIENT_ERROR</td><td></td><td>UNINTERPRETABLE</td><td>TIMEOUT</td><td>RESEND_PACKET</td><td colspan="24"></td></tr>
<tr><td>CLIENT_ACK</td><td colspan="28"></td></tr>
<tr><td>CLIENT_ABRT</td><td colspan="28"></td></tr>
<tr><td>CLIENT_GS</td><td colspan="28"></td></tr>
<tr><td>CIENT_THROTTLE</td><td colspan="4"></td><td colspan="24">WAIT_CYCLES</td></tr>
</table>

NO_ACK Server will not wait for an acknowledge from the client

RQ_PACKET_SIZE The number of pRU words requested in each packet - max 255 pRU words, i.e. 4 kB.

RQ_STREAM_SIZE Number of packets to transmit without listening for acknowledge - max $2^{16} - 1 = 65.535$ packets.

MIN_RQ Will not send a packet if the buffer has less than RQ_PACKET_SIZE. Pull: Server will send an ERROR. Semi push: Server will transmit an EOS. Full push: Server will wait until buffer is filled with required amount.

MAXIMIZE Maximize the number of pRU words transmitted, based on the data available. Can be used together with MIN_RQ.

NO_WAIT Server will not wait for the buffer to be filled with data if the buffer is empty or is filled with less than MIN_RQ. Result is EOS in both RQS and RQFS.

UNINTERPRETABLE The received packet was not possible to understand

TIMEOUT The client timed out while waiting for something

RESEND_PACKET Ask the server to resend the last packet - only available after CLIENT_RQR

WAIT_CYCLES The number of clock cycles between each stream packet transmitted from the server.

C.2 pDTP Server

Table C.4: *pDTP Server Header.*

<table>
<tr><td>Offsets</td><td>Octet</td><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td>Octet</td><td>Bit</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>0</td><td>0</td><td colspan="32">IPv4 (20 Octets?)</td></tr>
<tr><td>20</td><td>160</td><td colspan="32">UDP (8 Octets)</td></tr>
<tr><td>28</td><td>224</td><td colspan="4">DTP Server Opcode</td><td colspan="4">FLAGS</td><td colspan="16">DTP Packet ID / Buffer Fill Count</td><td colspan="8">Actual DTP Packet Size / Version Number</td></tr>
<tr><td>32</td><td>256</td><td colspan="32">ABS_TIME (System Clock Cycles)</td></tr>
<tr><td>36</td><td>288</td><td colspan="32">Payload (0 - 255 pRU words)</td></tr>
</table>

Table C.5: *pDTP Server Opcodes.*

Opcode Name	Opcode	Short Description	Long Description
SERVER_WRITE	0x0	Server Write Packet	Server sends a packet
SERVER_STREAM	0x1	Server Stream Packet	A part of a stream of packets
SERVER_ERROR	0x2	Server Error	Timeout while waiting for ack, uninterpretable received packet or no data available
SERVER_EOS	0x3	Server End-Of-Stream	Server is finished transmitting all possible packets
SERVER_STATUS	0x4	Server Status	Server Status

Table C.6: *pDTP Server Special Commands.*

<table>
<tr><td>Opcode / Bit</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>SERVER_WRITE</td><td>FULL</td><td>ALMOST_FULL</td><td></td><td></td><td colspan="16">PACKET_ID</td><td colspan="8">AC_PACKET_SIZE</td></tr>
<tr><td>SERVER_STREAM</td><td>FULL</td><td>ALMOST_FULL</td><td></td><td></td><td colspan="16">PACKET_ID</td><td colspan="8">AC_PACKET_SIZE</td></tr>
<tr><td>SERVER_ERROR</td><td>INVALID_RQ</td><td>MIN_RQ</td><td>EMPTY</td><td>TIMEOUT</td><td colspan="16">BUFFER_FILL_COUNT</td><td colspan="8"></td></tr>
<tr><td>SERVER_EOS</td><td></td><td>MIN_RQ</td><td colspan="2"></td><td colspan="16">BUFFER_FILL_COUNT</td><td colspan="8"></td></tr>
<tr><td>SERVER_STATUS</td><td>FULL</td><td>ALMOST_FULL</td><td>EMPTY</td><td></td><td colspan="16">BUFFER_FILL_COUNT</td><td colspan="8">VERSION_NUMBER</td></tr>
</table>

PACKET_ID Incremented counter of all transmitted packets

FULL The offload data buffer is full

ALMOST_FULL The offload data buffer has less than 20% space left

EMPTY The offload data buffer is empty

TIMEOUT Timeout while waiting for something or uninterpretable

NO_DATA No data was available on the server when receiving the request

BUFFER_FILL_COUNT Amount of pRU words stored in offload buffer

INVALID_RQ Client asked about something unknown to the server

VERSION_NUMBER Current version of pDTP protocol

C.2.1 Server Status

Table C.7: *pDTP Server Status Packet.*

<table>
<tr><td>Offsets</td><td>Octet</td><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td>Octet</td><td>Bit</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>0</td><td>0</td><td colspan="32">IPv4 (20 Octets)</td></tr>
<tr><td>20</td><td>160</td><td colspan="32">UDP (8 Octets)</td></tr>
<tr><td>28</td><td>224</td><td colspan="4">SERVER_STATUS</td><td>FULL</td><td>ALMOST_FULL</td><td>EMPTY</td><td></td><td colspan="16">BUFFER_FILL_COUNT</td><td colspan="8">VERSION_NUMBER</td></tr>
<tr><td>32</td><td>256</td><td colspan="32">ABS_TIME (System Clock Cycles)</td></tr>
<tr><td>36</td><td>288</td><td colspan="32">BUILD_DATE (YYMMDDHH)</td></tr>
<tr><td>40</td><td>320</td><td colspan="32">GIT_HASH</td></tr>
</table>

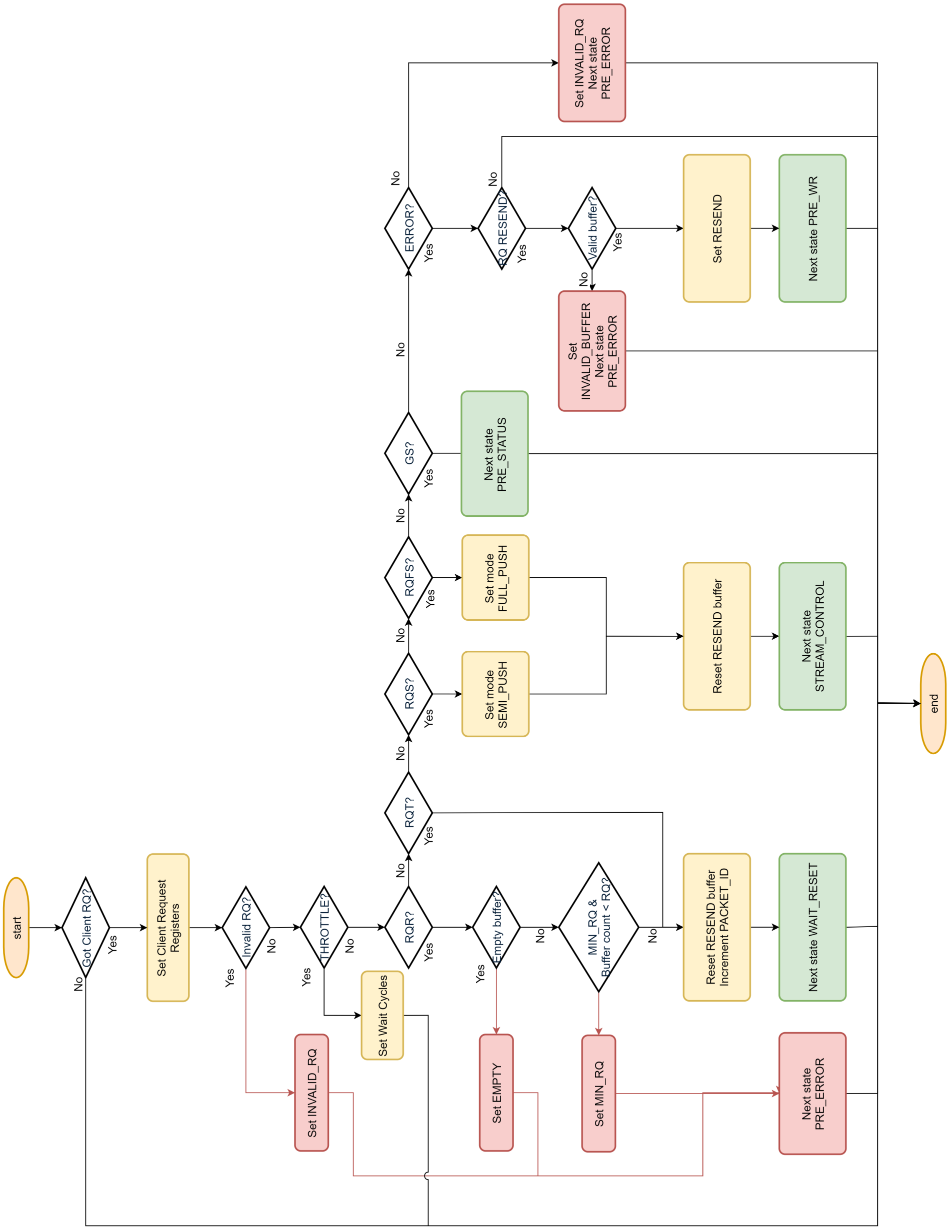


Figure C.1: *Flowchart of the IDLE state.*

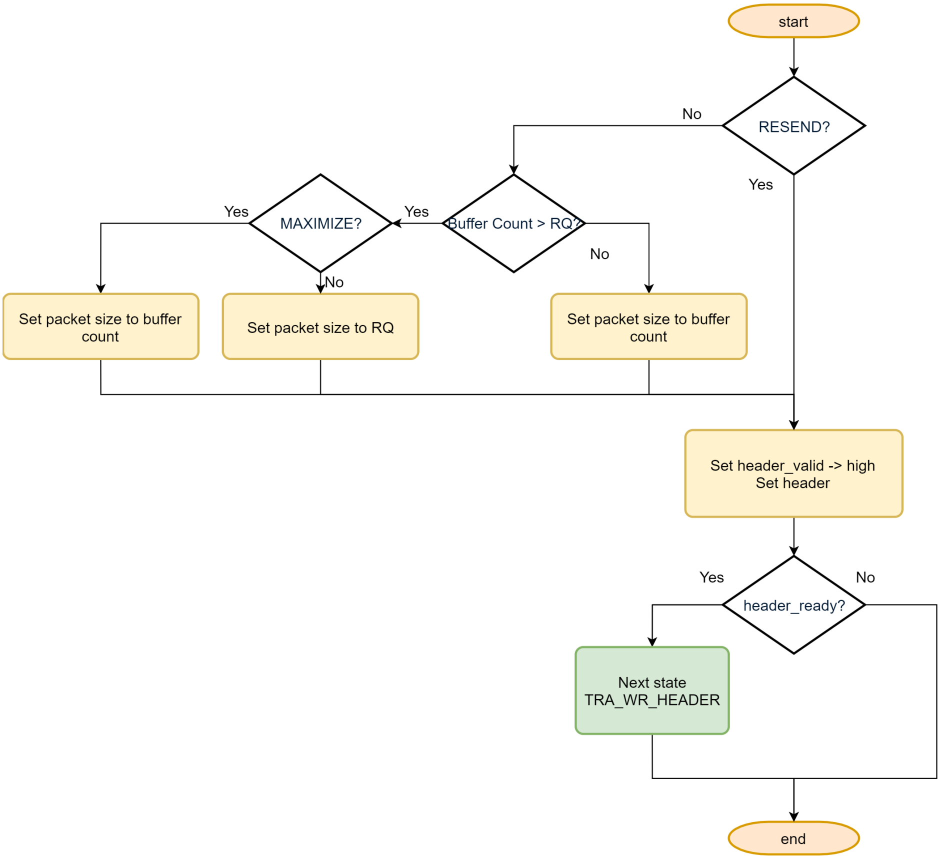


Figure C.2: *Flowchart of the PRE_WR state.*

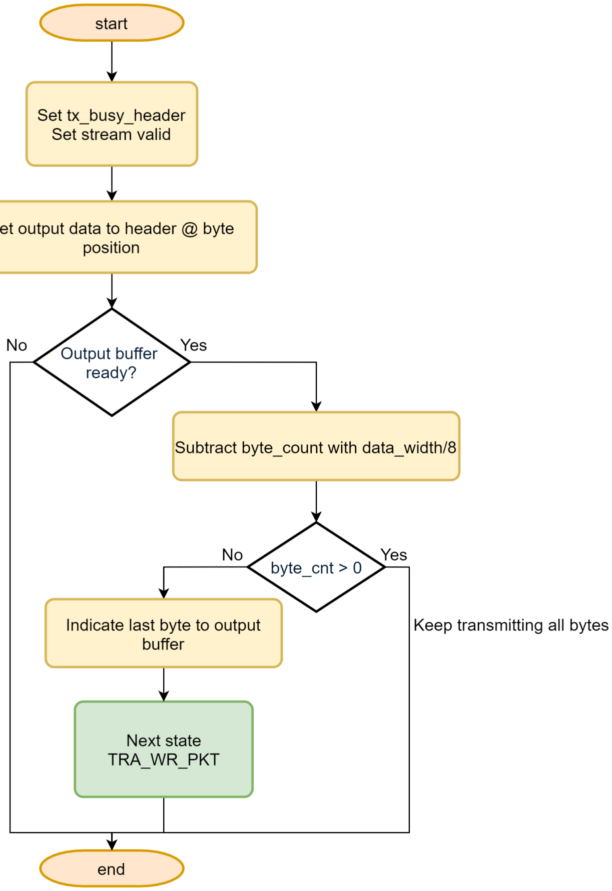


Figure C.3: *Flowchart of the TRA_WR_HEADER state.*

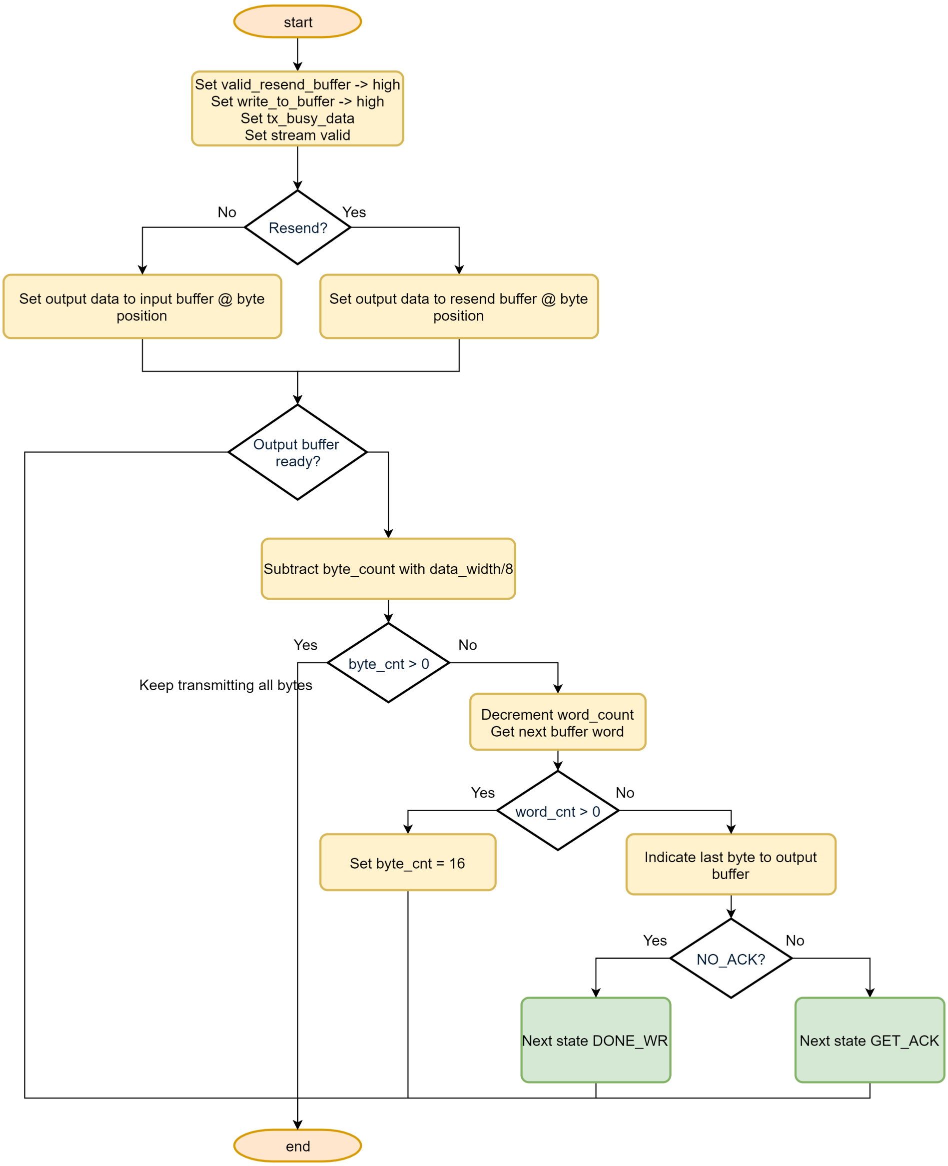


Figure C.4: *Flowchart of the TRA_WR_PKT state.*

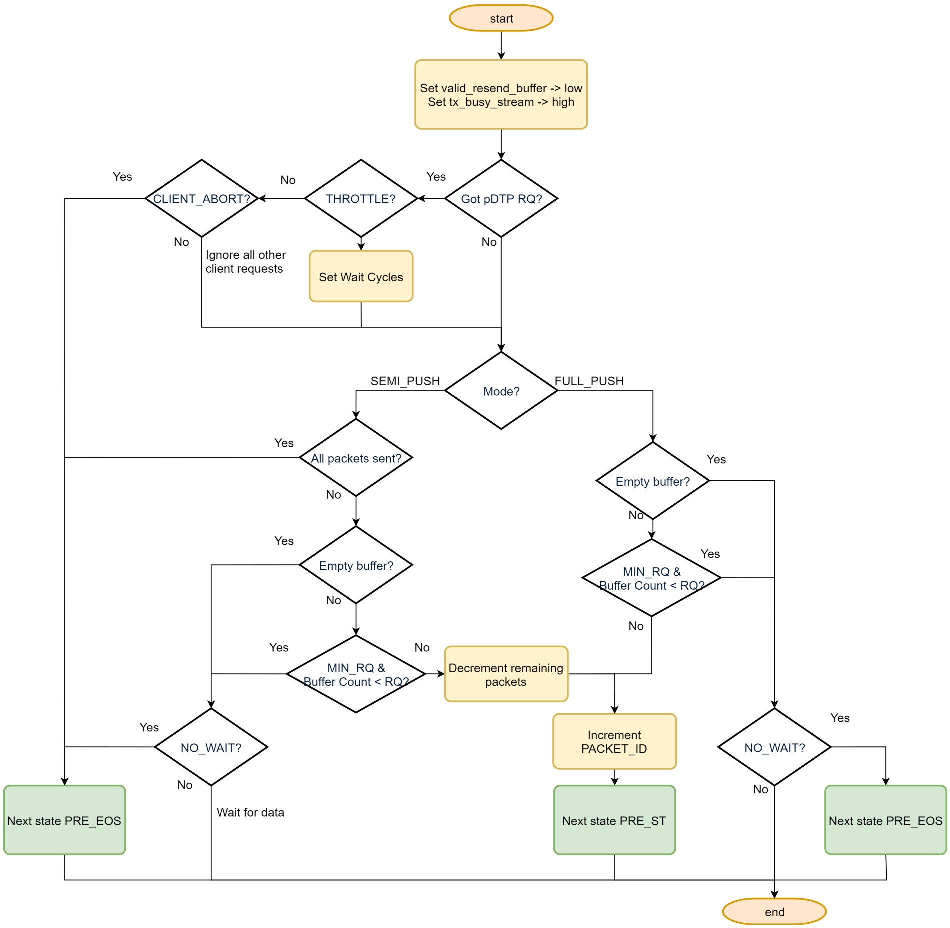


Figure C.5: *Flowchart of the STREAM_CONTROL state.*

Bibliography

[1] M. Jermann, “Particle Therapy Patient Statistics (per end of 2016),” PTCOG, Tech. Rep., 2017. [Online]. Available: https://bit.ly/3iNQmx5

[2] E. Dale and E. Waldeland, “Protonterapi – en realitet i Norge fra 2023,” Tidsskrift for Den norske legeforening, vol. 138, no. 13, sep 2018. DOI: 10.4045/tidsskr.18.0250

[3] T. Mitin and A. L. Zietman, “Promise and Pitfalls of Heavy-Particle Therapy,” Journal of Clinical Oncology, vol. 32, no. 26, pp. 2855–2863, sep 2014. DOI: 10.1200/JCO.2014.55.1945

[4] H. E. S. Pettersen, “A Digital Tracking Calorimeter for Proton Computed Tomography,” Ph.D. dissertation, University of Bergen, 2018. [Online]. Available: http://hdl.handle.net/1956/17757

[5] M. Filipak, “Comparison of dose profiles for proton v. x-ray radiotherapy,” 2012. [Online]. Available: https://bit.ly/33vGUsf

[6] H. Lynnebakken, “Jubler for partikkelterapi,” 2012. [Online]. Available: https://bit.ly/3iNIDz5

[7] A. J. Lomax, “Myths and realities of range uncertainty,” The British Journal of Radiology, vol. 93, no. 1107, p. 20190582, mar 2020. DOI: 10.1259/bjr.20190582

[8] M. Brada, M. Pijls-Johannesma, and D. De Ruysscher, “Proton Therapy in Clinical Practice: Current Clinical Evidence,” Journal

of Clinical Oncology, vol. 25, no. 8, pp. 965–970, mar 2007. DOI: 10.1200/JCO.2006.10.0131

[9] B. Schaffner and E. Pedroni, "The precision of proton range calculations in proton radiotherapy treatment planning: experimental verification of the relation between CT-HU and proton stopping power," Physics in Medicine and Biology, vol. 43, no. 6, pp. 1579–1592, jun 1998. DOI: 10.1088/0031-9155/43/6/016

[10] G. Poludniowski, N. M. Allinson, and P. M. Evans, "Proton radiography and tomography with application to proton therapy," The British Journal of Radiology, vol. 88, no. 1053, p. 20150134, sep 2015. DOI: 10.1259/bjr.20150134

[11] J. Rambo Sølie et al., "Image quality of list-mode proton imaging without front trackers," Physics in Medicine & Biology, vol. 65, no. 13, p. 135012, jul 2020. DOI: 10.1088/1361-6560/ab8ddb

[12] H. F. Sadrozinski et al., "Development of a head scanner for proton CT," in Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 699, jan 2013, pp. 205–210. DOI: 10.1016/j.nima.2012.04.029

[13] T. Lomax, "Towards daily adapted proton therapy," Physica Medica, vol. 30, p. e3, jan 2014. DOI: 10.1016/J.EJMP.2014.07.017

[14] M. Esposito et al., "PRaVDA: The first solid-state system for proton computed tomography," Physica Medica, vol. 55, pp. 149–154, nov 2018. DOI: 10.1016/j.ejmp.2018.10.020

[15] H. Pettersen 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, pp. 51–61, jul 2017. DOI: 10.1016/j.nima.2017.02.007

[16] H. E. S. Pettersen et al., “Design optimization of a pixel-based range telescope for proton computed tomography,” Physica Medica, vol. 63, pp. 87–97, jul 2019. DOI: 10.1016/j.ejmp.2019.05.026

[17] J. Alme et al., “A High-Granularity Digital Tracking Calorimeter Optimized for Proton CT,” Frontiers in Physics. DOI: 10.3389/fphy.2020.568243

[18] H. Pettersen et al., “Accuracy of parameterized proton range models; A comparison,” Radiation Physics and Chemistry, vol. 144, pp. 295–297, mar 2018. DOI: 10.1016/j.radphyschem.2017.08.028

[19] G. Tambave et al., “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, p. 162626, aug 2019. DOI: 10.1016/j.nima.2019.162626

[20] W. Snoeys, “CMOS monolithic active pixel sensors for high energy physics,” Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 765, pp. 167–171, nov 2014. DOI: 10.1016/J.NIMA.2014.07.017

[21] R. Turchetta et al., “CMOS Monolithic Active Pixel Sensors (MAPS): New ‘eyes’ for science,” Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 560, no. 1, pp. 139–142, may 2006. DOI: 10.1016/J.NIMA.2005.11.241

[22] G. Contin et al., “The STAR MAPS-based PiXeL detector,” Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 907, pp. 60–80, nov 2018. DOI: 10.1016/j.nima.2018.03.003

[23] C. Sauer, “Monolithic Active Pixel Sensors Particle tracking and identification at high rates,” Tech. Rep., 2017. [Online]. Available: https://bit.ly/2Y1Sutg

[24] H. Augustin et al., "MuPix7—A fast monolithic HV-CMOS pixel chip for Mu3e," Journal of Instrumentation, vol. 11, no. 11, pp. C11 029–C11 029, nov 2016. DOI: 10.1088/1748-0221/11/11/C11029

[25] T. Kugathasan, "Review on depleted CMOS," in Proceedings of Science, vol. 348. Sissa Medialab Srl, sep 2018, p. 042. DOI: 10.22323/1.348.0042

[26] B. Ristic, "CMOS Pixel Development for the ATLAS Experiment at HL-LHC," Springer Proc. Phys., vol. 213, pp. 426–430, 2018. DOI: 10.1007/978-981-13-1316-5_80

[27] J. W. van Hoorne, "Study and Development of a novel Silicon Pixel Detector for the Upgrade of the ALICE Inner Tracking System," Ph.D. dissertation, Technische Universität Wien, 2015. [Online]. Available: https://cds.cern.ch/record/2119197

[28] G. Aglieri Rinella, "The ALPIDE pixel sensor chip for the upgrade of the ALICE Inner Tracking System," Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 845, pp. 583–587, feb 2017. DOI: 10.1016/j.nima.2016.05.016

[29] The ALICE Collaboration, "ALPIDE Operations Manual," Tech. Rep., 2016.

[30] The ALICE Collaboration, "Conceptual Design Report for the Upgrade of the ALICE ITS," Tech. Rep., 2012. [Online]. Available: https://cds.cern.ch/record/1431539/files/LHCC-G-159.pdf

[31] V. Bezhenova and A. Michalowska-Forsyth, "Aspect ratio of radiation-hardened MOS transistors," e & i Elektrotechnik und Informationstechnik, vol. 135, no. 1, pp. 61–68, feb 2018. DOI: 10.1007/s00502-017-0575-2

[32] W. Snoeys, T. Gutierrez, and G. Anelli, "A new NMOS layout structure for radiation tolerance," IEEE Transactions on Nuclear Science, vol. 49, no. 4, pp. 1829–1833, aug 2002. DOI: 10.1109/TNS.2002.801534

[33] B. Abelev et al, "Technical Design Report for the Upgrade of the ALICE Inner Tracking System," Journal of Physics G: Nuclear and Particle Physics, vol. 41, no. 8, p. 087002, aug 2014. DOI: 10.1088/0954-3899/41/8/087002

[34] F. Reidt, "Studies for the ALICE Inner Tracking System Upgrade," 2016.

[35] G. A. Rinella, "The ALPIDE Pixel Sensor Chip for the Upgrade of the ALICE Inner Tracking System," ALICE Collaboration, Tech. Rep., feb 2016. [Online]. Available: https://bit.ly/3dDTzOK

[36] G. Aglieri et al., "Monolithic active pixel sensor development for the upgrade of the ALICE inner tracking system," Journal of Instrumentation, vol. 8, no. 12, pp. C12 041–C12 041, 2013. DOI: 10.1088/1748-0221/8/12/C12041

[37] The ALICE Collaboration, "Alice ITS Upgrade: Readout Electronic - WP10," CERN, Tech. Rep., 2016.

[38] P. A. Franaszek and A. X. Widmer, "Byte oriented DC balanced (0,4) 8B/10B partitioned block transmission code," 1984. [Online]. Available: https://bit.ly/3218Nbc

[39] T. Bodova, "High-Speed Signal and Power Distribution of a Digital Tracking Calorimeter for Proton Computed Tomography," Master's thesis, University of Bergen, aug 2020. [Online]. Available: http://hdl.handle.net/1956/23535

[40] V. Borshchov et al., "Innovative microelectronic technologies for high-energy physics experiments," Functional Materials, vol. 24, no. 1, pp. 143–153, 2017. DOI: 10.15407/fm24.01.143

[41] V. Borshchov et al., "Aluminium Microcable Technology for the Alice Silicon Strip Detector: A Status Report," in 8th Workshop on Electronics for LHC Experiments, 2002, pp. 144—-149. DOI: 10.5170/CERN-2002-003.144

[42] M. Oinonen et al., "Alice Silicon Strip Detector Module Assembly with Single-Point TAB Interconnections," 2005. [Online]. Available: http://cdsweb.cern.ch/record/920152

[43] E. Bogatin, Signal and Power Integrity - Simplified, 2nd ed. Prentice Hall, 2009.

[44] Microwaves101, "Transmission Line Model." [Online]. Available: https://bit.ly/3fXDxPe

[45] Microwaves101, "Transmission Line Loss." [Online]. Available: https://bit.ly/2DYhNWh

[46] Microwaves101, "Microstrip Loss Calculations." [Online]. Available: https://bit.ly/3g6Ozlm

[47] A. Athavale and C. Christensen, High-Speed Serial I/0 Made Simple, 1st ed., 2005. [Online]. Available: https://bit.ly/2E9h2K0

[48] Xilinx Inc., "UltraScale+ FPGA Product Tables and Product Selection Guide," 2017. [Online]. Available: https://bit.ly/33XVFGi

[49] Xilinx Inc., "UltraScale FPGA Product Tables and Product Selection Guide," 2016. [Online]. Available: https://bit.ly/2Y4hIr2

[50] M. Smerdon and Xilinx Inc., "Spartan-6 FPGAs: Performance, Power, and I/O Optimized for Cost-Sensitive Applications (WP396)," dec 2017. [Online]. Available: https://bit.ly/3hc2sQc

[51] Xilinx Inc., "Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics," Tech. Rep., 2013. [Online]. Available: https://bit.ly/3c05TaZ

[52] Xilinx Inc., "Native High-Speed I/O Interfaces Application Note (XAPP1274)," Tech. Rep., 2017. [Online]. Available: https://bit.ly/3fXEIOE

[53] C. E. Cummings, "Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog," in SNUG Boston, 2008.

[54] M. Litterick, “Pragmatic Simulation-Based Verification of Clock Domain Crossing Signals and Jitter using SystemVerilog Assertions,” in DVCon, 2006.

[55] T. Chelcea and S. M. Nowick, “Robust interfaces for mixed-timing systems,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, no. 8, pp. 857–873, aug 2004. DOI: 10.1109/TVLSI.2004.831476

[56] D. M. Hiemstra, V. Kirischian, and J. Brelski, “Single event upset characterization of the Kintex UltraScale field programmable gate array using proton irradiation,” in IEEE Radiation Effects Data Workshop, vol. 0. Institute of Electrical and Electronics Engineers Inc., jul 2016. DOI: 10.1109/NSREC.2016.7891743

[57] K. Røed, “Single Event Upsets in SRAM FPGA based readout electronics for the Time Projection Chamber in the ALICE experiment,” Ph.D. dissertation, The University of Bergen, dec 2009. [Online]. Available: http://hdl.handle.net/1956/3793

[58] K. Røed et al., “Fault injection as a test method for an FPGA in charge of data readout for a large tracking detector,” Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 629, no. 1, pp. 260–268, feb 2011. DOI: 10.1016/j.nima.2010.12.033

[59] M. R. Ersdal, “External scrubber implementation for the ALICE ITS Readout Unit,” in Proceedings of Topical Workshop on Electronics for Particle Physics — PoS(TWEPP2019). Trieste, Italy: Sissa Medialab, mar 2020, p. 136. DOI: 10.22323/1.370.0136

[60] C. Ramamurthy, S. Chellappa, and L. T. Clark, “Physical Design Methodologies for Soft Error Mitigation Using Redundancy,” in 2015 15th European Conference on Radiation and Its Effects on Components and Systems (RADECS). IEEE, 2015, pp. 1–5. DOI: 10.1109/RADECS.2015.7365647

[61] M. Berg, “FPGA Mitigation Strategies for Critical Applications,” NASA, Tech. Rep., nov 2018. [Online]. Available: https://go.nasa.gov/30ZmwQv

[62] A. Velure, “Integration, Commissioning and First Experience of ALICE ITS Control and Readout Electronics,” in Proceedings of Topical Workshop on Electronics for Particle Physics — PoS(TWEPP2019). Trieste, Italy: Sissa Medialab, mar 2020, p. 113. DOI: 10.22323/1.370.0113

[63] P. Moreira et al., “The GBT Project,” Topical Workshop on Electronics for Particle Physics, 2009. DOI: 10.5170/CERN-2009-006.342

[64] Altera, “The Need for Dynamic Phase Alignment in High-Speed FPGAs,” San Jose, CA, Tech. Rep., feb 2004. [Online]. Available: https://intel.ly/3104Ggg

[65] N. H. E. Weste and D. M. Harris, Integrated circuit design. Pearson, 2011.

[66] O. S. Grøttvik, “Design of High-Speed Digital Readout System for Use in Proton Computed Tomography,” Master’s thesis, The University of Bergen, jun 2017. [Online]. Available: http://hdl.handle.net/1956/16041

[67] Xilinx Inc., “7 Series FPGAs SelectIO Resources User Guide (UG471),” 2016. [Online]. Available: https://bit.ly/30Z6pCp

[68] Chuck Benz, “Chuck Benz’s ASIC/FPGA pages,” 2010. [Online]. Available: http://asics.chuckbenz.com/

[69] Xilinx Inc., “UltraScale Architecture SelectIO Resources User Guide (UG571 v1.12),” 2019. [Online]. Available: https://bit.ly/2PUwQml

[70] K. Mustafa and C. Sterzik, “AC-Coupling Between Differential LVPECL, LVDS, HSTL, and CML,” Texas Instruments, Tech. Rep., 2007. [Online]. Available: https://bit.ly/3g0F0US

[71] K. Mustafa and C. Sterzik, “DC-Coupling Between Differential LVPECL, LVDS, HSTL, and CM,” Texas Instruments, Tech. Rep., 2003. [Online]. Available: https://bit.ly/3iISciR

[72] Xilinx Inc., “Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics (DS922 v1.15),” 2019. [Online]. Available: https://bit.ly/3h18Dqu

[73] O. S. Groettvik et al., “Development of Readout Electronics for a Digital Tracking Calorimeter,” in Proceedings of Topical Workshop on Electronics for Particle Physics — PoS(TWEPP2019), vol. 370, mar 2020, p. 090. DOI: 10.22323/1.370.0090

[74] G. A. Rinella, “Private Correspondance,” 2020.

[75] A. Forencich, “Verilog Ethernet components for FPGA implementation.” [Online]. Available: https://github.com/alexforencich/verilog-ethernet

[76] K. E. S. Bohne, “Ethernet-Based Control System and Data Readout for a Proton Computed Tomography Prototype,” Master’s thesis, The University of Bergen, jun 2018. [Online]. Available: http://hdl.handle.net/1956/18466

[77] R. Frazier et al., “Software and firmware for controlling CMS trigger and readout hardware via gigabit Ethernet,” Physics Procedia, vol. 37, pp. 1892–1899, 2012. DOI: 10.1016/j.phpro.2012.02.516

[78] C. G. Larrea et al., “IPbus: a flexible Ethernet-based control system for xTCA hardware,” Journal of Instrumentation, vol. 10, no. 02, pp. C02 019–C02 019, feb 2015. DOI: 10.1088/1748-0221/10/02/C02019

[79] ARM, “AMBA AXI and ACE Protocol Specification,” Tech. Rep., 2011. [Online]. Available: https://bit.ly/35f3oyC

[80] OpenCores, “Wishbone B4,” Tech. Rep., 2010. [Online]. Available: https://bit.ly/2T9L7NM

[81] Intel, “Avalon Interface Specification,” Tech. Rep., 2020. [Online]. Available: https://intel.ly/3og9UOu

[82] Bitvis AS, “SBI.” [Online]. Available: https://bitvis.no/portfolio/sbi/

[83] Texas Instruments, “SN65MLVD080 SN65MLVD082 Datasheet,” Tech. Rep., 2005. [Online]. Available: https://www.ti.com/lit/ds/symlink/sn65mlvd080.pdf

[84] L. O. Damm, L. Lundberg, and D. Olsson, “Introducing test automation and test-driven development: An experience report,” in Electronic Notes in Theoretical Computer Science, vol. 116, no. SPEC.ISS., jan 2005, pp. 3–15. DOI: 10.1016/j.entcs.2004.02.090

[85] N. Johnson, “TDD And A New Paradigm For Hardware Verification,” XtremeEDA, Tech. Rep., 2011. [Online]. Available: https://bit.ly/32IE9TZ

[86] E. Tallaksen, “What is UVVM?” aug 2015. [Online]. Available: https://www.linkedin.com/pulse/what-uvvm-espen-tallaksen/

[87] R. Brun and F. Rademakers, “ROOT - An object oriented data analysis framework,” Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 389, no. 1-2, pp. 81–86, apr 1997. DOI: 10.1016/S0168-9002(97)00048-X

[88] MongoDB Inc., “The MongoDB 4.4 Manual,” 2020. [Online]. Available: https://docs.mongodb.com/manual/

Graphic design: Communication Division, UiB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230851814 (print)
9788230849491 (PDF)